Credit : Counterexample Guided Abstraction Refinement *

1, 2 Introduction and Preliminaries

Spurious counterexamples

CTL*: a superset of CTL (Computational Tree Logic) and LTL (Linear Temporal Logic)
ACTL* specification: formulas satisfiable by CTL* which only allow ∀ over paths

Let

φ an ACTL* specification (satisfiable formula)
P the program
Transition blocks B_i
Atoms(B_i) the set of atomic formulas that appear in the conditions (transition blocks B_i)
Atoms(φ) the set of atomic formulas appearing in the specification (satisfiable formula) φ
Atoms(P) = Atoms(B_i) U Atoms(φ)
program P corresponds to a labeled *Kripke structure* $M = (S, I, R, L)$
S = D the set of states
I ⊆ S the set of initial states
R ⊆ S x S the transition relation
L : S → $2^{Atoms(P)}$
L(d) = { f ∈ Atoms(P) | d |= f }
abstraction h for a program P is given by a surjection h : D → $\widehat{D}$
Let e, d ∈ D then d ≡ e iff h(d) = h(e)

Given program P and abstraction function h for P,
abstract Kripke structure: $\widehat{M} = (\widehat{S}, \widehat{I}, \widehat{R}, \widehat{L})$ where (existential abstraction)

1. $\widehat{S}$ is the abstract domain $\widehat{D}$
2. $\widehat{I}(\widehat{d})$ **<u>iff</u>** $\exists d\,(\,h(d) = \widehat{d} \;\wedge\; I(d)\,)$
3. $\widehat{R}(\widehat{d_1}, \widehat{d_2})$ **<u>iff</u>** $\exists d_1 \exists d_2\,(\,h(d_1) = \widehat{d_1} \;\wedge\; h(d_2) = \widehat{d_2} \;\wedge\; R(d_1, d_2)\,)$
4. $\widehat{L}(\widehat{d}) = \bigcup_{h(d) = \widehat{d}} L(d)$

Atomic formula **f**
represents an abstraction function h
If $\forall\ d, d' \in D$, $(d \equiv d') \Rightarrow (d \models \mathbf{f} \Leftrightarrow d' \models \mathbf{f})$

Let $\widehat{d}$ an abstract state
if all concrete states corresponding to $\widehat{d}$ satisfy all labels in $\widehat{L}(\widehat{d})$ then $\widehat{L}(\widehat{d})$ is consistent.
(if all concrete states corresponding to $\widehat{d}$ satisfy all labels in $\widehat{L}(\widehat{d})$ means
$\forall \quad d \in h^{-1}(\widehat{d})$ it holds that $d \models$ □ $\bigwedge$ f $\in \widehat{L}(\widehat{d})$ f

----
Theorem 1)
Given
- h an abstraction function
- φ an ACTL* specification (satisfiable formula)
- $\widehat{M} = (\widehat{S}, \widehat{I}, \widehat{R}, \widehat{L})$ abstract Kripke structure
- $\widehat{S} = \widehat{D}$ the set of abstract states
- Atoms(φ) the set of atomic formulas appearing in the specification (satisfiable formula) φ
- $Atoms(P) = Atoms$ (φ) $\cup$ $Atoms(B_i)$
- $\widehat{d} \in \widehat{D}$ or $\widehat{d} \in \widehat{S}$ an abstract state
- $\widehat{L} : \widehat{S} \rightarrow 2^{Atoms(P)}$
- $\widehat{L}(\widehat{d}) = \{$ f $\in Atoms(P) \mid \widehat{d} \models$ f $\}$
- abstraction h for a program P is given by a surjection $h : D \rightarrow \widehat{D}$ (could we say $h : S \rightarrow \widehat{S}$?)

