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

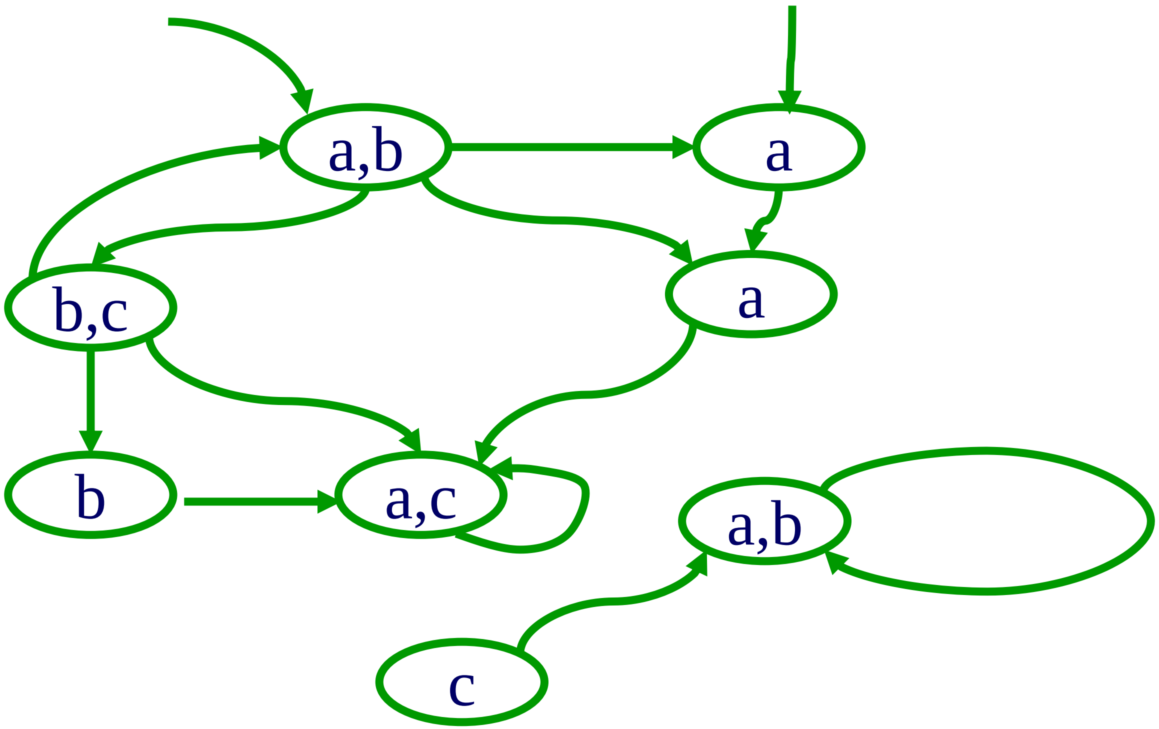
- hardware designs
- Communication protocols
- High level description of non finite state systems

Properties in temporal logic

- **mutual exclusion:**
  always $\neg(cs_1 \wedge cs_2)$
- **non starvation:**
  always (request $\Rightarrow$ **eventually** grant)
- **communication protocols:**
  ($\neg$ get-message) **until** send-message

Model of a system

Kripke structure / transition system



Model of systems

M=<S, I, R, L>

- **S** - Set of states.
- **I** $\subseteq$ S - Initial states.
- **R** $\subseteq$ S x 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

$s = s_0$ and 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

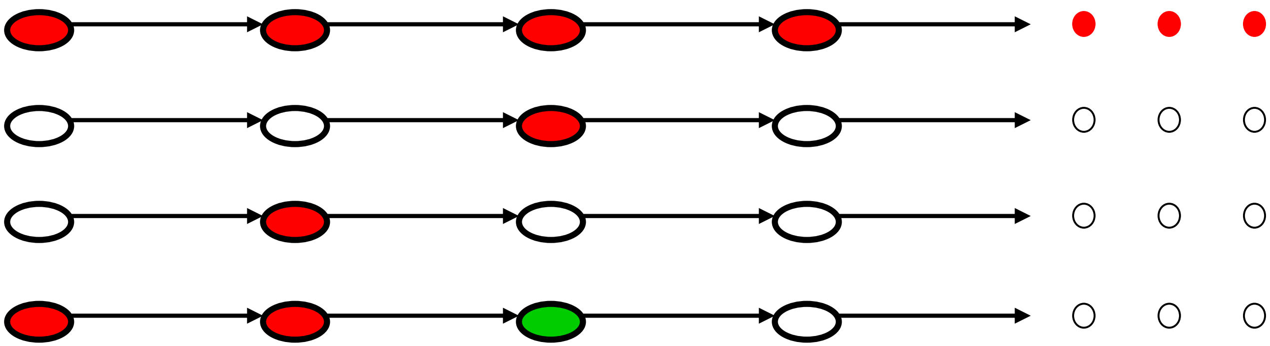
Temporal operators:

Gp

Fp

Xp

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: $f \wedge g$, $f \vee g$

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 $p \in AP$:**

  $s \models p \Leftrightarrow p \in L(s)$ $\quad$ $s \models \neg p \Leftrightarrow p \notin L(s)$
- $s \models f \wedge g \Leftrightarrow s \models f$ **and** $s \models g$
- $s \models f \vee g \Leftrightarrow s \models f$ **or** $s \models g$
- $s \models EXf \Leftrightarrow \exists \pi = s_0 s_1 \ldots$ **from** s: $s_1 \models f$
- $s \models E(f\, Ug) \Leftrightarrow \exists \pi = s_0 s_1 \ldots$ **from** s

  $\exists j \geq 0\ [\ s_j \models g$ **and** $\forall i : 0 \leq i < j\ [s_i \models f\ ]\ ]$
- $s \models EGf \Leftrightarrow \exists \pi = s_0 s_1 \ldots$ **from** $s\ \forall i \geq 0$: $s_i \models f$

Linear Temporal logic (LTL)

Formulas are of the form **Af** ,

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:** $\mathbf{AG}\ \neg(cs_1 \wedge cs_2)$
- **non starvation:** $\mathbf{AG}\ (\text{request} \Rightarrow \mathbf{AF}\ \text{grant})$
- **"sanity" check:** $\mathbf{EF}\ \text{request}$

LTL formulas:

- **fairness:** $\mathbf{A}(\mathbf{GF}\ \text{enabled} \Rightarrow \mathbf{GF}\ \text{executed})$
- $\mathbf{A}(x{=}a \wedge y{=}b \Rightarrow \mathbf{XXXX}\ z{=}a{+}b)$

Property types

	Universal	Existential
Safety	**AG**p	**EG**p
Liveness	**AF**p	**EF**p

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| × |f|)**

Model checking $\mathbf{f = EF\ g}$

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

and $\mathbf{S_g}$ the sets of states satisfying **g** in M

procedure CheckEF (S_g)

Q := emptyset; Q' := S_g ;

while Q ≠ Q' do

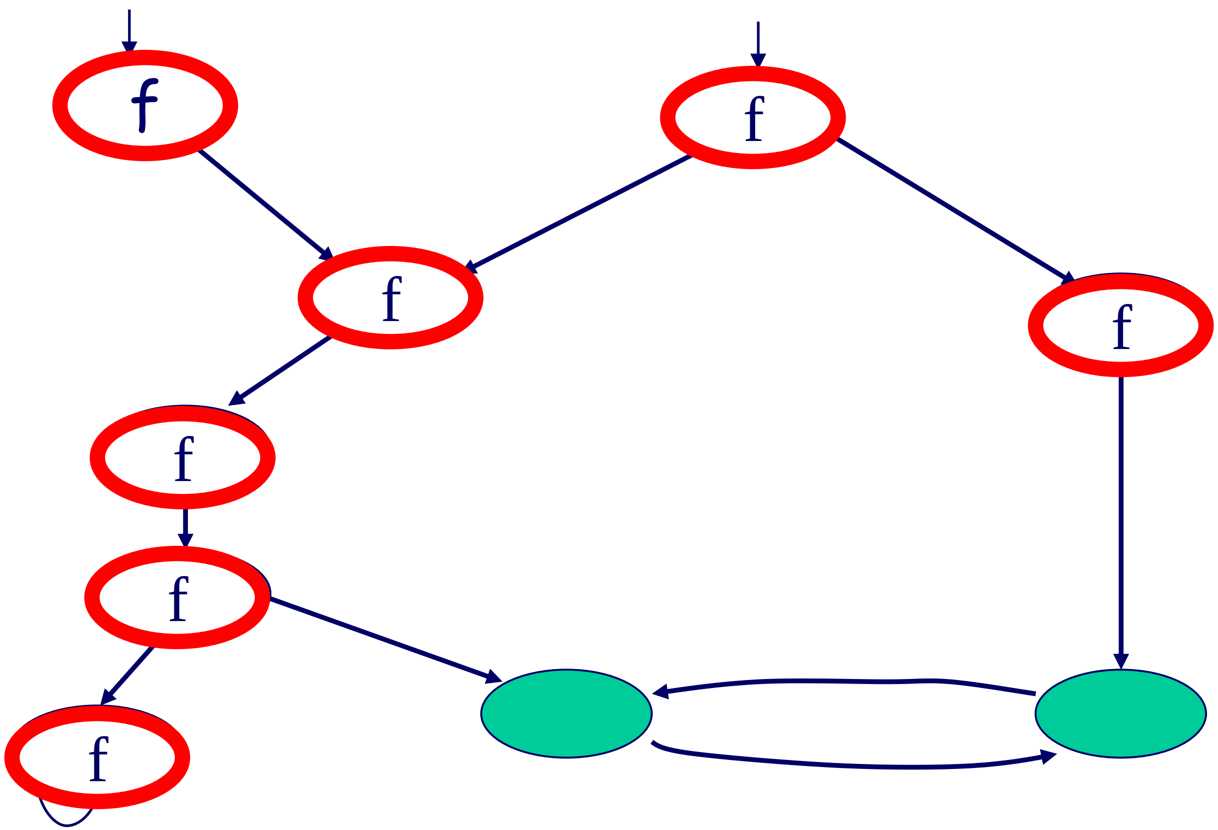
Q := Q';

