Model Checking
Foundations and Applications

Tevfik Bultan
Department of Computer Science
University of California, Santa Barbara
bultan@cs.ucsb.edu
http://www.cs.ucsb.edu/~bultan/

Outline

• Temporal Logics for Reactive Systems
• Automated Verification of Temporal Properties of Finite State Systems
• Temporal Properties = Fixpoints
• Symbolic Model Checking
  – SMV
• LTL Properties = Büchi automata
  – SPIN
• Infinite State Model Checking
• Model Checking Programs
  – SLAM project
  – Java Path Finder

Temporal Logics for Reactive Systems
[Pnueli FOCS 77, TCS 81]

Transformational systems
get input;
compute something;
return result;

Reactive systems
while (true) {
  receive some input,
send some output
}

• Transformational view follows
  from the initial use of computers
  as advanced calculators: A
  component receives some input,
does some calculation and then
  returns a result.

• Nowadays, the reactive system
  view seems more natural:
  components which continuously
  interact with each other and their
  environment without terminating

Transformational vs. Reactive Systems

Transformational systems
get input;
(pre-condition)
compute something;
(post-condition)
return result;

Reactive systems
while (true) {
  receive some input,
send some output
}

• Earlier work in verification uses
  the transformational view:
  – halting problem
  – Hoare logic
  – pre and post-conditions
  – partial vs. total correctness

• For reactive systems:
  – termination is not the main
    issue
  – pre and post-conditions are
    not enough

A Mutual Exclusion Protocol

Two concurrently executing processes are trying to enter a
critical section without violating mutual exclusion

Process 1:
while (true) {
  out: a := true; turn := true;
  wait: await (b = false or turn = false);
  cs: a := false;
}

Process 2:
while (true) {
  out: b := true; turn := false;
  wait: await (a = false or turn);
  cs: b := false;
}

Reactive Systems: A Very Simple Model

• We will use a very simple model for reactive systems

• A reactive system generates a set of execution paths

• An execution path is a concatenation of the states
  (configurations) of the system, starting from some initial state

• There is a transition relation which specifies the next-state
  relation, i.e., given a state what are the states that can
  follow that state

• We need an example

A Mutual Exclusion Protocol

Two concurrently executing processes are trying to enter a
critical section without violating mutual exclusion

Process 1:
while (true) {
  out: a := true; turn := true;
  wait: await (b = false or turn = false);
  cs: a := false;
}

Process 2:
while (true) {
  out: b := true; turn := false;
  wait: await (a = false or turn);
  cs: b := false;
}
State Space

The state space of a program can be captured by the valuations of the variables and the program counters - you have to wait until the next lecture for a discussion about the control stack (and recursion) and the heap (and dynamic memory allocation).

For our example, we have two program counters: pc1, pc2
- domains of the program counters: \{out, wait, cs\}
- three boolean variables: turn, a, b
- boolean domain: \{True, False\}

Each state of the program is a valuation of all the variables.

Transition Relation

Transition Relation specifies the next-state relation, i.e., given a state what are the states that can come after that state.

For example, given the initial state \( (o, o, F, F, F) \)
- Process 1 can execute:
  - out: \( a := \text{true}; \text{turn} := \text{true} \)
- or Process 2 can execute:
  - out: \( b := \text{true}; \text{turn} := \text{false} \)
- If process 1 executes, the next state is \( (w, o, T, T, F) \)
- If process 2 executes, the next state is \( (o, w, F, F, T) \)
- So the state pairs \( ((o, o, F, F, F), (w, o, T, T, F)) \) and \( ((o, o, F, F, F), (o, w, F, F, T)) \) are included in the transition relation.

