Symbolic model checking

Why?
Saves is from constructing a model's state space. Effective "cure" for state space explosion problem.

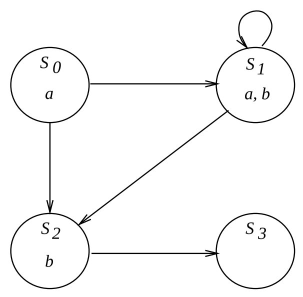
How?
Sets of states and transition relations are represented by formulas, and set operations are defined in terms of formula manipulations.

Data structures
BDDs - allow for efficient storage and manipulation of logic formulas.

Symbolic Model Checking

- A system state represents an interpretation (truth assignment) for a set of propositional variables V.

- Formulas represent sets of states that satisfy it
a - set of states in which a is true - $(\{s_0, s_1\})$
b - set of states in which b is true - $(\{s_1, s_2\})$
$a \vee b = \{s_0, s_1, s_2\}$



- State transitions are described by relations over two sets of variables, V (source state) and V' (destination state)

Transition from s_2 to s_3 is described by $(\neg a \wedge b \wedge \neg a' \wedge \neg b')$.

Transition from s_0 to s_1 and s_2, and from s_1 to s_2 and to itself is described by $(a \wedge b')$.

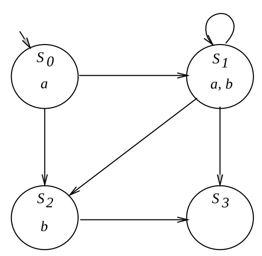
Relation R is described by $(a \wedge b') \vee (\neg a \wedge b \wedge \neg a' \wedge \neg b')$

Symbolic model checking (Cont'd)

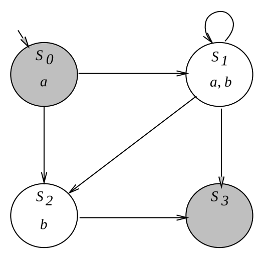
The meaning for CTL formulas can be redefined in terms of sets of states:

$$\begin{array}{rll}
s \models f & \text{iff} & s \in f \text{ where } f \in V \\
s \models \neg f & \text{iff} & s \in \neg f \\
s \models f \vee g & \text{iff} & s \in (f \vee g) \\
s \models \mathsf{EX} f & \text{iff} & s \in (\exists V'(R \wedge f(V/V'))) \\
s \models \mathsf{AX} f & \text{iff} & s \in \neg(\exists V'(R \wedge \neg f(V/V'))) \\
s \models \mathsf{E}(f \mathsf{U} g) & \text{iff} & s \in \mu y.(g \vee (f \wedge \mathsf{EX} \neg y)) \\
s \models \mathsf{A}(f \mathsf{U} g) & \text{iff} & s \in \mu y.(g \vee (f \wedge \mathsf{AX} y))
\end{array}$$

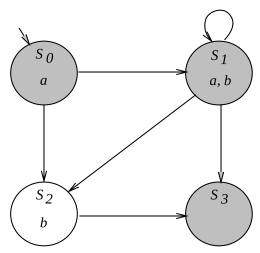
Example: $M, s_2 \models \mathsf{E}(a \mathsf{U} \neg b)$



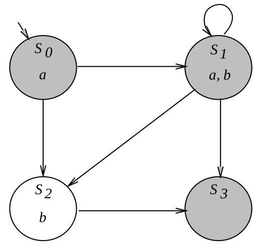
1. Model



2. ~b



3. ~b ∨ (a ∧ EX E[a U ~b])



4. ~b ∨ (a ∧ EX E[a U ~b]) ∨ (a ∧ EX EX E[a U~b])