Q' := Q ∪ { s | ∃s' [R(s,s') ∧ Q(s')] }

end while

S_f := Q ; return(S_f)

Example: **f = EF g**



Model checking $\mathbf{f = EG\ g}$

CheckEG gets $\mathbf{M = < S, I, R, L >}$ and $\mathbf{S_g}$

and returns $\mathbf{S_f}$

procedure CheckEG $(\mathbf{S_g})$

$\mathbf{Q := S\ ;\ \ Q' := S_g\ ;}$

while $\mathbf{Q \neq Q'}$ **do**

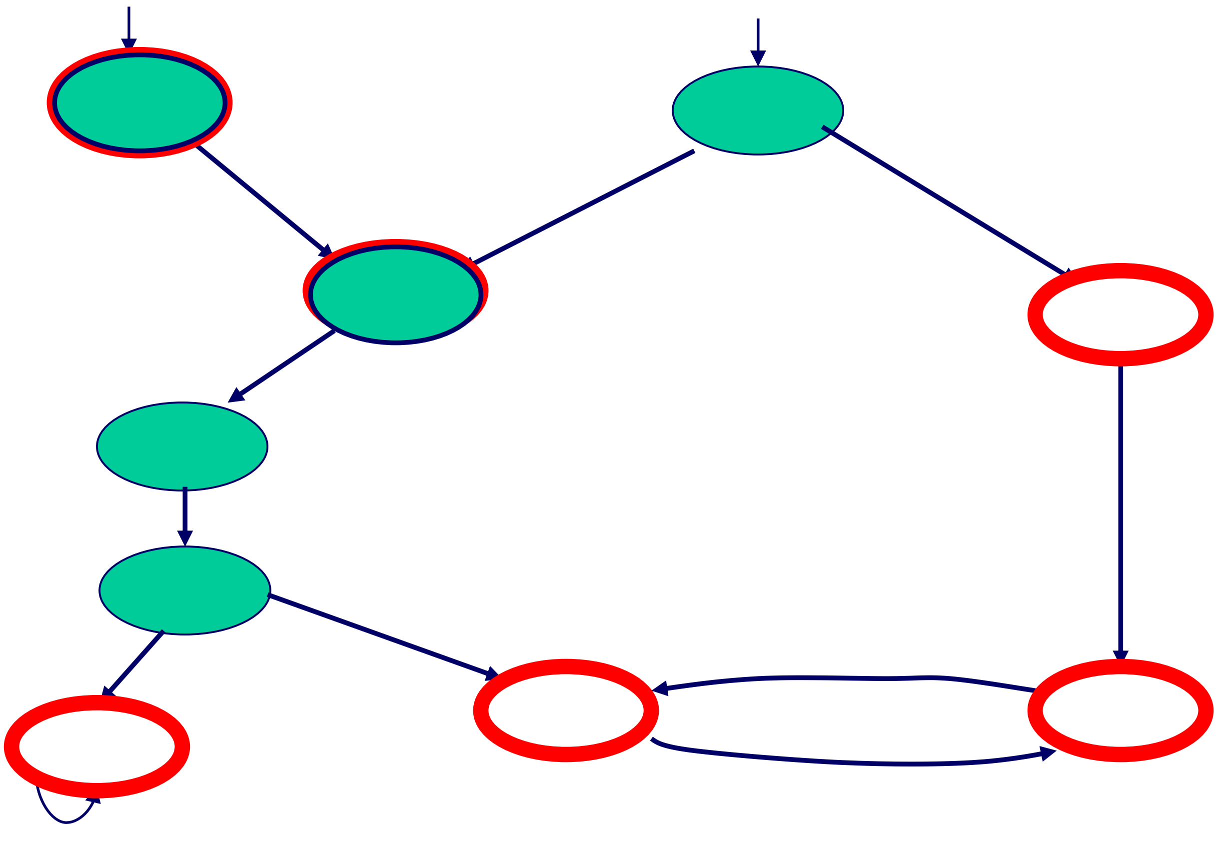
$\quad \mathbf{Q := Q';}$

$\quad \mathbf{Q' := Q \cap \{ s \mid \exists s' [R(s,s') \wedge Q(s')] \}}$

end while

$\mathbf{S_f := Q\ ;}$ **return**$(\mathbf{S_f})$

Example: $f = \mathbf{EG}\ g$



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**

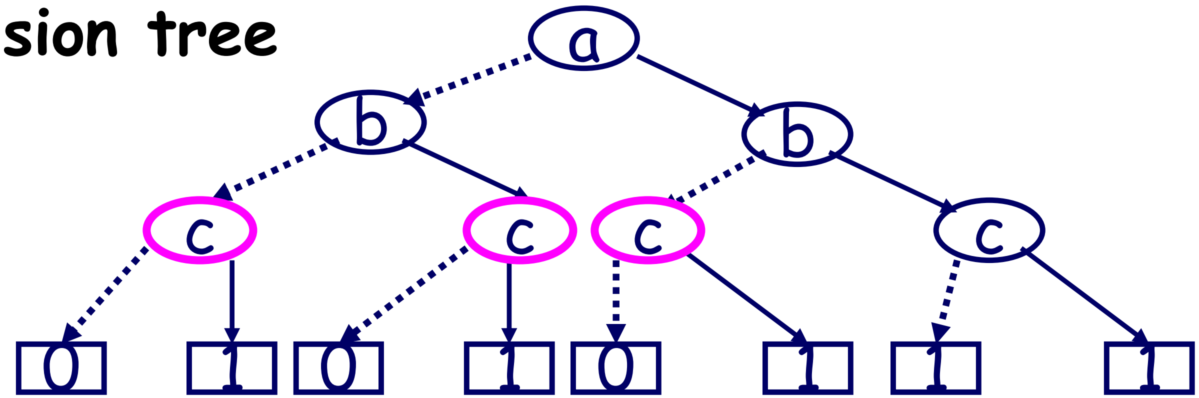
$$\mathbf{f_A(u)} = \begin{cases} \mathbf{1} & \text{if } u \in A \\ \mathbf{0} & \text{if } u \notin A \end{cases}$$

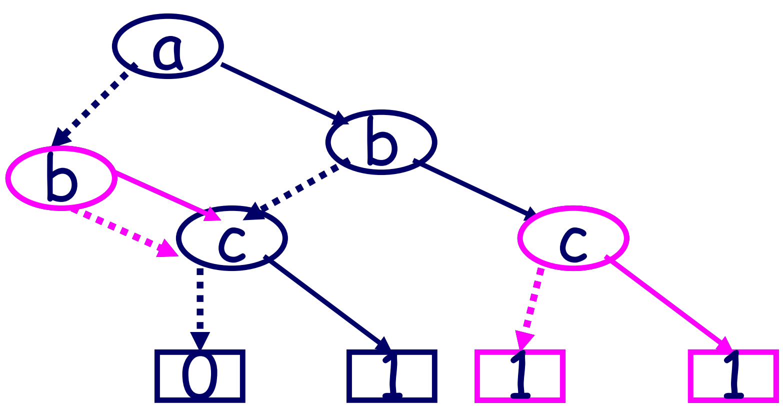
- If the elements of A are encoded by sequences over $\{0,1\}^n$ then $\mathbf{f_A}$ is a **Boolean function** and can be represented by a BDD

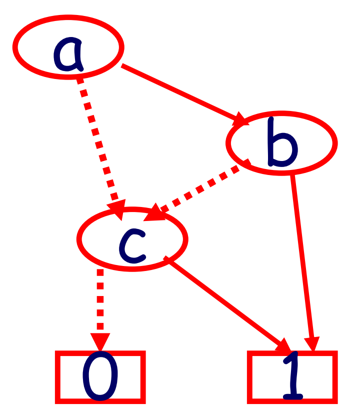
- Assume that **states** in model M are **encoded by $\{0,1\}^n$** and described by Boolean variables $\mathbf{v_1 \ldots v_n}$
- $\mathbf{S_f}$ can be represented by a BDD over $\mathbf{v_1 \ldots v_n}$
- **R** (a set of pairs of states **(s,s')**) can be represented by a BDD over $\mathbf{v_1 \ldots v_n\ v_1' \ldots v_n'}$

BDD for $f(a,b,c) = (a \wedge b) \vee c$

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 φ,

$M_1 \models \varphi \Leftrightarrow M_2 \models \varphi$

preorder on models that **weakly preserves** ACTL*

If $M_2 \geq M_1$ then for every **ACTL*** formula φ,

$M_2 \models \varphi \Rightarrow M_1 \models \varphi$

The simulation preorder [Milner]

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

$\mathbf{H \subseteq S_1 \times S_2}$ is a **simulation** iff

for every $(\mathbf{s_1}, \mathbf{s_2}) \in H$:

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

Notation: $\mathbf{s_1 \leq s_2}$

The simulation preorder [Milner]

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

$\mathbf{H \subseteq S_1 \times S_2}$ is a **simulation** iff

for every $(s_1, s_2) \in H$:

- $\forall p \in AP$: $s_2 \models p \Rightarrow s_1 \models p$
  $s_2 \models \neg p \Rightarrow s_1 \models \neg p$

- $\forall t_1 [(s_1, \mathbf{t_1}) \in R_1 \Rightarrow \exists t_2 [(s_2, \mathbf{t_2}) \in R_2 \wedge \mathbf{(t_1, t_2) \in H}]]$

Notation: $\mathbf{s_1 \leq s_2}$

Simulation preorder (cont.)

