Model Checking, Abstractions and Reductions

Orna Grumberg
Computer Science Department
Technion
Haifa, Israel
Overview

• Temporal logic model checking
• The state explosion problem
• Reducing the model of the system by abstractions
Program verification

Given a program and a specification, does the program satisfy the specification?

Not decidable!

We restrict the problem to a decidable one:

- **Finite-state** reactive systems
- **Propositional** temporal logics
Model Checking

An efficient procedure that receives

• Description of a finite-state system (model)
• Property written as a formula of propositional temporal logic

It returns yes, if the system has the property
It returns no + counterexample, otherwise
Finite state systems

- hardware designs
- Communication protocols
- High level description of non finite state systems
Properties in temporal logic

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

• non starvation:
  \textbf{always} (request \Rightarrow \textbf{eventually} grant)

• communication protocols:
  (\neg \text{get-message}) \textbf{until} send-message
Model of a system
Kripke structure / transition system
Model of systems

\[ M = \langle S, I, R, L \rangle \]

- \( S \) - Set of states.
- \( I \subseteq S \) - Initial states.
- \( R \subseteq S \times S \) - Total transition relation.
- \( L : S \rightarrow 2^{AP} \) - Labeling function.

\( AP \) – Set of atomic propositions
\( \pi = s_0 s_1 s_2 \ldots \) is a path in M from s iff

\[ s = s_0 \text{ and } \text{for every } i \geq 0: (s_i, s_{i+1}) \in R \]
Propositional temporal logic

In Negation Normal Form

AP – a set of atomic propositions

Temporal operators:

\( G p \) \hspace{1cm} \( F p \) \hspace{1cm} \( X p \) \hspace{1cm} \( pUq \)

Path quantifiers: \( A \) for all path \( E \) there exists a path
Computation Tree Logic (CTL)

**CTL operator:**

path quantifier + temporal operator

**Literals:** $p$, $\neg p$ for $p \in AP$

**Boolean operators:** $f \land g$, $f \lor g$

**Universal formulas:** $AX f$, $A(f U g)$, $AG f$, $AF f$

**Existential formulas:** $EX f$, $E(f U g)$, $EG f$, $EF f$
Semantics for CTL

• For $p \in AP$:
  $s \models p \iff p \in L(s)$
  $s \models \neg p \iff p \notin L(s)$
  $s \models f \land g \iff s \models f$ and $s \models g$
  $s \models f \lor g \iff s \models f$ or $s \models g$
  $s \models EXf \iff \exists \pi = s_0s_1\ldots$ from $s$: $s_1 \models f$
  $s \models E(f \lor g) \iff \exists \pi = s_0s_1\ldots$ from $s$
    $\exists j \geq 0 [ s_j \models g$ and $\forall i : 0 \leq i < j [s_i \models f ] ]$
  $s \models EGf \iff \exists \pi = s_0s_1\ldots$ from $s$ $\forall i \geq 0$: $s_i \models f$
Linear Temporal logic (LTL)

Formulas are of the form $\text{A}f$, where $f$ can include any nesting of temporal operators but no path quantifiers.
CTL*
Includes LTL and CTL and more

ACTL*, ACTL (LTL)
Universal fragments of CTL*, CTL

ECTL*, ECTL
Existential fragment of CTL*, CTL
Example formulas

CTL formulas:
• mutual exclusion:  $\text{AG} \neg (\text{cs}_1 \land \text{cs}_2)$
• non starvation:  $\text{AG} \ (\text{request} \Rightarrow \text{AF grant})$
• “sanity” check:  $\text{EF request}$

LTL formulas:
• fairness:  $\text{A} (\text{GF enabled} \Rightarrow \text{GF executed})$
• $\text{A} (x=a \land y=b \Rightarrow \text{XXXXX z=a+b})$
## Property types

<table>
<thead>
<tr>
<th></th>
<th>Universal</th>
<th>Existential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety</td>
<td>$\text{AG}_p$</td>
<td>$\text{EG}_p$</td>
</tr>
<tr>
<td>Liveness</td>
<td>$\text{AF}_p$</td>
<td>$\text{EF}_p$</td>
</tr>
</tbody>
</table>
Property types (cont.)

Combination of universal safety
and existential liveness:

“along every possible execution, in every state
there is a possible continuation that will
eventually reach a reset state”

AG EF reset
Model Checking $M \models f$

[Clarke, Emerson, Sistla 83]

• The **Model Checking** algorithm works **iteratively** on subformulas of $f$, from **simpler** subformulas to more **complex** ones

• When checking subformula $g$ of $f$ we assume that all subformulas of $g$ have already been checked

• For subformula $g$, the algorithm returns the **set of states** that satisfy $g$ ($S_g$)

• The algorithm has time complexity: $O(|M| \times |f|)$
Model checking $f = \text{EF } g$

Given a model $M = < S, I, R, L >$

and $S_g$ the sets of states satisfying $g$ in $M$

procedure $\text{CheckEF} \left( S_g \right)$

$Q := \text{emptyset}; \quad Q' := S_g$

while $Q \neq Q'$ do

$Q := Q'$;