Let
- Any atomic subformula of φ (something like Atoms(φ)) respects h

  Then the following holds:
  (i) $\widehat{L}(\widehat{d})$ is consistent for all abstract states $\widehat{d}$ in $\widehat{M}$
  (ii) $\widehat{M} \models \varphi \quad \Rightarrow \quad M \models \varphi$

Let domain $D = D_1 \times \ldots D_n$
Then: $h = (h_1, \ldots, h_n)$
Where also: $h = \{ h_i : Di \rightarrow \widehat{Di} \mid h(d_1, \ldots, d_n) = (h_1(d_1), \ldots, h_n(d_n)) \}$
($h_i : Di \rightarrow \widehat{Di}$ is a surjection)
and $\widehat{D} = \widehat{D_1} \times \ldots \widehat{D_n}$

The equivalence relations $\equiv_i$

corresponding to the individual surjections h_i

induce an equivalence relation $\equiv$

over the entire domain $D = D_1 \times \ldots D_n$

In the obvious manner:

$$(d_1, \ldots, d_n) \equiv (e_1, \ldots, e_n) \quad \underline{\textbf{iff}} \quad d_1 \equiv e_1 \wedge \ldots \wedge d_n \equiv e_n$$

In previous works:

$D_i = D_{vi}$ where D_{vi} is the set of all values possible for variable v_i

But any abstraction functions h cannot be described as:

Example:

Let $h : D \rightarrow \widehat{D}$

Let $D = \{0, 1, 2\} \times \{0, 1, 2\}$ and $\widehat{D} = \{0, 1\} \times \{0, 1\}$

$\Rightarrow h_1 : D_1 \rightarrow \widehat{D_1}$ and $h_2 : D_2 \rightarrow \widehat{D_2}$

where $h = (h_1, h_2)$, $D_1 = \{0, 1, 2\}$, $D_2 = \{0, 1, 2\}$

$\widehat{D_1} = \{0, 1\}$, $\widehat{D_2} = \{0, 1\}$

Then

$D = \{$ 00, 01, 02, 10, 11, 12, 20, 21, 22 $\}$ 9 members

D has 3 variables hence $D_v = \{0, 1, 2\}$

$\widehat{D} = \{$ 00, 01, 10, 11 $\}$ 4 members

$\widehat{D}$ has 2 variables hence $\widehat{D_v} = \{0, 1\}$

$\Rightarrow 4^9$ functions from D to $\widehat{D}$

$\Rightarrow 2^3$ functions from D_v to $\widehat{D_v}$

there are two sets of {0, 1, 2} and {0, 1}

$\Rightarrow 2(2^3) = 64$

$4^9 \neq 64$

So not always $D = D_v$

In this paper:

Set V of variables

Where $V = VC_1, \ldots, VC_m$ and each VC_i a variable cluster

The Domain of VC_i is $D_{VC_i} := \prod_{v \in VC_i} D_v$

$\Rightarrow D_V = D_{VC_1} \times \ldots \times D_{VC_m}$

Abstraction function: surjections on the domains $D_{VC_i} = D_i$

Conclusion:

In the previous solutions $D = D_v$ which cannot be used for multidimensional domain and abstract domains,

In this paper, $D_{VC_i} = D_i$

3 Overview of the paper's proposal:

Let

- program P
- ACTL* formula (specification) φ

Does Kripke structure M of program P satisfy φ?

Reminder : let

- Transition blocks B_i (conditions)
- Atoms(B_i) the set of atomic formulas that appear in the conditions B_i
- Atoms(φ) the set of atomic formulas appearing in the specification (satisfiable formula) φ
- Atoms(P) = Atoms(B_i) U Atoms(φ)

1. *Generating the initial abstraction h*

   Examining the transition blocks B_i
   corresponding to the set V of variables of the program P.
   Aiming to construct variable clusters VC_i (reminder $V = VC_1, \ldots, VC_m$)
   by considering the conditions used in the **case** statements

2. *Model-checking the abstract structure $\widehat{M}$:*

   Let $\widehat{M}$ abstract Kripke structure corresponding to abstraction h.
   We check whether $\widehat{M}$ |= φ.
   - If the check is affirmative, then we can conclude that M |= φ.
   - Suppose the check reveals that there is a counterexample T□.
     Check if T□ is an actual counterexample
     If T□ an actual counterexample, we report it to

3. *Refining the abstraction:*

Refining abstraction function h
that has admitted the spurious counterexample T□
by partitioning a *single* equivalence class $\equiv$
After refinement of h, $\widehat{M}$ corresponding to h
does not admit the spurious counterexample T□.