Transition System

A transition system \( T = (S, I, R) \) consists of
- a set of states \( S \)
- a set of initial states \( I \subseteq S \)
- and a transition relation \( R \subseteq S \times S \)

A common assumption in model checking
- \( R \) is total, i.e., for all \( s \in S \), there exists \( s' \) such that \( (s, s') \in R \)

Execution Paths

An execution path is an infinite sequence of states
\( x = s_0, s_1, s_2, \ldots \)
such that
\( s_0 \in I \) and for all \( i \geq 0 \), \( (s_i, s_{i+1}) \in R \)

Notation: For any path \( x \)
- \( x_i \) denotes the \( i \)'th state on the path (i.e., \( s_i \))
- \( x^i \) denotes the \( i \)'th suffix of the path (i.e., \( s_i, s_{i+1}, s_{i+2}, \ldots \))

State Space

Each state can be written as a tuple \( (pc1, pc2, turn, a, b) \)
- Initial states:
  \[ \{ (o, o, F, F, F), (o, o, F, F, T), (o, o, F, T, F), (o, o, F, T, T), (o, o, T, F, F), (o, o, T, F, T), (o, o, T, T, F), (o, o, T, T, T) \} \]
  - initially: \( pc1 = 0 \) and \( pc2 = 0 \)

- How many states total?
  \( 3 \times 3 \times 2 \times 2 \times 2 = 72 \)
eponential in the number of variables and the number of concurrent components.
Execution Paths

A possible execution path:

\[((o,o,F,F,F), (o,w,F,F,T), (o,c,F,F,T))\]^\omega

(\omega means repeat the above three states infinitely many times)

Temporal Logics

- Pnueli proposed using temporal logics for reasoning about the properties of reactive systems
- Temporal logics are a type of modal logics
  - Modal logics were developed to express modalities such as "necessity" or "possibility"
  - Temporal logics focus on the modality of temporal progression
- Temporal logics can be used to express, for example, that:
  - an assertion is an invariant (i.e., it is true all the time)
  - an assertion eventually becomes true (i.e., it will become true sometime in the future)

Temporal Logics

- We will assume that there is a set of basic (atomic) properties called \( AP \)
  - These are used to write the basic (non-temporal) assertions about the program
  - Examples: \( a=true, \ pc0=c, \ x=y+1 \)
- We will use the usual boolean connectives: \( \neg, \land, \lor \)
- We will also use four temporal operators:
  - Invariant \( p \) : \( G \ p \) (aka \( \Box \ p \)) (Globally)
  - Eventually \( p \) : \( F \ p \) (aka \( \lozenge \ p \)) (Future)
  - Next \( p \) : \( X \ p \) (aka \( \square \ p \)) (next)
  - \( p \) Until \( q \) : \( p \ U \ q \)

Atomic Properties

- In order to define the semantics we will need a function \( L \) which evaluates the truth of atomic properties on states:
  \[
  L : S \times AP \rightarrow \{ \text{True, False} \}
  \]
  \[
  L((o,o,F,F,F), pc1=o) = \text{True} \\
  L((o,o,F,F,F), pc1=w) = \text{False} \\
  L((o,o,F,F,F), turn) = \text{False} \\
  L((o,o,F,F,F), turn=false) = \text{True} \\
  L((o,o,F,F,F), pc1=0 \land pc2=0 \land \text{turn} \land \neg a \land \neg b ) = \text{True}
  \]

Linear Time Temporal Logic (LTL) Semantics

Given an execution path \( x \) and LTL properties \( p \) and \( q \)

- \( x \models p \) iff \( L((x_0, p) = \text{True}, \text{where } p \in AP) \)
- \( x \models \neg p \) iff \( \text{not } x \models p \)
- \( x \models p \land q \) iff \( x \models p \) and \( x \models q \)
- \( x \models p \lor q \) iff \( x \models p \) or \( x \models q \)
- \( x \models X p \) iff \( x' \models p \)
- \( x \models G p \) iff for all \( i, x' \models p \)
- \( x \models F p \) iff there exists an \( i \) such that \( x' \models p \)
- \( x \models p \ U q \) iff there exists an \( i \) such that \( x' \models q \) and for all \( j < i, x' \models p \)

LTL Properties

- \( X p \)
- \( G p \)
- \( F p \)
- \( p U q \)
Example Properties

- Mutual exclusion: \( G (\neg (pc1 = c \land pc2 = c)) \)
- Starvation freedom:
  \( G(pc1 = w \Rightarrow F(pc1 = c)) \land G(pc2 = w \Rightarrow F(pc2 = c)) \)

