Formal Verification Methods
3: Model Checking

John Harrison
Intel Corporation

Marktoberdorf 2003

Fri 1st August 2003 (11:25 – 12:10)

This presentation was not given in Marktoberdorf since it mostly duplicates Orna Grumberg's material

Summary

- Sequential circuits
- State transition systems
- Symbolic state representation
- Bounded model checking
- Unbounded reachability
- Temporal logic

Sequential circuits

We now generalize from *combinational* circuits, where we consider a fixed time interval.

In *sequential* circuits, there are state-holding elements, called *latches* or *flip-flops*.

We consider mainly *synchronous* circuits where the latches change value together according to a single *clock*.

However, there is also some interest in *asynchronous* circuits, and the techniques here can be applied there too.

Modelling sequential circuits

In combinational circuits, we considered the output(s) as Boolean function(s) of the inputs, with one basic Boolean value for each input.

To model combinational circuits, we introduce Boolean values for:

- The inputs (as before)
- The values of the latches

Each of these is considered to vary with time. Instead of just a value in $\mathbb{B}$, we consider it as a function from time to $\mathbb{B}$.

Since we consider sequential circuits, we consider the Boolean values as mappings $\mathbb{N} \to \mathbb{B}$.

Modelling sequential circuits

If there are n inputs and latches:

- The *state* $\sigma = (b_1, \ldots, b_n)$ of the circuit at time t is just the set of values of the inputs and latches at time t.
- The state space (set of possible states) is $\Sigma = \mathbb{B}^n$

We model the circuit itself as a transition relation $R \subseteq \Sigma \times \Sigma$:

- The relation $R(\sigma, \tau)$ holds if and only if the circuit when in state σ at time t may reach τ at time $t + 1$.
- The possible *trajectories* for the circuit are sequences $\sigma_0, \sigma_1, \sigma_2, \ldots$ with $R(\sigma_i, \sigma_{i+1})$ for all $t \in \mathbb{N}$.

Counter example

Consider a circuit with three latches v_0, v_1 and v_2 implementing a modulo-5 counter. State space $S = \mathbb{B} \times \mathbb{B} \times \mathbb{B}$, transition relation:

$$
\begin{aligned}
\{ \ & ((0,0,0),(0,0,1)),((0,0,1),(0,1,0)), \\
& ((0,1,0),(0,1,1)),((0,1,1),(1,0,0)),((1,0,0),(0,0,0))\}
\end{aligned}
$$

i.e, the following transitions are possible:

$$
\begin{array}{ll}
(0,0,0) \rightarrow (0,0,1) & (0,0,1) \rightarrow (0,1,0) \\
(0,1,0) \rightarrow (0,1,1) & (0,1,1) \rightarrow (1,0,0) \\
(1,0,0) \rightarrow (0,0,0) &
\end{array}
$$

Finite state transition systems

A circuit is nothing other than a finite state transition system (a.k.a. finite automaton, Kripke structure ...). Simply a finite state space Σ and a transition relation $R \subseteq \Sigma \times \Sigma$:

However, those arising from modelling circuits have the properties:

- The transition relation is deterministic, i.e. if $R(\sigma, \tau)$ and $R(\sigma, \tau')$ then $\tau = \tau'$.
- The state space is the Cartesian product of Boolean variables, $\Sigma = \mathbb{B}^n$.

Model checking works fine without determinism, and we can then apply it to other interesting state transition systems.

However, even then it's useful to represent the state space using Boolean variables ...

Symbolic state representation

Instead of *enumerating* all the elements of the transition relation, we can represent it *symbolically*.

- Use n Boolean variables v_1, ..., v_n for the 'current state'
- Use another n of them, v'_1, ..., v'_n for the 'next state'

We can then just represent the transition relation R as a Boolean formula $r(\overline{v}, \overline{v}')$. For the counter:

$$
\begin{aligned}
&(v'_0 \Leftrightarrow \neg v_0 \wedge \neg v_2) \wedge \\
&(v'_1 \Leftrightarrow \neg(v_0 \Leftrightarrow v_1)) \wedge \\
&(v'_2 \Leftrightarrow v_0 \wedge v_1)
\end{aligned}
$$

This is so useful that we use a Boolean parametrization of the state space for *any* transition system we are interested in.

Reachability

Many fundamentally interesting questions about finite state transition systems are about *reachability*:

- Starting from a state in $S \subseteq \Sigma$, can we reach a state in $S' \subseteq \Sigma$?

In the symbolic representation, subsets of the state space are represented by Boolean formulas.

For the counter, the formula:

$$v_0 \vee v_1 \vee v_2$$

represents the set of states where the count is nonzero.