Reminder:
a *single* equivalence class $\equiv$
equivalence classes of VC_i where $V = (VC_1, \ldots, VC_m)$ is denoted as $\equiv_V$

4 The Abstraction-Refinement Framework

4.1 Generating the Initial Abstraction

Let
- program P
- n variables in program P, the set of variables in P is $v = \{v_1, \ldots, v_n\}$
- Atomic formula f (notice $f \subseteq$ Atoms(P))

$var(f)$ the set of variables of program P appearing in f, meaning $var(f) \subseteq v$
Example: $var(x = y)$ is $\{x, y\}$
A set of atomic formulas U

$$var(U) = \bigcup_{f \in U} var(f)$$

Then

$\forall$ syntactic entities X, $var(X)$ will be the set of variables appearing in X
Two atomic formulas f_1 and f_2 interfere **iff** $var(f_1) \cap var(f_2) \neq \varnothing$
The equivalence relation on Atoms(P) is denoted by $\equiv_I$
Reflexivity ($x \sim x$)
Transitivity ($x \sim y$) $\wedge$ ($y \sim z$) $\Rightarrow$ ($x \sim z$)
$\equiv_I$ is a reflexive-transitive closure of the interference relation

Example on $\equiv_I$: ???
$var(f_1) \cap var(f_2) \equiv_I var(f_2) \cap var(f_3)$
implies $var(f_1) \cap var(f_3) \equiv_I var(f_2) \cap var(f_3)$

The equivalence class of an atomic formula $f \in$ Atoms(P)
is called the *formula cluster* of f
and is denoted by $[f]$

Let f_1 and f_2 two atomic formulas
Then $var(f_1) \cap var(f_2) \neq \varnothing$ implies $[f_1] = [f_2]$

Moreover,

a formula cluster $[f]$
induces an equivalence relation $\equiv_V$
on the set of variables V
in the following way:

$v_i \equiv_V v_j$ **iff** $\forall\ v_i \in var(f_i)\quad \forall\ v_j \in var(f_j)$, $[f_i] = [f_j]$

variable clusters VC_i (reminder $V = VC_1, \ldots , VC_m$): equivalence classes of $\equiv_V$

Example:

Consider a formula cluster $FCi = \{v1 > 3, v1 = v2\}$
the corresponding variable cluster is $VC_i = \{v_1, v_2\}$

Let

$\{FC_1, ..., FC_m\}$ set of formula clusters
$\{VC_1, ..., VC_m\}$ set of corresponding variable clusters

initial abstraction $h = (h_1, ..., h_m)$

where $\forall\ h_i$, $D_{VC_i} = \prod_{v \in VC_i} D_v$

variable cluster $VC_i = (v_{i_1}, \ldots , v_{i_k})$

D_{VC_i} the corresponding domain to VC_i

notice $v_{i_j} \in D_{VC_i}$

Since the variable clusters VC_i form <u>a partition of the set</u> of variables V,
it follows that $D = D_{VC_1} \times \cdots \times D_{VC_m}$

remember in the previous work:

the equivalence relations $\equiv_i$

corresponding to the individual surjections h_i

induce an equivalence relation $\equiv$

over the entire domain $D = D_1 \times \ldots D_n$

in the obvious manner:

$$(d_1, \ldots, d_n) \equiv (e_1, \ldots, e_n) \quad \underline{\textbf{iff}} \quad d_1 \equiv e_1 \wedge \ldots \wedge d_n \equiv e_n$$

remember $h = (h_1, \ldots, h_m)$,

hence for each formula cluster FC_i there's abstraction function h_i

In this paper:

the corresponding abstraction h_i is defined on D_{VC_i} as follows:

$$h_i \ (d_1, \ldots, d_k) = h_i \ (e_1, \ldots, e_k)$$

iff

$\forall$ atomic formulas $f \in FC_i$, $[(d_1, \ldots, d_k) \models f] \Leftrightarrow [(e_1, \ldots, e_k) \models f]$

In other words:

two values $(d_1, \ldots, d_k)$ and $(e_1, \ldots, e_k)$ in the same equivalence class

if they cannot be "distinguished" by all atomic formulas f such that $f \in FC_i$

Example 2 (initial abstraction h):

Let

program P

with three variables x, y, reset such that

x,y ∈ {0, 1, 2},

reset ∈ {TRUE,FALSE}

the set of atomic formulas:

Atoms(P) = {(reset = TRUE), (x = y), (x < y), (y = 2)}

two formula clusters

FC_1 = { (x = y), (x < y), (y = 2) }

FC_2 = { (reset = TRUE) }

with two corresponding variable clusters

VC_1 = {x, y}

VC_2 = {reset}

values (0, 0) and (1, 1) are in the same equivalence class because for all the atomic formulas $f \in FC_1$, (0,0) |= f **iff** (1,1) |= f

0 = 0	**iff**	1 = 1
0 < 0	**iff**	1 < 1
0 = 2	**iff**	1 = 2

domain {0, 1, 2} × {0, 1, 2} = { (0,0) , (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2)} is partitioned into a total of five equivalence classes by this criterion:

Equivalence class 0 = {(0, 0), (1, 1)}
Equivalence class 1 = {(0, 1)}
Equivalence class 2 = {(0, 2), (1, 2)}
Equivalence class 3 = {(1, 0), (2, 0), (2, 1)}
Equivalence class 4 = {(2, 2)}

// Start Program

flag = false
If FC_1 = { (x = y), (x < y), (y = 2) } holds for (x_0, y_0) and (x_1, y_1) then
Create Equivalence class 0
Put (x_0, y_0) and (x_1, y_1) in Equivalence class 0
Else
Create Equivalence class 0
Create Equivalence class 1
Put (x_0, y_0) in Equivalence class 0
Put (x_1, y_1) in Equivalence class 1

For i = 2 to n
For j = 0 to count.EquiClasses
If FC_1 = { (x = y), (x < y), (y = 2) } holds for (x_i, y_i) and $(x, y) \in$ Equivalence class j then
Put (x_i, y_i) in Equivalence class j
j = count.EquiClasses + 1
flag = true

If flag == false then
count.EquiClasses = count.EquiClasses + 1
Create Equivalence class count.EquiClasses
Put (x_i, y_i) in Equivalence class count.EquiClasses

// End Program

domain {TRUE, FALSE} :

FC = { TRUE }

Equivalence class true = {TRUE}

Equivalence class false = {FALSE}

Hence we have two abstraction functions h_1 and h_2 such as:

$$h_1 \ : \ \{0,\ 1,\ 2\}^2 \ \rightarrow \ \{ \ Equivalence\ Class\ 0, \\ Equivalence\ Class\ 1, \\ Equivalence\ Class\ 2, \\ Equivalence\ Class\ 3, \\ Equivalence\ Class\ 4 \ \}$$

$$h_2 \ : \ \{TRUE,\ FALSE\} \rightarrow \{ \ Equivalence\ class\ \ true\ , \\ Equivalence\ class\ \ false \ \}$$

To simplify h_1 : $h_1 \ : \ \{0,\ 1,\ 2\}^2 \ \rightarrow \ \{\ 0,\ 1,\ 2,\ 3,\ 4\}$

To simplify h_2 : $h_2 \ : \ \{TRUE,\ FALSE\} \rightarrow \{TRUE,\ FALSE\}$

Simplified h_1 output:

$h_1(0,0) \ = \ h_1(1,1) \ = \ 0$ (or = Equivalence Class 0)

$h_1(0,1) \ = \ 1$ (or = Equivalence Class 1)

$h_1(0,2) \ = \ h_1(1,2) \ = \ 2$ (or = Equivalence Class 2)

$h_1(1,0) \ = \ h_1(2,\ 0) \ = \ h_1(2,\ 1) \ = \ 3$ (or = Equivalence Class 3)

$h_1(2,\ 2) \ = \ 4$ (or = Equivalence Class 4)

h_2 identity function:

$$h_2(reset) \ = \ reset$$

Given the abstraction functions h_1 and h_2 ,
we use standard existential abstraction techniques to compute the abstract model $\widehat{M}$.

4.2 Model Checking the Abstract Model

Given

- an ACTL⋆ specification φ
- an abstraction function h (φ respects □ h)
- a program P with a finite set of variables □ $V = \{v_1, \cdots, v_n\}$

Let

- $\widehat{M}$ the abstract Kripke structure
- h the abstraction function that $\widehat{M}$ corresponds to

Does $\widehat{M}$ satisfy the specification (satisfiable formula) φ ?

If yes:

- By Theorem 1 we can conclude that the original Kripke structure M also satisfies φ.

