FINITE-STATE MACHINE OPTIMIZATION

©Giovanni De Micheli

Stanford University
Outline

- Modeling synchronous circuits:
  - *State-based* models.
  - *Structural* models.

- State-based optimization methods:
  - State minimization.
  - State encoding.
Synchronous Logic Circuits

• Interconnection of:
  
  – Combinational logic gates.

  – Synchronous delay elements:

    * E-T or M-S registers.

• Assumptions:

  – No direct combinational feedback.

  – Single-phase clocking.
Modeling synchronous circuits

- **State-based model:**
  - Model circuits as *finite-state machines*.
  - Represent by *state tables/diagrams*.
  - Apply exact/heuristic algorithms for:
    * *State minimization*.
    * *State encoding*.

- **Structural models:**
  - Represent circuit by synchronous logic network.
  - Apply:
    * *Retiming*.
    * *Logic transformations*. 
State-based optimization

FSM Specification

State Minimization

State Encoding

Combinational Optimization
A set of primary inputs patterns $X$.

A set of primary outputs patterns $Y$.

A set of states $S$.

A state transition function:

$\delta : X \times S \rightarrow S$.

An output function:

$\lambda : X \times S \rightarrow Y$ for Mealy models

$\lambda : S \rightarrow Y$ for Moore models.
State minimization

- Completely specified *finite-state machines*: 
  - No *don't care* conditions.
  - Easy to solve.

- Incompletely specified *finite-state machines*: 
  - Unspecified transitions and/or outputs.
  - Intractable problem.
State minimization
for completely specified FSMs

• Equivalent states:
  – Given any input sequence
    the corresponding output sequences match.

• Theorem:
  – Two states are equivalent iff:
    * they lead to identical outputs and
      their next-states are equivalent.

• Equivalence is transitive:
  – Partition states into equivalence classes.
  – Minimum finite-state machine is unique.
Example

<table>
<thead>
<tr>
<th>INPUT</th>
<th>STATE</th>
<th>N-STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$s_1$</td>
<td>$s_3$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_1$</td>
<td>$s_5$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_2$</td>
<td>$s_3$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_2$</td>
<td>$s_5$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_3$</td>
<td>$s_2$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_3$</td>
<td>$s_1$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_4$</td>
<td>$s_5$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_5$</td>
<td>$s_4$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_5$</td>
<td>$s_1$</td>
<td>0</td>
</tr>
</tbody>
</table>
Example
Algorithm

• Stepwise partition refinement.

• Initially:
  – All states in the same partition block.

• Then:
  – Refine partition blocks.

• At convergence:
  – Blocks identify equivalent states.
Algorithm

- \( \Pi_1 \): States belong to the same block when outputs are the same for any input.

- While further splitting is possible:
  - \( \Pi_{k+1} \): States belong to the same block if they were previously in the same block and their next-states are in the same block of \( \Pi_k \) for any input.
Example

- $\Pi_1 = \{\{s_1, s_2\}, \{s_3, s_4\}, \{s_5\}\}$.

- $\Pi_2 = \{\{s_1, s_2\}, \{s_3\}, \{s_4\}, \{s_5\}\}$.

- $\Pi_2 = \text{is a partition into equivalence classes:} $
  - States $\{s_1, s_2\}$ are equivalent.
Example

minimal finite-state machine

<table>
<thead>
<tr>
<th>INPUT</th>
<th>STATE</th>
<th>N-STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$s_{12}$</td>
<td>$s_3$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_{12}$</td>
<td>$s_5$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_3$</td>
<td>$s_{12}$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_3$</td>
<td>$s_{12}$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_4$</td>
<td>$s_5$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_5$</td>
<td>$s_4$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_5$</td>
<td>$s_{12}$</td>
<td>0</td>
</tr>
</tbody>
</table>
Example
Computational complexity

- Polynomially-bound algorithm.

- There can be at most $|S|$ partition refinements.

- Each refinement requires considering each state:
  
  - Complexity $O(|S|^2)$.

- Actual time may depend upon:
  
  - Data-structures.
  
  - Implementation details.
State minimization for incompletely specified FSMs

- **Applicable** input sequences:
  - All transitions are specified.

- **Compatible** states:
  - Given any applicable input sequence, the corresponding output sequences match.