Symbolic Model-Checking Algorithm

Procedure MC(p)
Case

```
$p \in A$                :  **return** $I_p^{-1}(\top)$
$p = \neg\varphi$         :  **return** $(S - \varphi)$
$p = \varphi \wedge \psi$  :  **return** $(\varphi \cap \psi)$
$p = \varphi \vee \psi$    :  **return** $(\varphi \cup \psi)$
$p = EX\varphi$           :  **return** $\mathtt{pre}(\varphi)$
$p = AX\varphi$           :  **return** $(S - \mathtt{pre}(S - \varphi))$
$p = E[\varphi U \psi]$    :  $Q_0 = \emptyset$
                             $Q_{i+1} = Q_i \cup (\psi \vee (\varphi \wedge EXQ_i))$
                             **return** $Q_n$ when $Q_n = Q_{n+1}$
$p = A[\varphi U \psi]$    :  $Q_0 = \emptyset$
                             $Q_{i+1} = Q_i \cup (\psi \vee (\varphi \wedge EXQ_i \wedge AXQ_i))$
                             **return** $Q_n$ when $Q_n = Q_{n+1}$
```

where $\mathtt{pre}(Q) \equiv \{s \mid t \in Q \wedge (s,t) \in R\}$ (all states that can reach elements in Q in one step).

Symbolic Model-Checking Algorithm on BDDs

Procedure MC(p)
Case

```
$p \in A$                :  **return** *Build*("p")
$p = \neg\varphi$         :  **return** *Apply*('$\neg$', MC($\varphi$))
$p = \varphi \wedge \psi$  :  **return** *Apply*('$\wedge$', MC($\varphi$), MC($\psi$))
$p = \varphi \vee \psi$    :  **return** *Apply*('$\wedge$', MC($\varphi$), MC($\psi$))
$p = EX\varphi$           :  **return** *Quantify*($V'$,
                                 *Apply*('$\wedge$', $R$, *Prime*(MC($\varphi$)))
$p = AX\varphi$           :  **return** *Apply*('$\neg$', MC($EX\ \neg\varphi$))
$p = E[\varphi U \psi]$    :  $Q_0$ = *Build*('$\bot$')
                             $Q_{i+1}$ = *Apply*('$\vee$', $Q_i$, *Apply*('$\vee$', MC($\psi$),
                                *Apply*('$\wedge$', MC($\varphi$), MC($EX\ Q_i$))))
                             **return** $Q_n$ when $Q_n = Q_{n+1}$
$p = A[\varphi U \psi]$    :  $Q_0$ = *Build*('$\bot$')
                             $Q_{i+1}$ = *Apply*('$\vee$', $Q_i$, *Apply*('$\vee$', MC($\psi$),
                                *Apply*('$\wedge$', MC($\varphi$),
                                  *Apply*('$\wedge$', MC($EX\ Q_i$), MC($AX\ Q_i$)))))
                             **return** $Q_n$ when $Q_n = Q_{n+1}$
```

Abstractions

Effective model-checking is impossible without the use of abstraction!

- "Machete style": decrease the number of processors, decrease the desired length of counterexample, etc.
  - Advantages:
  - Disadvantages:

- Remove variables that are not important to the property being verified (*slicing*)
  - One problem: non-determinism

- Abstract "big" variables (integers, large ranges) with variables with smaller ranges.
  - Replace integers with bits, bytes, if possible
  - Boolean predicates ($x > 0 \rightarrow$ PossX())
  - Modulo arithmetic
  - Any other criterion that correctly characterizes the property at hand

A lot of research work on property-preserving abstractions. Still, the burden is largely the developer's.

Pros and Cons of Model-Checking

- Often cannot express full requirements
  - instead, check several smaller properties
- Few real systems have sufficiently small state space to allow direct checking
  - must generally abstract them or "downscale" them. Abstractions may enable checking systems with virtually unlimited number of states
- Largely automatic and fast
- Produces counterexamples
- Can handle systems with 100-200 state variables
- Generally used for debugging rather than assurance
- Usually, find more problems by exploring *all* the behaviors of a downscaled system than by testing only *some* of the behaviors of full system.