If no:

- Firstly we assume the model checker is producing a counterexample $\widehat{T}$ corresponding to the abstract model $\widehat{M}$. Implying an spurious counterexample $\widehat{T}$ is assumed.

Identification of Spurious Path Counterexamples

Case 1)

$\widehat{T}$ is a path $\langle \widehat{s_1}, \cdots, \widehat{s_n} \rangle$

the set of abstract states $\widehat{s}$

the set of concrete states s such that $h(s) = \widehat{s}$ is denoted by $h^{-1}(\widehat{s})$

i.e., $h^{-1}(\widehat{s}) = \{ s \mid h(s) = \widehat{s} \}$

$h^{-1}{}_{path}$ is the same as h^{-1} , since h^{-1} is applied to a sequence

$$h^{-1}(\widehat{T}) = \{ \langle s_1, \cdots, s_n \rangle \mid \bigwedge^{n} h(s_i) = \widehat{s}_i \wedge I(s_1) \wedge \bigwedge^{n-1} R(s_i, s_{i+1}) \}$$

$i = 1$ $i = 1$

Algorithm to compute $h^{-1}(\widehat{T})$:

Let

$S_1 = h^{-1}(\widehat{s}_1) \cap I$

R transition relation on M

Define:

$S_i := Img(S_{i-1}, R) \cap h^{-1}(\widehat{s}_i)$, $1 < i \leq n$

Where:

$Img(S_{i-1}, R)$: forward image of S_{i-1}

with respect to transition relation R

S_1 through S_n is computed using Ordered Binary Decision Diagrams and Img(...)

Lemma 1)

(i) Abstract path $\widehat{T}$ corresponds to a concrete path T (concrete counterexample)

(ii) The set of concrete paths $T = h^{-1}(\widehat{T})$ is non-empty

(iii) $\forall\ 1 \leq i \leq n$, $S_i \neq \varnothing$.

Note: $h^{-1}(\widehat{T}) \neq \varnothing$ implies the counterexample T□ is spurious)

Lemma 1 implies :

$h^{-1}(\widehat{T}) \neq \varnothing$ then there exists a minimal i, $2 \leq i \leq n$ such that $S_i = \varnothing$

Side Note

بازکردن حلقه (انگلیسی: loop unwinding or Loop unrolling) از تکنیک‌های تبدیل حلقه یا لوپ در برنامه‌نویسی و بهینه‌سازی می‌باشد، که با استفاده از روش بازکردن حلقه، تلاش می‌کند تا سرعت اجرای برنامه را بهینه‌سازی نماید. این روش اغلب برای حلقه‌های کوتاه مناسب

است. پس از اینکه یک حلقه باز می‌شود، شرط حلقه برای چک کردن وجود ندارد و در هر مرحله اجرای حلقه، شاخه‌های کمتری اجرا می‌شوند. در مجموع با استفاده از تکنیک بازکردن حلقه، سرعت اجرای برنامه افزایش می‌یابد و از سوی دیگر، حجم کد برنامه افزایش خواهد یافت. تکنیک بازکردن حلقه، بخشی از روش‌های درستی‌یابی صوری (Formal Verification) است، که کاربرد ویژه آن در زمینه وارسی مدل می‌باشد.

Fig 3. An abstract counterexample

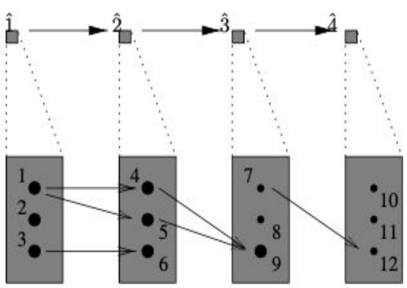


Fig 4. SplitPath checks spurious path

Algorithm SplitPath

$S \;=\; h^{-1}(\widehat{s_1}) \;\cap\; I$

$j \;:=\; 1$

While ($S \neq \varnothing$ and $j < n$)

$\quad j := j + 1$

$\quad S_{prev} := S$

$\quad S := Img(S_{prev}, R) \;\cap\; h^{-1}(\widehat{s_j})$

if $S \neq \varnothing$ **then** output counterexample (S) [genuine]

else output j, S_{prev} [spurious]

Example 3)