$\mathbf{H \subseteq S_1 \times S_2}$ is a **simulation** from M_1 to M_2 iff

H is a simulation and

for every $\mathbf{s_1}$ $\in I_1$ there is $\mathbf{s_2}$ $\in I_2$ s.t. $\mathbf{(s_1, s_2) \in H}$

Notation: $\mathbf{M_1 \leq M_2}$

Bisimulation relation [Park]

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

- $\forall p \in AP: \; p \in L(s_2) \Leftrightarrow p \in L(s_1)$
- $\forall t_1 [\, (s_1, \mathbf{t_1}) \in R_1 \Rightarrow \exists t_2 [\, (s_2, \mathbf{t_2}) \in R_2 \wedge \mathbf{(t_1, t_2) \in H} \,]\,]$

- $\forall \mathbf{t_2} [\, (\mathbf{s_2}, \mathbf{t_2}) \in \mathbf{R_2} \Rightarrow \exists \mathbf{t_1} [\, (\mathbf{s_1}, \mathbf{t_1}) \in \mathbf{R_1} \wedge (\mathbf{t_1}, \mathbf{t_2}) \in \mathbf{H} \,]\,]$

Notation: $\mathbf{s_1 \equiv s_2}$

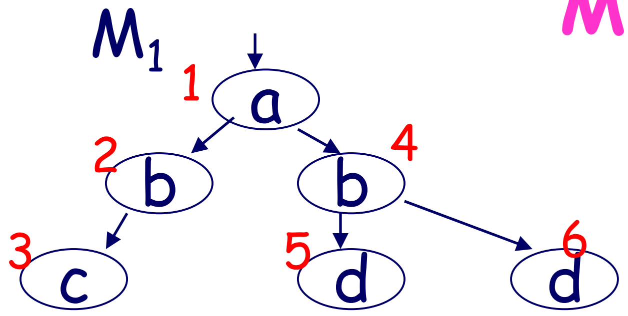
Bisimulation relation (cont.)

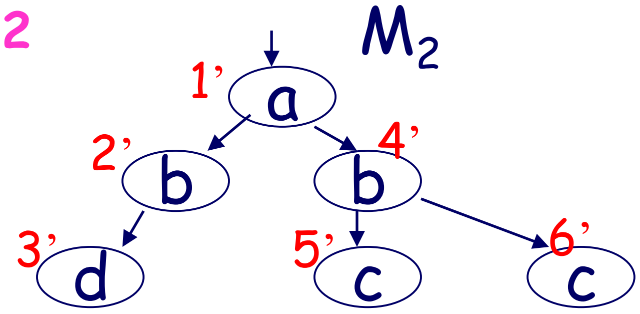
$H \subseteq S_1$ x S_2 is a **Bisimulation** between M_1 and M_2
iff H is a bisimulation and
for every **s_1** $\in I_1$ there is **s_2** $\in I_2$ s.t. **(s_1, s_2) $\in$ H** and
for every $s_2 \in I_2$ there is $s_1 \in I_1$ s.t. (s_1, s_2) $\in$ H

Notation: $M_1 \equiv M_2$

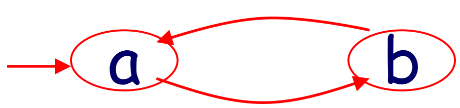
Bisimulation equivalence

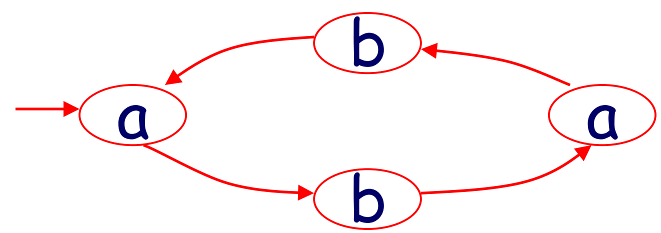
$$M_1 \equiv M_2$$





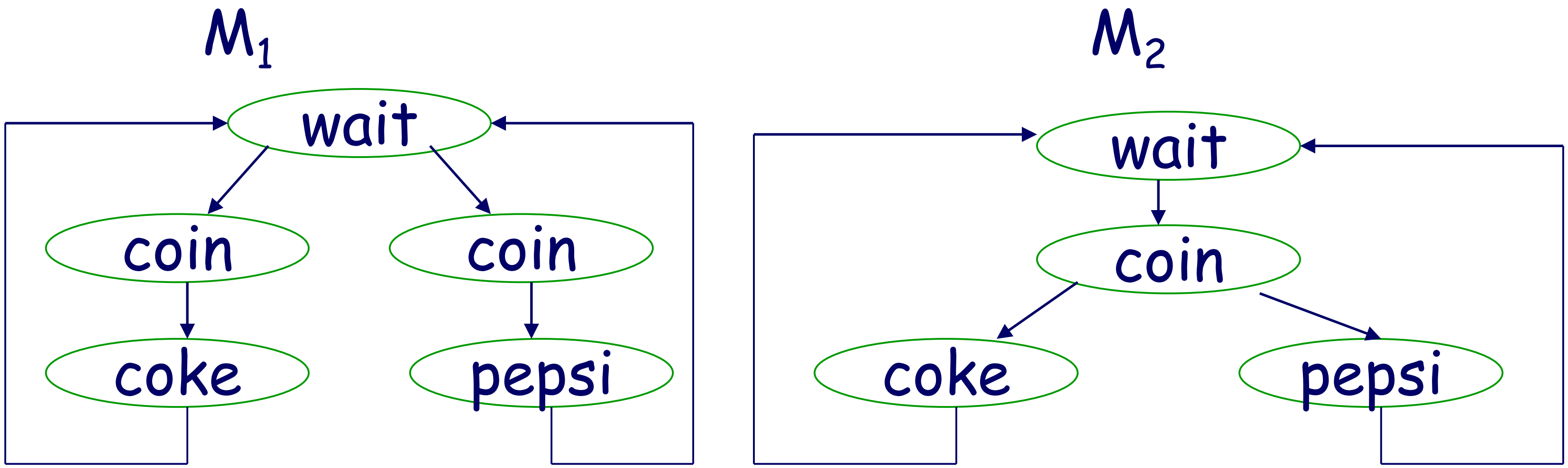
H={ (1,1'), (2,4'), (4,2'), (3,5'), (3,6'), (5,3'), (6,3') }

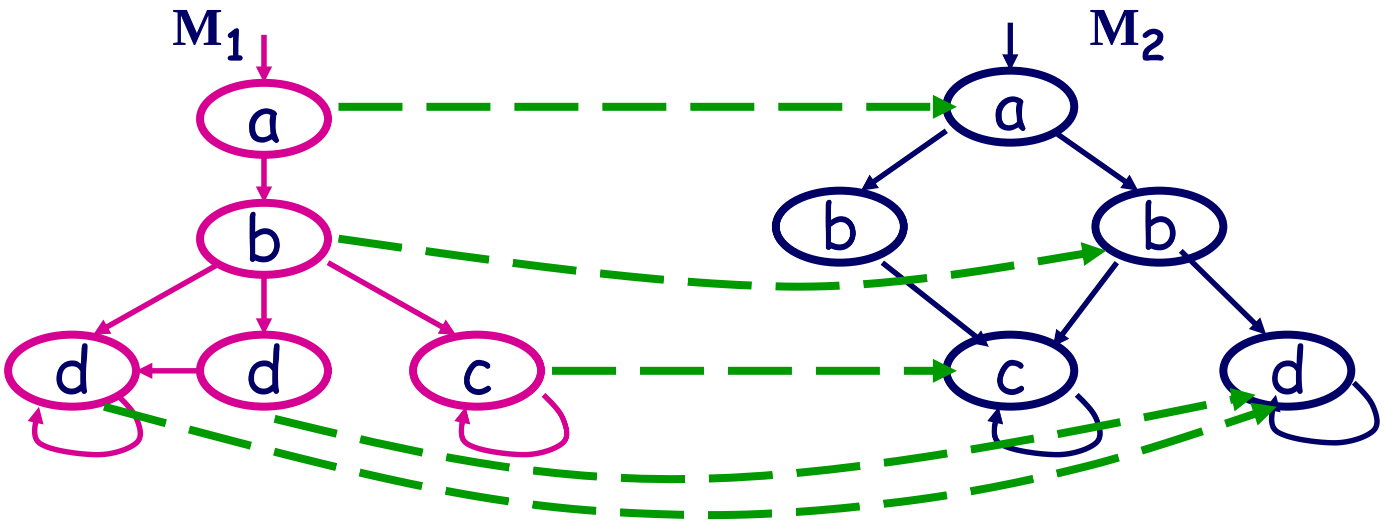




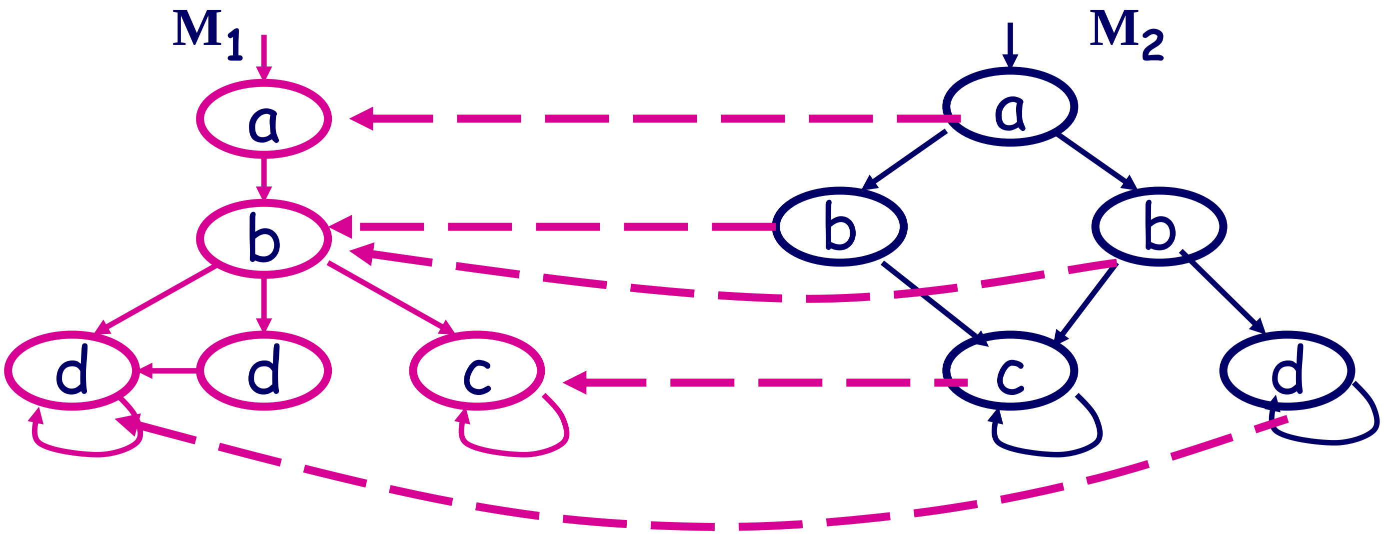
Simulation preorder

$$M_1 \leq M_2$$





$M_1 \leq M_2$



$M_1 \leq M_2$ and $M_1 \geq M_2$ but not $M_1 \equiv M_2$

(bi)simulation and logic preservation

Theorem:

If $\mathbf{M_1} \equiv \mathbf{M_2}$ then for every **CTL*** formula φ,

$M_1 \models \varphi \Leftrightarrow M_2 \models \varphi$

If $\mathbf{M_2} \geq \mathbf{M_1}$ then for every **ACTL*** formula φ,

$M_2 \models \varphi \Rightarrow M_1 \models \varphi$

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

E. Clarke, O. Grumberg, D. Long.
Model checking and abstraction,
TOPLAS, Vol. 16, No. 5, Sept. 1994

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) **$h: D \to A$**

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:

$A_1 = \{ \mathbf{a_{-}}, \mathbf{a_0}, \mathbf{a_{+}} \}$

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

Abstraction 2:

$A_2 = \{ \mathbf{a_{even}}, \mathbf{a_{odd}} \}$

$h_2(d) =$ if even($|d|$) then $\mathbf{a_{even}}$ else $\mathbf{a_{odd}}$

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 ~ s'**) iff $h(s) = h(s')$

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

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

$\mathbf{S_r} = A \times \ldots \times A$

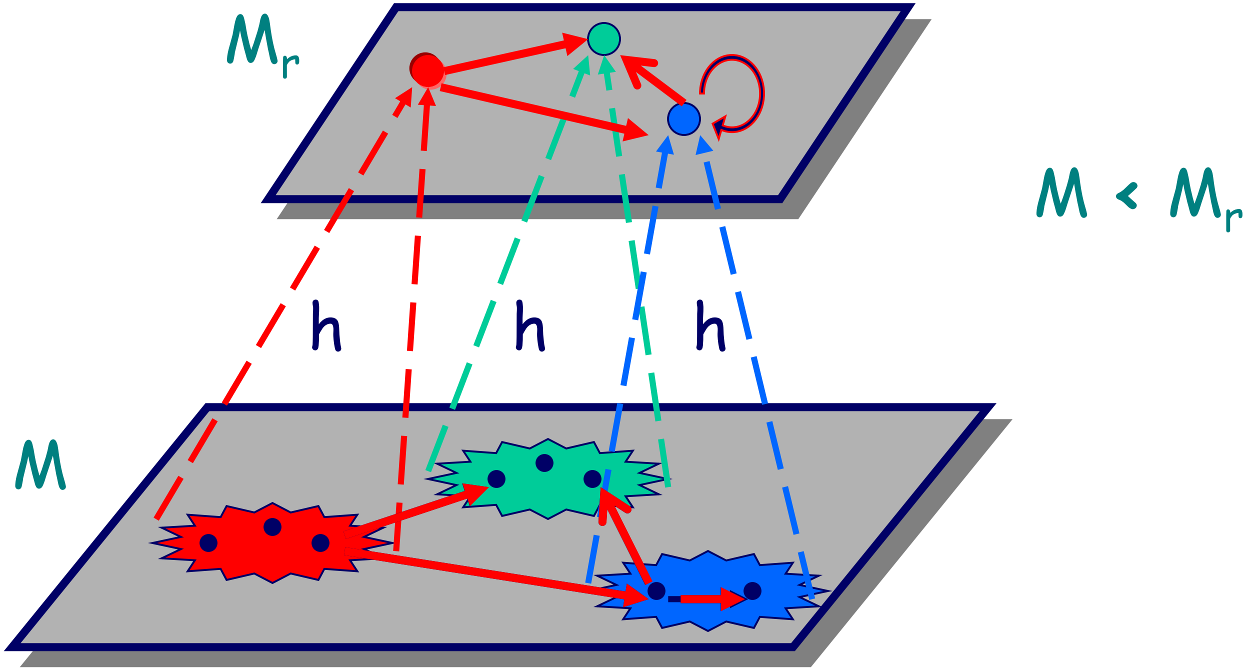
$\mathbf{s_r} \in \mathbf{I_r} \Leftrightarrow \exists\, \mathbf{s} \in I : h(\mathbf{s}) = s_r$

$\mathbf{(s_r,t_r)} \in \mathbf{R_r} \Leftrightarrow$

$\exists\, \mathbf{s,t}\ [h(\mathbf{s}) = s_r \wedge h(\mathbf{t}) = t_r \wedge (\mathbf{s,t}) \in R]$

For $s_r = (a_1,\ldots,a_n)$, $\mathbf{L_r(s_r)} = \{ (x_i^A = a_i) \mid i = 1, \ldots, n \}$

Existential Abstraction



Theorem:

$M_r \geq M$ by the simulation preorder

Corollary:

For every ACTL* formula φ:

If $M_r \models \varphi$ then $M \models \varphi$

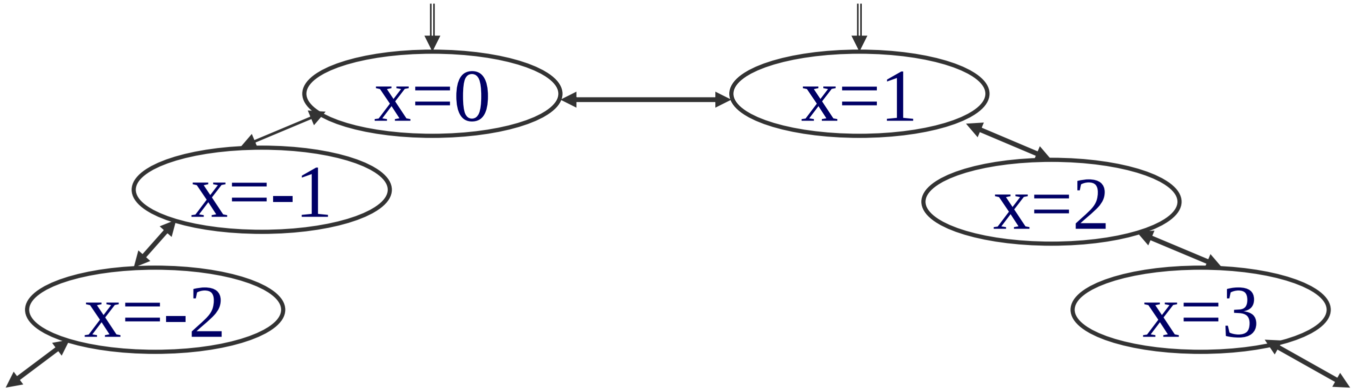
Example

Program with one variable **x** over the integers

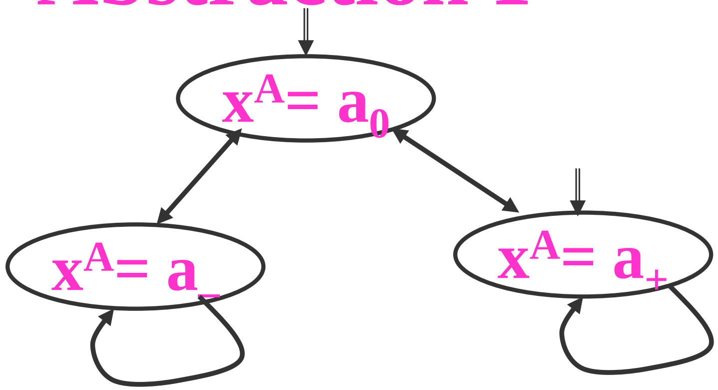
Initially x may be either **0** or **1**

At any step, **x** may non-deterministically either **decrease** or **increase** by 1

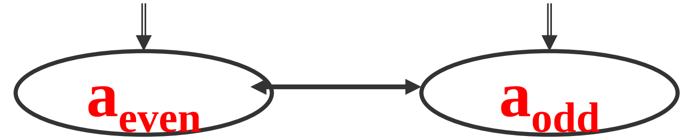
The concrete model



Abstraction 1



Abstraction 2



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** $\mathcal{I}(\mathcal{X})$ and $\mathcal{R}(\mathcal{X}, \mathcal{X}')$ where

$\mathcal{X}=(x_1,\ldots x_n)$ and $\mathcal{X}'=(x_1',\ldots x_n')$

Representing M by first-order formulas (cont)

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

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

$s \in I \Leftrightarrow \mathcal{I}[x_i \leftarrow d_i] = true$

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

$\mathcal{R}[x_i \leftarrow d_i, x_i' \leftarrow d_i'] = true$

Representing a program by formulas: example

statement: **k**: x:=e **k'**

Formula $\mathcal{R}$: $pc=k \wedge x'=e \wedge pc'=k'$

statement: **k**: if x=0 then **k_1**: x:=1 else **k_2**: x:=x+1 **k'**

Formula $\mathcal{R}$: $(\, pc=k \wedge x=0 \wedge x'=x \wedge pc' = k_1) \vee$

$(\, pc=k \wedge x \neq 0 \wedge x'=x \wedge pc' = k_2) \vee$

$(\, pc=k_1 \wedge x'=1 \wedge pc'=k') \vee$

$(\, pc=k_2 \wedge x'=x+1 \wedge pc'=k')$

Given a formula Φ over variables $\mathbf{x_1, ..., x_k}$

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

$$\exists \mathbf{x_1,...,x_k}\ (\ h(\mathbf{x_1}) = x_1^A \wedge ... \wedge h(\mathbf{x_k}) = x_k^A \wedge \Phi(\mathbf{x_1, ..., x_k})\ )$$

Let $\mathcal{I}(\mathcal{X})$ and $\mathcal{R}(\mathcal{X}, \mathcal{X}')$ be the formulas describing M.

