

# Formal Verification Methods 2: Symbolic Simulation

---

John Harrison  
Intel Corporation  
Marktoberdorf 2003

Thu 31st July 2003 (11:25 – 12:10)

## Summary

---

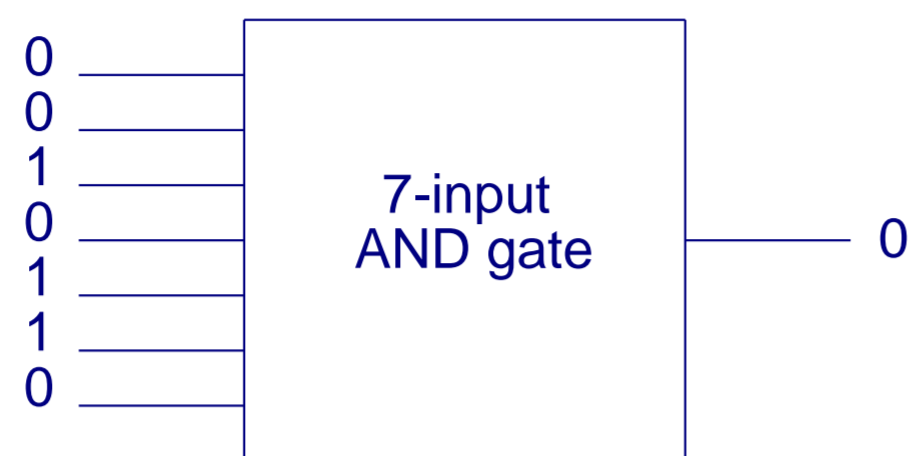
- Simulation
- Symbolic and ternary simulation
- BDDs
- Quaternary lattice
- Symbolic trajectory evaluation

## Simulation

---

The traditional method for testing and debugging hardware designs is *simulation*.

This is just testing, done on a formal circuit model.



Feed sets of arguments in as inputs, and check whether the output is as expected.

## Generalizations of Simulation

---

We can generalize basic simulation in two different ways:

- *Ternary* simulation, where as well as 0 and 1 we have a “don’t care” value X.
- *Symbolic* simulation, where inputs may be parametrized by Boolean variables, and outputs are functions of those variables.

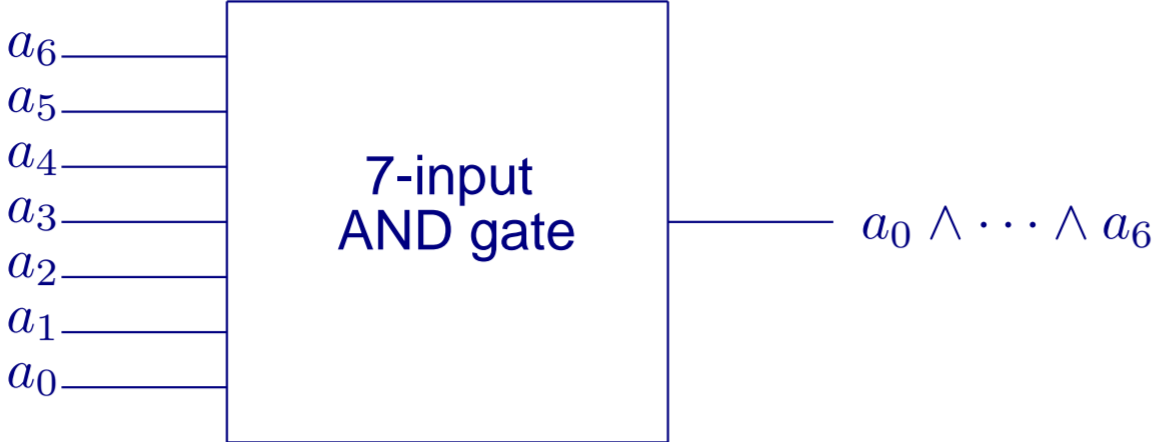
Rather surprisingly, it’s especially useful to do both at the same time, and have ternary values parametrized by Boolean variables.

This leads on to *symbolic trajectory evaluation* (STE) and its generalizations.

