Model Checking:  
From BDDs to Interpolation

Orna Grumberg  
Technion  
Haifa, Israel

Summer school at Bayrischzell 2011
Why (formal) verification?

• safety-critical applications: Bugs are unacceptable!
  - Air-traffic controllers
  - Medical equipment
  - Cars

• Bugs found in later stages of design are expensive, e.g. Intel’s Pentium bug in floating-point division

• Hardware and software systems grow in size and complexity: Subtle errors are hard to find by testing

• Pressure to reduce time-to-market

Automated tools for formal verification are needed
Formal Verification

Given
- a model of a (hardware or software) system and
- a formal specification

does the system model satisfy the specification?

Not decidable!

To enable automation, we restrict the problem to a decidable one:

- **Finite-state** reactive systems
- **Propositional** temporal logics
Finite state systems - examples

• Hardware designs
• Controllers (elevator, traffic-light)
• Communication protocols (when ignoring the message content)
• High level (abstracted) description of non finite state systems
Properties in temporal logic - examples

• mutual exclusion:
  \text{always} \neg (cs_1 \land cs_2)

• non starvation:
  \text{always} (\text{request} \Rightarrow \text{eventually granted})

• communication protocols:
  \neg \text{get-message) until send-message}
Model Checking \([CE81, QS82]\)

An efficient procedure that receives:
- A finite-state model describing a system
- A temporal logic formula describing a property

It returns
- yes, if the system has the property
- no + Counterexample, otherwise
Model Checking

- Emerging as an industrial standard tool for verification of hardware designs: Intel, IBM, Synopsis, ...

- Recently applied successfully also for software verification: SLAM (Microsoft), Java PathFinder and SPIN (NASA), BLAST (EPFL), CBMC (Oxford), ...
  - SLAM won the 2011 CAV award
Model of a system
Kripke structure / transition system

[Diagram of a model system with states labeled a, b, c, and transitions between them]
Temporal Logics

• Temporal Logics
  – Express properties of event orderings in time

• Linear Time
  - Every moment has a unique successor
  - Infinite sequences (words)
  - Linear Time Temporal Logic (LTL)

• Branching Time
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)
Propositional temporal logic

In Negation Normal Form
AP - a set of atomic propositions

Temporal operators:
- $Gp$
- $Fp$
- $Xp$
- $pUq$

Path quantifiers: $A$ for all path
$E$ there exists a path
CTL/CTL*

- **LTL** - interpreted over infinite computation paths
- **CTL** - interpreted over infinite computation trees
- **CTL** - Allows any combination of temporal operators and path quantifiers. Includes both LTL and CTL

**ACTL / ACTL**

The *universal* fragments of CTL/CTL* with only universal path quantifiers
CTL formulas: Example

- mutual exclusion: \[ AG \neg( cs_1 \land cs_2) \]
- non starvation: \[ AG( request \Rightarrow AF \text{ grant}) \]
- “sanity” check: \[ EF \text{ request} \]
Model checking

A basic operation: Image computation

Given a set of states $Q$, $\text{Image}(Q)$ returns the set of successors of $Q$

$\text{Image}(Q) = \{ s' \mid \exists s [ R(s, s') \land Q(s)] \}$
Model checking $AGq$ on $M$

- Iteratively compute the sets $S_j$ of states reachable from an initial state in $j$ steps.

- At each iteration check whether $S_j$ contains a state satisfying $\neg q$.
  - If so, declare a failure.

- Terminate when all states were found.
  \[ S_k \subseteq \bigcup_{i=0}^{k-1} S_i \]
  - Result: the set $\text{Reach}$ of reachable states.
Model checking $f = AG \ p$

Given a model $M = < S, I, R, L >$
and a set $S_p$ of states satisfying $q$ in $M$

procedure CheckAG ($S_p$)
    Reach = $\emptyset$
    $S_0 = I$
    $k = 0$
    while $S_k \not\subseteq$ Reach do
        If $S_k \cap S_p \neq \emptyset$ return ($M \not\models AGq$)
        $S_{k+1} = \text{Image}(S_k)$
        Reach = Reach $\cup S_k$
        $k = k + 1$
    end while
return(Reach, $M \models AGp$)
Model checking $AGq$