Then $[\mathcal{I}(\mathcal{X})]$ and $[\mathcal{R}(\mathcal{X}, \mathcal{X}')]$ describe $\mathbf{M_r}$

Note: $[\mathcal{I}(\mathcal{X})]$ and $[\mathcal{R}(\mathcal{X}, \mathcal{X}')]$ are formulas **over abstract variables**

Problem:

Given $[\mathcal{I}(\mathcal{X})]$ and $[\mathcal{R}(\mathcal{X}, \mathcal{X}')]$,

in order to determine if $\mathbf{s_r} \in \mathbf{I_r}$, 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 $(\mathbf{s_r}, \mathbf{t_r}) \in \mathbf{R_r}$ 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 Φ in negation normal form over basic predicates $\mathbf{p_i}$ and $\mathbf{\neg p_i}$, $\mathcal{T}(\Phi)$ simplifies $[\Phi]$ by "pushing" the **existential quantifiers inward:**

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

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

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

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

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

$\mathcal{T}(\exists \mathbf{x} \Phi) = \exists \mathbf{x}^{\mathbf{A}}\, \mathcal{T}(\Phi)$

Approximation model

Theorem:

$[\Phi] \Rightarrow \mathcal{T}(\Phi)$

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

Corollary:

The approximation model $\mathbf{M_a}$,

defined by $\mathcal{T}(\mathcal{I})$ and $\mathcal{T}(\mathcal{R})$ satisfies:

$\mathbf{M_a} \geq \mathbf{M_r} \geq \mathbf{M}$ 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 an 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** $[\mathbf{p_i}]$ and $[\neg \mathbf{p_i}]$ for every basic action (assignment or condition) in the program

Abstract predicates provided by the user: Example

statement: $x := y+z$

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

$A = \{a_{even}, a_{odd}\}$

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

$[p](a_{even}, a_{odd}, a_{odd})$ iff
$\exists x',y,z\ (h(x') = a_{even} \wedge h(y) = a_{odd} \wedge$
$\quad h(z)= a_{odd} \wedge x'=y+z)$

Useful abstractions
Modulo an integer m

Abstraction: **$h(i) = i \bmod m$**

Properties of modulo:

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

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

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

Specification:

$AG(\text{ waiting} \wedge \text{req} \wedge (in_1 \bmod m = i) \wedge (in_2 \bmod m = j)$

$\rightarrow A(\neg ack\ U\ (ack \wedge (overflow \vee$

$(output \bmod m = i + j \bmod m)))))$

Useful abstractions

logarithm

Abstraction: $h(i) = \lceil \log_2(i+1) \rceil$

(smallest number of bits to represent i>0)

Specification:

$AG\ (\ waiting \wedge req \wedge (\ h(in_1) + h(in_2) \leq 16)$
$\quad \rightarrow A\ (\neg ack\ U\ (\ ack \wedge \neg overflow)))$

$AG\ (\ waiting \wedge req \wedge (\ h(in_1) + h(in_2) \geq 18)$
$\quad \rightarrow A\ (\neg ack\ U\ (\ ack \wedge overflow)))$

Counterexample-guided refinement

Goal:

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

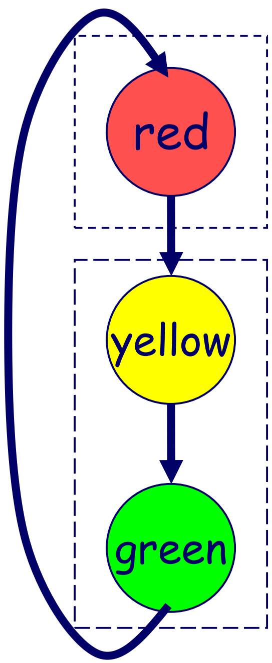
E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction Refinement, CAV'00

Traffic Light Example

Property:

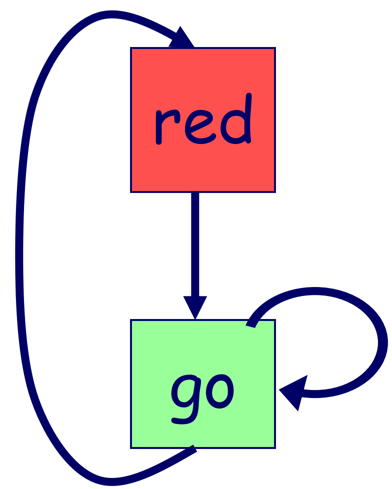
$\varphi = AG\ AF\ \neg(state=red)$

Abstraction function h maps green, yellow to go.



M

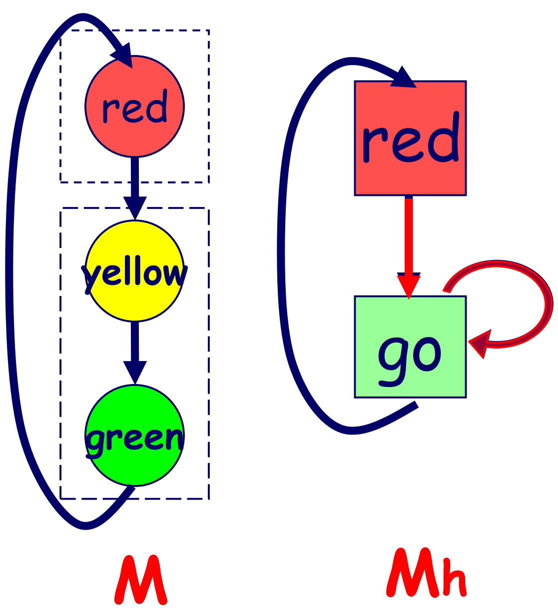
$$M \models \varphi \Leftarrow M_h \models \varphi$$



M_h

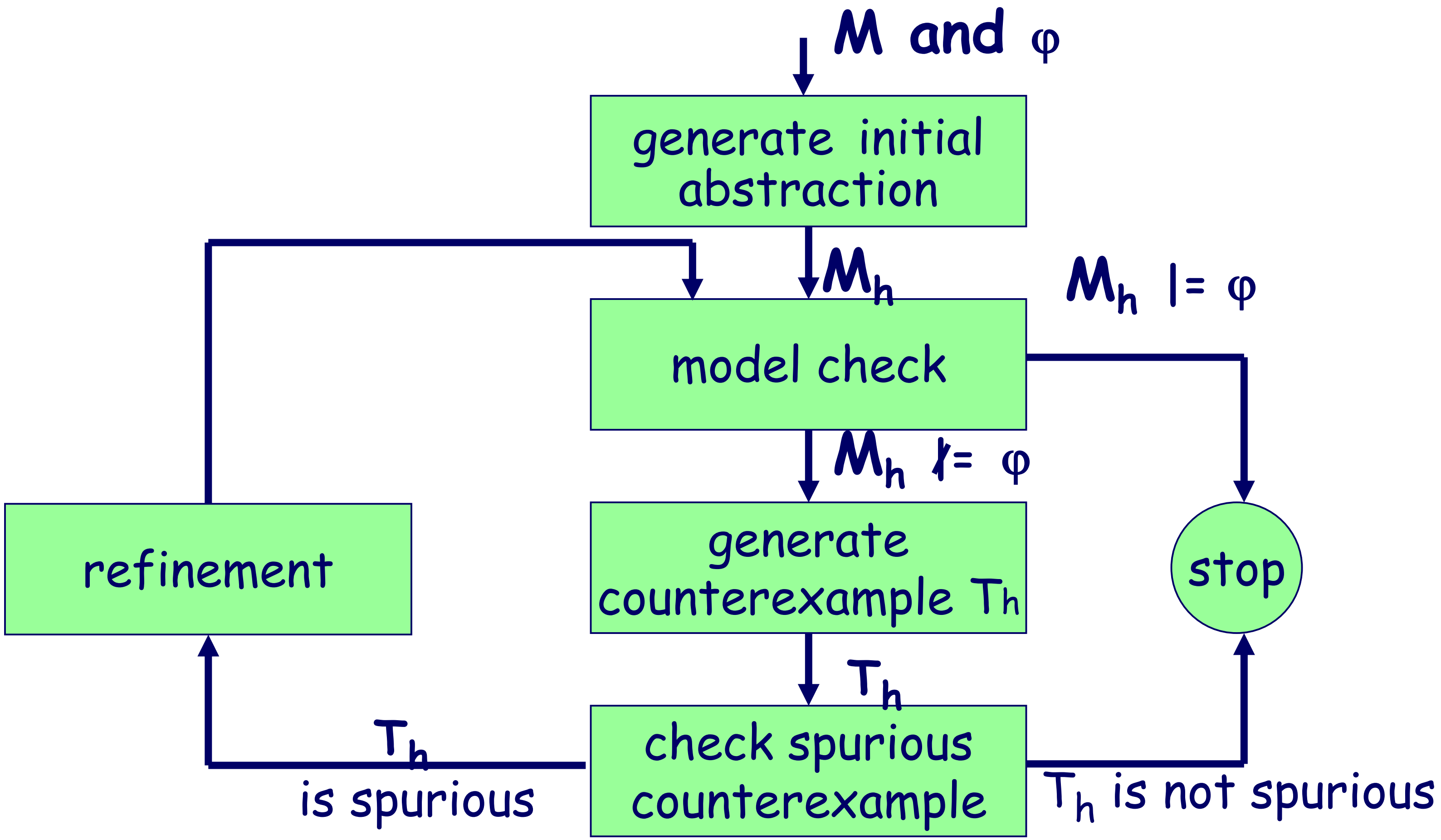
Traffic Light Example (Cont)

