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:
  $\text{always (request } \Rightarrow \text{ eventually granted)}$

- communication protocols:
  $(\neg \text{get-message}) \text{ until send-message}$
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
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:

- $Gp$:Always
- $Fp$:Someday
- $Xp$:Next
- $pUq$:Until

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

\textbf{ACTL / ACTL*}

The \textit{universal} fragments of CTL/CTL* with only universal path quantifiers
CTL formulas: Example

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

• non starvation: $\text{AG}(\text{request} \implies \text{AF grant})$

• “sanity” check: $\text{EF request}$
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 $S_k \not\subset Reach$ do
    If $S_k \cap S_p \neq \emptyset$ return $(M \models \not AGq)$
    $S_{k+1} = \text{Image}(S_k)$
    $Reach = Reach \cup S_k$
    $k = k+1$
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 --- $N_1 \land N_2 \land S_0$

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

M ⊨ AG ⊢ (C₁ ∧ C₂)

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

\[ M \models AG \rightarrow (C1 \land C2) \]

\[ S_0 \]
Mutual Exclusion Example

\[ M \models AG \rightarrow (C1 \land C2) \]

\[ S_1 \]
Mutual Exclusion Example

\[ M \models AG \rightarrow (C1 \land C2) \]

\[ S_2 \]
Mutual Exclusion Example

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

\( S_3 \)
Mutual Exclusion Example

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

\[ S_4 \subseteq S_0 \cup \ldots \cup S_3 \]
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 $v_1 \ldots v_n$

• Reach, $S_k$ can be represented by BDDs over $v_1 \ldots v_n$

• $R$ (a set of pairs of states $(s, s')$) can be represented by a BDD over $v_1 \ldots v_n$ $v_1' \ldots v_n'$
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 \( f(a,b,c) = (a \land b) \lor c \)
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 NP-complete, SAT solvers are based on heuristics.
SAT solvers

- Using heuristics, SAT tools can solve very large problems fast.
- They can handle systems represented by formulas with a few millions 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 $\text{AGp}$

- 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 $\varphi=\text{EF} \neg p$ holds within $k$ computation steps
- Check whether $f = f_{M,k} \land f_{\varphi,k}$ is satisfiable

If $f$ is satisfiable then $M \models \neg \text{AG}p$

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: \( AG (x = 0 \lor y = 0 \lor z = 0) \)
Propositional formula for \( k=2 \)

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

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

**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 $k$?

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

INIT $\rightarrow$ $S_1$ $\rightarrow$ $S_2$ $\rightarrow$ $S_3$

INIT $\rightarrow$ $I_1$ $\rightarrow$ $I_2$ $\rightarrow$ $I_3$

BAD $\neg q$
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 $\varphi_k$

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

\[ A \Rightarrow I \]

\[ I \land B \equiv F \]

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

- $I$ 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:

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

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

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

\( I_j \) 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

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

\[ INIT(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \neg q(V_2) \]
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

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

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land T(V_2, V_3) \land \neg q(V_3) \]
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:
  \[ M \models AGq \]
The same BMC formula partitioned in a different manner:

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

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

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

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

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

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

- \( I_j = \bigwedge_{k=j,N} I_j^k \) \([V_j \leftarrow V] \)

- The \textit{reachability vector} is:
  \( \hat{I} = (I_1, I_2, \ldots, I_N) \)
function FixpointReached ($\hat{I}$) // check $I_j \Rightarrow V_{k=1,j-1} I_k$

j=2
while ($j \leq \hat{I}.\text{length}$) do
  $R = V_{k=1,j-1} I_k$
  $\alpha = I_j \land \neg R$ // negation of $I_j \Rightarrow R$
  if ($\text{SAT}(\alpha) == \text{false}$) then return true
  end if
  j = j+1
end while
return false
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

\[
\begin{align*}
A & \implies I \\
I \wedge B & \equiv F
\end{align*}
\]

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

• 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

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

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

\[I_2(V_0) \land T(V_0, V_1) \land \neg q(V_1)\]
Using Interpolation (2)

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

\[ \mathcal{I}_1' \]