• Also called forward reachability analysis
Mutual Exclusion Example

- Two process mutual exclusion with shared semaphore
- Each process has three states
  - Non-critical (N)
  - Trying (T)
  - Critical (C)
- Semaphore can be available ($S_0$) or taken ($S_1$)
- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1 N_2 S_0$

\[
\begin{align*}
N_1 & \rightarrow T_1 \\
T_1 \wedge S_0 & \rightarrow C_1 \wedge S_1 \\
C_1 & \rightarrow N_1 \wedge S_0 \\
\parallel & \parallel \\
N_2 & \rightarrow T_2 \\
T_2 \wedge S_0 & \rightarrow C_2 \wedge S_1 \\
C_2 & \rightarrow N_2 \wedge S_0
\end{align*}
\]
Mutual Exclusion Example

\[ \text{M} \models \text{AG} \neg (C_1 \land C_2) \]

*The two processes are never in their critical states at the same time*
Mutual Exclusion Example

\[ M \models AG \neg (C_1 \land C_2) \]

\[ S_0 \]
Mutual Exclusion Example

\[ S_1 \]

\[ M \models AG \neg (C1 \land C2) \]
Mutual Exclusion Example

\[ M \models AG \neg (C_1 \land C_2) \]
Mutual Exclusion Example

\[ M \models AG \neg (C_1 \land C_2) \]

\[ S_3 \]
Mutual Exclusion Example

\[
\begin{align*}
M & \models AG \neg (C_1 \land C_2) \\
S_4 & \subseteq S_0 \cup \ldots \cup S_3
\end{align*}
\]
Main limitation:

The state explosion problem:
Model checking is efficient in time but suffers from high space requirements:

The number of states in the system model grows exponentially with
- the number of variables
- the number of components in the system
Symbolic model checking

A solution to the state explosion problem which uses Binary Decision Diagrams (BDDs) to represent the model and sets of states.

- Suitable mainly for hardware
- Can handle systems with hundreds of Boolean variables
Binary decision diagrams (BDDs)

• Data structure for representing Boolean functions
• Often concise in memory
• Canonical representation
• Most Boolean operations on BDDs can be done in polynomial time in the BDD size
BDDs in model checking

• Every set $A \subseteq U$ can be represented by its characteristic function

$$f_A(u) = \begin{cases} 
1 & \text{if } u \in A \\
0 & \text{if } u \notin A
\end{cases}$$

• If the elements of $A$ are encoded by sequences over $\{0,1\}^n$ then $f_A$ is a Boolean function and can be represented by a BDD
Representing a model with BDDs

- Assume that states in model M are encoded by \( \{0,1\}^n \) and described by Boolean variables \( v_1...v_n \)
- Reach, \( S_k \) can be represented by BDDs over \( v_1...v_n \)
- \( R \) (a set of pairs of states \((s,s')\)) can be represented by a BDD over \( v_1...v_n, v_1'...v_n' \)
Example: representing a model with BDDs

\[ S = \{ s_1, s_2, s_3 \} \]
\[ R = \{ (s_1, s_2), (s_2, s_2), (s_3, s_1) \} \]

State encoding:
\[ s_1: v_1v_2=00 \quad s_2: v_1v_2=01 \quad s_3: v_1v_2=11 \]

For \( A = \{ s_1, s_2 \} \) the Boolean formula representing \( A \):
\[ f_A(v_1, v_2) = (\neg v_1 \land \neg v_2) \lor (\neg v_1 \land v_2) = \neg v_1 \]
\( f_R(v_1, v_2, v'_1, v'_2) = \)

\((\neg v_1 \land \neg v_2 \land \neg v'_1 \land v'_2) \lor \)
\((\neg v_1 \land v_2 \land \neg v'_1 \land v'_2) \lor \)
\((v_1 \land v_2 \land \neg v'_1 \land \neg v'_2) \)

\( f_A \) and \( f_R \) can be represented by BDDs.
BDD for $f(a,b,c) = (a \land b) \lor c$

Decision tree

BDD
State explosion problem (cont.)