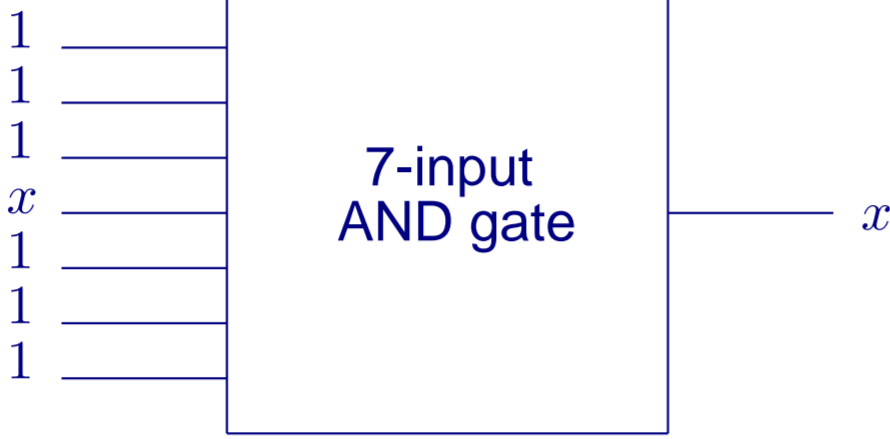
Example of symbolic simulation

---

We might use Boolean variables for all inputs:



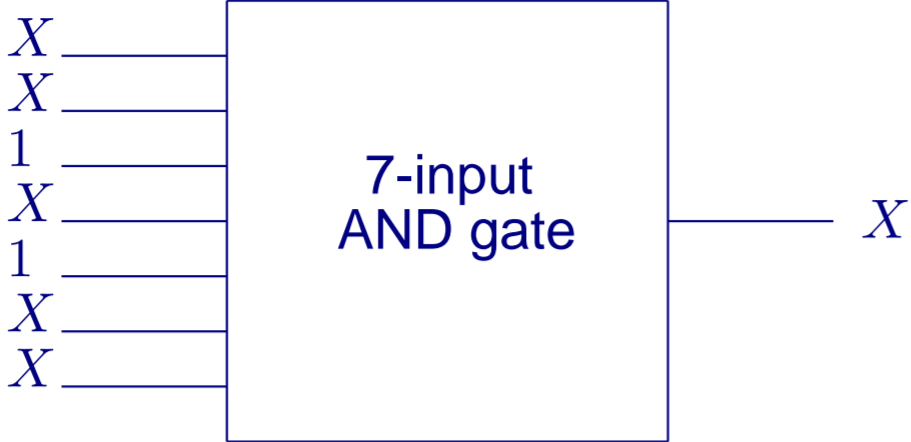
or just some of them:



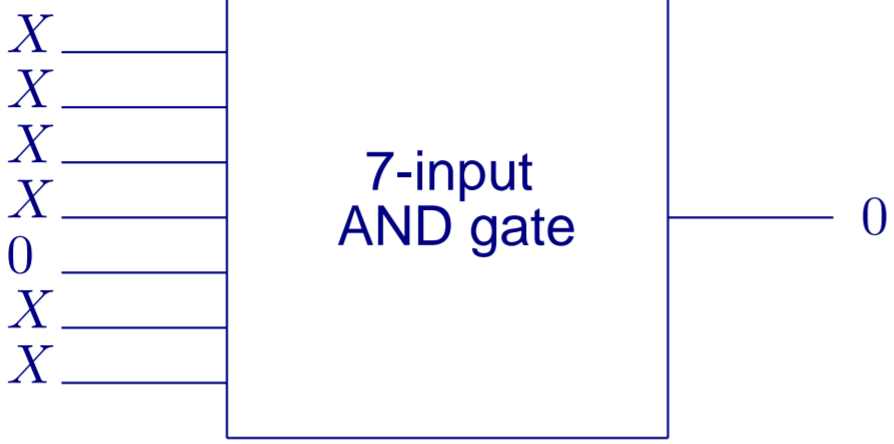
Example of ternary simulation

---

If some inputs are undefined, the output often is too:



but not always:



Consider the 7-input AND gate. To verify it exhaustively:

- In conventional simulation, we would need **128** test cases,  $0000000, 0000001, \dots, 1111111$ .
- In symbolic simulation, we only need **1** symbolic test case,  $a_0a_1a_2a_3a_4a_5a_6$ , but need to manipulate expressions, not just constants.
- In ternary simulation, we need **8** test cases,  $XXXXXX0, XXXXX0X, \dots, 0XXXXXX$  and  $1111111$ .