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

\[ \ldots \]

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

\[ INIT \land T(V_0, V_{11}) \land T(V_{1}, V_2) \land (\neg q_{111}) \land (\neg q_{222}) \]
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.
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.

\[ J_1 \quad S_1 \quad I_1 \]
Experimental Results

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

![Graph showing data points on a log-log scale]

Interpolation-Sequence Based MC

Interpolation Based MC
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>F_1</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>F_2</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>F_3</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>F_4</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>F_5</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>F_6</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>F_7</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>F_8</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.
Thank You
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:**

• **BDD-based model checking:**

• **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
• **Existential abstraction + data abstraction:**

• **Localization reduction:**
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
• 3-Valued BMC:
  A. Yadgar, A. Flaisher, O. Grumberg, and M. Lifshits, High Capacity (Bounded) Model Checking Using 3-Valued Abstraction

• A. Yadgar, New Approaches to Model Checking and to 3-valued abstraction and Refinement, Ph.d. Thesis, Technion, March 2010
Model Checking: From BDDs to Interpolation

Lecture 4

Orna Grumberg
Technion
Haifa, Israel

Summer school at Bayrischzell 2011
3-Valued Abstraction in (Bounded) Model Checking for Hardware

[Yadgar, Ph.d. thesis]
Motivation

• Increase capacity of (Bounded) Model Checking
  - By abstracting out parts of the model
• “Smart” abstraction
  - Automatic or manual
• “Easy” abstraction
  - Abstract out inputs or critical nodes
• Holy Grail: Change the level of BMC
Abstraction in Model Checking
Localization reduction

Over-approximating abstraction:
Abstract model contains more behaviors

- Property is **true** on abstract model $\Rightarrow$ Property is **true** on the concrete model
- Property is **false**: counterexample might be spurious
- Refinement is needed (CEGAR)
• Finding cutpoints: computationally expensive or needs human expertise

• False negative results: overhead in checking if counterexample is spurious
3-Valued Abstraction

- Add a third value “X” (“Unknown”)
Introducing X ("Unknown")

- Property is true on abstract model $\Rightarrow$ Property is true on the concrete model

- Property is false on abstract model $\Rightarrow$ Property is false on the concrete model

- Property is X $\Rightarrow$ needs refinement
3-Valued Abstraction

- Add a third value “X”
Outline

• LTL Model Checking – Automata Approach
  – Kripke Structures and LTL
  – Büchi Automata
  – BMC
• 3-Valued Abstraction
• 3-Valued BMC (X-BMC)
Kripke Structure

- $M = (S, s_0, R, L)$ over $AP$
- $L: S \rightarrow (AP \rightarrow \{0,1\})$ \( L: S \rightarrow \{0,1\}^{AP} \)
- Can describe hardware circuits

AP = \{a, b, c\}
Büchi Automata

- $B = (\Sigma, Q, q_0, \rho, \alpha)$
  - $\rho: Q \times \Sigma \rightarrow 2^Q$
  - $w \in \Sigma^\omega$
- Accepts $w$ iff there is an accepting run for $w$
  - Such that $\alpha$ is met infinitely often

$$\Sigma = \{0, 1\}^{\{a, b, c\}}, \quad \alpha = \{q_3\}$$

- $100, 100, 010, 110, 010, 110, 010, 110, \ldots$
- $010, 010, 010, 010, \ldots$
- $001, 100, 100, 100, \ldots$
Büchi Automata

- $\rho$ can be represented as a function $F:Q \times \Sigma \times N \rightarrow Q$
  - $q' = F(q, \sigma, nd)$

- $\rho(q_2, 110) = \{q_2, q_3\}$
- $F(q_2, 110, 0) = q_2$
- $F(q_2, 110, 1) = q_3$
Büchi for LTL

- Given $\varphi = A\psi$, build an automaton $B_{\neg\psi}$ for $\neg\psi$
- $\Sigma = \{0, 1\}^{AP}$

$P = AFc$

$\alpha = \{q_0\}$

$\pi = q_0, q_0, q_0, q_0, \ldots$
Model Checking

- Let $E = M \times B$  $F = S \times \alpha$
- Reduce Model Checking to Emptiness of $E$