• state of the art symbolic model checking can handle only systems with a few hundreds of Boolean variables

Other solutions for the state explosion problem are needed
SAT-based model checking

- Translates the model and the specification to a propositional formula
- Uses efficient tools for solving the satisfiability problem

Since the satisfiability problem is \textbf{NP-complete}, SAT solvers are based on heuristics.
SAT solvers

- Using heuristics, SAT tools can solve very large problems fast.
- They can handle systems with 1000 variables that create formulas with a few thousands of variables.

GRASP (Silva, Sakallah)
Prover (Stalmark)
Chaff (Malik)
MiniSat, …
Model Checking: From BDDs to Interpolation

Lecture 2

Orna Grumberg
Technion
Haifa, Israel

Summer school at Bayrischzell 2011
SAT-based model checking

• Translate the model and the specification to a propositional formula

• Use efficient tools (SAT solvers) for solving the satisfiability problem
Bounded model checking for checking AGp

- Unwind the model for k levels, i.e., construct all computation of length k
- If a state satisfying $\neg p$ is encountered, then produce a counterexample

The method is suitable for falsification, not verification
Bounded model checking with SAT

- Construct a formula $f_{M,k}$ describing all possible computations of $M$ of length $k$
- Construct a formula $f_{\varphi,k}$ expressing that $\varphi=EF\neg p$ holds within $k$ computation steps
- Check whether $f = f_{M,k} \land f_{\varphi,k}$ is satisfiable

If $f$ is satisfiable then $M \not\models AGp$
The satisfying assignment is a counterexample
Example - shift register

Shift register of 3 bits: \(<x, y, z>\)

Transition relation:
\(R(x, y, z, x', y', z') = x' = y \land y' = z \land z' = 1\)

Initial condition:
\(I(x, y, z) = x = 0 \lor y = 0 \lor z = 0\)

Specification: \(\text{AG} (x = 0 \lor y = 0 \lor z = 0)\)
Propositional formula for $k=2$

\[ f_M = (x_0=0 \lor y_0=0 \lor z_0=0) \land \\
    (x_1=y_0 \land y_1=z_0 \land z_1=1) \land \\
    (x_2=y_1 \land y_2=z_1 \land z_2=1) \]

\[ f_\varphi = \lor_{i=0,\ldots,2} (x_i=1 \land y_i=1 \land z_i=1) \]

Satisfying assignment: 101 011 111
This is a counter example!
A remark

In order to describe a computation of length $k$ by a propositional formula we need $k$ copies of the state variables. With BDDs we use only two copies of current and next states.
Bounded model checking

- Can handle **LTL** formulas, when interpreted over finite paths
- Can be used for **verification** by choosing \( k \) which is large enough so that every path of length \( k \) contains a cycle
- Using such a \( k \) is often **not practical** due to the size of the model
BDDs versus SAT

• SAT-based tools are mainly useful for **bug finding** while BDD-based tools are suitable for **full verification**

• some examples work better with BDDs and some with SAT.
Verification with SAT solvers
Interpolation-Sequence Based Model Checking [VG09]

Inspired by:
• forward reachability analysis

Combines:
• Bounded Model Checking
• Interpolation-sequence

Obtains:
• SAT-based model checking algorithm for full verification
Forward Reachability Analysis
Forward reachability analysis

• $S_j$ is the set of states reachable from some initial state in $j$ steps

• termination when
  - either a bad state satisfying $\neg q$ is found
  - or a fixpoint is reached:
    $$S_j \subseteq \bigcup_{i=0,j-1} S_i$$
Bounded Model Checking

• Does the system have a counterexample of length $k$?

\[ INIT(V_0) \land \neg q(V_0) \]
\[ INIT(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]
\[ INIT(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \neg q(V_2) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ INIT(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \ldots \land T(V_{k-1}, V_k) \land \neg q(V_k) \]
A Bit of Intuition

INIT → $S_1$ → $S_2$ → $S_3$

INIT → $I_1$ → $I_2$ → $I_3$
Interpolation

• If \( A \land B = \text{false} \), there exists an interpolant \( I \) for \((A,B)\) such that:

\[
A \Rightarrow I
I \land B = \text{false}
\]

I refers only to common variables of \(A,B\)
Interpolation (cont.)

