Abstraction Refinement

Abhik Roychoudhury
National University of Singapore

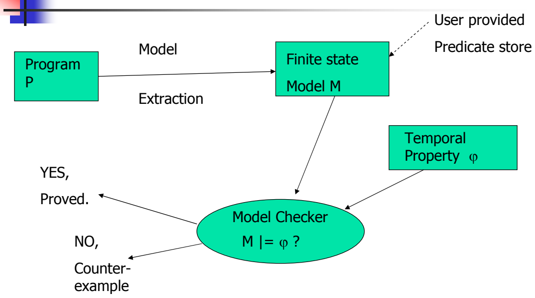
MC

- Model checking is a search based procedure applicable to only finite state systems.
- Extension to infinite state systems (arising out of infinite data domains) handled by abstraction of memory store.
- Requires human ingenuity in choice of the abstract predicates.

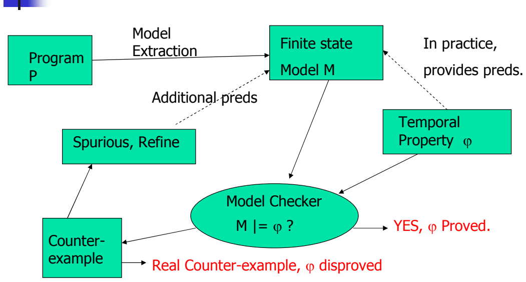
Abstraction Refinement

- Given a program P and a property f, very difficult to get the "right" abstraction which will be able to prove f (even if f is true).
- Instead start with a very coarse abstraction and model check the resultant abstract model.
- Counter-example generated may not correspond to any concrete trace of P.
  - Refine the abstract model.

Software Model Checking without Refinement



… and with Refinement



Infeasible paths

v=0;
if (v<0) then s1

v=0
s1

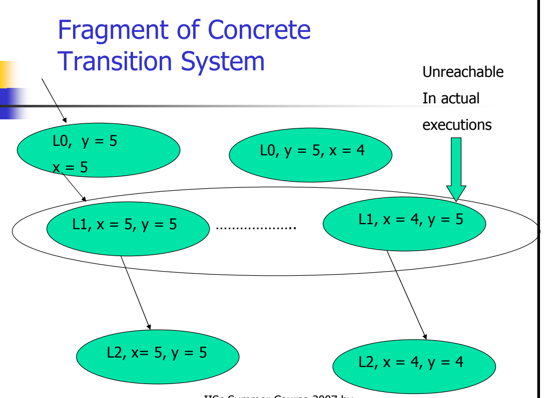
if(u<0) then S1
if (u>0) then S2.

S1
S2

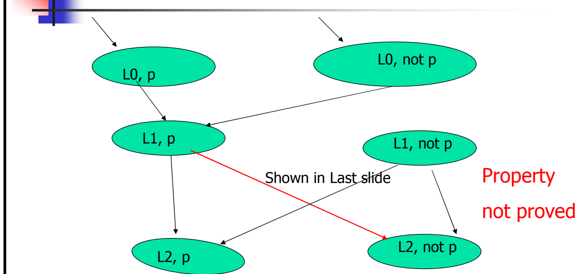
for (j:=1 to 1000) do
if j%2 == 0 then
S1 . .
else S2
endfor

An example program

- L0: x = 5
- L1: y = x
- L2
- Property G (pc = L2 ⇒ y = 5)
- Suppose we abstract with (y = 5)



Abstract Transition System



Abstract counter-example

- The following can be a counter-example trace returned by model checking
  - <L0,p>, <L1, p>, <L2, not p>
- But this does not correspond to any execution of the concrete program.
- This is a spurious counter-example
- Need to input new predicates for abstraction.

Abstraction refinement

- Generate the new predicates by analyzing the counter-example trace.
- A more informative view of the program's memory store is thus obtained.
- But how to establish a correspondence between the abstract counter-example and the concrete program ?

