19. Introduction to Symbolic Trajectory Evaluation

Jacob Abraham
Department of Electrical and Computer Engineering
The University of Texas at Austin
Verification of Digital Systems
Spring 2017
March 30, 2017

Motivation

- Equivalence checking between RTL and circuit schematics is difficult for some circuits (e.g., custom arrays)
  - Critical timing and self-timed control logic
  - Large number of bit-cells
  - Inherently complex sequential logic blocks
  - Dynamic logic
- Traditional tools fail on such circuits
  - Very large state space, too many initial state/input sequences for simulation-based tools
  - Boolean equivalence tools only check static cones of logic, do not capture dynamic behavior

Motivation

- Control for customer array structure

![Diagram showing logic gates and boolean function]

- With zero delay functional simulation OUT is 0
- OUT pulse fans out to array READ/WRITE control signals
- Need to define unit delay for all gates to get a pulse on OUT
- Verification for correctness fails in downstream logic without this pulse

Symbolic Trajectory Evaluation

- Symbolic Trajectory Evaluation (STE) is a high-performance simulation-based model checking technique, originally invented by Seger and Bryant
- STE uses a combination of three-valued simulation and symbolic simulation
- Let us look at the basics of STE
Example

- The specification says that out represents the output of a 3-input AND gate with in0, in1, and in2 as inputs
- Here is an implementation of the above spec

Scalar Simulation

- We need $2^n$ (8 here) simulation patterns
- One such pattern is the assertion
  - $(\text{in0 is } 0) \text{ and } (\text{in1 is } 1) \text{ and } (\text{in2 is } 0) \implies (\text{out is } 0)$
  - Every check like this is of the form $A \implies C$, where $A$ is an antecedent and $C$ is a consequent
  - All STE assertions are of this form – we will give formal definition of STE assertions and define their semantics
- Not practical for large scale designs and so we need to make it more efficient, but how?
### Three-valued Simulation

- **Observation**
  - Any input having assigned to 0, makes the output 0
  - Other inputs do not matter meaning could be 0 or 1 (X)
- If we introduce a new value (X = don’t care) in the simulation we need to evaluate the outputs of standard gates

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x \cdot y</th>
</tr>
</thead>
</table>
| 0 | 0 | 0  
| 0 | 1 | 0  
| 1 | 0 | 0  
| X | X | X  

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x + y</th>
</tr>
</thead>
</table>
| 0 | 0 | 0  
| 0 | 1 | 1  
| 1 | 0 | 1  
| X | X | X  

- Cannot afford $2^n$ simulation patterns and so we will use 3-valued simulations
- Simulation pattern becomes
  - (in0 is 0) and (in1 is X) and (in2 is X) \implies (out is 0)
  - This can be simplified into (in0 is 0) \implies (out is 0)
  - Still of the form $A \implies C$
- Now the total number of simulations needed has reduced to $n + 1$ (4 here)
  - (in1 is 0) \implies (out is 0)
  - (in2 is 0) \implies (out is 0)
  - (in0 is 1) and (in1 is 1) and (in2 is 1) \implies (out is 1)
- Can we do better?
Symbolic Simulation

- Assign symbols on inputs
- Now we need to do only one simulation
  - $(\text{in0 is } a) \text{ and } (\text{in1 is } b) \text{ and } (\text{in2 is } c) \implies (\text{out is } a.b.c)$
- Can we do better than 1 simulation?
- No, but...

Checking for correctness

- Boolean expressions can be represented as BDDs (or other data structures) for performing constant time checks
- Verification steps to check the following assertion
  - $(\text{in0 is } a) \text{ and } (\text{in1 is } b) \text{ and } (\text{in2 is } c) \implies (\text{out is } a.b.c)$
  - Implementation: From the antecedent assign symbols (equivalent to BDDs) to the inputs of the circuit
  - Calculate the symbolic expressions (as BDDs) in the circuit nodes until the symbolic value of the output is known
  - Specification: Calculate the consequent expression BDD
  - Compare the two BDDs for equivalence
- Can we reduce the complexity of this one simulation?
Three-valued symbolic simulation