$Q' := Q \cup \{ s | \exists s' \left[ R(s,s') \land Q(s') \right] \}$

end while

$S_f := Q ; \quad \text{return}(S_f)$
Example:  \( f = EF g \)
Model checking $f = \text{EG } g$

CheckEG gets $M = < S, I, R, L >$ and $S_g$
and returns $S_f$

procedure CheckEG ($S_g$)

$Q := S$ ; $Q' := S_g$ ;
while $Q \neq Q'$ do

$Q := Q'$;
$Q' := Q \cap \{ s | \exists s' [ R(s,s') \land Q(s') ] \}$

end while

$S_f := Q$ ; return($S_f$)
Example: $f = EG \ g$
Symbolic model checking
[Burch, Clarke, McMillan, Dill 1990]

If the model is given explicitly (e.g. by adjacent matrix) then only systems with about ten Boolean variables (~1000 states) can be handled.

Symbolic model checking uses Binary Decision Diagrams (BDDs) to represent the model and sets of states. It can handle systems with hundreds of Boolean variables.
Binary decision diagrams (BDDs) [Bryant 86]

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

• Every set $A$ 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
• Assume that **states** in model M are **encoded by** $\{0,1\}^n$ and described by Boolean variables $v_1...v_n$

• $S_f$ can be represented by a BDD over $v_1...v_n$

• $R$ (a set of pairs of states $(s,s')$) can be represented by a BDD over $v_1...v_n\ v_1'...v_n'$
BDD for $f(a,b,c) = (a \land b) \lor c$

Decision tree

BDD
State explosion problem

- Hardware designs are extremely large: $> 10^6$ registers
- State of the art symbolic model checking can handle medium size designs effectively: a few hundreds of Boolean variables

Other solutions for the state explosion problem are needed!
Possible solution
Replacing the system model by a smaller one (less states and transitions) that still preserves properties of interest

• Modular verification
• Symmetry
• Abstraction
We define:

equivalence between models that strongly preserves CTL*

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

$M_1 \models \varphi \iff M_2 \models \varphi$

preorder on models that weakly preserves ACTL*

If $M_2 \succeq M_1$ then for every ACTL* formula $\varphi$,

$M_2 \models \varphi \Rightarrow M_1 \models \varphi$
The simulation preorder [Milner]

Given two models $M_1 = (S_1, I_1, R_1, L_1)$, $M_2 = (S_2, I_2, R_2, L_2)$

$H \subseteq S_1 \times S_2$ is a simulation iff for every $(s_1, s_2) \in H$:

- $s_1$ and $s_2$ satisfy the same propositions
- For every successor $t_1$ of $s_1$ there is a successor $t_2$ of $s_2$ such that $(t_1, t_2) \in H$

Notation: $s_1 \leq s_2$
The simulation preorder [Milner]

Given two models \( M_1 = (S_1, I_1, R_1, L_1) \), \( M_2 = (S_2, I_2, R_2, L_2) \)

\( H \subseteq S_1 \times S_2 \) is a simulation iff
for every \( (s_1, s_2) \in H \):

- \( \forall p \in \text{AP}: s_2 \models p \Rightarrow s_1 \models p \)
  \( s_2 \models \neg p \Rightarrow s_1 \models \neg p \)

- \( \forall t_1 [ (s_1, t_1) \in R_1 \Rightarrow \exists t_2 [ (s_2, t_2) \in R_2 \land (t_1, t_2) \in H ] ] \)

Notation: \( s_1 \leq s_2 \)
Simulation preorder (cont.)

\[ H \subseteq S_1 \times S_2 \text{ is a simulation from } M_1 \text{ to } M_2 \text{ iff } \]

\[ H \text{ is a simulation and } \]

\[ \text{for every } s_1 \in I_1 \text{ there is } s_2 \in I_2 \text{ s.t. } (s_1, s_2) \in H \]

Notation:  \( M_1 \leq M_2 \)
For models $M_1$ and $M_2$, $H \subseteq S_1 \times S_2$ is a **bisimulation** iff for every $(s_1, s_2) \in H$:

- $\forall p \in AP : p \in L(s_2) \iff p \in L(s_1)$
- $\forall t_1 [ (s_1, t_1) \in R_1 \Rightarrow \exists t_2 [ (s_2, t_2) \in R_2 \land (t_1, t_2) \in H ] ]$

- $\forall t_2 [ (s_2, t_2) \in R_2 \Rightarrow \exists t_1 [ (s_1, t_1) \in R_1 \land (t_1, t_2) \in H ] ]$

**Notation:** $s_1 \equiv s_2$
Bisimulation relation (cont.)

$H \subseteq S_1 \times S_2$ is a **Bisimulation** between $M_1$ and $M_2$ iff $H$ is a bisimulation and
for every $s_1 \in I_1$ there is $s_2 \in I_2$ s.t. $(s_1, s_2) \in H$ and
for every $s_2 \in I_2$ there is $s_1 \in I_1$ s.t. $(s_1, s_2) \in H$