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



- Property:
  φ =**AG AF (state=red)**
- **M $\models \varphi$ but $M_h \not\models \varphi$**
- **Spurious** Counterexample:
  ⟨red,go,go, ...⟩

Our Abstraction Methodology



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 $\wedge$ $\neg$y=2 : y + 1;
  x = y : 0;
  else : y;
esac;

$FC_1 = \{x < y, x = y, y=2\}$, $FC_2 = \{reset=TRUE\}$

$VC_1 = \{x, y\}$, $VC_2 = \{reset\}$

Assume $x, y \in \{ 0, 1, 2 \}$

$reset \in \{ true, false \}$

Formulas in **FC_1** **cannot** distinguish **$\{x=0,y=0\}$** and **$\{x=1,y=1\}$,** therefore, **$\{x=0,y=0\}$** and **$\{x=1,y=1\}$** have the **same effect** on the control flow

Initial abstraction:

$\mathbf{h}(0,0) = \mathbf{h}(1,1) = \alpha$

Valuations **{ 0,1,2} × { 0,1,2}** of (x,y) are partitioned into **five** equivalence classes:

$\mathbf{h_1}(0,0) = \mathbf{h_1}(1,1) = \boldsymbol{\alpha}$

$\mathbf{h_1}(0,1) = \boldsymbol{\beta}$

$\mathbf{h_1}(0,2) = \mathbf{h_1}(1,2) = \boldsymbol{\gamma}$

$\mathbf{h_1}(1,0) = \mathbf{h_1}(2,0) = \mathbf{h_1}(2,0) = \boldsymbol{\delta}$

$\mathbf{h_1}(2,2) = \boldsymbol{\varepsilon}$

Valuations **{true, false}** of reset have two equivalence classes:

$\mathbf{h_2}$(true) = **true** $\mathbf{h_2}$(false) = **false**

Programs and specifications

atoms(P) is the set of **conditions** in the program **P** and **atomic formulas** in the specification **φ. atoms(P)** are defined over program variables.

Example: **$x+3<y$**

φ is an **ACTL*** formula over **atoms(P)**

A state **s** in the model of **P** is **labeled** with **f** ∈**atoms(P)** iff **s |= f**

Initial abstraction

Let $\{\mathbf{FC_1,\ldots,FC_m}\}$ be a set of formula clusters

Let $\{\mathbf{VC_1,\ldots,VC_m}\}$ be a set of variable clusters

The initial abstraction $\mathbf{h}=(\mathbf{h_1},\ldots,\mathbf{h_m})$ is defined by

$\mathbf{h_i(d_1\ldots d_k)} = \mathbf{h_i(e_1\ldots e_k)}$

iff for all $\mathbf{f \in FC_i}$,

$\mathbf{(d_1\ldots d_k)} \models \mathbf{f} \Leftrightarrow \mathbf{(e_1\ldots e_k)} \models \mathbf{f}$

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?

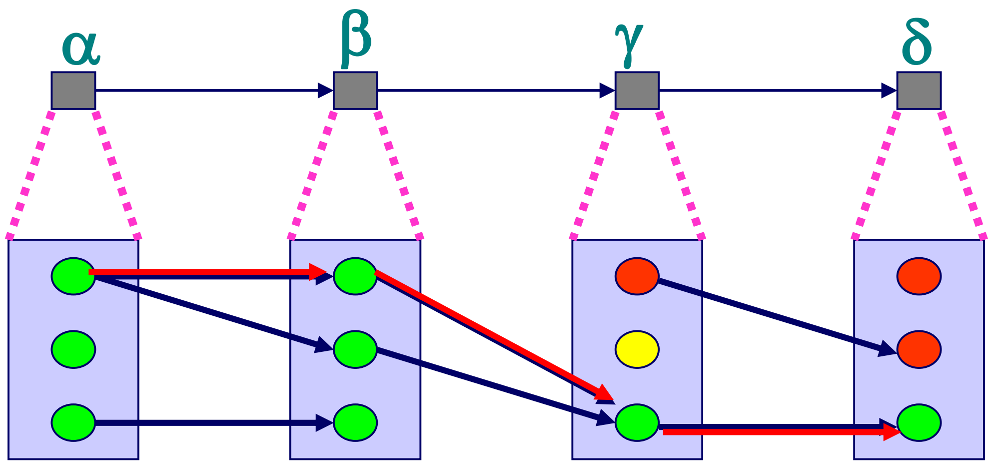
Path Counterexample

Assume that we have four abstract states

$\{1,2,3\} \leftrightarrow \alpha \qquad \{4,5,6\} \leftrightarrow \beta$

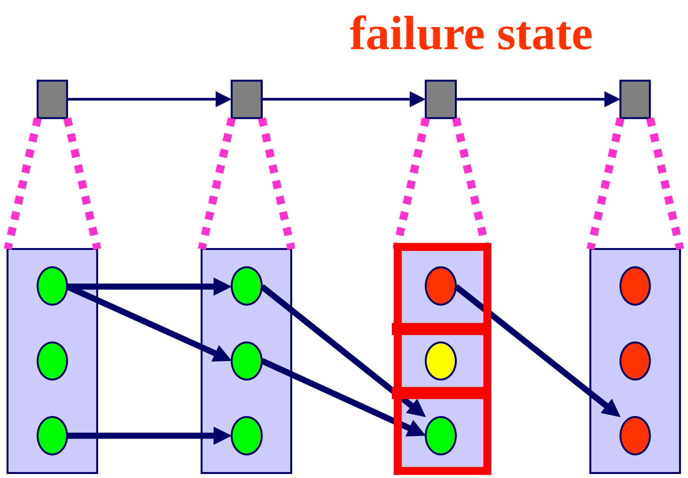
$\{7,8,9\} \leftrightarrow \gamma \qquad \{10,11,12\} \leftrightarrow \delta$

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



T_h is not spurious, therefore, $M \not\models \varphi$

Spurious Path Counterexample



T_h is spurious

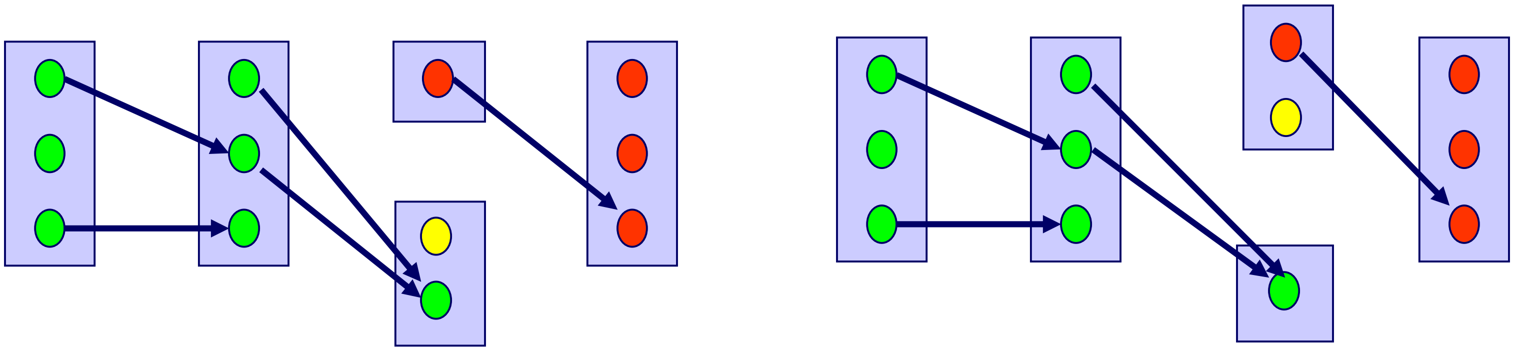
The concrete states mapped to the failure state are partitioned into 3 sets

states	dead-end	bad	irrelevant
reachable	yes	no	no
out edges	no	yes	no

Refining The Abstraction

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

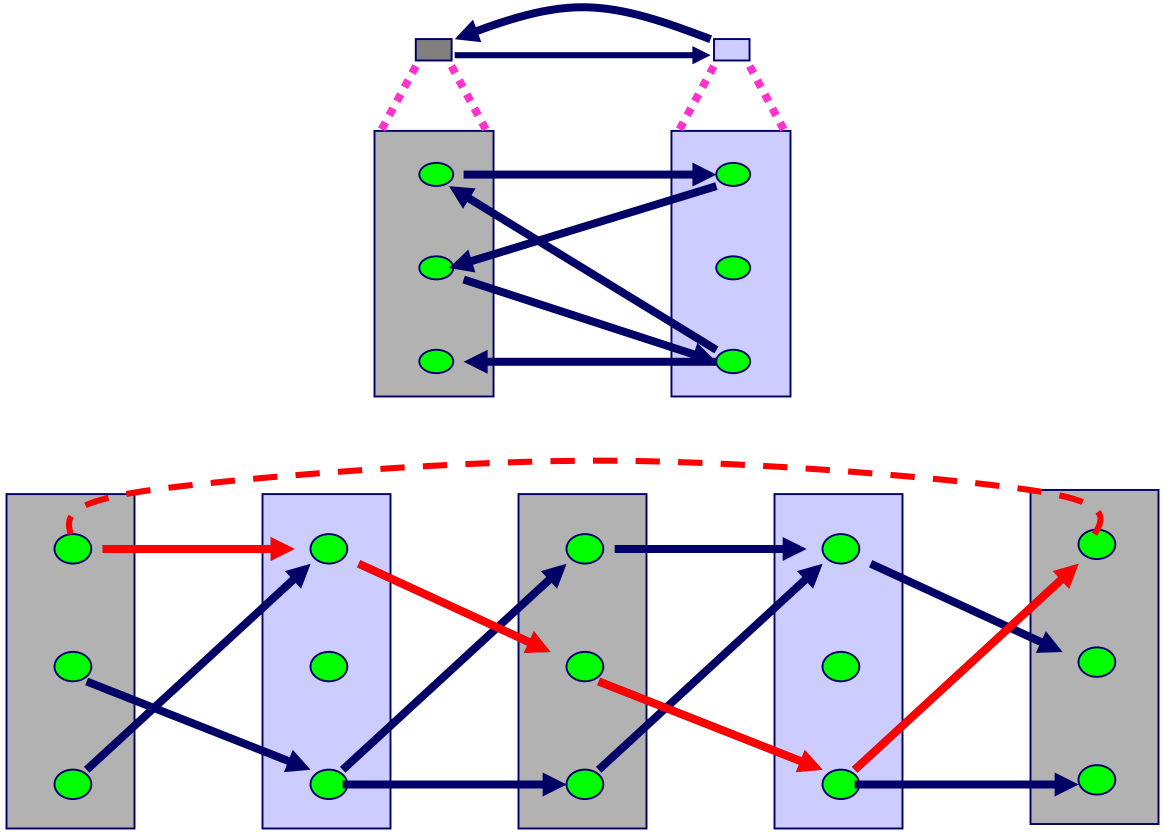
- For this example, two possible solutions.