Consider a program with only one variable with domain $D = \{1, \cdots, 12\}$
Assume the abstraction function $h(x) = \lfloor (x-1)/3 \rfloor + 1$
There are four abstract states corresponding to the —equivalence classes—

equivalence class s_1 = {1, 2, 3}
equivalence class s_2 = □{4, 5, 6}
equivalence class s_3 = {7, 8, 9}
equivalence class s_4 = {10, 11, 12}

The *transitions between states* (R) in the concrete model are indicated by the arrows in Figure 3;

- big dots denote reachable states
- small dots denote non-reachable states

abstract counterexample $\widehat{T}$ = ⟨□$\widehat{1}$, $\widehat{2}$□, □$\widehat{3}$, □$\widehat{4}$⟩ is spurious

domain $D = \{1, \cdots, 12\}$
abstraction function h is : $h(x) = \lfloor (x-1)/3 \rfloor + 1$

—— $h(s_1) = \widehat{s_1} = \widehat{1}$ => $h^{-1}(\widehat{s_1}) = \{1, 2, 3\}$, $I = \{1, 2, 3\}$
—— $h(s_2) = \widehat{s_2} = \widehat{2}$ => $h^{-1}(\widehat{s_2}) = \{4, 5, 6\}$, $Img(S_1, R) = \{4, 5, 6\}$
—— $h(s_3) = \widehat{s_3} = \widehat{3}$ => $h^{-1}(\widehat{s_3}) = \{7, 8, 9\}$, $Img(S_2, R) = \{9\}$
—— $h(s_4) = \widehat{s_4} = \widehat{4}$ => $h^{-1}(\widehat{s_4}) = \{10, 11, 12\}$, $Img(S_3, R) = \varnothing$

$S_1 = I \cap h^{-1}(\widehat{s_1}) = \{1, 2, 3\} \cap \{1, 2, 3\} = \{1, 2, 3\}$
$S_2 = Img(S_1, R) \cap h^{-1}(\widehat{s_2}) = \{4, 5, 6\} \cap \{4, 5, 6\} = \{4, 5, 6\}$
$S_3 = Img(S_2, R) \cap h^{-1}(\widehat{s_3}) = \{9\} \cap \{7, 8, 9\} = \{9\}$
$S_4 = Img(S_3, R) \cap h^{-1}(\widehat{s_4}) = \varnothing \cap \{10, 11, 12\} = \varnothing$

Notice that S_4 and therefore $Img(S_3, R)$ are both empty

Identification of Spurious Loop Counterexamples

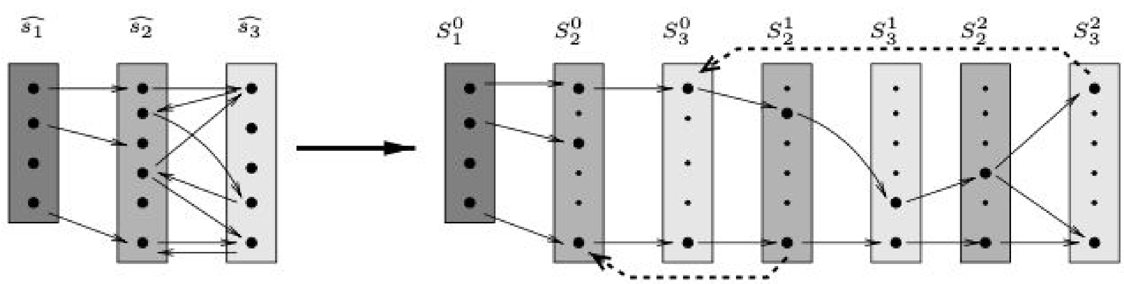
When abstract counterexample $\widehat{T}$ is a loop:

$$\widehat{T} = \langle \widehat{s_1}, \cdots, \widehat{s_i} \rangle \langle \widehat{s_{i+1}}, \cdots, \widehat{s_n} \rangle^{\omega}$$

Each arbitrary s_j is an actual state
Each arbitrary $\widehat{s_j}$ is an abstract state

Example 4)

Consider a loop $\langle \widehat{s_1} \rangle \langle \widehat{s_2}, \widehat{s_3} \rangle^{\omega}$
Does the abstract loop correspond to concrete loops?



An example cycle is indicated by a fat dashed arrow.

