3-Valued Abstraction and Its Applications in Model Checking

Orna Grumberg
Technion, Israel

Summerschool at Marktoberdorf, 2009

Outline

• Introduction to Model Checking and Abstraction
  – Temporal logic and model checking
  – Abstraction
  – 3-Valued abstraction

• 3-Valued abstraction for compositional verification

• 3-Valued abstraction in (Bounded) Model Checking for hardware
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:
  always $\neg (cs_1 \land cs_2)$

- non starvation:
  always (request $\Rightarrow$ eventually granted)

- communication protocols:
  $\neg$ get-message until send-message
Model Checking [EC81,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, Cadence, ...

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

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**

- **CTL** - Allows any combination of temporal operators and path quantifiers
- **CTL** - a useful sub-logic of **CTL**

**ACTL / ACTL**

The universal fragments of **CTL/CTL** with only universal path quantifiers

---

**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 --- $N_1, N_2, S_0$

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

![Diagram of mutual exclusion example]

\[ M \models AG EF (N_1 \land N_2 \land S_0) \]

No matter where you are there is always a way to get to the initial state.

**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
Solutions to the state-explosion problem

Symbolic model checking:
The model is represented symbolically
- BDD-based model checking
- SAT-based Bounded Model Checking (BMC)
- SAT-based Unbounded Model Checking

Other solutions to the state explosion problem

Small models replace the full, concrete model:
- Abstraction
- Compositional verification
- Partial order reduction
- Symmetry
Relations between small models and concrete models

Equivalence strongly preserves CTL*

If $M_1 \equiv M_2$ then for every CTL* formula $\varphi$, $M_1 \models \varphi \iff M_2 \models \varphi$

Bisimulation equivalence

$M_1 \equiv M_2$

Both models satisfy the CTL formula:

$\text{EX} (b \land AXc) \land \text{EX} (b \land AXd)$
Relations between small models and concrete models

preorder weakly preserves $\text{ACTL}^*$

If $M_2 \geq M_1$ then for every $\text{ACTL}^*$ formula $\varphi$, $M_2 \models \varphi \Rightarrow M_1 \models \varphi$

Simulation preorder

$M_1 \preceq M_2$

\[
\begin{array}{c}
\text{ACTL formula } \varphi - AG (b \rightarrow (AXc \lor AXd)) \\
M_2 \models \varphi \Rightarrow M_1 \models \varphi
\end{array}
\]
2-valued CounterExample-Guided Abstraction Refinement (CEGAR) [CGJLV02]

Abstraction-Refinement

- **Abstraction**: removes or simplifies details that are irrelevant to the property under consideration, thus reducing # of states

- **Refinement** might be needed
Abstraction preserving $\text{ACTL/ACTL}^*$

**Existential Abstraction:**
The abstract model is an **over-approximation** of the concrete model:

- The abstract model has **more behaviors**
- But no concrete behavior is lost

- Every $\text{ACTL/ACTL}^*$ property true in the abstract model is also true in the concrete model

---

Existential Abstraction

*Given an abstraction function $h : S \rightarrow S_A$, the concrete states are grouped and mapped into abstract states:*

![Diagram showing existential abstraction](image-url)
Widely used Abstractions $(S_h, h)$

- Localization reduction: each variable either keeps its concrete behavior or is fully abstracted (has free behavior) [Kurshan94]
- Predicate abstraction: concrete states are grouped together according to the set of predicates they satisfy [8597, 8599]
- Data abstraction: the domain of each variable is abstracted into a small abstract domain [CGL94, LON94]

Logic preservation Theorem

- Theorem $M_C \leq M_A$, therefore for every $\text{ACTL}^*$ formula $\varphi$,
  $$M_A \models \varphi \Rightarrow M_C \models \varphi$$
- However, the reverse may not be valid.
**Traffic Light Example**

Property:  
\[ \varphi = \text{AG AF (state=red)} \]

Abstraction function \( h \) maps green, yellow to go.

\[ M_C \models \varphi \iff M_A \models \varphi \]

**Traffic Light Example (Cont)**

If the abstract model invalidates a specification, the actual model may still satisfy the specification.

- Property:  
  \[ \varphi = \text{AG AF (state=red)} \]

- \( M_C \models \varphi \) but \( M_A \not\models \varphi \)

- Spurious Counterexample:  
  \[ \langle \text{red, go, go, ...} \rangle \]
The CEGAR Methodology

3-Valued Abstraction for Full CTL*
Abstract Models for CTL*

- Two transition relations [LT88]
- Kripke Modal Transition System (KMTS)
  - \( M = (S, S_0, R_{\text{must}}, R_{\text{may}}, L) \)
    - \( R_{\text{must}} \): an under-approximation
    - \( R_{\text{may}} \): an over-approximation
    - \( R_{\text{must}} \subseteq R_{\text{may}} \)

Abstract Models for CTL* (cont.)