**Notation:** $M_1 \equiv M_2$
Bisimulation equivalence

\[ M_1 \equiv M_2 \]

\[ H = \{ (1,1'), (2,4'), (4,2'), (3,5'), (3,6'), (5,3'), (6,3') \} \]
Simulation preorder

\[ M_1 \leq M_2 \]
$M_1 \leq M_2$
$M_1 \leq M_2$ and $M_1 \geq M_2$ but not $M_1 \equiv M_2$
Theorem:
If $M_1 \equiv M_2$ then for every $\text{CTL}^*$ formula $\varphi$, $M_1 \models \varphi \iff M_2 \models \varphi$

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

• They are one of the most useful ways to fight the state explosion problem

• They should preserve properties of interest: properties that hold for the abstract model should hold for the concrete model

• Abstractions should be constructed directly from the program
Data abstraction

Abstracts data information while still enabling to partially check properties referring to data

Data Abstraction

Given a program P with variables \( x_1, \ldots, x_n \)

\( \), each over domain \( D \),

the **concrete model** of P is defined over states

\[ (d_1, \ldots, d_n) \in D \times \ldots \times D \]

Choosing

- abstract domain \( A \)
- Abstraction mapping (surjection) \( h: D \rightarrow A \)

we get an **abstract model** over abstract states

\[ (a_1, \ldots, a_n) \in A \times \ldots \times A \]
Example

Given a program P with variable x over the integers

Abstraction 1:

\[ A_1 = \{ a_-, a_0, a_+ \} \]

\[ h_1(d) = \begin{cases} 
  a_+ & \text{if } d > 0 \\
  a_0 & \text{if } d = 0 \\
  a_- & \text{if } d < 0 
\end{cases} \]

Abstraction 2:

\[ A_2 = \{ a_{\text{even}}, a_{\text{odd}} \} \]

\[ h_2(d) = \text{if } \text{even}(|d|) \text{ then } a_{\text{even}} \text{ else } a_{\text{odd}} \]
Labeling by abstract atomic propositions

We assume that the states of the concrete model $M$ of $P$ are labeled by abstract atomic propositions of the form $(x^A = a)$ for $a \in A$.

$(x^A$ means that we refer to the abstract value of $x$)

for $s = (d_1, \ldots, d_n)$

$L(s) = \{ (x_i^A = a_i) \mid h(d_i) = a_i \}$
State equivalence

Given $M, A, h : D \rightarrow A$

$h((d_1,\ldots,d_n)) = (h(d_1),\ldots,h(d_n))$

States $s,s'$ in $S$ are **equivalent** ($s \sim s'$) iff $h(s) = h(s')$

An abstract state $(a_1,\ldots,a_n)$ **represents** the **equivalence class** of states $(d_1,\ldots,d_n)$ such that $h((d_1,\ldots,d_n)) = (a_1,\ldots,a_n)$
Reduced abstract model

Existential abstraction

Given $M, A, h : D \rightarrow A$

the reduced model $M_r = (S_r, I_r, R_r, L_r)$ is

$S_r = A \times ... \times A$

$s_r \in I_r \iff \exists s \in I : h(s) = s_r$

$(s_r, t_r) \in R_r \iff$

$\exists s, t \ [h(s) = s_r \land h(t) = t_r \land (s, t) \in R]$

For $s_r = (a_1, ..., a_n), \ L_r(s_r) = \{ (x_i^A = a_i) \mid i = 1, ..., n \}$
Existential Abstraction
Theorem:
$M_r \geq M$ by the simulation preorder

Corollary:
For every ACTL* formula $\phi$:
If $M_r \models \phi$ then $M \models \phi$
Example

Program with one variable \( x \) over the integers

Initially \( x \) may be either 0 or 1

At any step, \( x \) may non-deterministically either decrease or increase by 1
The concrete model

Abstraction 1
- $x^A = a_0$
- $x^A = a_-$
- $x^A = a_+$

Abstraction 2
- $a_{even}$
- $a_{odd}$
Representing M by first-order formulas

In order to show how to construct $M_r$ from the program text, we assume that the program is given by first order formulas $I(\mathcal{X})$ and $R(\mathcal{X}, \mathcal{X}')$ where

$\mathcal{X}=(x_1,\ldots,x_n)$ and $\mathcal{X}'=(x_1',\ldots,x_n')$
Representing $M$ by first-order formulas (cont)

$I(\mathcal{X})$ and $\mathcal{R}(\mathcal{X}, \mathcal{X}')$ describe the model $M=(S, I, R, L)$ as follows:

Let $s=(d_1, \ldots, d_n)$, $s'=(d_1', \ldots, d_n')$

$s \in I \iff I[x_i \leftarrow d_i] = \text{true}$

$(s, s') \in R \iff$

$\mathcal{R}[x_i \leftarrow d_i, x_i' \leftarrow d_i'] = \text{true}$
Representing a program by formulas: example

statement: $k$: $x := e \; k'$
Formula $\mathcal{R}$: $pc = k \land x' = e \land pc' = k'$

statement: $k$: if $x = 0$ then $k_1$: $x := 1$ else $k_2$: $x := x + 1 \; k'$
Formula $\mathcal{R}$: $(pc = k \land x = 0 \land x' = x \land pc' = k_1) \lor$
$(pc = k \land x \neq 0 \land x' = x \land pc' = k_2) \lor$
$(pc = k_1 \land x' = 1 \land pc' = k') \lor$
$(pc = k_2 \land x' = x + 1 \land pc' = k')$
Given a formula $\Phi$ over variables $x_1, ..., x_k$

$[\Phi] (x_1^A, ..., x_k^A) = \exists x_1, ..., x_k (h(x_1)=x_1^A \land ... \land h(x_k)=x_k^A \land \Phi(x_1, ..., x_k))$

Let $I(X)$ and $R(X, X')$ be the formulas describing $M$. Then $[I(X)]$ and $[R(X, X')]$ describe $M_r$.