- Theorem:
  - Two states are compatible iff:
    * they lead to identical outputs
      - (when both are specified)
    * and their next-states are compatible
      - (when both are specified).
State minimization
for incompletely specified FSMs

- Compatibility is not an \textit{equivalency} relation.

- \textit{Minimum finite-state machine} is not \textit{unique}.

- Implication relations make problem intractable.
## Example

<table>
<thead>
<tr>
<th>INPUT</th>
<th>STATE</th>
<th>N-STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$s_1$</td>
<td>$s_3$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_1$</td>
<td>$s_5$</td>
<td>*</td>
</tr>
<tr>
<td>0</td>
<td>$s_2$</td>
<td>$s_3$</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>$s_2$</td>
<td>$s_5$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_3$</td>
<td>$s_2$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_3$</td>
<td>$s_1$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_4$</td>
<td>$s_5$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_5$</td>
<td>$s_4$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_5$</td>
<td>$s_1$</td>
<td>0</td>
</tr>
</tbody>
</table>
Trivial method for the sake of illustration

• Consider all the possible *don’t care* assignments
  
  \(- n \; *don’t \; care* \; imply

  \(* \; 2^n \; completely \; specified \; FSMs.\)

  \(* \; 2^n \; solutions.\)

• Example:
  
  \(- \; Replace \; * \; by \; 1.\)

  \(* \; \Pi = \{\{s_1, s_2\}, \{s_3\}, \{s_4\}, \{s_5\}\}.\)

  \(- \; Replace \; * \; by \; 0.\)

  \(* \; \Pi = \{\{s_1, s_5\}, \{s_2, s_3, s_4\}\}.\)
Compatibility and implications

Example

- Compatible states \(\{s_1, s_2\}\).

- If \(\{s_3, s_4\}\) are compatible:
  
  - then \(\{s_1, s_5\}\) are compatible.

- Incompatible states \(\{s_2, s_5\}\).
Compatibility and implications

- Compatible pairs:
  - \( \{s_1, s_2\} \)
  - \( \{s_1, s_5\} \iff \{s_3, s_4\} \)
  - \( \{s_2, s_4\} \iff \{s_3, s_4\} \)
  - \( \{s_2, s_3\} \iff \{s_1, s_5\} \)
  - \( \{s_3, s_4\} \iff \{s_2, s_4\} \cup \{s_1, s_5\} \)

- Incompatible pairs:
  - \( \{s_2, s_5\}, \quad \{s_3, s_5\} \)
  - \( \{s_1, s_4\}, \quad \{s_4, s_5\} \)
  - \( \{s_1, s_3\} \)
Compatibility and implications

- A class of compatible states is such that all state pairs are compatible.

- A class is maximal:
  - If not subset of another class.

- Closure property:
  - A set of classes such that all compatibility implications are satisfied.

- The set of maximal compatibility classes:
  - Has the closure property.
  - May not provide a minimum solution.
Maximal compatible classes

• \( \{s_1, s_2\} \)

• \( \{s_1, s_5\} \leftarrow \{s_3, s_4\} \)

• \( \{s_2, s_3, s_4\} \leftarrow \{s_1, s_5\} \)

• Cover with MCC has cardinality 3.
Formulation of the state minimization problem

- A class is prime, if not subset of another class implying the same set or a subset of classes.

- Compute the prime compatibility classes.

- Select a minimum number of PCC such that:
  - all states are covered.
  - all implications are satisfied.

- Binate covering problem.
Prime compatible classes

- $\{s_1, s_2\}$

- $\{s_1, s_5\} \leftarrow \{s_3, s_4\}$

- $\{s_2, s_3, s_4\} \leftarrow \{s_1, s_5\}$

- Minimum cover: $\{\{s_1, s_5\}, \{s_2, s_3, s_4\}\}$. 

- Minimum cover has cardinality 2.
Heuristic algorithms

- Approximate the covering problem.
  - Preserve closure property.
  - Sacrifice minimality.

- Consider all maximal compatibility classes.
  - May not yield minimum.
State encoding

- Determine a binary encoding of the states:
  - that optimize machine implementation:
    * area.
    * cycle-time.

