Abstraction Refinement

CS 5219
Abhik Roychoudhury
National University of Singapore

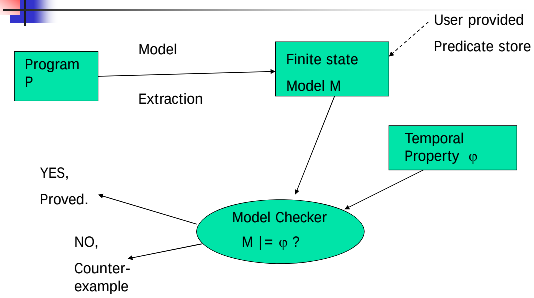
MC

- 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.

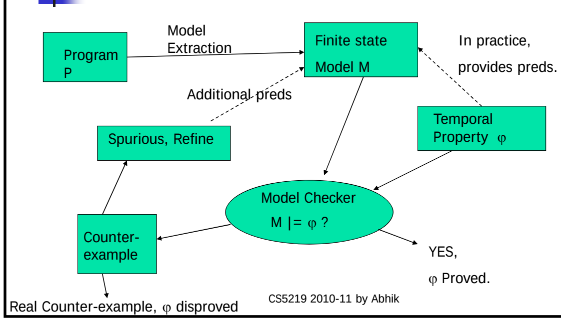
Abstraction Refinement

- 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

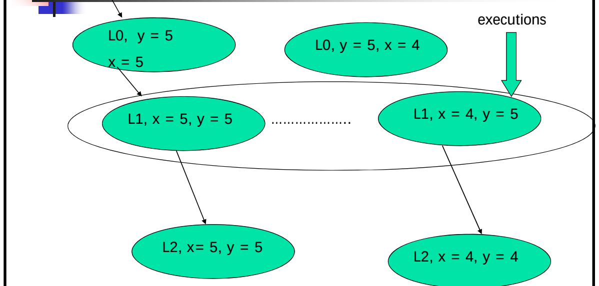
Infeasible paths

for (j:=1 to 100) do
if j%2 == 0 then
S1 . .
else S2
endfor

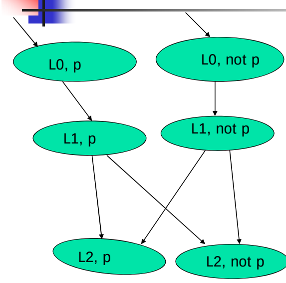
An example program

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

Fragment of Concrete Transition System



Abstract Transition System



L0: x = 5
L1: y = x
L2 if (y != 5)
Property G (pc = L2 ⇒ y = 5)

Property not proved !

p ≡ (y == 5)

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

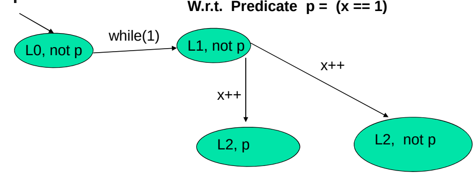
- Initially x == 0
- L0: while (1) {
- L1: x++;
- L2: while (x > 0) x - - ;
- L3 }

Property: AG(pc == L2 ⇒ x == 1)

A locational invariant

Initial Abstraction

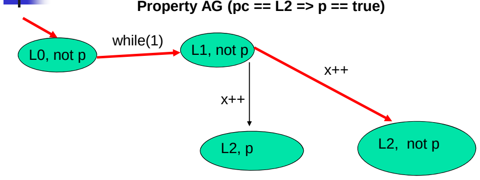
W.r.t. Predicate p = (x == 1)



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

Counter-example

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

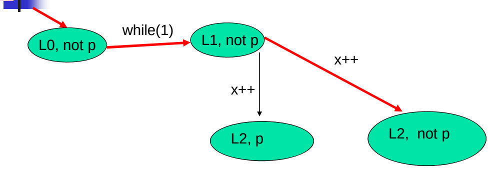


The predicate p denotes (x == 1)

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 reasoning later in the lecture.

Exact reasoning



(L2, x ≠1) ← (L1, x ≠ 0) ← (L0, x ≠ 0) ← Initially (x ≠ 0 ∧ x = 0)

-- the constraint to hold initially is unsatisfiable.

One step of exact reasoning

L2, x ≠1 ←(x++)— L1, x ≠ 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 ≠ 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

$X' = X + 1$

During backwards reasoning along the trace from the end of the trace
--- for every assignment X = e, replace X by e in the formula
--- for every branch with condition c, conjoin formula with c.

For assignment X = e, formula f becomes $f[X \rightarrow e]$
For branch with condition c, formula f becomes $f \wedge c$

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 $\models \varphi$ to
  - Case 1: Success. We have proved M $\models \varphi$
  - 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

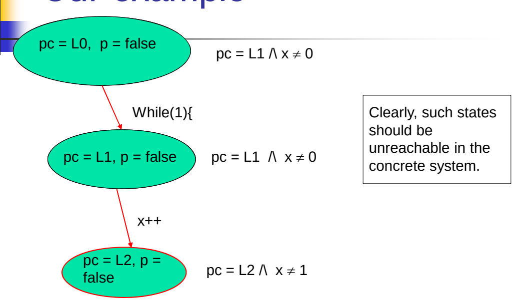
What if spurious ?

- 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, still somewhat active topic of research.
  - 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.

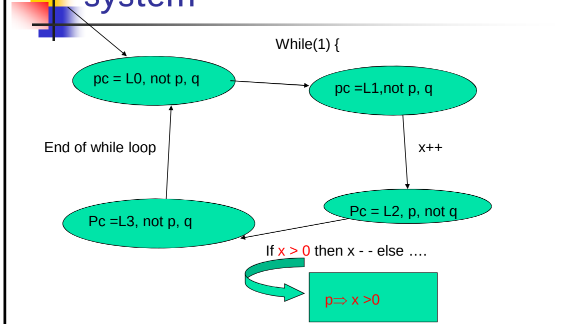
Our example



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 ⇒ x == 1)
- … yields no counter-example trace.
- Constitutes a proof of
- M |= AG(pc == L2 ⇒ x == 1)
- Where M is the transition system corresponding to original program.

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 makes the constraint satisfiable
  - Should we choose it?
- Is it sufficient to choose predicates from the formula which makes the formula uns

- **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 := a - 1; 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 /\ … /\ …) \/ (¬(p = q) /\ … /\ …)

More details

- Computation of SP
  - Forward simulation of the trace with non-concrete input values.
  - Maintain a variable valuation store as well as constraint store
    - Please check out the reading

Example

assume(b>0)		b>0
c = 2*b	<(c,2b)>	b > 0
a = b	<(c,2b),(a,b)>	b >0
a = a -1	<(c,2b),(a,b-1)>	b > 0
assume(a<b)	<(c,2b),(a,b-1)>	b > 0 /\ b-1<b
assume(c=a)	<(c,2b),(a,b-1)>	b>0 /\ b-1<b /\ 2b=b-1

Try it out – (1)

- Consider the program
- x = 0; x = x + 1; x = x + 1;
- if (x > 2){ 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 > 2 is insufficient to prove this property. You need to construct the abstract transition system for this purpose.

Try it out – (2)

- Refine your abstraction { x > 2 }
- 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.