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



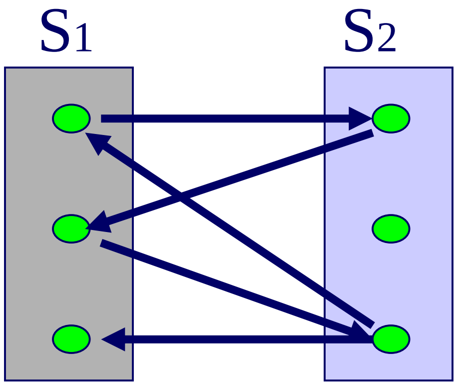
length of loop = 4

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 |= EG TRUE

Spurious Loop Counterexample

- If an abstract loop counterexample is **spurious**, loop unwinding will reach **empty** set
- Let $\mathbf{T_{unwind}}$ be the unwound loop by $|\mathbf{S_1}|$ times.
- Theorem: The loop counterexample is spurious iff $\mathbf{T_{unwind}}$ 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 **φ** whose counterexample is either path or loop, our algorithm will find a model $\mathbf{M_a}$ such that

$$\mathbf{M_a} \models \varphi \Leftrightarrow \mathbf{M} \models \varphi$$

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 **cannnot** 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,
TOPLAS Vol. 19, No. 2, March 1997.

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

Lit = $AP \cup \{\neg p \mid p \in AP\}$

L : $S \rightarrow 2^{\mathbf{Lit}}$ - labeling function so that

$p \in L(s) \Rightarrow \neg p \notin L(s)$ and

$\neg p \in L(s) \Rightarrow p \notin L(s)$

But not required: $p \in L(s) \Leftrightarrow \neg p \notin L(s)$

Galois connection

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

- α and γ 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: **$a \leq a' \Leftrightarrow \gamma(a) \leq \gamma(a')$**

then for all a, $\alpha(\gamma(a)) = a$ and (α, γ) 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

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

γ: $A \rightarrow C$ 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

$\boldsymbol{\alpha : 2^S \rightarrow S_A}$

concretization mapping

$\boldsymbol{\gamma : S_A \rightarrow 2^S}$

We use **Galois insertion**

Remarks

For every set of concrete states $\mathbf{C} \subseteq S$, $\boldsymbol{\gamma(\alpha(C)) \supseteq C}$.
Therefore, for every **C** there is an abstract state **a** with $\boldsymbol{\gamma(a) \supseteq C}$. In particular, $\mathbf{S_A}$ must contain a "top" state $\top$ so that $\boldsymbol{\gamma(\top) = S}$.

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

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

Example

Abstract states:

A = { grt_5, leq_5, ⊤ }

γ(grt_5) = {s ∈ S | s(x) > 5 }

γ(leq_5) = {s ∈ S | s(x) ≤ 5 }

The set {s ∈ S | s(x) > 6 } could be mapped to either grt_5 or ⊤, but grt_5 is **more precise**, and therefore a better choice

{s ∈ S | s(x) >0 } must be mapped to ⊤

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

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

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

If R is a transition relation

$R^{\exists\exists}(X,Y)$ 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 $\mathbf{S_A}$, to construct the **most precise** model $\mathbf{M_A = (S_A, I_A, R_A, L_A)}$ such that for every **CTL*** formula φ and abstract state $\mathbf{a} \in S_A$,

$$\mathbf{M_A, a \models \varphi \Rightarrow M, \gamma(a) \models \varphi}$$

L_A

For $p \in Lit$ **:**

$p \in L_A(a) \Leftrightarrow \forall s \in \gamma(a)\mathbf{:}\ p \in L(s)$

Note: it is possible that $p \notin L_A(a)$ and $\neg p \notin L_A(a)$

The definition guarantees for every $p \in Lit$ **:**

$\mathbf{a \models p \Rightarrow \gamma(a) \models p}$

I_A

$\mathbf{I_A = \{ \alpha(s) \mid s \in I \}}$

($\boldsymbol{\alpha}$**(s)** means $\boldsymbol{\alpha}$**({s})**)

Guarantees that $\mathbf{M_A \models \varphi \Rightarrow M \models \varphi}$

Explanation:

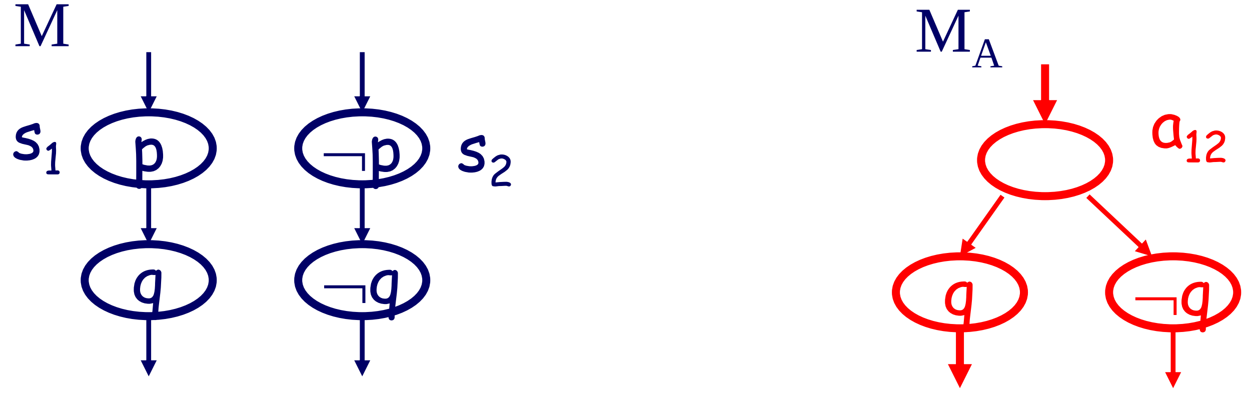
$\mathbf{M_A \models \varphi \Rightarrow \forall a \in I_A\text{: } M_A, a \models \varphi \Rightarrow}$

$\mathbf{\forall a \in I_A\text{: } M, \gamma(a) \models \varphi \Rightarrow \forall s \in I : M, s \models \varphi \Rightarrow M \models \varphi}$

More on I_A

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

Example:



$M \models A(\neg p \vee AX\ q)$ but not ($M_A \models A(\neg p \vee AX\ q)$)

R_A

We define **two** abstract transition relations:

R^A preserves **A**CTL*

R^E preserves **E**CTL*

Putting them together in the same model will preserve **full CTL***

R^A

In order to preserve ACTL* we may **add** more transitions, but **never lose** one.

Possible definition:

$R^A(a, b) \Leftrightarrow R^{\exists\exists}(\gamma(a), \gamma(b))$

R^A (cont.)

A more precise definition:

adds **less** transitions to **more precise** abstract states

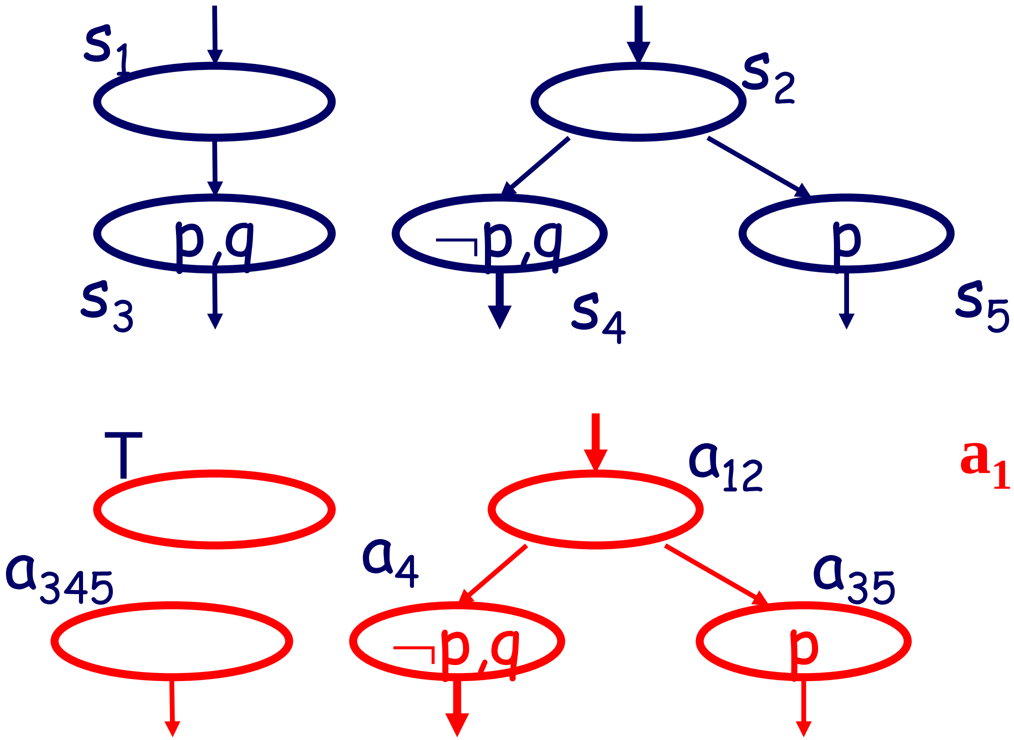
$R^A(a, b) \Leftrightarrow$

$\exists \mathbf{Y} \subseteq S \; [\, \alpha(\mathbf{Y})=b \wedge$

$\mathbf{Y}$ is a **minimal** set that satisfies $R^{\exists\exists}\; (\gamma(a), \mathbf{Y})]$

Note: Y is always a **singleton**

R^A (cont.)