Note: $[I(X)]$ and $[R(X, X')]$ are formulas over abstract variables
Problem:

Given $[\mathcal{I}(\mathcal{X})]$ and $[\mathcal{R}(\mathcal{X},\mathcal{X}')]$, in order to determine if $s_r \in I_r$, we need to find a state $s \in I$ (a satisfying assignment for $\mathcal{I}(\mathcal{X})$) so that $h(s) = s_r$.

Similarly, for $(s_r, t_r) \in R_r$ we look for a satisfying assignment for $\mathcal{R}(\mathcal{X},\mathcal{X}')$.

This is a difficult task due to the size and complexity of the two formulas.
Simplifying the formulas

For $\Phi$ in negation normal form over basic predicates $p_i$ and $\neg p_i$, $\mathcal{T}(\Phi)$ simplifies $[\Phi]$ by “pushing” the existential quantifiers inward:

$\mathcal{T}(p_i(x_1,\ldots,x_n)) = [p_i](x_1^A,\ldots,x_n^A)$

$\mathcal{T}(\neg p_i(x_1,\ldots,x_n)) = [\neg p_i](x_1^A,\ldots,x_n^A)$

$\mathcal{T}(\Phi_1 \land \Phi_2) = \mathcal{T}(\Phi_1) \land \mathcal{T}(\Phi_2)$

$\mathcal{T}(\Phi_1 \lor \Phi_2) = \mathcal{T}(\Phi_1) \lor \mathcal{T}(\Phi_2)$

$\mathcal{T}(\forall x \Phi) = \forall x^A \mathcal{T}(\Phi)$

$\mathcal{T}(\exists x \Phi) = \exists x^A \mathcal{T}(\Phi)$
Approximation model

**Theorem:**

\[ [\Phi] \Rightarrow \mathcal{I}(\Phi) \]

In particular, \[ [I] \Rightarrow \mathcal{I}(I) \] and \[ [R] \Rightarrow \mathcal{I}(R) \]

**Corollary:**

The approximation model \( M_a \), defined by \( \mathcal{I}(I) \) and \( \mathcal{I}(R) \) satisfies:

\[ M_a \geq M_r \geq M \] by the simulation preorder
Approximation model (cont.)

• Defined over the **same set** of abstract states as \( M_r \)

• **Easier to compute** since existential quantifiers are applied to simpler formulas

• **Less precise**: has more initial states and more transitions than \( M_r \)
Computing approximation model from the text

• **No need to construct formulas.** The approximation model can be constructed **directly from the program text**

• **The user should provide abstract predicates** \([p_i]\) and \([\neg p_i]\) for every basic action (assignment or condition) in the program
Abstract predicates provided by the user: Example

statement: \( x := y + z \)

predicate \( p(x',y,z): x' = y + z \)

\( A = \{ a_{\text{even}}, a_{\text{odd}} \} \)

\[
[p](x'^A, y^A, z^A) = \{ (a_{\text{even}}, a_{\text{odd}}, a_{\text{odd}}),
(a_{\text{even}}, a_{\text{even}}, a_{\text{even}}), (a_{\text{odd}}, a_{\text{odd}}, a_{\text{even}}), (a_{\text{odd}}, a_{\text{even}}, a_{\text{odd}}) \}
\]

\[
[p](a_{\text{even}}, a_{\text{odd}}, a_{\text{odd}}) \text{ iff } \exists x', y, z ( h(x') = a_{\text{even}} \land h(y) = a_{\text{odd}} \land h(z) = a_{\text{odd}} \land x' = y + z )
\]
Useful abstractions
Modulo an integer m

Abstraction: $h(i) = i \mod m$

Properties of modulo:

$((i \mod m) + (j \mod m)) \mod m = i + j \mod m$

$((i \mod m) - (j \mod m)) \mod m = i - j \mod m$

$((i \mod m) \times (j \mod m)) \mod m = i \times j \mod m$

Specification:

$AG(\text{waiting} \land \text{req} \land (\text{in}_1 \mod m = i) \land (\text{in}_2 \mod m = j) \rightarrow A(\neg \text{ack} \lor (\text{ack} \land (\text{overflow} \lor (\text{output} \mod m = i + j \mod m))))))$
Useful abstractions
logarithm

Abstraction: $h(i) = \lceil \log_2(i+1) \rceil$
(smallest number of bits to represent $i>0$)

Specification:
AG ( waiting $\land$ req $\land$ ( $h(\text{in}_1) + h(\text{in}_2) \leq 16$))
$\rightarrow$A ($\neg$ack U ($\neg$ack $\land$ $\neg$overflow)))

AG ( waiting $\land$ req $\land$ ( $h(\text{in}_1) + h(\text{in}_2) \geq 18$))
$\rightarrow$A ($\neg$ack U (ack $\land$ overflow)))
Counterexample-guided refinement

Goal:

• To produce abstraction **automatically**
• To use **counter example** in order to refine the abstraction

Traffic Light Example

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

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

\[ M \models \varphi \iff M_h \models \varphi \]
If the abstract model invalidates a specification, the actual model may still satisfy the specification.