Observe:

(i) A given abstract loop may correspond to several concrete loops of different size
(ii) Each of these loops may start at different stages of the unwinding

(iii) The unwinding eventually becomes periodic (in our case $S^0{}_3 = S^2{}_3$),
but only after several stages of the unwinding.

size of the loop's period = least common multiple (Loop1Size, … , LoopNSize)

- Hence size of the loop's period in general exponential
- Hence naive algorithms may have exponential time complexity due to an exponential number of loop unwindings.

for $\widehat{T} = \langle \widehat{s_1}, \cdots, \widehat{s_i} \rangle \langle \widehat{s_{i+1}}, \cdots, \widehat{s_n} \rangle^{\omega}$

the number of unwindings = $min_{\,i+1 \le j \le n} \ | \ h^{-1} (\widehat{s_j}) \ |$

Meaning if there's a state set $s_j = h^{-1} (\widehat{s_j})$ with set size x ($| s_j | = | h^{-1} (\widehat{s_j}) |$ = x) and x is the minimum set size amongst all $s_1 = h^{-1} (\widehat{s_{i+1}})$, …, $s_n = h^{-1} (\widehat{s_n})$ then the number of unwindings is equal to x

the number of unwindings is at most the number of concrete states | s_j | for any abstract state $\widehat{s_k}$ in the loop.

Let $\widehat{T}_{unwind}$ denote a unwinded loop counterexample

i.e. let $\widehat{T}_{unwind} = \langle \widehat{s_1}, \cdots, \widehat{s_i} \rangle \langle \widehat{s_{i+1}}, \cdots, \widehat{s_n} \rangle^{min}$

Then theorem 2 holds.

Theorem 2)

the followings are equivalent:

(i) $\widehat{T}$ corresponds to a concrete counterexample

(ii) $h_{path}^{-1} (\widehat{T}_{unwind}) \neq \varnothing$

4.3 Refining the Abstraction

Path Refining:

Let

a - abstract counterexample $\widehat{T} = \langle \Box\, \widehat{s_1}, \ldots, \Box\, \widehat{s_n} \rangle$ a path.

b - $\widehat{T}$ does not correspond to a real counterexample,

b implies, by Lemma 1 part (iii), that

$\exists$ set $S_i \subseteq h^{-1}(\widehat{s_i}) = s_i$ with 1 ≤ i < n such that $Img(S_i, R) \cap h^{-1}(\widehat{s_{i+1}}) = \varnothing$ and S_i is reachable from initial state set $h^{-1}(\widehat{s_1}) \cap I$.

since there is a transition R from abstract state $\widehat{s_i}$ to abstract state $\widehat{s_{i+1}}$ in the abstract model then there is at least one transition R from concrete state $s_i = h^{-1}(\widehat{s_i})$ to concrete state $s_{i+1} = h^{-1}(\widehat{s_{i+1}})$, even though there is no transition R from S_i to $h^{-1}(\widehat{s_{i+1}})$

We partition the concrete state $s_i = h^{-1}(\widehat{s_i})$ into three subsets $S_{i\,,\,0}$, $S_{i\,,\,1}$, and $S_{i\,,\,x}$ where:

- $S_{i\,,\,0} = S_i$
- $S_{i\,,\,1} = \{ \text{s} \in [\,h^{-1}(\widehat{s_i}) = s_i\,] \mid \exists\ s' \in h^{-1}(\widehat{s_{i+1}}).\, R(s, s') \}$
- $S_{i\,,\,x} = [\,h^{-1}(\widehat{s_i}) = s_i\,] \setminus S_{i\,,\,0} \cup S_{i\,,\,1}$

In simple words:

- $S_{i\,,\,0}$ the set of initial states
- $S_{i\,,\,1}$ the set of states in $h^{-1}(\widehat{s_i}) = s_i$ that are not not reachable from initial states
- $S_{i\,,\,x}$ the set of every other states that is in $h^{-1}(\widehat{s_i}) = s_i$ but not in $S_{i\,,\,0}$ or in $S_{i\,,\,1}$

Remembering Example 3, S_3 would be:

$S_{3\,,\,0}$ = {9}

$S_{3\,,\,1}$ = {7}

$S_{3\,,\,x}$ = {8}

$S_{i\,,\,1} \neq \varnothing \rightarrow$ there is a spurious transition $\widehat{s_i} \rightarrow \widehat{s_{i+1}}$

This causes spurious counterexample $\widehat{T}$

Hence we need a refined abstraction function h such that

if $s \in [\,h^{-1}(\widehat{s_i}) = s_i\,]$ then $\neg(\ s \in S_{i\,,\,0} \wedge s \in S_{i\,,\,1}\ )$

Theorem 3.