Labeling function:
- \( L: S \rightarrow 2^{\text{Literals}} \)
- \( \text{Literals} = \text{AP} \cup \{ \neg p \mid p \in \text{AP} \} \)
- At most one of \( p \) and \( \neg p \) is in \( L(s) \).
  - Concrete: exactly one of \( p \) and \( \neg p \) is in \( L(s) \).
  - KMTS: possibly none of them is in \( L(s) \).
Abstract Models for CTL* (cont.)

must and may transitions:

\[ M_A \]

\[ M_C \]

may: over approximation (∃∃)

3-Valued Semantics

- Additional truth value: \( \bot \) (indefinite)
- Abstraction preserves both truth and falsity
- (abstract) \( s_a \) represents (concrete) \( s_c \):
  - \( \varphi \) is true in \( s_a \) \( \Rightarrow \) \( \varphi \) is true in \( s_c \)
  - \( \varphi \) is false in \( s_a \) \( \Rightarrow \) \( \varphi \) is false in \( s_c \)
  - \( \varphi \) is \( \bot \) in \( s_a \) \( \Rightarrow \) the value of \( \varphi \) in \( s_c \) is unknown

[B699]
3-Valued Semantics

- Universal properties ($A_\psi$):
  - **Truth** is examined along all **may**-successors
  - **Falsity** is shown by a single **must**-successor

- Existential properties ($E_\psi$):
  - **Truth** is shown by a single **must**-successor
  - **Falsity** is examined along all **may**-successors

Compositional Verification and 3-Valued Abstraction Join Forces [S607]
We describe

- How to join forces of two popular solutions:
  - Abstraction-Refinement
  - Compositional reasoning

In order to obtain
- fully automatic
- compositional model checking
- for the full \( \mu \)-calculus

Compositional Verification

The system is composed of \( M_1 \| \ldots \| M_n \)
- "divide and conquer" approach: try to verify each component separately
- Problem: usually impossible due to dependencies
  - a component is typically designed to satisfy its requirements in specific environments (contexts)

\( \Rightarrow \) More sophisticated schemes are needed
Assume-Guarantee (AG) paradigm

Introduces assumptions representing a component's environment

\[
\begin{array}{c}
M_1 \\
\hline
A
\end{array}
\begin{array}{c}
M_2
\end{array}
\]

1. check if a component $M_1$ guarantees $\varphi$ when it is a part of a system satisfying assumption $A$.
2. discharge assumption: show that the remaining components (the environment) satisfy $A$.

Main challenge: How to construct assumptions?

---

Automatic Compositional Framework

- Previous work: based on the Assume-Guarantee (AG) paradigm and on assumption generation via learning, for universal safety properties [CGP03, AMN05, CCST05,...]

- Our approach: based on techniques from 3-valued model checking, applicable to the full mu-calculus
General Idea

- View $M_i$ as a 3-valued abstraction $M_i \uparrow$ of $M_1 \parallel \ldots \parallel M_n$ and check each $M_i \uparrow$ separately using a 3-valued semantics:
  - $\mathbf{tt}$ and $\mathbf{ff}$ are definite: hold also in $M_1 \parallel \ldots \parallel M_n$
  - $\perp$ is indefinite: value in $M_1 \parallel \ldots \parallel M_n$ is unknown

- If no $M_i \uparrow$ returned a definite result, identify the parts which are indefinite and compose only them

Composition of Models $M_1 \parallel M_2$

$M_1 = (AP_1, S_1, s_1^0, R_1, L_1), M_2 = (AP_2, S_2, s_2^0, R_2, L_2)$

- Components synchronize on the joint labelling of the states $AP_1 \cap AP_2$
\begin{itemize}
\item \( o = \text{output} \)
\item \( i = \text{input} \)
\item \( r = \text{receive} \)
\end{itemize}

**Example**

\[ M_1: \quad s_0 \xrightarrow{-r, -o} s_1 \xrightarrow{r, -o} r, o \]
\[ M_2: \quad t_0 \xrightarrow{-i, -r} t_1 \xrightarrow{i, r} -i, r, t_2 \]

\[ M_1 \parallel M_2: \]

\[ s_1 t_1 \xrightarrow{s_0 t_0} -r, -o, i \quad r, -o, -i \quad s_2 t_1 \]
\[ r, o, i \quad r, -o, -i \quad s_1 t_2 \quad s_2 t_2 \]
• This composition is suitable for describing hardware designs