- **Property:**
  \[ \varphi = AG \ AF \ (state=red) \]

- \( M \models \varphi \) but \( M_h \not\models \varphi \)

- **Spurious Counterexample:**
  \( \langle red, go, go, \ldots \rangle \)
Our Abstraction Methodology

- **Generate initial abstraction:**
  - \( M \) and \( \varphi \)
  - Model check
    - \( M_h \sim\varphi \)
  - Generate counterexample \( T_h \)
    - Check spurious counterexample
      - \( T_h \) is spurious
      - Is spurious
      - Refinement
      - Stop
      - \( M_h \) \( \not\sim\varphi \)

- **Is not spurious:**
  - Stop
Generating the Initial Abstraction

Basic Idea

- Extract atomic formulas from control flow
- Group formulas into formula clusters
- Generate abstraction for each cluster

Intuition: We consider the correlation between variables only when they appear in control flow.
Formula Cluster Example

init(x) := 0
next(x) := case
    reset=TRUE : 0;
    x < y : x + 1;
    x = y : 0;
else : x;
esac;

init(y) := 1;
next(y) := case
    reset=TRUE : 0;
    x=y ∧ ¬y=2 : y + 1;
    x = y : 0;
else : y;
esac;

FC₁ = \{ x < y , x = y , y=2 \}, \quad FC₂ = \{ \text{reset}=\text{TRUE} \}

VC₁ = \{ x , y \}, \quad VC₂ = \{ \text{reset} \}
Assume $x, y \in \{0, 1, 2\}$
\[\text{reset} \in \{\text{true, false}\}\]

Formulas in $\mathbf{F C}_1$ cannot distinguish
\[\{x=0, y=0\} \text{ and } \{x=1, y=1\},\]
therefore, \[\{x=0, y=0\} \text{ and } \{x=1, y=1\}\]
have the same effect on the control flow

Initial abstraction:
\[h(0,0) = h(1,1) = \alpha\]
Valuations \( \{0,1,2\} \times \{0,1,2\} \) of \((x,y)\) are partitioned into five equivalence classes:

\[
h_1(0,0) = h_1(1,1) = \alpha \\
h_1(0,1) = \beta \\
h_1(0,2) = h_1(1,2) = \gamma \\
h_1(1,0) = h_1(2,0) = h_1(2,0) = \delta \\
h_1(2,2) = \varepsilon
\]

Valuations \{true, false\} of \(\text{reset}\) have two equivalence classes:

\[
h_2(\text{true}) = \text{true} \quad h_2(\text{false}) = \text{false}
\]
Programs and specifications

$\text{atoms}(P)$ is the set of \textbf{conditions} in the program $P$ and \textbf{atomic formulas} in the specification $\varphi$. $\text{atoms}(P)$ are defined over program variables.

Example: $x+3<y$

$\varphi$ is an $\text{ACTL}^*$ formula over $\text{atoms}(P)$

A state $s$ in the model of $P$ is \textbf{labeled} with $f \in \text{atoms}(P)$ iff $s \models f$
Initial abstraction

Let \{FC_1, \ldots, FC_m\} be a set of formula clusters
Let \{VC_1, \ldots, VC_m\} be a set of variable clusters
The initial abstraction \(h=(h_1, \ldots, h_m)\) is defined by

\[ h_i(d_1 \ldots d_k) = h_i(e_1 \ldots e_k) \]
iff for all \(f \in FC_i\),
\(d_1 \ldots d_k \models f \iff e_1 \ldots e_k \models f\)
Model Check The Abstract Model

Given a generated abstraction function $h$,

- $M_h$ is built by using existential abstraction
- If not ($M_h \models \varphi$), then the model checker generates a counterexample trace ($T_h$)
- Current model checkers generate paths or loops.
- Question: is $T_h$ spurious?
Path Counterexample

Assume that we have four abstract states

\{1,2,3\} \leftrightarrow \alpha \quad \{4,5,6\} \leftrightarrow \beta
\{7,8,9\} \leftrightarrow \gamma \quad \{10,11,12\} \leftrightarrow \delta

Abstract counterexample \( T_h = \langle \alpha, \beta, \gamma, \delta \rangle \)

\( T_h \) is not spurious, therefore, \( M \nvdash \varphi \)
Spurious Path Counterexample

The concrete states mapped to the failure state are partitioned into 3 sets.

<table>
<thead>
<tr>
<th>states</th>
<th>dead-end</th>
<th>bad</th>
<th>irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>reachable</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>out edges</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
Refining The Abstraction

- **Goal**: refine $h$ so that the dead-end states and bad states do not belong to the same abstract state.