Given the execution path:
\[
\begin{align*}
  (s_1, s_2, s_3, s_4, s_5) &= ((o, o, F, F, F), (o, w, F, F, T), (o, c, F, F, T)) \\
  x |\models pc1 = o \\
  x |\models X (pc2 = w) \\
  x |\models F (pc2 = c) \\
  x |\models (\neg \text{turn}) U (pc2 = c \land \text{b}) \\
  x |\models G (\neg (pc1 = c \land pc2 = c)) \\
  x |\models G(pc1 = w \Rightarrow F(pc1 = c)) \land G(pc2 = w \Rightarrow F(pc2 = c))
\end{align*}
\]

LTL Equivalences

- We do not really need all four temporal operators
  - \( X \) and \( U \) are enough (i.e., \( X, U, AP \) and boolean connectives form a basis for LTL)
  - \( F p = true U p \)
  - \( G p = \neg (F \neg p) = \neg (true U \neg p) \)

LTL Model Checking

- Given a transition system \( T \) and an LTL property \( p \)
  \( T |\models p \iff \text{for all execution paths } x \text{ in } T, x |\models p \)

For example:
\[
\begin{align*}
  &G (\neg (pc1 = c \land pc2 = c)) \\
  &G(pc1 = w \Rightarrow F(pc1 = c)) \land G(pc2 = w \Rightarrow F(pc2 = c))
\end{align*}
\]

Model checking problem: Given a transition system \( T \) and an LTL property \( p \), determine if \( T |\models p \) (i.e., if \( T |\models p \))

Linear Time vs. Branching Time

- In linear time logics given we look at execution paths individually
- In branching time logics we view the computation as a tree
  - computation tree: unroll the transition relation

Computation Tree Logic (CTL)

- In CTL we quantify over the paths in the computation tree
- We use the same four temporal operators: \( X, G, F, U \)
- However we attach path quantifiers to these temporal operators:
  - \( A \) : for all paths
  - \( E \) : there exists a path
- We end up with eight temporal operators:
  - \( AX, EX, AG, EG, AF, EF, AU, EU \)

CTL Semantics

Given a state \( s \) and CTL properties \( p \) and \( q \)
\[
\begin{align*}
  s |\models p &\iff L(s, p) = \text{True, where } p \in AP \\
  s |\models \neg p &\iff \text{not } x |\models p \\
  s |\models p \land q &\iff s |\models p \text{ and } s |\models q \\
  s |\models p \lor q &\iff s |\models p \text{ or } s |\models q \\
  s_0 |\models EX p &\iff \text{there exists a path } s_0, s_1, s_2, \ldots \text{ such that } s_1 |\models p \\
  s_0 |\models AX p &\iff \text{for all paths } s_0, s_1, s_2, \ldots, s_1 |\models p
\end{align*}
\]
### CTL Semantics

- **CTL basis:** EX, EU, EG
  
  - **AX p = ¬ EX ¬ p**
  - **AG p = ¬ EF ¬ p**
  - **AF p = ¬ EG ¬ p**

- **p AU q = ¬ ((¬q EU (¬p ∧ ¬q)) ∨ EG ¬ q)**
  - **EF p = True EU p**

- **Another CTL basis:** EX, EU, AU

### CTL Properties

**Transition System**

- **Computation Tree**

<table>
<thead>
<tr>
<th>State</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3</td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td></td>
</tr>
</tbody>
</table>

### CTL Equivalences

- **CTL basis:** EX, EU, EG
  - **AX p = ¬ EX ¬ p**
  - **AG p = ¬ EF ¬ p**
  - **AF p = ¬ EG ¬ p**

### CTL Model Checking

- **Model checking problem:** Given a transition system T and a CTL property p, determine if T is a model for p (i.e., if T |=p)

  - For example:
    - **T |=? AG (¬ (pc1=c ∧ pc2=c))**
    - **T |=? AG(pc1=w ⇒ AF(pc1=c)) ∧ AG(pc2=w ⇒ AF(pc2=c))**

- **Question:** Are CTL and LTL equivalent?