- In STE we combine the efficiency of three-valued simulation with the preciseness of symbolic simulations.
- Use this combination by expressing the four 3-valued simulation runs as only one 3-valued symbolic simulation run.
- Each assignment of 0 and 1 to two variables represents one 3-valued simulation run and since there are 4 possible assignments, and 4 3-valued simulation runs we can achieve this reduction (symbolic encoding).
- Symbolically,

\[
\begin{align*}
((\neg p \cdot \neg q) \to \text{in}0 \text{ is } 0) \text{ and } \\
((\neg p \cdot q) \to \text{in}1 \text{ is } 0) \text{ and } \\
((p \cdot \neg q) \to \text{in}2 \text{ is } 0) \text{ and } \\
((p \cdot q) \to \text{(in}0 \text{ is } 1) \text{ and (in}1 \text{ is } 1) \text{ and (in}2 \text{ is } 1)) \\
\implies \text{(out is } p.q))
\end{align*}
\]

Three-valued symbolic simulation

- Assertion has the symbolic form \( P \rightarrow A \implies C \) where \( P \) is a boolean expression (predicate).
- Logically, this is an implication.
- Simulation-wise this assertion is going to assign values of nodes from \( A \) in situations where \( P \) is true otherwise, the nodes are kept at Xs.
Dual-rail encoding

- The value of a 3-valued variable $x$ can be represented by 2 boolean variables $x = (x_0, x_1)$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$(x_0, x_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>1</td>
<td>(0, 1)</td>
</tr>
<tr>
<td>X</td>
<td>(0, 0)</td>
</tr>
</tbody>
</table>

- Operators can be defined as follows
  - NEG: $\neg x = \neg(x_0, x_1) = (x_1, x_0)$
  - For 2 3-valued variables $x = (x_0, x_1)$ and $y = (y_0, y_1)$ we can calculate
    - AND: $x.y = (x_0, x_1). (y_0, y_1) = (x_0 + y_0, x_1.y_1)$
    - OR: $x + y = (x_0, x_1) + (y_0, y_1) = (x_0.y_0, x_1 + y_1)$

Dual-rail encoding for circuit inputs

- So, this expression implies a few values for $in_0$, $in_1$, $in_2$
  - $((\neg p.\neg q) \rightarrow in_0$ is 0) and
  - $((\neg p.q) \rightarrow in_1$ is 0) and
  - $((p.\neg q) \rightarrow in_2$ is 0) and
  - $((p.q) \rightarrow (in_0$ is 1) and (in1 is 1) and (in2 is 1))
  $\implies$ (out is $(p.q)$)
  - if $(\neg p.\neg q)$ then assign $in_0$ to 0
  - else if $(p.q)$ then assign $in_0$ 1 else keep $in_0$ at an X
- Similar for $in_1$ and $in_2$
- Since $in_0$ is a 3-valued variable its dual-rail encoding will be $in_0 = ((\neg p.\neg q), (p.q))$
- So, now the verification can be done by one simulation run with $\lceil \log_2(n) \rceil$ (2 here) number of variables
Inaccuracy

We have come from $2^n$ simulations to one simulation using $\lceil \log_2(n) \rceil$ variables so what is the catch?

The STE assertion is

$$(\text{in0 is } a) \text{ and } (\text{in1 is } a) \implies (\text{out is } a)$$

Without specifying anything about sel in the antecedent it will be kept as an X making out X

Inaccuracy can be reduced by variables

The STE assertion that will resolve the issue is

$$(\text{sel is } b) \text{ and } (\text{in0 is } a) \text{ and } (\text{in1 is } a) \implies (\text{out is } a)$$

Bottomline: there is a reduction in number of variables but under some circumstances one will need more variables for accuracy
STE Theory: information ordering

- X is “unknown (under-constrained) value of 0 or 1”
- T is “conflicting (over-constrained) value 0 and 1”
- The information ordering forms a lattice that can extend to $n$ variables ($X \preceq 0, X \preceq 1, 0 \preceq T$ and $1 \preceq T$)
- Simulators need to use values $V = \{0, 1, X, T\}$
- Simulators use $\neg T = T, x.T = T, T.x = T, x + T = T$ and $T + x = T$
- Logic gates are monotonic w.r.t. information ordering