- Modeling:
  - Two-level circuits.
  - Multiple-level circuits.
Two-level circuit models

- Sum of product representation.
  - PLA implementation.

- Area:
  - # of products × # I/Os.

- Delay:
  - Twice # of products plus # I/Os.

- Note:
  - # products of a minimum implementation.
  - # I/Os depends on encoding length.
State encoding for two-level models

- Symbolic minimization of state table.

- Constrained encoding problems.
  - Exact and heuristic methods.

- Applicable to large finite-state machines.
Symbolic minimization

- Extension of two-level logic optimization.
- Reduce the number of rows of a table, that can have symbolic fields.
- Reduction exploits:
  - Combination of input symbols in the same field.
  - Covering of output symbols.
State encoding of *finite-state machines*

- Given a (minimum) state table of a *finite-state machine*:
  - find a consistent encoding of the states
    - that preserves the cover minimality
    - with minimum number of bits.
Example

<table>
<thead>
<tr>
<th>INPUT</th>
<th>P-STATE</th>
<th>N-STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$s_1$</td>
<td>$s_3$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_1$</td>
<td>$s_3$</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>$s_2$</td>
<td>$s_3$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_2$</td>
<td>$s_1$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_3$</td>
<td>$s_5$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$s_3$</td>
<td>$s_4$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$s_4$</td>
<td>$s_2$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_4$</td>
<td>$s_3$</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>$s_5$</td>
<td>$s_2$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$s_5$</td>
<td>$s_5$</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

• Minimum symbolic cover:

\[
\begin{array}{cccc}
* & s_1 s_2 s_4 & s_3 & 0 \\
1 & s_2 & s_1 & 1 \\
0 & s_4 s_5 & s_2 & 1 \\
1 & s_3 & s_4 & 1 \\
\end{array}
\]

• Covering constraints:

– \( s_1 \) and \( s_2 \) cover \( s_3 \)

– \( s_5 \) is covered by all other states.

• Encoding constraint matrices:

\[
A = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
Example

- Encoding matrix (one row per state):

$$E = \begin{bmatrix}
1 & 1 & 1 \\
1 & 0 & 1 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}$$

- Encoded cover of combinational component:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>1**</td>
<td>001</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>111</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>*00</td>
<td>101</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>100</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multiple-level circuit models

- *Logic network* representation.
  - Logic gate interconnection.

- Area:
  - # of literals.

- Delay:
  - Critical path length.

- Note
  - # literals and CP in a *minimum* network.
State encoding
for multiple-level models

- Cube-extraction heuristics [Mustang-Devadas].

- Rationale:
  - When two (or more) states have a transition to the same next-state:
    - Keep the distance of their encoding short.
    - Extract a large common cube.

- Exploit first stage of logic.

- Works fine because most FSM logic is shallow.
Example

- 5-state FSM (3-bits).
  
  - $s_1 \rightarrow s_3$ with input $i$.
  
  - $s_2 \rightarrow s_3$ with input $i'$.

- Encoding:
  
  - $s_1 \rightarrow 000 = a'b'c'$.
  
  - $s_2 \rightarrow 001 = a'b'c$.

- Transition:
  
  - $ia'b'c' + i'a'b'c = a'b'(ic + i'c')$
  
  - 6 literals instead of 8.
Algorithm

- Examine all state pairs:
  - Complete graph with $|V| = |S|$.

- Add weight on edges:
  - Model desired code proximity.

- Embed graph in the Boolean space.
Difficulties

- The number of occurrences of common factors depends on the next-state encoding.

- The extraction of common cubes interact with each other.
Algorithm implementation

- Fanout-oriented algorithm:
  - Consider present states and outputs.
  - Maximize the size of the most frequent common cubes.

- Fanin-oriented algorithm:
  - Consider next states and inputs.
  - Maximize the frequency of the largest common cubes.
Finite-state machine decomposition

- Classic problem.
  - Based on partition theory.
  - Recently done at symbolic level.

- Different topologies:
  - Cascade, parallel, general.

- Recent heuristic algorithms:
  - Factorization [Devadas].
Example
Summary

- *Finite-state machine* optimization is commonly used.
  - Large body of research.

- State reduction/encoding correlates well to area minimization.

- Performance-oriented methods are still being researched.