### CTL vs. LTL

- **CTL and LTL are not equivalent**
  - There are properties that can be expressed in LTL but cannot be expressed in CTL
    - For example: FG p
  - There are properties that can be expressed in CTL but cannot be expressed in LTL
    - For example: AG(EF p)

- Hence, expressive power of CTL and LTL are not comparable

### CTL*

- **CTL** is a temporal logic which is strictly more powerful than CTL and LTL

  - CTL* uses the temporal operators X, F, G, U and the path quantifiers A and E, but temporal operators can also be used without path quantifiers

  - The following CTL* property cannot be expressed in CTL or LTL
    - **A(FG p) ∨ AG(EF p)**
Automated Verification of Finite State Systems
[Clarke and Emerson 81], [Quelle and Sifakis 82]

CTL Model checking problem: Given a transition system \( T = (S, I, R) \), and a CTL formula \( f \), does the transition system satisfy the property?

CTL model checking problem can be solved in \( O(|f| \times (|S|+|R|)) \)

Note that the complexity is linear in the size of the transition system

Recall that the size of the transition system is exponential in the number of variables and concurrent components (this is called the state space explosion problem)

CTL Model Checking Algorithm

• Translate the formula to a formula which uses the basis
  – \( \text{EX} \ p, \ \text{EG} \ p, \ p \text{ EU } q \)

• Start from the innermost (non-atomic) subformulas and label the states in the transition system with the subformulas that hold in that state
  – Initially states are labeled with atomic properties

• Each (temporal or boolean) operator has to be processed once

• Computation of each subformula takes \( O(|S|+|R|) \)

CTL Model Checking Algorithm

• \( \text{EX} \ p \) is easy to do in \( O(|S|+|R|) \)
  – All the nodes which have a next state labeled with \( p \) should be labeled with \( \text{EX} \ p \)

• \( p \text{ EU } q \): Find the states which are the source of a path where \( p \text{ U } q \) holds
  – Equivalently, find the nodes which reach a node that is labeled with \( q \) by a path where each node is labeled with \( p \)
  – Label such nodes with \( p \text{ EU } q \)
  – It is a reachability problem which can be solved in \( O(|S|+|R|) \)

CTL Model Checking Algorithm

• \( \text{EG} \ p \): Find infinite paths where each node is labeled with \( p \) and label nodes in such paths with \( \text{EG} \ p \)
  – First remove all the states which do not satisfy \( p \) from the transition graph
  – Compute the strongly connected components of the remaining graph and then find the nodes which can reach the strongly connected components (both of which can be done in \( O(|S|+|R|) \)
  – Label the nodes in the strongly connected components and that can reach the strongly connected components with \( \text{EG} \ p \)

Verification vs. Falsification

• Verification:
  – Show: initial states \( \subseteq \) truth set of \( p \)

• Falsification:
  – Find: a state \( \subseteq \) initial states \( \cap \) truth set of \( \neg p \)
    – Generate a counter-example starting from that state

• CTL model checking algorithm can also generate a counter-example path if the property is not satisfied
  – without increasing the complexity

• The ability to find counter-examples is one of the biggest strengths of the model checkers

What About LTL and CTL* Model Checking?

• The complexity of the model checking problem for LTL and CTL* are:
  – \( (|S|+|R|) \times 2^{O(|f|)} \)

• Typically the size of the formula is much smaller than the size of the transition system
  – So the exponential complexity in the size of the formula is not very significant in practice
### Temporal Properties ≡ Fixpoints

[Emerson and Clarke 80]

Here are some interesting CTL equivalences:

\[
\begin{align*}
AG p & = p \land AX AG p \\
EG p & = p \land EX EG p \\
AF p & = p \lor AX AF p \\
EF p & = p \lor EX EF p \\
p AU q & = q \lor (p \land AX (p AU q)) \\
p EU q & = q \lor (p \land EX (p EU q))
\end{align*}
\]

Note that we wrote the CTL temporal operators in terms of themselves and EX and AX operators.

