Equivalence Checking of Sequential Circuits

Sanjit Seshia
EECS
UC Berkeley

With thanks to K. Keutzer, R. Rutenbar

Today’s Lecture

• What we know:
  – How to check two combinational circuits for equivalence

• What we need:
  – Checking equivalence of sequential circuits
  – E.g., a circuit and its retimed version

• Today’s lecture is about using Boolean function manipulation & BDDs for doing this
  – Basics
  – Sequential equivalence checking: the problem
  – Algorithms
Recap: Cofactors

A Boolean function $F$ of $n$ variables $x_1, x_2, \ldots, x_n$

$$F : \{0,1\}^n \rightarrow \{0,1\}$$

Cofactors of $F$:

$$F_{x_1} (x_2, \ldots, x_n) = ?$$
$$F_{x_1'} (x_2, \ldots, x_n) = ?$$

Two Operations on Cofactors

Given: $F(x_1, \ldots, x_n)$

Define

1. $C(x_2, \ldots, x_n) = F_{x_1} \cdot F_{x_1'}$ ← “Consensus”
2. $S(x_2, \ldots, x_n) = F_{x_1} + F_{x_1'}$ ← “Smoothing”

What do $C$ and $S$ look like in terms of the ON-sets of $F_{x_1}$ and $F_{x_1'}$?
**Example**

\[ F(a,b,c) = ab + bc + ac \]

\[ F_a = b + c \]
\[ F_{a'} = bc \]

\[ C(b,c) = ? \]
\[ S(b,c) = ? \]

---

**Quantification**

- **Consensus also called “universal quantification”**
  
  \[ C(x_2, \ldots, x_n) = F_{x_1} \cdot F_{x_1'} \]
  
  \[ = \forall x_1 F(x_1, x_2, \ldots, x_n) \] ("for all \( x_1 \) …")

- **Smoothing also called “existential quantification”**
  
  \[ S(x_2, \ldots, x_n) = F_{x_1} + F_{x_1'} \]
  
  \[ = \exists x_1 F(x_1, x_2, \ldots, x_n) \] ("there exists \( x_1 \) …")
Back to Equivalence Checking . . .

Equivalence Checking: Simple Case

``specification''

implementation
Retimed circuits

Circuits are equivalent but it is not possible to show that they are equivalent using Boolean equivalence

Encoding Problems

Some logic specifications are “symbolic” rather than binary-valued

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>+</td>
</tr>
<tr>
<td>SUB</td>
<td>-</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive-OR</td>
</tr>
<tr>
<td>INC</td>
<td>Increment</td>
</tr>
</tbody>
</table>

Can assign any binary op code to the symbolic values, so long as they are different
### Different State Encodings

<table>
<thead>
<tr>
<th>Circuit 1</th>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td></td>
<td>00</td>
</tr>
<tr>
<td>SUB</td>
<td></td>
<td>01</td>
</tr>
<tr>
<td>XOR</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit 2</th>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>SUB</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>XOR</td>
<td></td>
<td>00</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>01</td>
</tr>
</tbody>
</table>

Different state encodings make circuits no longer amenable to combinational logic equivalence checking.

### Different Encodings

**ALU `ADD's on 00**

- `x` and `y` inputs are 32 bits.
- `alu_out` is the output.
- `clk` is the clock input.

**ALU `ADD's on 11**

- `x` and `y` inputs are 32 bits.
- `alu_out` is the output.
- `clk` is the clock input.
A Fresh Look at Equivalence Checking

Given: Two sequential circuits, with same inputs and outputs
- But state bits might differ

Let’s view this problem mathematically (“formally”):
A combinational circuit is a Boolean function.
A sequential circuit is a __________

What’s in a Finite-State Machine (FSM) ?

[Diagram showing a Finite-State Machine with FF, Next State logic, Output logic, inputs, clk, and outputs]
Finite-state machine (FSM) Equivalence

Equivalence checking problem:

Given: 2 FSMs, with same inputs/outputs

To check:
The output behavior of both machines is identical
- over all time points, starting from a common “initial” / “reset” state
- for every sequence of inputs

Visualizing the Problem

Q1. What goes inside the boxes?
Q2. How can we decide if the output is always 1?
What goes in the boxes

From the finite-state machine description, we write Boolean equations that describe
1. Next state as a function of present state & inputs
2. Output as a function of present state & inputs

• Most often this is how the system is most easily described

Example: FSM1

Denote next state encoding as \( p'q + \) and output as \( z \)

\[
\begin{align*}
p'(x, p, q) &= ? \\
pq' + p'x &\ \ \ \ \ \text{(a)} \\
q'(x, p, q) &= ? \\
p'x' + p'q &\ \ \ \ \ \text{(b)} \\
z(x, p, q) &= ? \\
pq &\ \ \ \ \ \text{(c)}
\end{align*}
\]
Example: FSM 2 (different state encoding)

Denote next state encoding as \( a^*b^*c^*d^* \) and output as \( z \)

\[
\begin{align*}
\text{abcd} = 1000 & \quad \text{abcd} = 0100 \\
A/0 & \quad B/0 \\
x = 0 & \quad x = 0 \\
x = 1 & \quad x = 1 \\
x = 0,1 & \quad x = 1 \\
\end{align*}
\]

\[
\begin{align*}
\text{abcd} = 0010 & \quad \text{abcd} = 0001 \\
C/0 & \quad D/1 \\
\end{align*}
\]

\[a^*(x, a, b, c, d) = ? \]
\[b^*(x, a, b, c, d) = ? \]
\[c^*(x, a, b, c, d) = ? \]
\[d^*(x, a, b, c, d) = ? \]

\[z(x, a, b, c, d) = d \]

NOTE: We never start with a state graph like the one above – WHY?