(i) The problem of finding the coarsest abstract function h refinement is NP-hard;

(ii) $S_{i,x} = \varnothing$ *then the problem of finding the coarsest abstract function h refinement can be solved in polynomial time.*

Note:
Partition Into Cliques can be reduced to *the problem of finding the coarsest abstract function h refinement*

For when $S_{i,x} = \varnothing$
Let

P_{j+}, P_{j-} projection functions such that

where

$s = (d_1,\ldots,d_m)$,

$P_{j+}(s) = d_j$ and $P_{j-}(s) = (d_1,\ldots, d_{j-1}, d_{j+1},\ldots,d_m)$

Then:

$\mathrm{proj}(S_{i,0}, j, a) = \{ P_{j-}(s) \mid P_{j+}(s) = a,\ s \in S_{i,0} \}$

Hence $\mathrm{proj}(S_{i,0}, j, a) \neq \mathrm{proj}(S_{i,0}, j, b)$ in the *abstraction function h refinement algorithm*

Means that

$\exists\ P_{j-}(s) = (d_1,\ldots, d_{j-1}, d_{j+1},\ldots,d_m) \in \mathrm{proj}(S_{i,0}, j, a) \wedge P_{j-}(s) = (d_1,\ldots, d_{j-1}, d_{j+1},\ldots,d_m) \notin \mathrm{proj}(S_{i,0}, j, b)$

According to definition:

$\mathrm{proj}(S_{i,0},j,a)$ implies $s_1 = (d_1,\ldots,d_{j-1},a, d_{j+1},\ldots,d_m) \in S_{i,0}$

Hence

$s_2 = (d_1,\ldots,d_{j-1},b,d_{j+1},\ldots,d_m) \notin S_{i,0}$

i.e. $s_2 = (d_1,\ldots,d_{j-1},b,d_{j+1},\ldots,d_m) \in S_{i,1}$

s_1 and s_2 only different in jth element

Hence to separate s1 and s2 into different equivalence classes

a and b have to be in different equivalence classes of $\equiv'_j$, i.e., $a \not\equiv'_j b$.

Lemma 2.

When $S_{i,x} = \varnothing$*, the relation* $\equiv'_j$ *computed by* **PolyRefine** *is an equivalence relation which refines* $\equiv_j$ *and separates* $S_{i,0}$ *and* $S_{i,1}$*. Furthermore, the equivalence relation* $\equiv'_j$ *is the coarsest refinement of* $\equiv_j$ *.*

In this paper, the following heuristics is used:

- Merging the states in Si,x into Si,1,
- using the algorithm PolyRefine to find the coarsest refinement that separates the sets Si,0 and Si,1 ∪ Si,x.

This heuristics isn't optimal but has given good practical results to the authors
Works the same manner for SplitLoop once its unwinded

Theorem 4. *Given a model* M *and an* ACTL* *specification* φ *whose counterexample* *is either path or loop, our algorithm will find an abstract model* $\widehat{M}$ *such that* $\widehat{M} \models \varphi \Leftrightarrow M \models \varphi$.

5 Performance Improvements

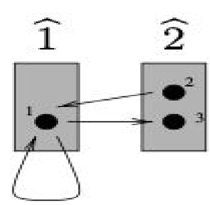


Fig. 8. A spurious loop counterexample $\langle \widehat{1}, \widehat{2} \rangle^{\omega}$

Two-phase Refinement Algorithms

When spurious path (1,1,1,...) is detected in abstract model but this is needed in concrete one, we use :

$$S_{\text{local}} := \left(\bigcup_{1 \leq i \leq n} h^{-1}(\widehat{s_i})\right)$$

Approximation

Using *early approximation* to avoid turning the whole abstract model $\widehat{M}$ to concrete model M

Abstractions For Distant Variables

Using user provided constant to completely abstract variables whose distance from the specification in the *variable de- pendency graph* is greater than the user intends