---

### Functionals

- Given a transition system \( T = (S, I, R) \), we will define functions from sets of states to sets of states
  \[ F : 2^S \rightarrow 2^S \]
- For example, one such function is the EX operator (which computes the precondition of a set of states)
  \[ EX(p) = \{ s | (s, s') \in R \text{ and } s' \in p \} \]

Abuse of notation: I am using \( p \) to denote the set of states which satisfy the property \( p \)

---

### Lattice

The set of states of the transition system forms a lattice:

- lattice \( 2^S \)
- partial order \( \subseteq \)
- bottom element \( \emptyset \)
- top element \( S \)
- Least upper bound \( \cup \) (aka join) operator
- Greatest lower bound \( \cap \) (aka meet) operator

---

### Temporal Properties ≡ Fixpoints

Based on the equivalence \( EF p = p \lor EX EF p \)

we observe that \( EF p \) is a fixpoint of the following function:

\[
F_y = p \lor EX y
\]

In fact, \( EF p \) is the least fixpoint of \( F \), which is written as:

\[ EF p = \mu y . p \lor EX y \]

Based on the equivalence \( EG p = p \land AX EG p \)

we observe that \( EG p \) is a fixpoint of the following function:

\[
F_y = p \land EX y
\]

In fact, \( EG p \) is the greatest fixpoint of \( F \), which is written as:

\[ EG p = \nu y . p \land EX y \]

---

### Fixpoint Characterizations

<table>
<thead>
<tr>
<th>Fixpoint Characterization</th>
<th>Equivalences</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG p = ( y \land y \lor AX y )</td>
<td>AG p = ( p \land AX AG p )</td>
</tr>
<tr>
<td>EG p = ( y \land y \lor EX y )</td>
<td>EG p = ( p \land EX EG p )</td>
</tr>
<tr>
<td>AF p = ( \mu y . p \lor AX y )</td>
<td>AF p = ( p \lor AX AF p )</td>
</tr>
<tr>
<td>EF p = ( \mu y . p \lor EX y )</td>
<td>EF p = ( p \lor EX EF p )</td>
</tr>
<tr>
<td>p AU q = ( \mu y . q \lor (p \land AX (p AU q)) )</td>
<td>p AU q = ( q \lor (p \land AX (p AU q)) )</td>
</tr>
<tr>
<td>p EU q = ( \mu y . q \lor (p \land EX (p EU q)) )</td>
<td>p EU q = ( q \lor (p \land EX (p EU q)) )</td>
</tr>
</tbody>
</table>
Least and Greatest Fixpoints

The least and greatest fixpoint operators are defined as:

\[ \mu y . F y = \bigcap \{ y \mid F y \subseteq y \} \] (glb of all the reductive elements)

\[ \nu y . F y = \bigcup \{ y \mid F y \supseteq y \} \] (lub of all the extensive elements)

The least fixpoint \( \mu y . F y \) is the limit of the following sequence (assuming \( F \) is \( \cup \)-continuous):

\[ \emptyset, F \emptyset, F^2 \emptyset, F^3 \emptyset, \ldots \]

The greatest fixpoint \( \nu y . F y \) is the limit of the following sequence (assuming \( F \) is \( \cap \)-continuous):

\[ S, pEX S, pEX(p \land EX S), pEX(p \land EX(p \land EX S)), \ldots \]

If \( S \) is finite, then we can compute the least and greatest fixpoints using these sequences.

EF and EG computations

Then, \( EF p = \mu y . p \lor EX y \) is the limit of the sequence:

\[ \emptyset, p \lor EX \emptyset, p \lor EX(p \lor EX \emptyset), p \lor EX(p \lor EX(p \lor EX(\ldots))) \]

which is equivalent to

\[ \emptyset, p \lor EX p, \emptyset \lor EX(p \lor EX(\ldots)) \]

Similarly, \( EG p = \nu y . p \land EX y \) is the limit of the sequence:

\[ S, p \land EX S, p \land EX(p \land EX S), p \land EX(p \land EX(p \land EX S)), \ldots \]

which is equivalent to

\[ S, p, p \land EX p, p \land EX(p \land EX(p \land EX p)), \ldots \]

\[ \mu \text{-Calculus} \]

\( \mu \)-Calculus is a temporal logic which consist of the following:

- Atomic properties \( AP \)
- Boolean connectives: \( \neg, \land, \lor \)
- Precondition operator: \( EX \)
- Least and greatest fixpoint operators: \( \mu y . F y \) and \( \nu y . F y \)