An Example

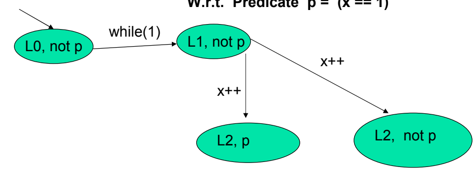
- Initially x == 0
- L0: while (1) {
- L1: x++;
- L2: while (x > 0) x - - ;
- L3 }

Property: AG(pc == L2 $\Rightarrow$ x == 1)

A locational invariant

Initial Abstraction

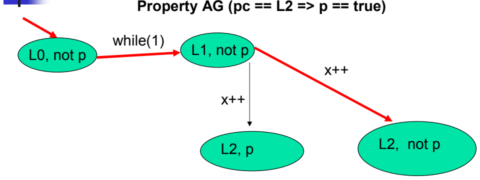
W.r.t. Predicate p = (x == 1)



No need to traverse further, counter-example trace found.

Counter-example

Property AG (pc == L2 => p == true)

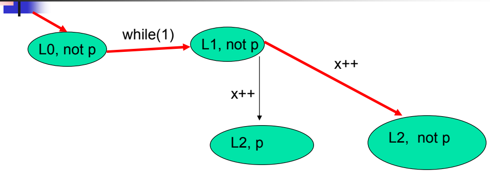


The predicate p denotes (x == 1)

Counter-example verification

- The counter-example may be spurious because our abstraction was too coarse.
  - The sequence of statements in the control-flow graph constitute an infeasible path in Control Flow Graph.
  - Not part of any concrete execution trace in the program.
- How to check whether the produced counter-example trace is spurious ?
  - Backwards or forwards exact reasoning on the counter-example trace.
  - Backwards reasoning shown now, forwards later in the lecture

Exact reasoning



(L2, x $\neq$1) $\leftarrow$ (L1, x $\neq$ 0) $\leftarrow$ (L0, x $\neq$ 0) $\leftarrow$ Initially (x $\neq$ 0 $\wedge$ x = 0)

-- the constraint to hold initially is unsatisfiable.

One step of exact reasoning

L2, x $\neq$1 $\xleftarrow{x++}$ L1, x $\neq$ 0

What is the weakest constraint on data states that should hold at L1, such that when control moves to L2 (by executing x++), the data state at L2 is guaranteed to satisfy x $\neq$ 1 ?

-- Weakest pre-condition (WP) computation.

-- We repeat the WP computation until we reach the end of the trace OR the constraint accumulated becomes unsatisfiable.

-- Corresponds to Real counter-example OR spurious counter-example.

Effect constraint of each statement $\psi(X, X')$

e.g. effect constraint of $X{+}{+}$ is

$$X' = X + 1$$

$$wp_{c(x)}(X) = \forall X' \; \psi(X, X') \Rightarrow c(X')$$

$$X' = X+1 \Rightarrow X' > 4$$

$$X > 3$$

More examples of spuriousness

w = 0;

- if (v == 0) { w++};
- if (u == 0) { w++};
- if (u == 0 && v == 0 && w < 2){error}

- Counter-example
  - A trace which executes the "error"

Constraint Propagation

TRACE	CONSTRAINT STORE
w= 0	
Assume(v \= 0)	$u = 0 \wedge v = 0 \wedge w < 1 \wedge v \neq 0$
	$u = 0 \wedge v = 0 \wedge w < 1 \wedge u = 0$
Assume(u == 0)	
w++	$u = 0 \wedge v = 0 \wedge w < 1$
Assume(u == 0 /\ v == 0 /\ w < 2)	$u = 0 \wedge v = 0 \wedge w < 2$

So, what do we know ?

- We are verifying an invariant φ against an infinite state system M.
- We abstracted (the data states of) M w.r.t. p1,…,pk to get M1
  - For every trace c1,c2,…,cn (statement sequences) in M, there is a trace c1,c2…,cn in M1 (not vice-versa)
- Model check M1 |= φ to
  - Case 1: Success. We have proved M |= φ
  - Case 2: We get a counter-example trace $\sigma 1$
    - Need to check whether $\sigma 1$ is "spurious"

What is "spurious" ?