$a_{12} \models AX\ (p \vee q)$

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

R^E

In order to preserve ECTL* we may **eliminate** some transitions, but **never add** non-real ones.

Possible definition:

$R^E(a, b) \Leftrightarrow R^{\forall\exists}(\gamma(a), \gamma(b))$

R^E (cont.)

A more precise definition:

keeps **more** transitions to

more precise abstract states

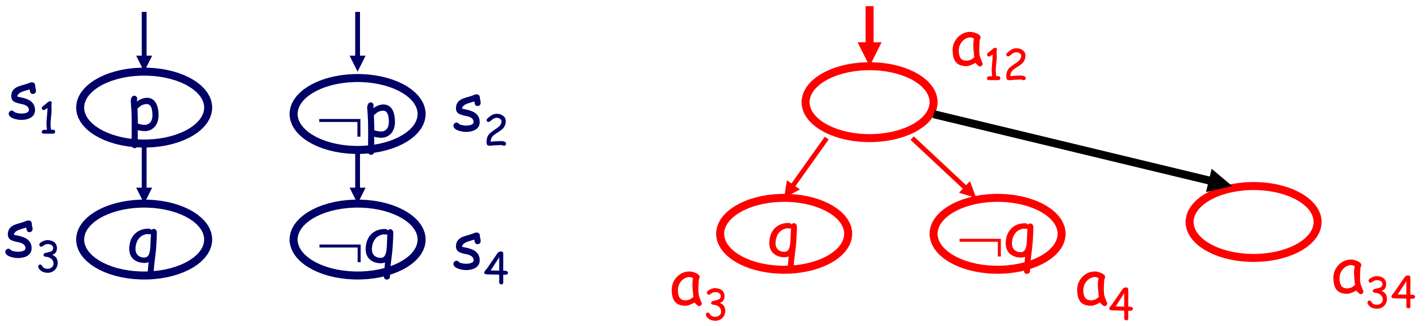
$R^E(a, b) \Leftrightarrow$

$\exists$ **Y** $\subseteq S$ [α(**Y**)=b $\wedge$

[**Y** is a **minimal** set that satisfies $R^{\forall\exists}$ (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

$\mathbf{M_A = (S_A, I_A, R^A, R^E, L_A)}$

A-path is a path over $\mathbf{R^A}$-transitions

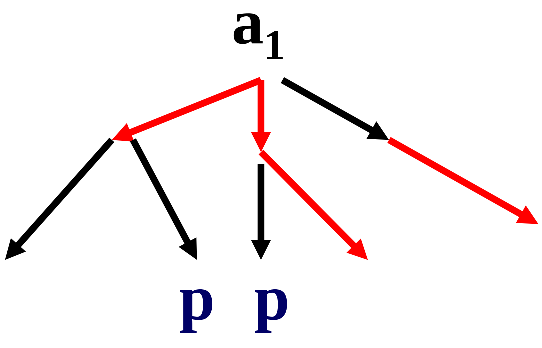
E-path is a path over $\mathbf{R^E}$-transitions

$M_A, a \models \mathbf{A}X\ f \Leftrightarrow \forall b\ [\ (a,b) \in R^{\mathbf{A}} \rightarrow M_A, b \models f\ ]$

$M_A, a \models \mathbf{E}X\ f \Leftrightarrow \exists b\ [\ (a,b) \in R^{\mathbf{E}} \wedge M_A, b \models f\ ]$

Model checking on mixed models

CTL model checking works iteratively, from simpler subformulas to more complex ones. Each subformula will be checked on either $\mathbf{R^A}$ or $\mathbf{R^E}$, according to the main operator of the formula



$\mathbf{a_1} \models \mathbf{AX\ EXp}$

We have constructed $\mathbf{M_A}$, which **given** $\mathbf{S_A}$, is the best model satisfying **for every φ in CTL***

$\mathbf{M_A} \models \varphi \Rightarrow \mathbf{M} \models \varphi$

If **not (** $\mathbf{M_A} \models \varphi$ **)** then we can check whether

$\mathbf{M_A} \models \neg \varphi$.

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

Approximations

As in other abstractions:

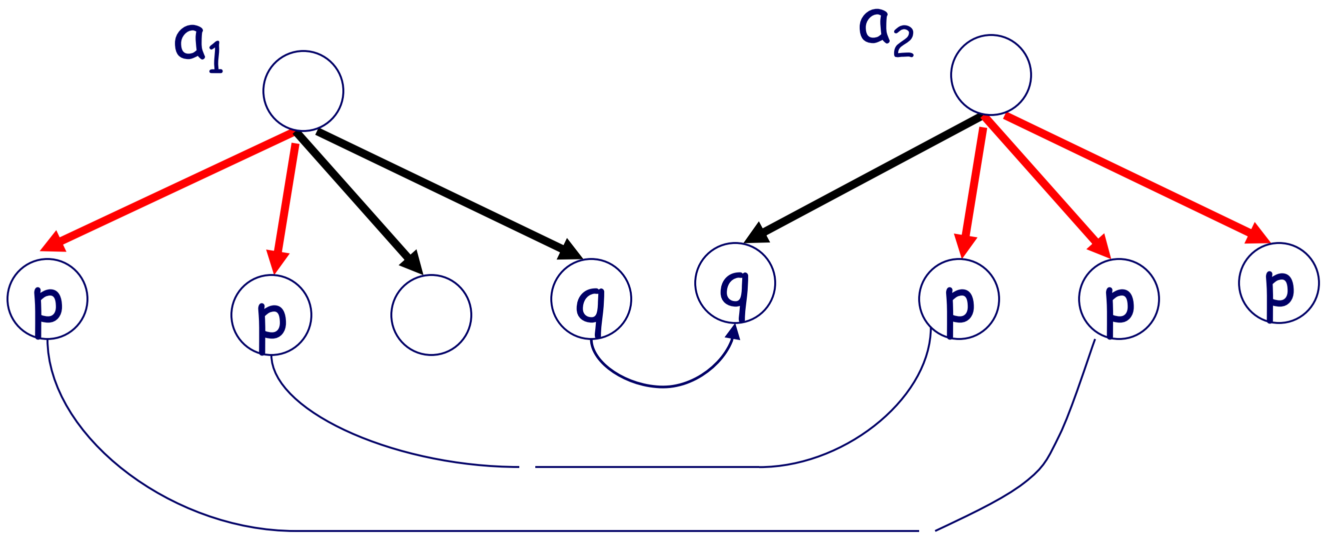
- We would like to construct the abstraction **directly** from the **program text**
- Best abstraction is too **difficult** to compute
- We therefore construct **approximation** to the abstraction

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" ≥ A' by the mixed simulation

then for every CTL* formula φ

$\mathbf{A'' \models \varphi \quad \Rightarrow \quad A' \models \varphi}$

Corollary:

If $\mathbf{A \geq M_A}$ by the mixed simulation

then $\mathbf{A \models \varphi \quad \Rightarrow \quad M \models \varphi}$

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.

$\mathbf{c_i^A(a)} \Leftrightarrow \exists s \in \gamma(a)\text{:}\ \ c_i(s)$

$\mathbf{t_i^A(a, b)} \Leftrightarrow \exists \mathbf{Y} \subseteq S\ \ [\alpha(\mathbf{Y})=b \wedge$

$\mathbf{Y}$ is a **minimal** set that satisfies $\mathbf{t_i}^{\exists\exists}$ $(\gamma(a), \mathbf{Y})]$

$\mathbf{c_i^E(a)} \Leftrightarrow \forall s \in \gamma(a)\text{:}\ c_i(s)$

$\mathbf{t_i^E(a, b)} \Leftrightarrow \exists \mathbf{Y} \subseteq S\ \ [\alpha(\mathbf{Y})=b \wedge$

$\mathbf{Y}$ is a **minimal** set that satisfies $\mathbf{t_i}^{\forall\exists}$ $(\gamma(a), \mathbf{Y})]$

Approximation for R^A and R^E

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

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

Example

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

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

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

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

Example

Program: $\{$ even(x) $\rightarrow$ x' := x/2

even(x) $\rightarrow$ x' := x+1 $\}$

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

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

$\mathbf{R'^E} = \{$ (even, odd), **(even, $\top$)** $\}$

Lemma

- $R^A \subseteq R'^A$
- 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
$\mathbf{t_i^A}$ and $\mathbf{t_i^E}$:
Replace transition **(a, b)** by transition **(a, b')**
with $\mathbf{b \leq b'}$ 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, Kurshan94, Balarin & Sangiovanni-Vincentelli 93, Lind-Nielsen & Andersen 99)
- Predicate abstraction and Theorem proving
  (Das & Dill & Park 99, Graf & Saidi 97, Uribe 99)

The End