Any CTL* property can be expressed in \( \mu \)-calculus

Symbolic Model Checking

[McMillan et al. LICS 90]

- 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)
Example Mutual Exclusion Protocol

Two concurrently executing processes are trying to enter a critical section without violating mutual exclusion

Process 1:
while (true) {
  out: a := true; turn := true;
  wait: await (b = false or turn = false);
  cs: a := false;
}

Process 2:
while (true) {
  out: b := true; turn := false;
  wait: await (a = false or turn);
  cs: b := false;
}

State Space

- Encode the state space using only boolean variables
  - Two program counters: pc1, pc2 with domains {out, wait, cs}
    - Use two boolean variable per program counter: pc10, pc11, pc20, pc21
    - Encoding:
      - pc1 = out
      - pc1 = wait
      - pc1 = cs
  - The other three variables are booleans: turn, a, b

Initial States

- We can write the initial states as a boolean logic formula
  - recall that, initially: pc1=out and pc2=out

| ¬pc1 ∧ ¬pc1 ∧ ¬pc2 ∧ ¬pc2 ∧ turn ∧ ¬a ∧ ¬b |

Transition Relation

- We can use boolean logic formulas to encode the transition relation
- We will use two sets of variables:
  - Current state variables: pc1, pc2, turn, a, b
  - Next state variables: pc1', pc2', turn, a', b'

For example, we can write a boolean logic formula for the statement:

\[ \text{cs: } a := false; \]

as follows:

\[ \text{pc1 ∧ pc1' ∧ ¬pc1' ∧ ¬pc1 ∧ (pc2' ↔ pc2) ∧ (pc2' ↔ pc2)} \]

- Call this formula \( R_{1c} \)
**Symbolic Pre-condition Computation**

- Remember the function
  \[ \text{EX}: 2^3 \rightarrow 2^3 \]
  which is defined as:
  \[ \text{EX}(p) = \{ s | (s, s') \in R \land s' \in p \} \]
- We can symbolically compute pre as follows
  \[ \text{EX}(p) = 3V' R \land p[V' / V] \]
  - \( V \) : current-state boolean variables
  - \( V' \) : next-state boolean variables
  - \( p[V' / V] \) : rename variables in \( p \) by replacing current-state variables with the corresponding next-state variables
  - \( 3V' f \) : existentially quantify out all the variables in \( V' \) from \( f \)

**Existential Quantifier Elimination**

- Given a boolean formula \( f \) and a single variable \( v \)
  \[ 3v f = \forall v [F\forall v] \]
  i.e., to existentially quantify out a variable, first set it to true then set it to false and then take the disjunction of the two results
- Example: \( f = \neg x \land y \land x' \land y' \)
  \[ 3V' f = 3x' (3y' (\neg x \land y \land x' \land y')) \]
  \[ = 3x' ([T][y'] \lor \neg x \land y \land x' \land y'))[F\forall y'] \]
  \[ = 3x' ([\neg x \land y \land x'] \land (\neg x \land y \land x') \lor \neg x \land y \land x' \land F) \]
  \[ = 3x' \neg x \land y \land x' \]
  \[ = \neg x \land y \land x' \land [T][y'] \lor \neg x \land y \land x' \land y \land F \]
  \[ = \neg x \land y \land y \land F \]
  \[ = \neg x \land y \]

**Renaming**

- Assume that we have two variables \( x, y \).
- Then, \( V = \{ x, y \} \) and \( V' = \{ x', y' \} \)
- Renaming example:
  Given \( p = x \land y \):
  \[ p[V' / V] = x \land y [V' / V] = x' \land y' \]

**An Extremely Simple Example**

Variables: \( x, y \): boolean

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

Initial condition:
\[ I = \neg x \land \neg y \]

Transition relation (negates one variable at a time):
\[ R = (x = \neg x \land y = y \lor x = x \land y' = \neg y) \land x' \land y' \land x = x \land y = y \land \neg y \land x' \land y' \land x = x \land y = y \land \neg y \land x = x \land y = y \land \neg y \land x' \land y' \]