- Example:
  \[ A = p \land q, \quad B = \neg q \land r, \quad I = q \]

- Interpolants from proofs
  
  given a resolution refutation (proof of unsatisfiability) of \( A \land B \),
  
  I can be derived in linear time.

(Pudlak,Krajicek,97)
Interpolation In The Context of Model Checking

• Given the following BMC formula $\varphi^k$

$$\begin{align*}
&\text{INIT}(V_0) \land T(V_0,V_1) \land T(V_1,V_2) \land \ldots \land T(V_{k-1},V_k) \land \neg q(V_k) \\
&\downarrow I \\
&A \Rightarrow I \\
&I \land B \equiv F
\end{align*}$$

I is over the common variables of A and B, i.e. $V_1$
Interpolation in the context of model checking

• $I$ is over $V_1$

• $A \Rightarrow I$
  
  - $I$ over-approximates the set $S_1$

• $I \land B \equiv F$

  - States in $I$ cannot reach a bug in $k-1$ steps
The same BMC formula partitioned in a different manner:

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land T(V_2, V_3) \land \ldots \land T(V_{k-1}, V_k) \land \neg q(V_k) \]

\[ I_1 \quad I_2 \quad I_3 \quad I_{k-1} \quad I_k \]

\[ I_0 = T, I_{k+1} = F \]

\[ I_{j-1} \land A_j \Rightarrow I_j \]

\( I_j \) is over the common variables of \( A_1, \ldots, A_j \) and \( A_{j+1}, \ldots, A_{k+1} \), i.e. \( V_j \)
Interpolation-Sequence (2)

- Can easily be computed. For $1 \leq j < n$
  - $A = A_1 \land \ldots \land A_j$
  - $B = A_{j+1} \land \ldots \land A_n$
  - $I_j$ is the interpolant for the pair $(A,B)$
Interpolation-Sequence Based Model Checking
Using Interpolation-Sequence

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

\[ \text{INIT} (V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \neg q(V_2) \]
Combining Interpolation-Sequence and BMC

• A way to do reachability analysis using a SAT solver.
• Uses the original BMC loop and adds an inclusion check for full verification.
• Similar sets to those computed by Forward Reachability Analysis but over-approximated.
Computing Reachable States with a SAT Solver

• Use BMC to search for bugs.
• Partition the checked BMC formula and extract the interpolation sequence

\[ INIT(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \ldots \land T(V_{N-1}, T_N) \land \neg q(V_N) \]
The Analogy to Forward Reachability Analysis

\[ \text{INIT} (V_0) \land T(V_0, V_1) \land T(Q(V_1, V_2)) \land T_q(V_2, V_3) \land \neg q(V_3) \]
Model Checking: From BDDs to Interpolation

Lecture 3

Orna Grumberg
Technion
Haifa, Israel

Summer school at Bayrischzell 2011
Verification with SAT solvers
Combining Interpolation-Sequence and BMC

- Uses BMC for bug finding
- Uses Interpolation-sequence for computing over-approximation of sets $S_j$ of reachable states
- Uses SAT solver for inclusion check for full verification
Combining Interpolation-Sequence and BMC

Always terminates

• either when BMC finds a bug:
  \[ M \not\models AGq \]

• or when all reachable states has been found:
  \[ M \models AGq \]
Interpolation-Sequence

- The same BMC formula partitioned in a different manner:

\[ \begin{align*}
\text{INIT}(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land T(V_2, V_3) \land \ldots \land T(V_{k-1}, V_k) \land \neg q(V_k) \\
I_1 \quad I_2 \quad I_3 \quad I_{k-1} \quad I_k
\end{align*} \]

\[ I_0 = T, \quad I_{k+1} = F \]

\[ I_{j-1} \land A_j \Rightarrow I_j \]