If we combine symbolic and ternary simulation, we can *parametrize* the 8 test cases by just 3 Boolean variables.

This makes the manipulation of expressions much more economical.

## Representing Boolean expressions

---

We could just represent Boolean expressions as formulas in the usual way.

- Need to check equivalence of output expression and expectation
- Essentially the same approach as in previous lecture

Main alternative is to use a *canonical* representation for Boolean formulas.

- Testing equivalence is trivial (are the expressions the same?)
- There is more work in composing the operations internally

The most popular canonical representation is (reduced ordered) *binary decision diagrams* (BDDs).



## BDDs

---

(Reduced Ordered) Binary decision diagrams represent Boolean functions as decision trees, but share common subtrees to give a directed acyclic graph structure.

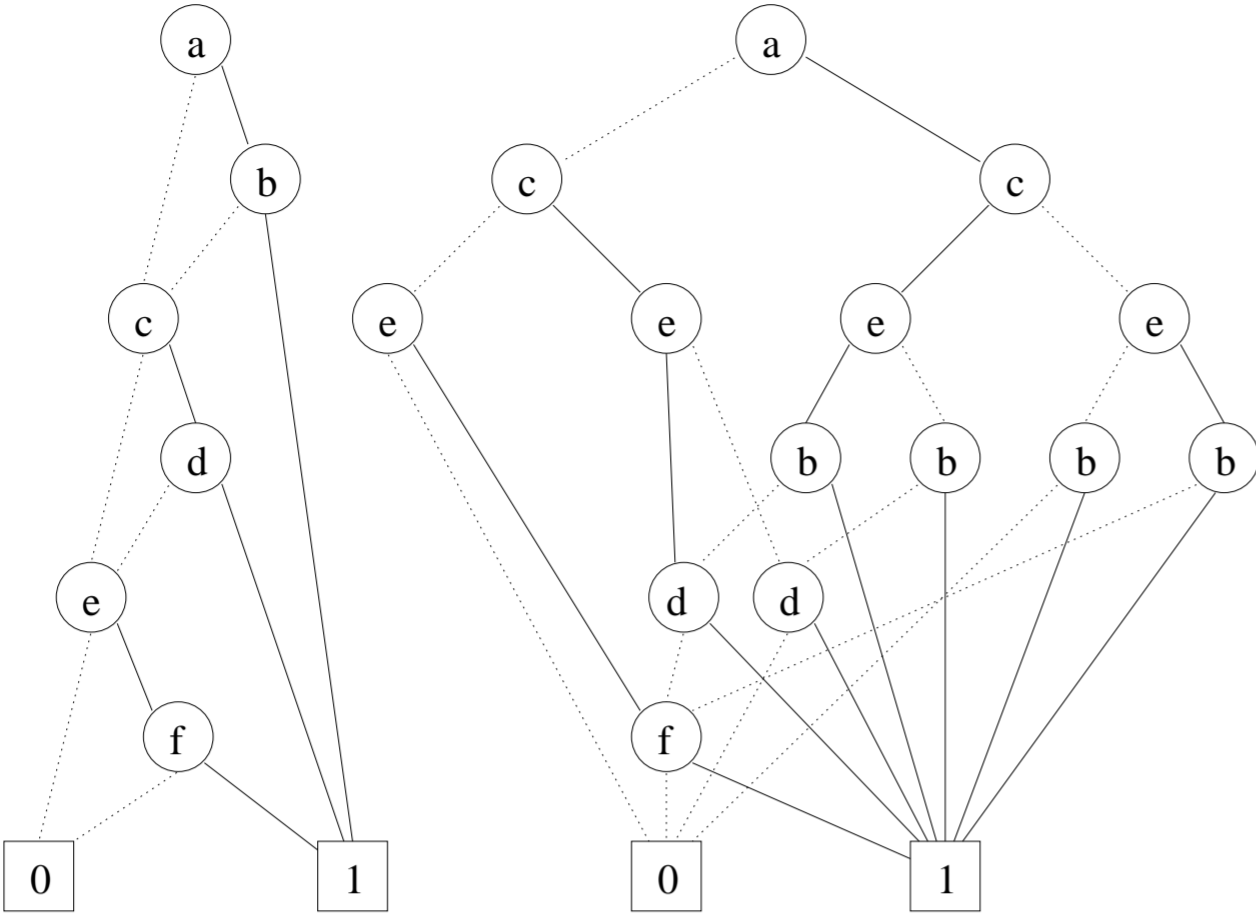
A canonical variable ordering is imposed, so variables occur in the same order along all paths.