- For this example, two possible solutions.
General Refinement Problem

• The **optimal** refinement is hard to find

• **Coarser** refinements are **safer**
  • the refined abstract machine is still small

• **Theorem:** Finding the coarsest refinement is NP-hard.

• **Heuristic:** Treat all the irrelevant states as bad states
  • in practice, this works very well
Loop Counterexample

length of loop = 4
Loop Counterexample (cont)

Important observations

• The size of a concrete loop may be different from the abstract loop

• An abstract loop may correspond to several concrete loops

• Naïve unwinding may be exponential
Spurious Loop Counterexample

Restrict original model $M$ to $S_1 \cup S_2$, i.e., $K = M \downarrow (S_1 \cup S_2)$, then

There is a loop counterexample if and only if

$K \models EG \text{ TRUE}$
Spurious Loop Counterexample

• If an abstract loop counterexample is spurious, loop unwinding will reach empty set

• Let $T_{\text{unwind}}$ be the unwound loop by $|S_1|$ times.

• Theorem: The loop counterexample is spurious iff $T_{\text{unwind}}$ is spurious.

Use refinement algorithm for path counterexample!
Completeness

- Our methodology refines the abstraction until either the property is proved or counterexamples are found.

- **Theorem:** Given a model $M$ and an ACTL* specification $\phi$ whose counterexample is either path or loop, our algorithm will find a model $M_a$ such that

  $$M_a \models \phi \iff M \models \phi$$
Experiment : Fujitsu Design

The multimedia processor is very complicated

- Description includes 61,500 lines of Verilog code
- Manual abstraction by Fujitsu engineers reduces the code to 10,600 lines with 500 registers
- We translated this abstracted code into 9,500 lines of SMV code
Experiment (cont.)

We tried to verify this using state-of-art model checkers

• NuSMV+COI cannot verify the design

• Bwolen Yang’s SMV cannot verify the design

• Our approach abstracted 144 symbolic variables, used 3 refinement steps, and found a bug
Abstract Interpretation

We show how abstractions preserving temporal logics can be defined within the framework of abstract interpretation

D. Dams, R. Gerth, O. Grumberg,
Abstract interpretation of reactive systems,
Abstract interpretation (cont.)

We define abstractions that preserve:

– Existential properties (ECTL*)
– Universal properties (ACTL*)
– Both (CTL*)

We define:

• **Canonical abstraction** that preserves maximum number of temporal properties
• **Approximations**
Abstract interpretation (cont.)

Using abstract interpretation we can obtain abstract models which are more precise (and therefore preserve more properties) than the existential abstraction presented before.
The Abstract Interpretation Framework

• Developed by Cousot & Cousot for compiler optimization
• Constructs an abstract model directly from the program text
• Classical abstract interpretations preserve properties of states. Here we are interested in properties of computations
Model

\[ M = ( S, I, R, L ) \] where \( S, I, R \) – as before

\[ \text{Lit} = \text{AP} \cup \{ \neg p \mid p \in \text{AP} \} \]

\[ L : S \rightarrow 2^{\text{Lit}} \] - labeling function so that

\[ p \in L(s) \Rightarrow \neg p \not\in L(s) \] and

\[ \neg p \in L(s) \Rightarrow p \not\in L(s) \]

But not required: \( p \in L(s) \iff \neg p \not\in L(s) \)
Galois connection

( \( \alpha: C \to A, \gamma: A \to C \)) is a **Galois connection** from \((C, \leq)\) to \((A, \leq)\) iff

- \(\alpha\) and \(\gamma\) are total and monotonic
- for all \(c \in C\), \(\gamma(\alpha(c)) \geq c\)
- for all \(a \in A\), \(\alpha(\gamma(a)) \leq a\)

If \(\leq\) on \(A\) is defined by: \(a \leq a' \iff \gamma(a) \leq \gamma(a')\)

then for all \(a\), \(\alpha(\gamma(a)) = a\) and 
\((\alpha, \gamma)\) is a **Galois insertion**
For the partially ordered sets $(C, \leq)$ and $(A, \leq)$: the concrete and abstract domains

$a \leq a'$ - $a$ is more precise than $a'$
   $a'$ approximates $a$

c $\leq$ $c'$ - $c$ is more precise than $c'$
   $c'$ approximates $c$

$\alpha$: $C \rightarrow A$ maps each $c$ to its most precise (least) abstraction

$\gamma$: $A \rightarrow C$ maps each $a$ to the most general (greatest) $c$ that is abstracted by $a$
Our abstract Interpretation

For model M with state set $S$

• Choose an abstract domain $S_A$
  – $S_A$ must contain the top element $T$

• Define:

  abstraction mapping
  $\alpha : 2^S \rightarrow S_A$

  concretization mapping
  $\gamma : S_A \rightarrow 2^S$

We use Galois insertion
Remarks
For every set of concrete states $C \subseteq S$, $\gamma(\alpha(C)) \supseteq C$. Therefore, for every $C$ there is an abstract state $a$ with $\gamma(a) \supseteq C$. In particular, $S_A$ must contain a “top” state $T$ so that $\gamma(T) = S$.

