Recall: Fixed-Point Semantics
We will develop the solution first with a closed system (or equivalently, with known external inputs)

Today we consider open systems.
Problem 1: Non-constructive, but seemingly combinational circuits

Logically, the output of the AND gate is 0. But if the input $x$ is zero, and the inverters have delay, then this circuit will oscillate.

The problem is that, with methods presented so far, we have to test whether the circuit is constructive for all possible inputs.

[Malik, Trans. on CAD, 1994]

Problem 2: Circuits that are constructive for some inputs, and may only get those inputs.

First need to find which inputs are problematic.
Then need to determine whether those inputs can occur (reachability analysis on a state machine)

[Shiple, Berry, and Touati, DATE, 1996]
Recall: The Synchronous Abstraction

- Execution is a sequence of “ticks” of a global clock.
- All components in a network execute “simultaneously” and “instantaneously”
- The behavior at each tick is the solution to a fixed-point problem:

\[ F_i(s) = s \]

Recall: Constructive Semantics

Solution procedure:
- Start with signals “unknown” at all nodes except inputs.
- Evaluate components (gates) in arbitrary order repeatedly until no further progress is made.
- If the result has all signals “known,” then declare the constructive solution.
- To know statically whether a circuit is constructive, it seems we have to try this for all possible inputs!
Symbolic Execution

Solution procedure:

- Start with unknown function of the inputs at all nodes except inputs.
- Update the functions in arbitrary order repeatedly until no further progress is made.
- If the result has all functions known, then declare the circuit constructive.

Symbolic Execution

Assume a single binary input (for now). For each node in the circuit, define a function from that input to the node value of this form:

\[ f_a : \{0, 1\} \rightarrow \{\bot, 0, 1\} \]

These give the outputs as a function of x only.
Symbolic Execution Strategy

Start with all nodes except inputs being given by the unknown function:

\[ f_y(x) = \perp \]

Then update these functions iteratively until convergence. But how to update the functions?

Idea

Represent each function of the form:

\[ f_a : \{0, 1\} \rightarrow \{\perp, 0, 1\} \]

using two characteristic functions of the form:

\[ f_a^0 : \{0, 1\} \rightarrow \{0, 1\} \]

\[ f_a^1 : \{0, 1\} \rightarrow \{0, 1\} \]

where

\[ f_a^0(x) = \begin{cases} 1 & \text{if } f_a(x) = 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ f_a^1(x) = \begin{cases} 1 & \text{if } f_a(x) = 1 \\ 0 & \text{otherwise} \end{cases} \]

Represent using BDDs.

[Shiple, Berry, and Touati, DATE, 1996]
Symbolic Execution Strategy using Characteristic Functions

Start with all nodes except inputs being given by the unknown function:

\[
\begin{align*}
  f_0^0(x) &= 0 \\
  f_1^0(x) &= 0 \\
  f_0^1(x) &= 0 \\
  f_1^1(x) &= 0 \\
  f_c^0(x) &= f_c^1(x) = 0
\end{align*}
\]

Then update these functions iteratively until convergence. But how to update the functions?

Operating on Characteristic Functions

Gates relate characteristic functions of the outputs with those of the inputs:

\[
\begin{align*}
  f_0^0(x) &= f_a^0(x) \\
  f_1^0(x) &= f_a^0(x) + f_b^0(x) \\
  f_0^1(x) &= f_a^1(x) \\
  f_1^1(x) &= f_a^1(x) \cdot f_b^1(x) \\
  f_c^0(x) &= f_a^0(x) \cdot f_b^0(x) \\
  f_c^1(x) &= f_a^1(x) + f_b^1(x)
\end{align*}
\]
Symbolic Execution Strategy using Characteristic Functions

Update nodes in arbitrary order:

\[ f_y^0(x) = 0 \]
\[ f_y^1(x) = 0 \]
\[ f_1^1(x) = f_1^1(x) = x \]
\[ f_z^1(x) = f_z^1(x) = 0 \]

\[ f_y^0(x) = f_y^0(x) = x \]
\[ f_y^1(x) = f_y^1(x) = 0 \]

etc.