$AP = \{a, b, c\}$

$w = 110 \cdot 110 \cdot 110 \ldots$

$B_{\neg P}$

$c = 0$

$\alpha = \{q_0\}$

$F = \{(110, q_0), (001, q_0)\}$
Model Checking

- Fair Paths in $E$

$\alpha$

$\alpha$

$\text{SCC}$
Bounded Model Checking (BMC)

- Build a propositional representation of E
  - Describe paths of bounded length

\[ \varphi^i_M (\overline{v}_0 \ldots \overline{v}_i) = I^M_0 (\overline{v}_0) \land \bigwedge_{0 \leq j < i} R_M (\overline{v}_j, \overline{v}_{j+1}) \]

\[ \varphi^i_B (\overline{v}_0 \ldots \overline{v}_i) = I^B_0 (\overline{v}_0) \land \bigwedge_{0 \leq j < i} R_B (\overline{v}_j, \overline{v}_{j+1}) \land \text{fair}_i \]

\[ \text{fair}_i (\overline{v}_0 \ldots \overline{v}_i) = \bigvee_{0 \leq l < i} ((\overline{v}_l = v_i) \land \bigvee_{l \leq j \leq i} \alpha_E (\overline{v}_j)) \]

\[ \varphi^i_i (\overline{v}_0 \ldots \overline{v}_i) = \varphi^i_M \land \varphi^i_B \]
Check finite paths in $E$

$BMC(M, P)$

$i \leftarrow 0$

while (true) {
    if SAT ($\varphi_i$) return false
    inc($i$)
}

• Check finite paths in $E$

$BMC(M, P)$

$i \leftarrow 0$

while (true) {
    if SAT ($\varphi_i$) return false
    inc($i$)
}

$BMC (M, P)$
3-Valued logic

- Ternary domain $D = \{0, 1, X\}$
  - $X$ is “unknown” (not “don’t care”)

- Ternary operators agree with Boolean operators on Boolean values
3-Valued Abstraction

- Ternary domain $D = \{0, 1, X\}$
  - $X$ is “unknown” (not “don’t care”)

$[M \models P] = 1 \implies [M' \models P] = 1$

$[M' \models P] = 0 \implies [M \models P] = 0$
3-Valued Kripke Structure

- $M' = (S', s'_0, R', L')$ over $AP$
- $L': S' \rightarrow \{0, 1, X\}^{AP}$

$AP = \{a, b, c\}$
3-Valued LTL

- Over AP
- $P = A\psi$

$\pi \models \psi \in \{0, 1, X\}$

$[M \models P] = \begin{cases} 
1 & \forall \pi, [\pi \models \psi] = 1 \\
0 & \exists \pi, [\pi \models \psi] = 0 \\
X & otherwise
\end{cases}$
3-Valued Büchi

- $\Sigma = \{0, 1, X\}^{AP}$
- 3-Valued transition function $F'$ for $\rho$
  - $F': Q \times \Sigma \times N \rightarrow Q$
  - Ternary variables and operators

$$F'(q_3, 11X, 0) = q_1$$
3-Valued Model Checking

$E' = M' \times B'$

- A short loop is a witness for a long concrete loop
  - Lower the bound required for finding bugs
3-Valued Model Checking

\[ \alpha \in \{0, 1, X\} \]

- Checking might yield an “unknown” result.
\[ \phi^i_M(\overline{v}_0 \ldots \overline{v}_i) = I^M_0(\overline{v}_0) \land \bigwedge_{0 \leq j < i} R_M(\overline{v}_j, \overline{v}_{j+1}) \]

\[ \phi^i_B(\overline{v}_0 \ldots \overline{v}_i) = I^B_0(\overline{v}_0) \land \bigwedge_{0 \leq j < i} R_B(\overline{v}_j, \overline{v}_{j+1}) \land \text{fair}_i \]

\[ \text{fair}_i(\overline{v}_0 \ldots \overline{v}_i) = \bigvee_{0 \leq l < i} \left( (\overline{v}_l = \overline{v}_j) \land \bigvee_{l \leq j \leq i} \alpha_E(\overline{v}_j) \right) \]
X-BMC

