Remember Symbolic Model Checking

• Represent sets of states and the transition relation as Boolean logic formulas

• Fixpoint computation becomes formula manipulation
  – pre-condition (EX) computation: Existential variable elimination
  – conjunction (intersection), disjunction (union) and negation (set difference), and equivalence check

• Use an efficient data structure for boolean logic formulas
  – Binary Decision Diagrams (BDDs)
An Extremely Simple Example

Variables: x, y: boolean

Set of states:
S = {(F,F), (F,T), (T,F), (T,T)}
S ≡ True

Initial condition:
I ≡ ¬x ∧ ¬y

Transition relation (negates one variable at a time):
R ≡ x’ = ¬x ∧ y’ = y ∨ x’ = x ∧ y’ = ¬y

(= means ↔)
An Extremely Simple Example

• Assume that we want to check if this transition system satisfies the property $\text{AG}(\neg x \lor \neg y)$
• Instead of checking $\text{AG}(\neg x \lor \neg y)$ we can check $\text{EF}(x \land y)$
  – Since $\text{AG}(\neg x \lor \neg y) \equiv \neg \text{EF}(x \land y)$
    
    $I \subseteq \text{AG}(\neg x \lor \neg y)$ if and only if $I \cap \text{EF}(x \land y) = \emptyset$

• If we find an initial state which satisfies $\text{EF}(x \land y)$ (i.e., there exists a path from an initial state where eventually $x$ and $y$ both become true at the same time)
  – Then we conclude that the property $\text{AG}(\neg x \lor \neg y)$ does not hold for this transition system

• If there is no such initial state, then property $\text{AG}(\neg x \lor \neg y)$ holds for this transition system
An Extremely Simple Example

Given \( p \equiv x \land y \), compute \( \text{EX}(p) \)

\[
\text{EX}(p) \equiv \exists V' \ R \land p[V' / V]
\equiv \exists V' \ R \land x' \land y'
\equiv \exists V' \ (x' = \neg x \land y' = y \lor x' = x \land y' = \neg y) \land x' \land y'
\equiv \exists V' \ (x' = \neg x \land y' = y) \land x' \land y' \lor (x' = x \land y' = \neg y) \land x' \land y'
\equiv \exists V' \ \neg x \land y \land x' \land y' \lor x \land \neg y \land x' \land y'
\equiv \neg x \land y \lor x \land \neg y
\]

\[
\text{EX}(x \land y) \equiv \neg x \land y \lor x \land \neg y
\]

In other words \( \text{EX}((T,T)) \equiv \{(F,T), (T,F)\} \)
An Extremely Simple Example

Let’s compute compute $EF(x \land y)$

The fixpoint sequence is

False, $x \land y$, $x \land y \lor EX(x \land y)$, $x \land y \lor EX(x \land y) \lor EX(x \land y)$, ...

If we do the EX computations, we get:

False, $x \land y$, $x \land y \lor \neg x \land y \lor x \land \neg y$, True

$EF(x \land y) \equiv True \equiv \{(F,F),(F,T),(T,F),(T,T)\}$

This transition system violates the property $AG(\neg x \lor \neg y)$ since it has an initial state that satisfies the property $EF(x \land y)$
Bounded Model Checking

• Represent sets of states and the transition relation as Boolean logic formulas

• Instead of computing the fixpoints, unroll the transition relation up to certain fixed bound and search for violations of the property within that bound

• Transform this search to a Boolean satisfiability problem and solve it using a SAT solver
Same Extremely Simple Example

Variables: \( x, y \): boolean

Set of states:
\[ S = \{(F,F), (F,T), (T,F), (T,T)\} \]
\[ S \equiv \text{True} \]

Initial condition:
\[ I(x,y) \equiv \neg x \land \neg y \]

Transition relation (negates one variable at a time):
\[ R(x,y,x',y') \equiv x' = \neg x \land y' = y \lor x' = x \land y' = \neg y \quad (= \text{means } \leftrightarrow) \]
Bounded Model Checking

• Assume that we like to check that if the initial states satisfy the formula $EF(x \land y)$

• Instead of computing a backward fixpoint, we will unroll the transition relation a fixed number of times starting from the initial states

• **For each unrolling we will create a new set of variables:**
  – The initial states of the system will be characterized with the variables $x_0$ and $y_0$
  – The states of the system after executing one transition will be characterized with the variables $x_1$ and $y_1$
  – The states of the system after executing two transitions will be characterized with the variables $x_2$ and $y_2$
Unrolling the Transition Relation