Circuit model

- Set of circuit nodes is $N$ (example, in0, in1, out, the inputs and output of an AND gate)
- A state is an assignment of values from $V$ to circuit nodes, $s: N \rightarrow V$ (example, assignment $s(in0) = X, s(in1) = 1, s(out) = X$)
- Circuit state is a collection of such values of the circuit nodes $S = \langle \forall n \in N : s(n) \rangle$ (example, $\langle X1X \rangle$)
- Note that in STE circuit state includes all nodes and not just latch nodes
- Closure function $F: S \times S$ (example, can be derived from the AND function extended to $V$)
- Note that the closure function is not the same as the traditional next state function
- $F$ propagates given values to other nodes
- $F$ can be easily constructed from the netlist logic
Properties of $F$

- $F$ is monotonic
  - $s_1 \preceq s_2$ implies $F(s_1) \preceq F(s_2)$
  - no second thoughts
- $F$ is an idempotent
  - $F(F(s)) = F(s)$
  - complete one-step simulation
- $F$ is extensive
  - $s \leq F(s)$
  - does not invent information

Sequences of States

A Sequence $\sigma$ is a function that maps time to States,
$\sigma : \text{Time} \rightarrow S$

Example: $\sigma = \sigma(0), \sigma(1), \ldots = s_0, s_1, \ldots$

Information ordering extends to states and sequences

- each circuit node is assigned $\{0, 1, X, \top\}$
- node values are weaker or stronger based on position of the value in the information ordering lattice
- state $s_0$ is weaker than state $s_1$, written $s_0 \preceq s_1$ iff all node values in $s_0$ are weaker than corresponding values in $s_1$,
  mathematically, $s_0 \preceq s_1 \equiv \forall n \ (\text{node}), \ s_0(n) \preceq s_1(n)$
- sequence $\sigma_0$ is weaker than sequence $\sigma_1$, written $\sigma_0 \preceq \sigma_1$ iff all states in $\sigma_0$ are weaker than corresponding states in $\sigma_1$
  mathematically, $\sigma_0 \preceq \sigma_1 \equiv \forall t \ (\text{time}), \ \sigma_0(t) \preceq \sigma_1(t)$
Closure function over Sequences

In STE, a circuit is simulated over multiple time steps. During simulation, information is propagated forwards through the circuit and through time, from each time step \( t \) to time step \( t + 1 \). Closure function is extensible over multi-step sequences. Given a closure function \( F : S \rightarrow S \) we can define \( F^* : Seq \rightarrow Seq \):

- This connects all state holding elements as well as all circuit nodes.
- \( F^* \) is monotonic: \( \sigma_0 \preceq \sigma_1 \) implies \( F^*(\sigma_0) \preceq F^*(\sigma_1) \).
- \( F^* \) is idempotent: \( F^*(F^*(\sigma)) = F^*(\sigma) \).
- \( F^* \) is extensive: \( \sigma \preceq F^*(\sigma) \).

Trajectory Evaluation Logic (TEL)

STE assertions are of the form \( A \implies C \), where \( A \) and \( C \) are Trajectory Formulas in the language of TEL with the following syntax:

**Definition**

\[
A, C ::= \begin{align*}
& n \text{ is } 0 \\
& | \ n \text{ is } 1 \\
& | \ A_1 \text{ and } A_2 \\
& | \ P \rightarrow A \\
& | \ \neg A
\end{align*}
\]

**Notes**

- \( P \) is a predicate over a set of symbolic variables \( V \) that are time-independent.
- The notion of time is in the form of the next time operator.
- We can assign symbolic expressions to node values because \( (n \text{ is } P) \) is short form of \( (P \rightarrow n \text{ is } 1) \text{ and } (\neg P \rightarrow n \text{ is } 0) \).
Semantics of TEL

The meaning of a TEL formula is defined by a satisfaction relation. Given a boolean evaluation $\phi$ of a set of symbolic variables $V$ and a sequence of states $\sigma \in \text{Seq}$ satisfaction of a TEL formula $A$ by a sequence $\sigma$ and a valuation $\phi$ is defined as follows:

<table>
<thead>
<tr>
<th>Definition</th>
<th>( \phi, \sigma \models n \text{ is } 0 )</th>
<th>( \phi, \sigma \models n \text{ is } 1 )</th>
<th>( \phi, \sigma \models A_1 \text{ and } A_2 )</th>
<th>( \phi, \sigma \models P \rightarrow A )</th>
<th>( \phi, \sigma \models N \overline{A} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>iff ( 0 \leq \sigma(0)(n) ), value of node $n$ in state $\sigma(0)$</td>
<td>iff ( 1 \leq \sigma(0)(n) )</td>
<td>iff $\phi, \sigma \models A_1$ and $\phi, \sigma \models A_2$</td>
<td>iff $\phi \models P$ implies $\phi, \sigma \models A$</td>
<td>iff $\phi, \sigma^1 \models A$</td>
</tr>
</tbody>
</table>

Notes

- $\phi \models P$ meaning standard propositional satisfiability
- If $\sigma = \sigma(0), \sigma(1), \sigma(2), \ldots$ then time-shifted sequence $\sigma^1 = \sigma(1), \sigma(2), \ldots$

Property

- Given a sequence $\sigma$ and a valuation $\phi$ if a TEL $A$ is satisfied, meaning, $\phi, \sigma \models A$ then for all sequences $\sigma'$ that are stronger than $\sigma$, $A$ holds as well, symbolically, if $\phi, \sigma \models A$ and $\sigma \preceq \sigma'$ then $\phi, \sigma' \models A$
- This means that we need not verify a TEL $A$ for all simulation sequences but a representative weakest sequence.
When a circuit is simulated the chain of states found during the execution forms a sequence.

A trajectory is a sequence in which no more information can be derived by forward propagation of the closure function $F^*$, mathematically, $\sigma$ is a trajectory iff $F^*(\sigma) = \sigma$.

Therefore, a trajectory is a fixed-point of the closure function $F^*$ over time.

Since $F^*$ is idempotent, $F^*(\sigma) = F^*(F^*(\sigma))$, applying $F^*$ to any sequence gives a trajectory.

A circuit with closure function $F^*$ satisfies a trajectory assertion $A \implies C$ iff for every valuation $\phi$ of the symbolic variables and for every 3-valued trajectory $\sigma$ of $F^*$ we have

$$\phi, \sigma \models A \implies \phi, \sigma \models C$$
A simplification and the Fundamental Theorem of STE

- The simulator performs one simulator run
- The simulator cannot check if $C$ holds for all trajectories
- It calculates the weakest trajectory $\sigma$ in which $A$ is satisfied and checks if $C$ holds on that particular trajectory
- By previous observation $C$ is going to hold on all trajectories that are stronger than $\sigma$
- This implies that it is enough to check $C$ only for the weakest trajectory satisfying $A$ instead of all trajectories

STE Example: Memory verification

- Memory with address width $k$ and data width $n$ needs $n.2^k$ state holding elements (state based model checkers)
- STE would need $n + k$ variables: let us see how
STE Example: Memory verification

Assertion for a memory
\[(\text{wr is 1}) \text{ and } (\text{addr}[0] \text{ is } a0) \text{ and } (\text{addr}[1] \text{ is } a1) \text{ and } (\text{in is } d)\]
and \(N ((\text{rd is 1}) \text{ and } (\text{addr}[0] \text{ is } a0) \text{ and } (\text{addr}[1] \text{ is } a1)) \implies N (\text{out is } d)\)

Observations
- Symbolic variables are \(a0, a1, d\) (total number is \(n + k\))
- Assertion starts from all Xs, writes into symbolic location \(a0a1\) with symbolic data \(d\) and then reads from the same symbolic location expecting to read out the symbolic data \(d\)

STE Example: Memory verification: time 0 and 1

Time 0: before the simulation begins all locations are Xs

Time 1: once written the symbolic values in the memory locations are
- \(\text{loc0} = \text{if } (\neg a0.\neg a1 == \text{true}) \text{ then } \text{new loc0 value is } d \text{ else } X\)
- \(\text{loc1} = \text{if } (\neg a0.a1) \text{ then } d \text{ else } X\)
- \(\text{loc2} = \text{if } (a0.\neg a1) \text{ then } d \text{ else } X\)
- \(\text{loc3} = \text{if } (a0.a1) \text{ then } d \text{ else } X\)
STE Example: Memory verification: time 2