\( I_j \) is over the common variables of \( A_1, \ldots, A_j \) and \( A_{j+1}, \ldots, A_{k+1} \), i.e. \( V_j \)}
Using Interpolation-Sequence

\[ INIT(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

\[ INIT(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \neg q(V_2) \]
Checking if a “fixpoint” has been reached

- $I_j \Rightarrow V_{k=1..j-1} I_k$

- Similar to checking fixpoint in forward reachability analysis:
  $S_j \subseteq U_{k=1..j-1} S_k$

- But here we check inclusion for every $2 \leq j \leq N$
  - No monotonicity because of the approximation

- “Fixpoint” is checked with a SAT solver
The Analogy to Forward Reachability Analysis

\[ \text{INIT} (V_0) \land T(V_0, V_1) \land E(V_1, V_2) \land I_q(V_2, V_3) \land \neg q(V_3) \]
Notation:

If no counterexample of length $N$ or less exists in $M$, then:

- $I_j^k$ is the $j$-th element in the interpolation-sequence extracted from the BMC-partition of $\varphi^k$

- $I_j = \Lambda_{k=j,N} I_j^k [Vj \leftarrow V]$

- The reachability vector is: $\hat{I} = (I_1, I_2, ..., I_N)$
function UpdateReachable( $\hat{I}$, $\hat{I}^k$ )

j=1

while (j < k) do

$I_j = I_j \land I_j^k$

$\hat{I}[j] = I_j$

end while

$\hat{I}[k] = I_k^k$

end function
function FixpointReached (\hat{\mathcal{I}}) // check $I_j \Rightarrow V_{k=1,j-1} I_k$

j=2
while (j ≤ \hat{\mathcal{I}}.length) do
  R = $V_{k=1,j-1} I_k$
  $\alpha = I_j \land \neg R$ // negation of $I_j \Rightarrow R$
  if (SAT($\alpha$)==false) then return true
  end if
  j = j+1
end while
return false
end function
Function ISB(M, f)  // f = AGq
    k = 0
    result = BMC(M, f, 0)
    if (result == cex) then return cex
    \hat{I} = \phi  // the reachability vector
    while (true) do
        k = k+1
        result = BMC(M, f, k)
        if (result==cex) then return cex
        \hat{I}^k = (T, I_1^k, ..., I_k^k, F)
        UpdateReachable(\hat{I}, \hat{I}^k)
        if (FixpointReached(\hat{I}) == true) then
            return true
        end if
    end while
end function
Interpolation-Based Model Checking [McM03]
Interpolation In The Context of Model Checking

- We can check several bounds with one formula
- Given a BMC formula with possibly several bad states

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \ldots \land T(V_{k-1}, V_k) \land (\neg q(V_1) \lor \ldots \lor \neg q(V_k)) \]

\[ A \Rightarrow I \]

\[ I \land B \equiv F \]

I is over the common variables of A and B, i.e. V_1
Interpolation In The Context of Model Checking

• The interpolant represents an over-approximation of reachable states after one transition.
• Also, there is no path of length $k-1$ or less that can reach a bad state.
Using Interpolation

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

\[ I_1(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

\[ I_2(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

BAD

\(-q\)
Using Interpolation (2)

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land (\neg q(V_1) \lor \neg q(V_2)) \]

\[ I_1'(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land (\neg q(V_1) \lor \neg q(V_2)) \]

\[ \cdot \]

\[ I_k'(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land (\neg q(V_1) \lor \neg q(V_2)) \]
The Analogy to Forward Reachability Analysis

(INIT (V₀) ∧ T(V₀, V₁) ∧ T(V₁, V₂) ∧ (¬q(W₁) ∧ ¬q(W₂)))
• When calculating the interpolant for the $i$-th iteration, for bound $k$ the following holds:
  - The interpolant represents an over-approximation of reachable states after $i$ transitions.
  - Also, it cannot reach a bad state in $k-1+i$ steps or less.
    • It is similar to $I_i$ calculated in ISB after $k+i$ iterations.
Algorithm

Check the INIT states.
N = 1
Reachable = INIT
While (true)
  while ( BMC(M,f,Reachable,1,N) == false )
    I = getInterpolant();
    if ( I \Rightarrow Reachable )
      return true;
    else
      Reachable = Reachable \lor I;
      if (Reachable == INIT)
        return false;
      else
        N++;

McMillan’s Method

• The computation itself is different.
  – Uses basic interpolation.
  – Successive calls to BMC for the same bound.
  – Not incremental.