- **Initial states:** \( I(x_0, y_0) \equiv \neg x_0 \land \neg y_0 \)
- **Unrolling the transition relation once (bound k=1):**
  \[
  I(x_0, y_0) \land R(x_0, y_0, x_1, y_1)
  \]
  \[
  \equiv \neg x_0 \land \neg y_0 \land (x_1 = \neg x_0 \land y_1 = y_0 \lor x_1 = x_0 \land y_1 = \neg y_0)
  \]
- **Unrolling the transition relation twice (bound k=2):**
  \[
  I(x_0, y_0) \land R(x_0, y_0, x_1, y_1) \land R(x_1, y_1, x_2, y_2)
  \]
  \[
  \equiv \neg x_0 \land \neg y_0 \land (x_1 = \neg x_0 \land y_1 = y_0 \lor x_1 = x_0 \land y_1 = \neg y_0)
  \]
  \[
  \land (x_2 = \neg x_1 \land y_2 = y_1 \lor x_2 = x_1 \land y_2 = \neg y_1)
  \]
- **Unrolling the transition relation thrice (bound k=3):**
  \[
  I(x_0, y_0) \land R(x_0, y_0, x_1, y_1) \land R(x_1, y_1, x_2, y_2) \land R(x_2, y_2, x_3, y_3)
  \]
  \[
  \equiv \neg x_0 \land \neg y_0 \land (x_1 = \neg x_0 \land y_1 = y_0 \lor x_1 = x_0 \land y_1 = \neg y_0)
  \]
  \[
  \land (x_2 = \neg x_1 \land y_2 = y_1 \lor x_2 = x_1 \land y_2 = \neg y_1)
  \]
  \[
  \land (x_3 = \neg x_2 \land y_3 = y_2 \lor x_3 = x_2 \land y_3 = \neg y_2)
  \]
Expressing the Property

• How do we represent the property we wish to verify?

• Remember the property: We were interested in finding out if some initial state satisfies EF(x ∧ y)

  – This is equivalent to checking if x ∧ y holds in some reachable state

  – If we are doing bounded model checking with bound k=3, we can express this property as:

    \[ x_0 ∧ y_0 ∨ x_1 ∧ y_1 ∨ x_2 ∧ y_2 ∨ x_3 ∧ y_3 \]
Converting to Satisfiability

• We end up with the following formula for bound k=3:

\[ F \equiv l(x_0, y_0) \land R(x_0, y_0, x_1, y_1) \land R(x_1, y_1, x_2, y_2) \land R(x_2, y_2, x_3, y_3) \]

\[ \land (x_0 \land y_0 \lor x_1 \land y_1 \lor x_2 \land y_2 \lor x_3 \land y_3) \]

\[ \equiv \neg x_0 \land \neg y_0 \land (x_1 = \neg x_0 \land y_1 = y_0 \lor x_1 = x_0 \land y_1 = \neg y_0) \]

\[ \land (x_2 = \neg x_1 \land y_2 = y_1 \lor x_2 = x_1 \land y_2 = \neg y_1) \]

\[ \land (x_3 = \neg x_2 \land y_3 = y_2 \lor x_3 = x_2 \land y_3 = \neg y_2) \]

\[ \land (x_0 \land y_0 \lor x_1 \land y_1 \lor x_2 \land y_2 \lor x_3 \land y_3) \]

• Here is the main observation: if F is a satisfiable formula then there exists an initial state which satisfies EF(x \land y)

  – A satisfying assignment to the boolean variables in F corresponds to a counter-example for AG(\neg x \lor \neg y) (i.e., a witness for EF(x \land y))
The Result

\[ F \equiv \neg x_0 \land \neg y_0 \land (x_1 = \neg x_0 \land y_1 = y_0 \lor x_1 = x_0 \land y_1 = \neg y_0) \]
\[ \land (x_2 = \neg x_1 \land y_2 = y_1 \lor x_2 = x_1 \land y_2 = \neg y_1) \]
\[ \land (x_3 = \neg x_2 \land y_3 = y_2 \lor x_3 = x_2 \land y_3 = \neg y_2) \]
\[ \land (x_0 \lor y_0 \land x_1 \land y_1 \land x_2 \land y_2 \land x_3 \land y_3) \]

Here is a satisfying assignment:

\[ x_0 = F, \; y_0 = F, \; x_1 = F, \; y_1 = T, \; x_2 = T, \; y_2 = T, \; x_3 = F, \; y_3 = T \]

which corresponds to the (bounded) path:

\( (F,F), \; (F,T), \; (T,T), \; (F,T) \)
What Can We Guarantee?

• We converted checking property AG(p) to Boolean SAT solving by looking for bounded paths that satisfy EF(¬p)
• Note that we are checking only for bounded paths (paths which have at most k+1 distinct states)
  – So if the property is violated by only paths with more than k+1 distinct states, we would not find a counter-example using bounded model checking
  – Hence if we do not find a counter-example using bounded model checking we are not sure that the property holds
• However, if we find a counter-example, then we are sure that the property is violated since the generated counter-example is never spurious (i.e., it is always a concrete counter-example)
Bounded Model Checking for LTL

• It is possible to extend the basic ideas we discussed for verifying properties of the form $\text{AG}(p)$ to all LTL (and even ACTL*) properties.