Time 2: outputs are assigned depending on which location is read

- out0 // ("if I am addressed then value of loc0 else 0")
  = if (¬a0.¬a1) then loc0 else 0
  = if (¬a0.¬a1) then (if(¬a0.¬a1) then d else X) else 0
  = if (¬a0.¬a1) then d else 0
- Similarly, out1 = if (¬a0.a1) then d else 0
- out2 = if (a0.¬a1) then d else 0
- out3 = if (a0.a1) then d else 0

Simplifying, out = out0 OR out1 OR out2 OR out3 = d

Abstraction Refinement

A failed STE assertion can be

- a real counter-example showing an error in the circuit
- a failure that resulted from imprecision in STE
  - Under-constrained inputs
  - Quaternary state set unions
  - Existentially quantified-out symbolic variables

Corrective action can consist of monotonic abstraction refinement

- Constraining inputs with symbolic constants/variables
- Model refinement: introducing precise nodes
- Specification refinement: assertion graph transformations
Example of Symbolic Trajectory Evaluation (STE) (Bryant and Seger)

**Buffered Register**

```
<table>
<thead>
<tr>
<th></th>
<th>clk</th>
<th>din</th>
<th>wr</th>
<th>rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>DATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>out</td>
<td>DATA</td>
<td>DATA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Simulation-based model checking

**Symbolic Trajectory Evaluation at Freescale**

- VERSYS symbolic trajectory evaluation tool developed at Motorola/Freescale
  - Based on VOSS (from CMU/UBC)
- Trajectory formulas
  - Boolean expressions with the temporal next-time operator
  - Ternary values states represented by a Boolean encoding
- Properties of type: Antecedent $\implies$ Consequent
  - Antecedent, Consequent are trajectory formulas
  - Antecedent sets up stimulus, state of the circuit
  - Consequent specifies constraint on the state sequence
- Used to verify PowerPC arrays at Motorola/Freescale in 8 – 10% of the design time
- Bugs found during array equivalence checking
  - Incorrect clock regenerators feeding latches
  - Control logic errors in READ/WRITE enables
  - Violation of “one-hot” property assumptions
  - Scan chain hookup errors
  - Potential circuit-related problems such as glitches and races
Other issues with STE

- LTL with finite number of next time $N$ operators
- No notion of initial states
- No concept of reachable states

Solution 1: Induction

- We need to prove that $B$ should hold for all reachable states
- First prove the induction basis $I \implies B$ where $I$ represents the predicate for the initial states
- Next prove the induction step $B \implies N B$ meaning if $B$ holds now it holds on the next time step
- This concludes that $B$ holds for all reachable states
- Crucial step: theorem proving where the meta-reasoning is proved
Solution 2: Generalized STE (Yang and Seger)

- Specification is a graph that generates an infinite number of assertions
- Simulation is performed through fixed-point computation
- GSTE is essentially partitioned model checking by using antecedents on specification graph edges
- Instead of exploring all possible state space, antecedents guide symbolic simulation through trajectories under verification

Industrial application of GSTE

Use in the verification flow

Results on the Pentium 4 verification (Yang)

<table>
<thead>
<tr>
<th></th>
<th>Latches</th>
<th>Gates</th>
<th>Spec. Vars.</th>
<th>Time (Sec.)</th>
<th>Mem. (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>718</td>
<td>17367</td>
<td>68</td>
<td>122</td>
<td>36</td>
</tr>
<tr>
<td>2</td>
<td>7506</td>
<td>62735</td>
<td>95</td>
<td>5220</td>
<td>260</td>
</tr>
<tr>
<td>3</td>
<td>22433</td>
<td>187928</td>
<td>401</td>
<td>117</td>
<td>509</td>
</tr>
<tr>
<td>4</td>
<td>22433</td>
<td>187928</td>
<td>103</td>
<td>500</td>
<td>240</td>
</tr>
<tr>
<td>5</td>
<td>34899</td>
<td>406630</td>
<td>24</td>
<td>451</td>
<td>361</td>
</tr>
<tr>
<td>6</td>
<td>46682</td>
<td>241854</td>
<td>282</td>
<td>132</td>
<td>295</td>
</tr>
</tbody>
</table>
References