This makes BDDs a canonical representation, so comparison of Boolean functions is constant time.

Moreover, Boolean operations are polynomial time, and they are often surprisingly compact for functions of interest.

Some functions such as multipliers have no efficient representation.

The variable ordering can make a big difference!



## Quaternary simulation

---

It's theoretically convenient to generalize ternary to *quaternary* simulation, introducing an 'overconstrained' value  $T$ .

We can think of each quaternary value as standing for a set of possible values:

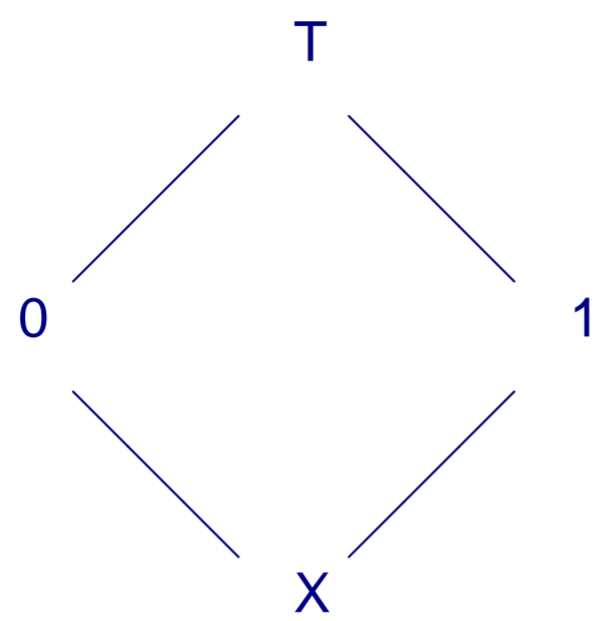
$$\begin{aligned} T &= \{\} \\ 0 &= \{0\} \\ 1 &= \{1\} \\ X &= \{0, 1\} \end{aligned}$$

This is essentially a simple case of an abstraction mapping.

## Quaternary lattice

---

We consider the quaternary values laid out in an information ordering:



The lattice ordering indicates that one value has 'more information' than another.

## Extended truth tables

---

The truth-tables for basic gates are extended:

$p$	$q$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
X	X	X	X	X	X
X	0	0	X	X	X
X	1	X	1	1	X
0	X	0	X	1	X
0	0	0	0	1	1
0	1	0	1	1	0
1	X	X	1	X	X
1	0	0	1	0	0
1	1	1	1	1	1

Composing gates in this simple way, we may lose information.

## Parametrizing quaternary values

---

Parametrize sets of quaternary values using Boolean variables:

$$a_0 = \begin{cases} 1 & \text{if } p \wedge q \wedge r \\ 0 & \text{if } \neg p \wedge \neg q \wedge \neg r \\ X & \text{otherwise} \end{cases}$$

...

$$a_6 = \begin{cases} 1 & \text{if } p \wedge q \wedge r \\ 0 & \text{if } p \wedge q \wedge \neg r \\ X & \text{otherwise} \end{cases}$$

We can represent quaternary values as a pair of Boolean values during simulation, and thus use the standard BDD representation in terms of the parameters.

## Symbolic trajectory evaluation

---

Symbolic trajectory evaluation (STE) is a further development of ternary symbolic simulation.

The user can write specifications in a restricted *temporal logic*, specifying the behaviour over bounded-length *trajectories* (sequences of circuit states).

A typical specification would be: if the current state satisfies a property  $P$ , then after  $n$  time steps, the state will satisfy the property  $Q$ .

The circuit can then be checked against this specification by symbolic quaternary simulation.

STE is used extensively at Intel.

## Summary

---

- Simulation is the standard technique for testing and debugging circuit designs
- The two generalizations to *ternary* and *symbolic* simulation are independently valuable.
- A canonical representation of Boolean functions using BDDs often works well
- We can generalize simulation to quaternary simulation, considering it as an abstraction mapping
- STE arises by combining symbolic and ternary simulation, and using it to check properties expressed in a simple temporal logic.