- Each trace in M (concrete system) has a corresponding trace with same statement sequence in M1 (abstract system).
- A trace in M1 may not have a corresponding trace with same statement sequence in M.
- Does the counter-example trace $\sigma 1$ in M1 have a corresponding trace σ with same statement sequence in M ?
  - If not , then $\sigma 1$ is a spurious counter-example

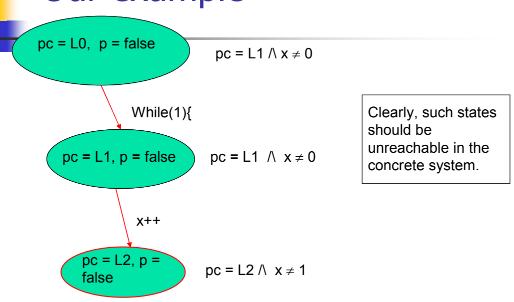
What if spurious ?

- So, we discussed how to check whether an obtained counter-example is spurious.
- If $\sigma 1$ is not spurious, then we have proved that M (concrete sys.) does not satisfy φ
- If $\sigma 1$ is spurious, we need to refine the abstraction of M
  - Original abs: Predicates p_1,…,p_k
  - New abs: Preds p_1,…,p_k, **p_(k+1),…,p_n**

But how do we …

- … compute the new preds p_(k+1),…,p_n ?
  - No satisfactory answer, active topic of research in the verification community.
  - All existing approaches are based on analysis of the spurious counter-example trace $\sigma 1$
  - Concretize the abstract states of $\sigma 1$ to get constraints on concrete data states.
  - But several ways to glean the new predicates from these constraints.
    - We will just look at some possible heuristics.

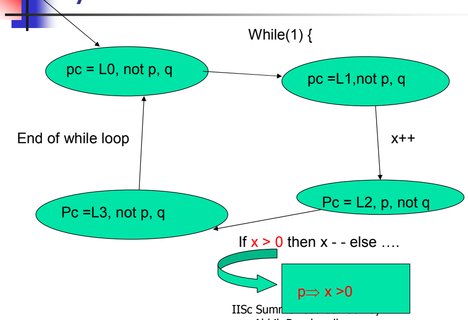
Our example



New predicates

- Based on the spurious trace, we choose another predicate q = (x = 0)
  - No clear answer why, different research papers give different heuristic 'justifications'.
- Again abstract the concrete program w.r.t. the predicates
  - p = (x = 1)
  - q = (x = 0)

New abstract transition system



Final result

- Model checking the new abstract transition system w.r.t.
  - AG(pc == L2 $\Rightarrow$ x == 1)
- … yields no counter-example trace.
- Constitutes a proof of
- M |= AG(pc == L2 $\Rightarrow$ x == 1)
- Where M is the transition system corresponding to original program.

Counter-example trace of Ex.

- assume($x \neq 0$)
- assume(true)
- x := x + 1
- assume(x = 1)

- assume represents a conditional
  - Terminates exec if condition false

Symbolic constants

- A variable must always be defined before it is used.

- $x = \theta$
- assume ($x \neq 0$)
- x := x + 1
- assume(x = 1)

- Allows us to maintain variable valuations in terms of symbolic constants. Initializations taken care of.

IISc Summer Course 2007 by
Abhik Roychoudhury

Pgm, Valuations, Constraints

$x = \theta$	(x, θ)	
assume ($x \neq 0$)	(x, θ)	$\theta \neq 0$
x := x + 1	$(x, \theta + 1)$	
assume(x = 1)	$(x, \theta + 1)$	$\theta \neq 0 \wedge \theta + 1 = 1$

Checking of infeasibility is the easy step.

But how to find the new predicates ?

IISc Summer Course 2007 by
Abhik Roychoudhury

A sufficient set of predicates

- From the valuations we get
  - $x = \theta$
  - $x = \theta + 1$
- From the constraints we get
  - $\theta \neq 0$ and its negation
- This produces
  - x = 0 (and its negation)
  - x = 1 (and its negation)

IISc Summer Course 2007 by
Abhik Roychoudhury

Sufficient set of predicates

- All. Preds. Obtained from
  - All variable valuations (in terms of symbolic constants), current or otherwise.
  - All constraints (also in terms of symb. constants)
- Sufficient to explain infeasibility of counter-example.
- Any preds. among this sufficient set which are not included in the current abstraction can be added as refinement.
- But more succinct explanations can exist, leading to parsimonious refinements …