• Same ideas are applicable to other synchronization models, e.g. Labeled Transition Systems (LTS) that synchronize on the joint actions and interleave the local transitions

Example

- o = output
- i = input
- r = receive

M_1: s_0 \rightarrow \neg r, \neg o \rightarrow s_1 \rightarrow r, \neg o \rightarrow r, o

M_2: t_0 \rightarrow \neg i, \neg r \rightarrow t_1 \rightarrow i, r \rightarrow \neg i, r

□(\neg i \lor \Diamond o): in all successors, input signal implies that there exists a successor producing the output signal
• o = output
• i = input
• r = receive

\[ M_1 \parallel M_2 : \]

\[ s_0 \xrightarrow{t_0} \langle r, \neg o, \neg i \rangle \]

\[ s_1 \xrightarrow{t_1} \langle r, \neg o, i \rangle \]

\[ s_2 \xrightarrow{t_1} \langle r, o, i \rangle \]

\[ s_1 \xrightarrow{t_2} \langle r, \neg o, \neg i \rangle \]

\[ s_2 \xrightarrow{t_2} \langle r, o, \neg i \rangle \]

\[ \Box(\neg i \lor \Diamond o) : \]

in all successors, input signal implies that there exists a successor producing the output signal.

---

View \( M_i \) as a \[ M_1 \uparrow \]

Transitions become (unshared) literals.

\[ M_1 \uparrow : \]

\[ s_0 \xrightarrow{t_0} \langle r, \neg o \rangle \]

\[ s_1 \rightarrow \langle r, o \rangle \]

\[ s_2 \rightarrow \langle r, o \rangle \]

\[ \bullet Value of inp. unknown \]

\[ s_i \] in \( M_1 \uparrow \) abstracts all states \((s_i, t)\) in \( M_1 \parallel M_2 \)

Model check using 3-valued MC-based

[S603, GLLS05, GLLS07]
Model Checking

**MC Graph:**

$\mathcal{M}_1 \uparrow: s_0 \vdash r, \neg o$

$\mathcal{M}_1 \uparrow: s_1 \vdash r, \neg o$

$\mathcal{M}_1 \uparrow: s_2 \vdash r, o$

$[[ \square(\neg i \vee \diamond o) ]](s_0) = ?$

**State of the model:**

**Formula that we want to evaluate in $s_0$:**

$\mathcal{M}_1 \uparrow: s_0 \vdash \square(\neg i \vee \diamond o)$

$\mathcal{M}_1 \uparrow: s_0 \vdash \neg i \vee \diamond o$

$\mathcal{M}_1 \uparrow: s_1 \vdash \neg i \vee \diamond o$

$\mathcal{M}_1 \uparrow: s_2 \vdash \neg i \vee \diamond o$

$\mathcal{M}_1 \uparrow: s_0 \vdash \neg i$

$\mathcal{M}_1 \uparrow: s_0 \vdash \diamond o$

$\mathcal{M}_1 \uparrow: s_1 \vdash \neg i$

$\mathcal{M}_1 \uparrow: s_1 \vdash \diamond o$

$\mathcal{M}_1 \uparrow: s_2 \vdash \neg i$

$\mathcal{M}_1 \uparrow: s_2 \vdash \diamond o$

$\mathcal{M}_1 \uparrow: s_1 \vdash \neg i$

$\mathcal{M}_1 \uparrow: s_1 \vdash \diamond o$

$\mathcal{M}_1 \uparrow: s_2 \vdash \neg i$

$\mathcal{M}_1 \uparrow: s_2 \vdash \diamond o$
3-Valued Model Checking Results

- $\top$ and $\bot$ are definite: hold in the concrete model as well
  - In our case: hold in $M_1 \parallel M_2$

- $\bot$ is indefinite
  - result on $M_1$ is indefinite
Similarly for $M_2 \uparrow$:

\[ t_0 \vdash \Box (\neg i \lor o) \]

- \hspace{1cm} tt
- \hspace{1cm} t_0 \vdash \neg i \lor o
- \hspace{1cm} tt
- \hspace{1cm} t_0 \vdash \neg i
- \hspace{1cm} t_1 \vdash \neg i \lor o
- \hspace{1cm} t_1 \vdash \neg i
- \hspace{1cm} t_2 \vdash \neg i \lor o
- \hspace{1cm} t_2 \vdash \neg i
- \hspace{1cm} t_2 \vdash o
- \hspace{1cm} t_2 \vdash o