\( \text{EX}(x \land y) = \neg x \land y \lor x \land y \lor \neg y \)

In other words \( \text{EX}(\{(T,T)\}) = \{(F,T), (T,F)\} \)

**An Extremely Simple Example**

Let’s compute \( \text{EF}(x \land y) \)

The fixpoint sequence is
- False, \( x \land y \land \text{EX}(x \land y) \land \text{EX}(x \land y) \lor x \land y \land \text{EX}(x \land y) \)

If we do the EX computations, we get:
False
\[ \begin{array}{c|c|c}
0 & 1 & 2 \\
\hline
\text{False} & x \land y \land \neg x \land y \land \neg y \land x = x \land y = y \land \neg y \land x' \land y' & \text{True} \\
\end{array} \]

\( \text{EF}(x \land y) = \text{True} \)

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

- Based on our results, for our extremely simple transition system \( T=(S,I,R) \) we have

\[
I \subseteq \text{EF}(x \land y) \quad (\subseteq \text{corresponds to implication}) \text{ hence:}
\]

\[
T \models \text{EF}(x \land y)
\]

(i.e., there exists a path from each initial state where eventually \( x \) and \( y \) both become true at the same time)

\[
I \not\subseteq \text{EX}(x \land y) \quad \text{hence:}
\]

\[
T \not\models \text{EX}(x \land y)
\]

(i.e., there does not exist a path from each initial state where in the next state \( x \) and \( y \) both become true)

Let’s try one more property \( \text{AF}(x \land y) \)

To check this property we first convert it to a formula which uses only the temporal operators in our basis:

\[
\text{AF}(x \land y) \equiv \neg \text{EG}(\neg (x \land y))
\]

If we can find an initial state which satisfies \( \text{EG}(\neg (x \land y)) \), then we know that the transition system \( T \), does not satisfy the property \( \text{AF}(x \land y) \)

Symbolic CTL Model Checking Algorithm

- Translate the formula to a formula which uses the basis

\(- \quad \text{EX} \ p, \ \text{EG} \ p, \ p \ \text{EU} \ q\)

- Atomic formulas can be interpreted directly on the state representation

- For \( \text{EX} \ p \) compute the precondition using existential variable elimination as we discussed

- For \( \text{EG} \) and \( \text{EU} \) compute the fixpoints iteratively

Symbolic Model Checking Algorithm

- \( \text{Check}(f : \text{CTL formula}) : \text{boolean logic formula} \)

\[
\text{case: } f \in \text{AP} \quad \text{return } f;
\]

\[
\text{case: } f = \neg p \quad \text{return } \neg \text{Check}(p);
\]

\[
\text{case: } f = p \land q \quad \text{return } \text{Check}(p) \land \text{Check}(q);
\]

\[
\text{case: } f = p \lor q \quad \text{return } \text{Check}(p) \lor \text{Check}(q);
\]

\[
\text{case: } f = \text{EX} \ p \quad \text{return } 3V' R \land \text{Check}(p) [V' / V];
\]
Symbolic Model Checking Algorithm

Check(f)
...
  case: f = p EU q
      Y := False;
      P := Check(p);
      Q := Check(q);
      Y' := Q ∨ P ∧ Check(EX(Y));
      while (Y ≠ Y') {
          Y := Y';
          Y' := Q ∨ P ∧ Check(EX(Y));
      }
  return Y;

Binary Decision Diagrams (BDDs)

• Ordered Binary Decision Diagrams (BDDs)
  – An efficient data structure for boolean formula manipulation
  – There are BDD packages available: (for example CUDD from Colorado University)
• BDD data structure can be used to implement the symbolic model checking algorithm discussed above
  • BDDs are a canonical representation for boolean logic formulas
    – given two boolean logic formulas F and G, if F and G are equivalent their BDD representations will be identical

Binary Decision Trees

Fix a variable order, in each level of the tree branch on the value of the variable in that level

• Examples for boolean formulas on two variables
  Variable order: x, y

BDDs

• Repeatedly apply the following transformations to a binary decision tree:
  – Remove duplicate terminals
  – Remove duplicate non-terminals
  – Remove redundant tests
• These transformations transform the tree to a directed acyclic graph