\[ f_0^0(x) = f_0^0(x) \cdot f_0^0(x) = 0 \cdot x = 0 \]
\[ f_1^1(x) = f_1^1(x) + f_1^1(x) = 0 + x = x \]

Symbolic Execution Strategy using Characteristic Functions Convergence

Quickly converge to these characteristic functions:

\[ f_y^0(x) = x \]
\[ f_y^1(x) = 0 \]
\[ f_c^0(x) = x \]
\[ f_c^1(x) = 0 \]

\[ f_z^0(x) = 0 \]
\[ f_z^1(x) = x \]

How do we know whether the circuit is constructive?
Symbolic Execution Strategy using Characteristic Functions Convergence

Quickly converge to these characteristic functions:

\[
\begin{align*}
\forall x, y, z, a, \ b, \ c \in \{0, 1\} : \ f_0^0(x) &= x \\
\ f_0^1(x) &= 0 \\
\ f_1^0(x) &= \bar{x} \\
\ f_1^1(x) &= x \\
\ f_c^0(x) &= x \\
\ f_c^1(x) &= 0 \\
\ f_a^0(x) &= 0 \\
\ f_a^1(x) &= x \\
\ f_x^0(x) &= \bar{x} \\
\ f_x^1(x) &= x 
\end{align*}
\]

Circuit is constructive iff at all nodes \( a \) we have for all \( x \)
\[
\forall x, y, z, a, \ b, \ c \in \{0, 1\} : \ f_a^0(x) + f_a^1(x) = 1
\]
i.e. the value is known! (Checking this is a SAT problem)

Does the procedure always converge?
Is the answer unique?

Consider a poset \( \{0, 1\} \) where \( 0 \prec 1 \).
This induces a poset that is the set of functions of form:
\[
f^i_a : \{0, 1\} \rightarrow \{0, 1\}
\]
This poset is pointed, with bottom being the function
\[
f^i_\perp(x) = 0
\]
This poset is finite, with structure much like the Scott order (it is a complete lower semilattice). The Knaster-Tarski fixed-point theorem applies. Extends easily to tuples of functions.
Gate Operations on Characteristic Functions are Monotonic Functions!

These are monotonic in the sense that if you know more about the inputs, then you learn more about the outputs:

\[
(f^0_a, f^1_a) \leq (g^0_a, g^1_a) \Rightarrow (f^0_c, f^1_c) \leq (g^0_c, g^1_c)
\]

\[
((f^0_a, f^0_b), (f^1_a, f^1_b)) \leq ((g^0_a, g^0_b), (g^1_a, g^1_b)) \Rightarrow (f^0_c, f^1_c) \leq (g^0_c, g^1_c)
\]

Extending to State Machines

The state of the latches is is the state of a state machine.

[Shiple, Berry, and Touati, DATE, 1996]
Extending to State Machines

The states are (a, b).

Note that state (1, 1) is problematic (the circuit is not constructive with a=b=1).

But this state is not reachable if the initial state is not (1, 1).

[Shiple, Berry, and Touati, DATE, 1996]

Notation for State Machines

Guard is a combinational circuit, action may produce outputs.

In each state, this defines a combinational component. When composing state machines (with feedback loops), we would like to ensure that the composition constructive.

Symbolic execution allows us to do that statically (vs. detecting non-constructive compositions at run time).
Constructive Composition of State Machines

**Figure 6.2:** A simple well-formed feedback model.

[Lee & Seshia, *Introduction to Embedded Systems, 2010*]

Constructive Composition of State Machines

**Figure 6.4:** An ill-formed feedback model that has no fixed point in state s2.

[Lee & Seshia, *Introduction to Embedded Systems, 2010*]
Constructive Composition of State Machines

Figure 6.5: An ill-formed feedback model that has more than one fixed point in state s1.

[Lee & Seshia, Introduction to Embedded Systems, 2010]

Constructive Composition of State Machines

Figure 6.6: A well-formed feedback model that is not constructive.

[Lee & Seshia, Introduction to Embedded Systems, 2010]
Synchronous Languages: Stateful systems

Example: SCADE is based on Lustre and Esterel. Synchronous composition of state machines.

Used for safety-critical systems.

References