• Create 3-Valued propositional formulae (dual rail)

\[ BMC(M',\psi) \] 

\[ i \leftarrow 0 \]

\[ \text{while (true) } \] 

\[ \text{if } SAT(\phi^i_{M'} = 1 \land \phi^i_B = 1) \text{ return } false \]

\[ \text{if } SAT(\phi^i_{M'} = 1 \land \phi^i_B = X) \text{ return } X \]

\[ \text{inc}(i) \]
Holy Grail - Revisited
## Experimental Results (EXE Cluster)

<table>
<thead>
<tr>
<th>Property</th>
<th>Result</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td># Latches</td>
<td>133K</td>
<td>132K</td>
</tr>
<tr>
<td># Gates</td>
<td>6.1M</td>
<td>6.0M</td>
</tr>
</tbody>
</table>

### XBMC

<table>
<thead>
<tr>
<th>Property</th>
<th>Result</th>
<th>EXE</th>
<th>Abs 1</th>
<th>Abs 2</th>
<th>Abs 3</th>
<th>Abs 4</th>
<th>Abs 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>fail</td>
<td>266</td>
<td>281</td>
<td>270</td>
<td>254</td>
<td>103</td>
<td>105</td>
</tr>
<tr>
<td>P2</td>
<td>pass</td>
<td>262</td>
<td>271</td>
<td>265</td>
<td>244</td>
<td>212</td>
<td>205</td>
</tr>
<tr>
<td>P3</td>
<td>fail</td>
<td>264</td>
<td>280</td>
<td>249</td>
<td>282</td>
<td>285</td>
<td>103</td>
</tr>
<tr>
<td>P4</td>
<td>pass</td>
<td>412</td>
<td>365</td>
<td>342</td>
<td>323</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>fail</td>
<td>278</td>
<td>267</td>
<td>252</td>
<td>264</td>
<td>110</td>
<td>108</td>
</tr>
<tr>
<td>P6</td>
<td>pass</td>
<td>654</td>
<td>640</td>
<td>631</td>
<td>615</td>
<td>587</td>
<td>552</td>
</tr>
</tbody>
</table>

### BMC

<table>
<thead>
<tr>
<th>Property</th>
<th>Result</th>
<th>EXE</th>
<th>Abs 1</th>
<th>Abs 2</th>
<th>Abs 3</th>
<th>Abs 4</th>
<th>Abs 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>fail</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>12280</td>
<td>525</td>
</tr>
<tr>
<td>P2</td>
<td>pass</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>479</td>
<td>411</td>
</tr>
<tr>
<td>P3</td>
<td>fail</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>408</td>
</tr>
<tr>
<td>P4</td>
<td>F/N</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>F/N</td>
</tr>
<tr>
<td>P5</td>
<td>fail</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>908</td>
</tr>
<tr>
<td>P6</td>
<td>pass</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>M/O</td>
<td>2241</td>
</tr>
</tbody>
</table>
Conclusion

• 3-Valued Abstraction
  - Models, specification and automata
  - Automatic or manual abstraction
  - Abstraction of inputs to the model

• 3-Valued Bounded Model Checking
  - Enhanced performance
  - Increased capacity
  - Reduced counterexample lengths
  - Insensitive to size of irrelevant parts of the model
  - Allows checking higher level models
    • Change in methodology (!)

• Unbounded Model Checking (Induction)

• Automatic Refinement
Thank You
Model checking:

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


- E.M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT press, 1999
• **BDDs:**

• **BDD-based model checking:**

• **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
• **Existential abstraction + data abstraction:**

• **Localization reduction:**
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
• **3-Valued BMC:**
  A. Yadgar, A. Flaisher, O. Grumberg, and M. Lifshits, High Capacity (Bounded) Model Checking Using 3-Valued Abstraction

• A. Yadgar, New Approaches to Model Checking and to 3-valued abstraction and Refinement, Ph.d. Thesis, Technion, March 2010