3-Valued Model Checking Results

- $\bot$ on both components
  - $\Rightarrow$ Refinement is needed
  - $\Rightarrow$ consider the composition

But only the parts of the abstract models for which the model checking result is $\bot$ are identified and composed
Identify the indefinite parts:

- Construct $\bot$-subgraph, top-down
- For each $\bot$-node keep only witnessing sons:
  - $\lor$, $\diamond$: keep $tt$-sons + $\bot$-sons
  - $\land$, $\Box$: keep $ff$-sons + $\bot$-sons

Remaining sons suffice to determine result

Compose indefinite parts:

- Same subformula
- Composable states
**Product Graph:**

- $M_1 \parallel M_2$
- $(s_0, t_0) \leftarrow \Box(\neg i \lor o)$
- All transitions are real transitions
- $(s_1, t_1) \leftarrow \neg i \lor o$
- Substantially smaller than full game-graph for $M_1 \parallel M_2$
- $(s_1, t_1) \leftarrow o$
- $(s_2, t_1) \leftarrow o$

Terminal nodes are all colored $tt$ or $ff$

---

**Color Product Graph:**

- $M_1 \parallel M_2$
- $(s_0, t_0) \leftarrow \Box(\neg i \lor o)$
- All transitions are real transitions
- $(s_1, t_1) \leftarrow \neg i \lor o$
- verified
- $(s_1, t_1) \leftarrow o$
- $(s_2, t_1) \leftarrow o$

Terminal nodes are all colored $tt$ or $ff$
Compositional Model Checking

For each $i = 1,2$:
- Lift $M_i$ to $M_i^\uparrow$
- Construct model checking graph for $M_i^\uparrow$
- Apply 3-valued coloring

If both results are indefinite:
- Construct ?-subgraphs
- Compose ?-subgraphs to obtain product graph
- Color product graph

Summary

- New ingredient to compositional model checking: uses a MC graph to identify and focus on the parts of the components where their composition is necessary.
  - orthogonal to the AG approach

- Automatic compositional abstraction-refinement framework, which is incremental.

- Applicable to the full mu-calculus.
More background:
SAT-Based Bounded Model Checking (BMC)
[BCCFZ99]

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 \text{NP-complete}, SAT solvers are based on heuristics.
**SAT tools**

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

**GRASP** (Silva, Sakallah)
**Prover** (Stalmark)
**Chaff** (Malik)
**MiniSAT**

---

**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 counter example

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 $f_\varphi$, expressing $\varphi = \text{EF} \neg p$
- Check if $f = f_{M,k} \land f_\varphi$ 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: $\langle x, y, z \rangle$

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 $k=2$

$$f_M = \left( x_0 = 0 \lor y_0 = 0 \lor z_0 = 0 \right) \land$$
$$\left( x_1 = y_0 \land y_1 = z_0 \land z_1 = 1 \right) \land$$
$$\left( x_2 = y_1 \land y_2 = z_1 \land z_2 = 1 \right)$$

$$f_\varphi = \bigvee_{i=0\ldots2} \left( x_i = 1 \land y_i = 1 \land z_i = 1 \right)$$

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

---

3-Valued Abstraction in (Bounded) Model Checking for Hardware [YFGL09]
**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 ⇒
  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 ⇒ Property is true on the concrete model

• Property is false on abstract model ⇒ Property is false on the concrete model

• Property is X ⇒ needs refinement
3-Valued Abstraction

- Add a third value “X”

Outline

- Model Checking - Automata Approach
  - Kripke Structures
  - 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

**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

\[ \Sigma = \{0,1\}^{\{a,b,c\}}, \quad \alpha = \{q_3\} \]
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 \( E \)

Model Checking

- Fair Paths in \( E \)
Bounded Model Checking (BMC)

- Build a propositional representation of $E$
  - Describe paths of bounded length
    $$\varphi^i_M(\vec{v}_0...\vec{v}_i) = I^M_0(\vec{v}_0) \land_{0 \leq j \leq i} R^i_0(\vec{v}_j, \vec{v}_{j+1})$$
    $$\varphi^i_B(\vec{v}_0...\vec{v}_i) = I^B_0(\vec{v}_0) \land_{0 \leq j \leq i} R^i_B(\vec{v}_j, \vec{v}_{j+1}) \land \text{fair}$$
    $$\text{fair}(\vec{v}_0...\vec{v}_i) = \bigvee_{0 \leq j \leq i} ((\vec{v}_i = v_i) \land \bigvee_{l \leq j \leq i} \alpha^i_E(\vec{v}_j))$$
    $$\varphi_i(\vec{v}_0...\vec{v}_i) = \varphi^i_M \land \varphi^i_B$$