Not necessarily, for every set $C$ there is a different abstract state $a$.

For example: $S_A = \{ T \}$ with $\gamma(T) = S$ and for every $C \subseteq S$, $\alpha(C) = T$ is a correct abstraction (even though meaningless)
Example

Abstract states:

\[ A = \{ \text{grt}_5, \text{leq}_5, T \} \]

\[ \gamma(\text{grt}_5) = \{ s \in S \mid s(x) > 5 \} \]

\[ \gamma(\text{leq}_5) = \{ s \in S \mid s(x) \leq 5 \} \]

The set \( \{ s \in S \mid s(x) > 6 \} \) could be mapped to either \( \text{grt}_5 \) or \( T \), but \( \text{grt}_5 \) is more precise, and therefore a better choice

\[ \{ s \in S \mid s(x) > 0 \} \] must be mapped to \( T \)
Relation transformers

Given sets A and B and a relation $R \subseteq A \times B$, the relations $R^\exists\exists$, $R^\forall\exists \subseteq 2^A \times 2^B$ are defined:

$$R^\exists\exists = \{(X,Y) \mid \exists x \in X \exists y \in Y \ R(x,y)\}$$

$$R^\forall\exists = \{(X,Y) \mid \forall x \in X \exists y \in Y \ R(x,y) \}$$
If R is a transition relation

\( R^\exists(X,Y) \) iff there exists some state in X that makes a transition to some state in Y

\( R^\forall \exists(X,Y) \) iff every state in X makes a transition to some state in Y
Goal

Given a set of abstract states $S_A$, to construct the most precise model $M_A = (S_A, I_A, R_A, L_A)$ such that for every $CTL^*$ formula $\varphi$ and abstract state $a \in S_A$,

$$M_A, a \models \varphi \Rightarrow M, \gamma(a) \models \varphi$$
For \( p \in \text{Lit} \):

\[ p \in L_A(a) \iff \forall s \in \gamma(a): p \in L(s) \]

**Note:** it is possible that \( p \not\in L_A(a) \) and \( \neg p \not\in L_A(a) \)

The definition guarantees for every \( p \in \text{Lit} \):

\[ a \models p \Rightarrow \gamma(a) \models p \]
\[ I_A = \{ \alpha(s) | s \in I \} \]

(\(\alpha(s)\) means \(\alpha(\{s\})\))

Guarantees that \(M_A |= \phi \Rightarrow M |= \phi\)

**Explanation:**
\(M_A |= \phi \Rightarrow \forall a \in I_A : M_A, a |= \phi \Rightarrow \forall a \in I_A : M, \gamma(a) |= \phi \Rightarrow \forall s \in I : M, s |= \phi \Rightarrow M |= \phi\)
More on $I_A$

An alternative definition: $I_A = \alpha(I)$ is less precise.

Example:

$M \models A(\neg p \lor AX q )$ but not $(M_A \models A(\neg p \lor AX q ))$
We define two abstract transition relations:

$\mathcal{R}_A$ preserves $\mathsf{ACTL}^*$

$\mathcal{R}_E$ preserves $\mathsf{ECTL}^*$

Putting them together in the same model will preserve full $\mathsf{CTL}^*$
In order to preserve ACTL* we may add more transitions, but never lose one.

Possible definition:

\( R^A (a, b) \iff R^\exists \exists (\gamma(a), \gamma(b)) \)
A more precise definition:

adds less transitions to more precise abstract states

\[ R^A(a, b) \iff \exists Y \subseteq S \ [ \alpha(Y)=b \land Y \text{ is a minimal set that satisfies } R^{\exists \exists} (\gamma(a), Y) ] \]

**Note:** \( Y \) is always a singleton
\( R^A \) (cont.)

\[ \alpha(s_1) = \alpha(s_2) = a_{12} \quad \alpha(s_3) = \alpha(s_5) = a_{35} \quad \alpha(s_4) = a_4 \]

\[ a_{12} |\!|= AX \ (p \lor q) \]
In order to preserve ECTL* we may eliminate some transitions, but never add non-real ones.

Possible definition:
\[ R^E (a, b) \iff R^{\forall \exists} (\gamma(a), \gamma(b)) \]
R^E (cont.)

A more precise definition:

keeps more transitions to

more precise abstract states

R^E(a, b) ⇔

∃ Y ⊆ S [α(Y) = b ∧

[Y is a minimal set that satisfies R^∀∃ (γ(a), Y)]
**$R^A$ and $R^E$**

- Because of minimality, not necessarily $R^E \subseteq R^A$

- Minimality is not necessary for correctness of abstraction. We will later give it up in order to compute abstract models more easily.
**Mixed model**

\[ M_A = (S_A, I_A, R^A, R^E, L_A) \]

**A-path** is a path over \( R^A \)-transitions

**E-path** is a path over \( R^E \)-transitions

\[ M_A, a \models AX f \iff \forall b \ [ (a,b) \in R^A \rightarrow M_A, b \models f ] \]

\[ M_A, a \models EX f \iff \exists b \ [ (a,b) \in R^E \land M_A, b \models f ] \]
Model checking on mixed models

CTL model checking works iteratively, from simpler subformulas to more complex ones. Each subformula will be checked on either $R^A$ or $R^E$, according to the main operator of the formula.
We have constructed $M_A$, which given $S_A$, is the best model satisfying for every $\varphi$ in CTL*

$M_A \models \varphi \implies M \models \varphi$

If not ($M_A \models \varphi$) then we can check whether $M_A \models \neg \varphi$.