Back to the Problem

FSM 1

\[
\begin{align*}
p^* &= pq' + p'x \\
q^* &= p'x' + p'q \\
z &= pq \\
\end{align*}
\]

FSM 2

\[
\begin{align*}
a^* &= d \\
b^* &= ax' + bx' \\
c^* &= ax + c \\
d^* &= bx \\
z &= d \\
\end{align*}
\]

Q1. What goes inside the boxes? ✓
Q2. How can we decide if the output is always 1?
Rephrasing the Problem

Is the output always 1?

Can the output ever be 0?

Solved using “reachability analysis”
  - Is there a state that the combined FSM can reach such that the output is 0?

Performing Reachability Analysis

3 Main ideas:
1. Represent sets as Boolean functions
   - Use BDDs
2. Represent FSMs “symbolically”
   - FSM = set of states and set of transitions
   - FSM can be encoded using BDDs
3. Perform Symbolic Reachability Analysis
   - Start in initial state
   - Compute set of states reachable from initial state in 1, 2, 3, … clock ticks
   - This computation must terminate – WHY?
1. Sets as Boolean functions

A Boolean function $F$ of $n$ variables $x_1, x_2, \ldots, x_n$

$$F : \{0,1\}^n \rightarrow \{0,1\}$$

can be represented as set

Similarly, for a set of size $\leq 2^n$, you can encode each element as a string of $\leq n$ bits

- Each string can be viewed as a minterm
- View the set as the ON-SET of a Boolean function

Set Operations as Boolean Operations

- $A \cup B = ?$
- $A \cap B = ?$
- $A \subset B = ?$
- Is $A$ empty?
2. Symbolic Encoding of FSM

FSM is

• Set of states
  – Each state is a minterm
  – This is what we want to compute!

• Set of transitions
  – To compute set of reachable states, we first need a way of encoding transitions
  – WHY NOT just enumerate all the states by repeatedly evaluating equations, starting from an initial state?

Encoding Transitions

Define a new function, $\delta$, called the “transition relation”

$\delta$ (current state $s$, input $x$, next state $s^+$)

= 1 if we can go to $s^+$ from $s$ on $x$
= 0 otherwise

i.e. $\delta$ encodes all legal transitions (“edges” in the state graph)
3. Reachability Analysis

Given:
1. A minterm corresponding to initial state \( R_0 \)
2. \( \delta \)

To find:
All states reachable from \( R_0 \) in 1, 2, 3, … clock ticks

Strategy: Denote set of states reachable from \( R_0 \) in \( k \) (or less) clock ticks as \( R_k \)
- Express \( R_k \) as a function of \( R_{k-1} \) and \( \delta \) and solve recurrence relation
  - Remember: Every set is represented as a Boolean function (BDD)
What’s the initial state?

Start state for combined FSM is \( pqabcd = 001000 = R_0 \)

Computing \( R_k, k \geq 1 \)

What’s the relation between \( R_k \) and \( R_{k+1} \)?

(Think in terms of sets)
Computing $R_k$, $k \geq 1$

To get from $R_k$ to $R_{k+1}$, there must be some triple $(s, x, s^+)$ such that:

1. $s \in R_k$
2. $s^+ \in R_{k+1}$
3. $\delta(s, x, s^+) = 1$

Looking at it another way...

Suppose I gave you a $s^+$ and asked you whether it was in $R_{k+1}$, i.e.: Is $R_{k+1}(s^+) = 1$?

Can you phrase the answer to this question in terms of $R_k$ and $\delta$? (say in English)

Either
1. $s^+$ is in $R_k$, i.e., $R_k(s^+) = 1$
Or
2. There exist current state $s$ and input $x$ such that:
   • $R_k(s) = 1$
   • $\delta(s, x, s^+) = 1$
Writing out an equation for $R_{k+1}$

$$R_{k+1}(s^*) = R_k(s^*) + \exists s, x \{ R_k(s) \cdot \delta(s, x, s^*) \}$$

Either
1. $s^*$ is in $R_k$, i.e., $R_k(s^*) = 1$
   Or
2. There exist current state $s$ and input $x$ such that:
   • $R_k(s) = 1$
   • $\delta(s, x, s^*) = 1$

Computing $R_k$

Start with $R_0$

Repeatedly compute $R_{k+1}$ as:
$$R_{k+1}(s^*) = R_k(s^*) + \exists s, x \{ R_k(s) \cdot \delta(s, x, s^*) \}$$

Note: everything is represented as a Boolean function

When do we stop?
Termination

When $R_k$ and $R_{k+1}$ are the same

Why is this guaranteed to happen?

Recap of Reachability Analysis

1. Compute start state $R_0$
2. Compute expression for $\delta$
3. Repeatedly compute $R_k$ until termination criterion is true
4. Resulting $R_k$ for largest $k$ is the set of all states reachable from $R_0$
**Sequential Equivalence Checking**

1. Connect the two FSMs to form combined FSM
2. Compute combined start state $R_0$
3. Compute expression for $\delta$
4. Repeatedly compute $R_k$ until termination criterion is true
5. Resulting $R_k$ for largest $k$ is the set of all states reachable from $R_0$
6. Check if any of these states can generate output 0 (showing that the two FSM outputs are different)

**Summary**

- Sequential equivalence checking can be done using FSM reachability analysis
- In practice, very computationally intensive
  - Memory intensive: BDDs can grow quite big
- Currently limited to a few hundred state bits
- Scaling this up is an active area of research
  - New techniques based on SAT solving are available