- Check finite paths in $E$
  $\text{BMC}(M, P)$
  $\quad i \leftarrow 0$
  $\quad \text{while(true) }$
  $\quad \quad \text{if SAT(} \varphi_i \text{) return false}$
  $\quad \quad \text{inc}(i)$
  $\quad \text{while(true) }$
3-Valued logic

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

- 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”)

\[ [M'] = P \Rightarrow [M] = 1 \]
\[ [M'] = P = 0 \Rightarrow [M] = 0 \]
3-Valued Kripke Structure

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

\[
\begin{align*}
S_0 & \xrightarrow{1X0} S_1 \\
\text{a=1} & \\
\text{b=X} & \\
\text{c=0} & 
\end{align*}
\]

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

3-Valued LTL

- Over $AP$
- $P = A\psi$

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

\[
[M'] = P = \begin{cases} 
1 & \forall \pi, [\pi] = \psi \Rightarrow 1 \\
0 & \exists \pi, [\pi] = \psi \Rightarrow 0 \\
X & \text{otherwise}
\end{cases}
\]
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

\[ \alpha(\overline{v_j}) \in \{0, 1, X\} \]

- Checking might yield an "unknown" result.

X-BMC
BMC - Reminder

\[ \varphi^i_M(\overline{v}_0...\overline{v}_i) = \bigwedge_{0 \leq j < i} R^i_M(\overline{v}_j, \overline{v}_{j+1}) \]

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

\[ \text{fair}_i(\overline{v}_0...\overline{v}_i) = \bigvee_{0 < j < i} (\overline{v}_i = \overline{v}_j) \land \bigvee_{l < j < q} \alpha_E(\overline{v}_j) \]

X-BMC

- Create 3-Valued propositional formulae (dual rail)

\[ \text{BMC}(M', \psi) \{ \]
\[ i \leftarrow 0 \]
\[ \text{while(true) \{ \]
\[ \text{if SAT}(\varphi^i_{M'} = 1 \land \varphi^i_B = 1) \text{ return false} \]
\[ \text{if SAT}(\varphi^i_{M'} = 1 \land \varphi^i_B = X) \text{ return X} \]
\[ \text{inc}(i) \]
\[ \text{\}} \]
Holy Grail - Revisited

Experimental Results (EXE Cluster)

<table>
<thead>
<tr>
<th>Property</th>
<th>Model</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>X B/C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>222</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>X</td>
<td>X</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>
<tr>
<td>B/C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>fail</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>122800</td>
<td>525</td>
<td>168</td>
</tr>
<tr>
<td>P2</td>
<td>pass</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>479</td>
<td>411</td>
<td>235</td>
</tr>
<tr>
<td>P3</td>
<td>fail</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>408</td>
</tr>
<tr>
<td>P4</td>
<td>F/N</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>F/N</td>
<td>F/N</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>fail</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>908</td>
<td>632</td>
</tr>
<tr>
<td>P6</td>
<td>pass</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>2241</td>
<td>199</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 (1)
- Unbounded Model Checking (Induction)
- Automatic Refinement

Conclusion (Final)

We introduced 3-valued abstraction and demonstrated its usefulness in two different applications:

- Compositional verification
- (Bounded) model checking for hardware

3-valued abstract models are:

- More precise
- Enable verification and falsification
- Avoid false negative results
Thank You

- **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:**  
  E. M. Clarke, O. Grumberg, D. E. Long,  
  Model Checking and Abstraction, TOPLAS, 1994.

• **Localization reduction:**  
  R. P. Kurshan, Computer-Aided Verification  
  of coordinating processes - the automata  

• **Predicate abstraction:**  
  S. Graf and H. Saidi, Construction of abstract  
  state graphs with PVS, CAV'97

  H. Saidi and N. Shankar, Abstract and Model Check  
  while you Prove, CAV'99

• **BDD-based CEGAR:**  
  Clarke, Grumberg, Jha, Lu, Veith, Counterexample-  
  Guided Abstraction Refinement, CAV2000, JACM2003
• 3-Valued Abstraction-Refinement:
  S. Shoham and O. Grumberg, A Game-Based Framework for CTL Counterexamples and Abstraction-Refinement, CAV’03

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