Good News About BDDs

• Given BDDs for two boolean logic formulas F and G
  – The BDDs for F ∧ G and F ∨ G are of size |F| × |G| (and can be computed in that time)
  – The BDD for ¬F can be computed in and is of size |F| (and can be computed in that time)
  – F =? G can be checked in constant time
  – Satisfiability of F can be checked in constant time
  • No, this does not mean that you can solve SAT in constant time
Bad News About BDDs

- The size of a BDD can be exponential in the number of boolean variables

- The sizes of the BDDs are very sensitive to the variable ordering. Bad variable ordering can cause exponential increase in the size of the BDD

- There are functions which have BDDs that are exponential for any variable ordering (for example binary multiplication)

- Pre condition computation requires existential variable elimination
  - Existential variable elimination can cause an exponential blow-up in the size of the BDD

BDDs are Sensitive to Variable Ordering

Identity relation for two variables: \((x' \leftrightarrow x) \land (y' \leftrightarrow y)\)

<table>
<thead>
<tr>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

For \(n\) variables, \(3^n + 2\) nodes

For \(n\) variables, \(3 \times 2^{n-1}\) nodes

SMV [McMillan 93]

- BDD-based symbolic model checker
- Finite state
- Temporal logic: CTL
- Focus: hardware verification
  - Later applied to software specifications, protocols, etc.
- SMV has its own input specification language
  - concurrency: synchronous, asynchronous
  - shared variables
  - boolean and enumerated variables
  - bounded integer variables (binary encoding)
  - SMV is not efficient for integers, but it can be fixed
  - fixed size arrays

Example Mutual Exclusion Protocol

Two concurrently executing processes are trying to enter a critical section without violating mutual exclusion

Process 1:
```
while (true) {
  out:  a := true; turn := true;
  wait: await (b = false or turn = false);
  cs:   a := false;
}
||
```

Process 2:
```
while (true) {
  out:  b := true; turn := false;
  wait: await (a = false or turn);
  cs:   b := false;
}
```

Example Mutual Exclusion Protocol in SMV

```
MODULE process1(a,b,turn)
VAR
  pc: {out, wait, cs};
ASSIGN
  init(pc) := out;
  next(pc) :=
    case
      pc=out : wait;
      pc=wait & (!b | !turn) : cs;
      pc=cs : out;
    esac;
  next(turn) :=
    case
      pc=out : 1;
      1 : turn;
    esac;
  next(a) :=
    case
      pc=out : 1;
      pc=cs : 0;
      1 : a;
    esac;
  next(b) := b;
FAIRNESS
running
```

```
MODULE process2(a,b,turn)
VAR
  pc: {out, wait, cs};
ASSIGN
  init(pc) := out;
  next(pc) :=
    case
      pc=out : wait;
      pc=wait & (!a | turn) : cs;
      pc=cs : out;
    esac;
  next(turn) :=
    case
      pc=out : 0;
      1 : turn;
    esac;
  next(b) :=
    case
      pc=out : 1;
      pc=cs : 0;
      1 : b;
    esac;
  next(a) := a;
FAIRNESS
running
```

```
MODULE main
VAR
  a : boolean;
  b : boolean;
  turn : boolean;
  p1 : process process1(a,b,turn);
  p2 : process process2(a,b,turn);
SPEC
  AG(!(p1.pc=cs & p2.pc=cs))
  -- AG(p1.pc=wait -> AF(p1.pc=cs)) & AG(p2.pc=wait -> AF(p2.pc=cs))
```

Here is the output when I run SMV on this example to check the mutual exclusion property

```
% smv mutex.smv
-- specification AG (!(p1.pc = cs & p2.pc = cs)) is true
resources used:
user time: 0.01 s, system time: 0 s
BDD nodes allocated: 692
Bytes allocated: 1245184
BDD nodes representing transition relation: 143 + 6
```

Example Mutual Exclusion Protocol in SMV

```
Example Mutual Exclusion Protocol in SMV

Let's insert an error

c change pc=wait & (b | !turn) : cs;
to pc=wait & (!b | turn) : cs;