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

---

**State minimization** for completely specified FSMs

- 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.

---

**Formal finite-state machine model**

- 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.

---

- 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>s₁</td>
<td>s₃</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>s₁</td>
<td>s₅</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>s₂</td>
<td>s₃</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>s₂</td>
<td>s₅</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>s₃</td>
<td>s₂</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>s₃</td>
<td>s₁</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>s₄</td>
<td>s₄</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>s₄</td>
<td>s₅</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>s₅</td>
<td>s₄</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>s₅</td>
<td>s₁</td>
<td>0</td>
</tr>
</tbody>
</table>

Algorithm

- Stepwise partition refinement.

- Initially:
  - All states in the same partition block.

- Then:
  - Refine partition blocks.

- At convergence:
  - Blocks identify equivalent states.

Example

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

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
\text{INPUT} & \text{STATE} & \text{N-STATE} & \text{OUTPUT} \\
\hline
0 & \text{s}_{12} & \text{s}_3 & 1 \\
1 & \text{s}_{12} & \text{s}_5 & 1 \\
0 & \text{s}_3 & \text{s}_{12} & 0 \\
1 & \text{s}_3 & \text{s}_5 & 1 \\
0 & \text{s}_4 & \text{s}_4 & 1 \\
1 & \text{s}_4 & \text{s}_{12} & 0 \\
\hline
\end{tabular}
\end{center}

Computational complexity

- Polynomials-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).

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>s₁</td>
<td>s₃</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>s₁</td>
<td>s₅</td>
<td>*</td>
</tr>
<tr>
<td>0</td>
<td>s₂</td>
<td>s₃</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>s₂</td>
<td>s₅</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>s₃</td>
<td>s₂</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>s₃</td>
<td>s₁</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>s₄</td>
<td>s₄</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>s₄</td>
<td>s₅</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>s₅</td>
<td>s₄</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>s₅</td>
<td>s₁</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.
    - \( II = \{\{s₁, s₂\}, \{s₃\}, \{s₄\}, \{s₅\}\} \).
  - Replace * by 0.
    - \( II = \{\{s₁, s₅\}, \{s₂, s₃, s₄\}\} \).
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 \( \{s_2, s_5\} \).

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

• Incompatible pairs:
  - \( \{s_2, s_5\}, \{s_3, s_5\} \)
  - \( \{s_1, s_4\}, \{s_4, s_5\} \)
  - \( \{s_1, s_3\} \)

 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

- \( \{s_1, s_2\} \)
- \( \{s_1, s_5\} \leq \{s_3, s_4\} \)
- \( \{s_2, s_3, s_4\} \leq \{s_1, s_5\} \)
- 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

- $\{s_1, s_2\}$
- $\{s_1, s_5\} \subseteq \{s_3, s_4\}$
- $\{s_2, s_3, s_4\} \subseteq \{s_1, s_5\}$
- 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 \times \# 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

**Combinational Circuit**

- **Primary Inputs**: \( s_1, s_2, s_3 \)
- **Primary Outputs**: \( s_5 \)
- **State**: \( s_0, s_1, s_2, s_3, s_4, s_5 \)

### Table: Primary State Inputs Outputs

<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>

### Minimum Symbolic Cover:

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

### Covering Constraints:

- \( s_1 \) and \( s_2 \) cover \( s_3 \)
- \( s_5 \) is covered by all other states.

### Encoding Constraint Matrices:

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

### 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:
  - $ia'b'c' + i'd'b'c = a'b'(ic + i'c')$
  - 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

- 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.