• The basic observation is that we can define a bounded semantics for LTL properties so that if a path satisfies an LTL property based on the bounded semantics, then it satisfies the property based on the unbounded semantics.
  – This is why a counter-example found on a bounded path is guaranteed to be a real counter-example.
  – However, this does not guarantee correctness.
Bounded Model Checking: Proving Correctness

• One can also show that given an LTL property \( f \), if \( E f \) holds for a finite state transition system, then \( E f \) also holds for that transition system using bounded semantics for some bound \( k \)

• So if we keep increasing the bound, then we are guaranteed to find a path that satisfies the formula
  – And, if we do not find a path that satisfies the formula, then we decide that the formula is not satisfied by the transition system
  – Is there a problem here?
Proving Correctness

• We can modify the bounded model checking algorithm as follows:
  – Start from an initial bound.
  – If no counter-examples are found using the current bound, increment the bound and try again.

• The problem is: We do not know when to stop
Proving Correctness

• If we can find a way to figure out when we should stop then we would be able to provide guarantee of correctness.

• There is a way to define a diameter of a transition system so that a property holds for the transition system if and only if it is not violated on a path bounded by the diameter.

• So if we do bounded model checking using the diameter of the system as our bound, then we can guarantee correctness if no counter-example is found.
Bounded Model Checking

• What are the differences between bounded model checking and BDD-based symbolic model checking?
  – In bounded model checking we are using a SAT solver instead of a BDD library
  – In symbolic model checking we do not unroll the transition relation as in bounded model checking
  – In bounded model checking we do not compute the fixpoint as in symbolic model checking
  – In symbolic model checking for finite state systems both verification and falsification results are guaranteed
    • In bounded model checking we can only guarantee the falsification results, in order to guarantee the verification results we need to know the diameter of the system
Bounded Model Checking

• Boolean satisfiability problem (SAT) is an NP-complete problem

• A bounded model checker needs an efficient SAT solver
  – zChaff SAT solver is one of the most commonly used ones
  – However, in the worst case any SAT solver we know will take exponential time

• Most SAT solvers require their input to be in Conjunctive Normal Form (CNF)
  – So the final formula has to be converted to CNF
Bounded Model Checking

• Similar to BDD-based symbolic model checking, bounded model checking was also first used for hardware verification.

• Later on, it was applied to software verification.
Bounded Model Checking for Software

CBMC is a bounded model checker for ANSI-C programs

- Handles function calls using inlining
- Unwinds the loops a fixed number of times
- Allows user input to be modeled using non-determinism
  - So that a program can be checked for a set of inputs rather than a single input
- Allows specification of assertions which are checked using the bounded model checking
Loops

- Unwind the loop n times by duplicating the loop body n times
  - Each copy is guarded using an if statement that checks the loop condition
- At the end of the n repetitions an unwinding assertion is added which is the negation of the loop condition
  - Hence if the loop iterates more than n times in some execution, the unwinding assertion will be violated and we know that we need to increase the bound in order to guarantee correctness
- A similar strategy is used for recursive function calls
  - The recursion is unwound up to a certain bound and then an assertion is generated stating that the recursion does not go any deeper
A Simple Loop Example

Original code

```plaintext
x=0;
while (x < 2) {
    y=y+x;
    x++;
}
```

Unwinding the loop 3 times

```plaintext
x=0;
if (x < 2) {
    y=y+x;
    x++;
}
if (x < 2) {
    y=y+x;
    x++;
}
if (x < 2) {
    y=y+x;
    x++;
}
```

Unwinding assertion:

```plaintext
assert (! (x < 2))
```
From Code to SAT

• After eliminating loops and recursion, CBMC converts the input program to the static single assignment (SSA) form
  – In SSA each variable appears at the left hand side of an assignment only once
  – This is a standard program transformation that is performed by creating new variables
• In the resulting program each variable is assigned a value only once and all the branches are forward branches (there is no backward edge in the control flow graph)
• CBMC generates a Boolean logic formula from the program using bit vectors to represent variables
Another Simple Example

Original code

```c
x=x+y;
if (x!=1)
  x=2;
else
  x++;
assert(x<=3);
```

Convert to static single assignment

```c
x_1=x_0+y_0;
if (x_1!=1)
  x_2=2;
else
  x_3=x_1+1;
x_4=(x_1!=1)?x_2:x_3;
assert(x_4<=3);
```

Generate constraints

\[
C \equiv x_1=x_0+y_0 \land x_2=2 \land x_3=x_1+1 \land (x_1!=1 \land x_4=x_2 \lor x_1=1 \land x_4=x_3)
\]

\[
P \equiv x_4 <= 3
\]

Check if \( C \land \neg P \) is satisfiable, if it is then the assertion is violated

\( C \land \neg P \) is converted to boolean logic using a bit vector representation for the integer variables \( y_0, x_0, x_1, x_2, x_3, x_4 \)
Bounded Verification Approaches

• What we have discussed above is bounded verification by bounding the number of steps of the execution.
• For this approach to work, the variable domains also need to be bounded, otherwise we cannot convert the problems to boolean SAT
• Bounding the execution steps and bounding the data domain are two orthogonal approaches.
  – When people say bounded verification it may refer to either of these
  – When people say bounded model checking, it typically refers to bounding the execution steps