• The sets computed are different.
Experimental Results

- Experiments were conducted on two future CPU designs from Intel (two different architectures)
Experimental Results – Falsification
Experimental Results - Verification
## Experiments Results – Analysis

<table>
<thead>
<tr>
<th>Spec</th>
<th>#Vars</th>
<th>Bound (Ours)</th>
<th>Bound (M)</th>
<th>#Int (Ours)</th>
<th>#Int (M)</th>
<th>#BMC (Ours)</th>
<th>#BMC (M)</th>
<th>Time [s] (Ours)</th>
<th>Time [s] (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F₁</td>
<td>3406</td>
<td>16</td>
<td>15</td>
<td>136</td>
<td>80</td>
<td>16</td>
<td>80</td>
<td>970</td>
<td>5518</td>
</tr>
<tr>
<td>F₂</td>
<td>1753</td>
<td>9</td>
<td>8</td>
<td>45</td>
<td>40</td>
<td>9</td>
<td>40</td>
<td>91</td>
<td>388</td>
</tr>
<tr>
<td>F₃</td>
<td>1753</td>
<td>16</td>
<td>15</td>
<td>136</td>
<td>94</td>
<td>16</td>
<td>94</td>
<td>473</td>
<td>1901</td>
</tr>
<tr>
<td>F₄</td>
<td>3406</td>
<td>6</td>
<td>5</td>
<td>21</td>
<td>13</td>
<td>6</td>
<td>13</td>
<td>68</td>
<td>208</td>
</tr>
<tr>
<td>F₅</td>
<td>1761</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>F₆</td>
<td>3972</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>19</td>
<td>14</td>
</tr>
<tr>
<td>F₇</td>
<td>2197</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2544</td>
<td>1340</td>
</tr>
<tr>
<td>F₈</td>
<td>4894</td>
<td>5</td>
<td>1</td>
<td>15</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>635</td>
<td>101</td>
</tr>
</tbody>
</table>
Analysis

- False properties is always faster.
- True properties – results vary. Heavier properties favor ISB where the easier favor IB.
- Some properties cannot be verified by one method but can be verified by the other and vise-versa.
Conclusions

• A new SAT-based method for **unbounded** model checking.
  - BMC is used for falsification.
  - Simulating forward reachability analysis for verification.

• Method was successfully applied to industrial sized systems.
End of lecture 3
Model checking:

• E.M. Clarke, A. Emerson, Synthesis of Synchronization Skeletons for Branching Time Temporal Logic, workshop on Logic of programs, 1981

• J-P. Queille, J. Sifakis, Specification and Verification of Concurrent Systems in CESAR, international symposium on programming, 1982

• E.M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT press, 1999
• **BDDs:**
  R. E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, IEEE transactions on Computers, 1986

• **BDD-based model checking:**
  J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic Model Checking: $10^{20}$ States and Beyond, LICS'90

• **SAT-based Bounded model checking:**
  Symbolic model checking using SAT procedures instead of BDDs, A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, Y. Zhu, DAC'99
• **3-Valued BMC:**
  A. Yadgar, A. Flaisher, O. Grumberg, and M. Lifshits, *High Capacity (Bounded) Model Checking Using 3-Valued Abstraction*

Interpolation based model checking:

• K. McMillan, Interpolation and SAT-Based Model Checking, CAV’03

• T. Henzinger, R. Jhala, R. Majumdar, K. McMillan, Abstractions from Proofs, POPL’04

• Y. Vizel and O. Grumberg, Interpolation-Sequence Based Model Checking, FMCAD’09
Exercise 1

Write 2 CTL formulas.

1. \( f_1 \) is true in a state iff
   the state is the start of a path along which \( p \) holds at least twice

2. \( f_2 \) is true in a state iff
   the state is the start of a path along which \( p \) holds exactly twice