If by 'reachable' we mean reachable in 1 cycle, then in the symbolic representation we just need to ask if the formula

$$s(\overline{v}) \wedge r(\overline{v}, \overline{v}') \wedge s'(\overline{v'})$$

is satisfiable. For example, if we ask 'can we get from a state where the count is zero to a state where it is nonzero':

$$\begin{aligned}
&\neg v_0 \wedge \neg v_1 \wedge \neg v_2 \wedge \\
&(v_0' \Leftrightarrow \neg v_0 \wedge \neg v_2) \wedge \\
&(v_1' \Leftrightarrow \neg(v_0 \Leftrightarrow v_1)) \wedge \\
&(v_2' \Leftrightarrow v_0 \wedge v_1) \wedge \\
&(v_0' \vee v_1' \vee v_2')
\end{aligned}$$

More generally, we can consider reachability in k states for some fixed k. Duplicate the set of state variables

$$\overline{v^{(0)}}, \overline{v^{(1)}}, \ldots, \overline{v^{(k)}}$$

and ask if the following formula is satisfiable:

$$\begin{array}{l} s(\overline{v^{(0)}}) \wedge \\ r(\overline{v^{(0)}}, \overline{v^{(1)}}) \wedge r(\overline{v^{(1)}}, \overline{v^{(2)}}) \wedge \cdots \wedge r(\overline{v^{(k-1)}}, \overline{v^{(k)}}) \wedge \\ s'(\overline{v^{(k)}}) \end{array}$$

Because such efficient satisfiability-testing methods are available, this is usually much more efficient than using non-symbolic representations.

Unbounded reachability

What if we consider reachability in *any finite* number of steps?

Essentially we need some sort of reflexive transitive closure operation. There are two main variants:

- *Forward reachability*: find S^*, the set of states reachable from S by some finite number of R-transitions, and see if $S^* \cap S' \neq \emptyset$.
- *Backward reachability*: find S'_*, the set of states that can reach a state in S' by some finite number of R-transitions, and see if $S \cap S'_* \neq \emptyset$.

Sometimes one or the other is more efficient.

We focus on *backward* reachability, because it generalizes to more complicated temporal properties.

Fixpoint computation

Given a set of states S, we can find S_* by iterating:

$$\begin{aligned} S_0 &= \emptyset \\ S_{i+1} &= S \cup \{a \mid \exists b \in S_i.\, R(a,b)\} \end{aligned}$$

We always have $S_i \subseteq S_{i+1} \subseteq \Sigma$.

Since Σ is finite, we eventually reach a fixed point $S_* = S_k$ for some k.

Relational product

We can translate this fixpoint computation into the symbolic representation using BDD operations for basic logical connectives.

BDDs are canonical so we can recognize immediately when a fixpoint is reached.

$$\begin{aligned} s_0 &= \bot \\ s_{i+1} &= s \vee Pre(s_i) \end{aligned}$$

The only new component is the 'relational product' operation Pre:

$$Pre(s) = \exists v_1', \ldots, v_n'.\ \ r[v_1, \ldots, v_n, v_1', \ldots, v_n'] \wedge s[v_1', \ldots, v_n']$$

This is quite easy to implement as a BDD operation — though it's often the main computational bottleneck.

Temporal Operators

Instead of just using propositional logic, we can introduce additional *temporal operators*:

$EX(p)$	There is a successor state where p
$AX(p)$	In all successor states, p
$EF(p)$	There is a path along which p somewhere
$EG(p)$	There is a path along which always p
$AF(p)$	Along all paths, p somewhere
$AG(p)$	For all paths, always p

In this context, p_* is simply the 'semantics' of the temporal formula $EF(p)$, i.e. the set of states satisfying $EF(p)$.

We can state more interesting properties than pure reachability, e.g. request-acknowledge $AG(r \Rightarrow AF(a))$

Together with more complicated binary temporal ‘until’ connectives, this gives *Computation Tree Logic*.

The semantics of all the CTL operators can be found using very similar fixpoint computations to $EF(p)$. For $EG(p)$ we do:

$$\begin{aligned} s_0 &= \top \\ s_{i+1} &= s \wedge Pre(s_i) \end{aligned}$$

for $EX(p)$ we just need $Pre(p)$, and we can deal with the others using duality, e.g.

$$AG(p) \mapsto \neg(EF(\neg p))$$

This process of finding a semantics for a CTL formula w.r.t. a transition system is called *CTL model checking*.

There are many variants of temporal logic. The two main classifications are:

- Branching time (e.g. CTL)
- Linear time (e.g. LTL).

In branching time logics, we can explicitly quantify over the set of possible successor states (E or A).

In linear time logics, we just consider *all* paths.

Neither LTL nor CTL includes the other (e.g. we can express ‘along all paths, p is true infinitely often’ only in LTL).

There are generalizations that take in both, e.g. CTL* and the modal μ-calculus.

Applications

Model checking can be useful in verifying or finding bugs in designs, and is widely used in digital circuits.

It can also be used to analyze software, protocols and anything else that can be modelled with a finite state transition system.

The main drawback is that even with the symbolic representation, it is not feasible to make the computations on really large systems.

STE usually handles large systems better because of its built-in abstraction, but can only consider properties in a very restricted temporal logic.

Commonly, STE is used for data, and CTL for control.

Summary

- We can model sequential circuits, and also many other things, as finite state transition systems.
- A symbolic (e.g. BDD) representation often makes it feasible to analyze surprisingly large systems.
- The most basic, and useful, operations, is reachability, and this can be computed on the symbolic representation using the relational product.
- This generalizes directly to temporal logics like CTL, giving a useful *model checking* algorithm.
- Various temporal logics exist, with different expressive powers.
- Temporal logic model checking and STE complement each other well, and there is active research into generalizations of STE.