IISc Summer Course 2007 by
Abhik Roychoudhury

Possible explanations

- assume(b> 0)
- c := 2*b
- a := b
- a := a – 1
- assume (a < b)
- assume (a = c)

- Expl 1: $b>0 \wedge c = 2b \wedge a < b \wedge a = c$
- Expl 2: Expl 1 $\wedge\ a = b \wedge a = b - 1$

IISc Summer Course 2007 by
Abhik Roychoudhury

Constructing Explanations

- Start from the end (or beginning of the trace)
  - Strongest post condition (SP), [next slide]
  - Or Weakest Pre condition (WP) [discussed]
- Perform exact reasoning at each step until you hit unsatisfiability
- Greedily remove one constraint at a time from the unsatisfiable constraint store until it becomes satisfiable
  - Is that sufficient ?

SP along a trace

assume(b> 0)	*b > 0*
c := 2*b	*b > 0, c = 2b*
a := b	*b > 0, c = 2b, a = b*
a := a – 1	*b> 0, c = 2b, a = b-1*
assume (a < b)	*b>0, c = 2b, a = b-1, a<b*
assume (a = c)	*b>0, c = 2b, a = b-1, a<b, a = c*

- Conjunction shown with comma.

Choosing predicates

- *b>0, c = 2b, a = b-1, a<b, a = c*
- Removing a = b-1 keeps constraint unsatisfiable
  - Not a good choice?
- Is it sufficient to choose predicates in this way --- predicates contributing in "unsatisfiable core".
-
- **Exercise:** Try to work out the backwards traversal and investigate choices of predicates.

Choosing predicates

a := b ;	a = b
a := a – 1;	a = b – 1
assume(a ≥ b)	a = b-1, a ≥ b

- If we choose a = b-1, a ≥ b as new refinement it may not suffice.
- The effect of a := b can only be accurately captured by the pred (a = b)
- So, we need all predicates whose **transformation** leads to one of the predicates causing unsatisfiability.

Exercise

- Try verifying absence of error in
  - a := b; a--; if (a >b) { error}
- Using the predicates
  - {a ≥ b}
  - { a ≥ b, a = b – 1}
- Feel free to use forwards or backwards counter-example analysis …

Additional: Dealing with pointers

```
int *p, *q;
void main(){
    if (*p == 3){
        *q = 2;
        if (*p == 2){
            *p = 3;
            if (*q == 2){
                ERROR
            }
        }
    }
}
```

p may or may not be aliased to q

Is the ERROR state ever reachable ?

Use pointer analysis

- Can p ever alias to q
  - Static analysis, flow insensitive.
- If yes, then need to consider both the aliased and non-aliased cases
  - Corresponding to truth of p=q which is also maintained as a predicate.
  - Infeasible constraint store has disjunction
    - $(p = q \wedge \ldots \wedge \ldots) \vee (\neg(p = q) \wedge \ldots \wedge \ldots)$
- Exercise: What are the predicates which will rule out all counterexamples of the program shown in previous slide ?

Optional Reading

- Counter-example guided Abstraction refinement (additional reading)
  - - by Edmund Clarke et. Al, CAV 2000.
  - http://www-2.cs.cmu.edu/~emc/papers.htm
  - One of the first papers to develop abstraction refinement. Try summarizing it if you are really interested in this topic.
- Regular reading appears in Lesson Plan (see Course Web-page).

Class Exercise – (1)

- Consider the program
- x = 0; x = x + 1; x = x - 1;
- if (x != 0){ error }

- Suppose we want to prove that the ``error'' location is never reached, that is, any trace reaching ``error'' is a counter-example. Show that the predicate abstraction x == 0 is insufficient to prove this property. You need to construct the abstract transition system for this purpose.

Class Exercise– (2)

- Refine your abstraction { x == 0 }
- by traversing the counter-example obtained.
- Show and explain all steps. Your refined abstraction should be sufficient to prove the unreachability of the ``error'' location – i.e. all spurious counter-examples should have been explained by the refined predicate abstraction.