If neither holds then $S_A$ is too coarse to give the answer.
Approximations

As in other abstractions:

• We would like to construct the abstraction **directly** from the **program text**
• Best abstraction is too **difficult** to compute
• We therefore construct **approximation** to the abstraction
Mixed simulation

$H_A \subseteq S_A \times S_A$ is defined over mixed abstract models, each with state set $S_A$

Mixed simulation is similar to simulation, except that the condition on $(s_1, s_2) \in H$ saying that $s_2$ has “more” successors than $s_1$ is replaced for $(a_1, a_2) \in H_A$ by

- $a_2$ has “more” A-successors than $a_1$
- $a_2$ has “less” E-successors than $a_1$
\( a_2 \geq a_1 \) by the mixed simulation

\( a_2 \models AXp \Rightarrow a_1 \models AXp \)

\( a_2 \models Exq \Rightarrow a_1 \models EXq \)
Theorem:
If $A'$ and $A''$ are mixed models and $A'' \geq A'$ by the mixed simulation
then for every CTL* formula $\varphi$
$A'' \models \varphi \ \Rightarrow \ A' \models \varphi$

Corollary:
If $A \geq M_A$ by the mixed simulation
then $A \models \varphi \ \Rightarrow \ M \models \varphi$
Computing abstraction from the program text

Assume a program that repeatedly computes a set of transitions:

\[ \{ c_i(x) \rightarrow t_i(x,x') \mid i \in J \} \].

Being in state \( s \), it chooses nondeterministically a transition \( i \) for which \( c_i(s) \) is true.

The transition results in state \( s' \) for which \( t_i(s, s') \) is true.
c_i^A(a) \iff \exists s \in \gamma(a): \ c_i(s)

t_i^A(a, b) \iff \exists Y \subseteq S \ [\alpha(Y) = b \land

\text{Y is a \textbf{minimal} set that satisfies } t_i^{\exists \exists} (\gamma(a), Y)]

c_i^E(a) \iff \forall s \in \gamma(a): \ c_i(s)

t_i^E(a, b) \iff \exists Y \subseteq S \ [\alpha(Y) = b \land

\text{Y is a \textbf{minimal} set that satisfies } t_i^{\forall \exists} (\gamma(a), Y)]
Approximation for $R^A$ and $R^E$

$R'^A = \{ (a,b) \mid \exists i \in J: c_i^A(a) \land t_i^A(a, b) \}$

$R'^E = \{ (a,b) \mid \exists i \in J: c_i^E(a) \land t_i^E(a, b) \}$

Example

Program: $\{ x=4 \rightarrow x' := x/4 \}$

$S_A = \{ \text{even}, \text{odd}, \top \}$

$R^A = \{ (\text{even}, \text{odd}), (\top, \text{odd}) \}$

$R'^A = \{ (\text{even}, \text{odd}), (\top, \text{odd}), (\text{even}, \text{even}) \}$
Example

Program: { even(x) → x' := x/2
          even(x) → x' := x+1  }

$S_A = \{ \text{even, odd, T } \}$

$R^E = \{ (\text{even, odd}) \}$

$R'E = \{ (\text{even, odd}), (\text{even, T}) \}$

Lemma

• $R^A \subseteq R'E$

• For all $a, b \in S_A$ [ $R^E(a,b) \Rightarrow \exists b'' \leq b \ [ R^E(a, b'') ]$]
Further approximation

Give up minimality in the definition of $t_i^A$ and $t_i^E$:

Replace transition $(a, b)$ by transition $(a, b')$ with $b \leq b'$ by the mixed simulation.

- Easier to compute from the program text.
- Still preserves (possibly less) CTL* formulas.
State-of-the-art Abstraction

- **Abstract interpretation**
  (Cousot & Cousot 77, Loiseaux & Graf & Sifakis & Bouajjani & Bensalem 95, Graf 94)

- **(Bi)-simulation reduction**
  (Bouajjani & Fernandez & Halbwachs 90, Lee & Yannakakis 92, Fisler & Vardi 98, Bustan & Grumberg 00)

- **Formula-dependent equivalence**
  (Aziz & Singhal & Shiple & Sangiovanni-Vincentelli 94)

- **Compositional minimization**
  (Aziz & Singhal & Swamy & Brayton 94)
State-of-the-art Abstraction (Cont)

- **Uninterpreted functions**
  
  (Burch & Dill 94, Berzin & Biere & Clarke & Zhu 98, Bryant & German & Velve 99)

- **Abstraction and refinement**
  
  (Dams & Gerth & Grumberg 93, Kurshan 94, Balarin & Sangiovanni-Vincentelli 93, Lind-Nielsen & Andersen 99)

- **Predicate abstraction and Theorem proving**
  
  (Das & Dill & Park 99, Graf & Saidi 97, Uribe 99)
The End