Abstraction Refinement

CS 5219
Abhik Roychoudhury
National University of Singapore

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

User provided
Predicate store

Program $P$ $\Rightarrow$ Model $M$

Temporal Property $\varphi$

Model Checker $M \models \varphi$?

YES, $\varphi$ Proved.

NO, Counter-example

Program $P$ $\Rightarrow$ Model $M$

Additional preds

Sporadic, Refine

Model Checker $M \models \varphi$?

YES, $\varphi$ Proved.

In practice, provides preds.

Real Counter-example, $\varphi$ disproved
Infeasible paths

An example program

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

Fragment of Concrete Transition System

- L0: x = 5
- L1: y = x
- L2

Abstract Transition System

- L0: x = 5
- L1: y = x
- L2

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

Counter-example

Property \( AG( pc == L2 \Rightarrow p == true) \)

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

One step of exact reasoning

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.
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 \( c \), conjunct formula with \( c \).

For assignment \( X = e \), formula \( f \) becomes \( f[X \rightarrow e] \).
For branch with condition \( c \), formula \( f \) becomes \( f \land c \).

--- What do we know?
--- We are verifying an invariant \( \varphi \) against an infinite state system \( M \).
--- We abstracted (the data states of) \( M \) w.r.t. \( p_1, \ldots, p_k \) to get \( M_1 \).
--- For every trace \( c_1, c_2, \ldots, c_n \) (statement sequences) in \( M \), there is a trace \( c_1, c_2, \ldots, c_n \) in \( M_1 \) (not vice-versa).
--- Model check \( M_1 \models \varphi \) to
--- Case 1: Success. We have proved \( M \models \varphi \).
--- Case 2: We get a counter-example trace \( \sigma \).
--- Need to check whether \( \sigma \) is “spurious”.

--- What is “spurious”?
--- Each trace in \( M \) (concrete system) has a corresponding trace with same statement sequence in \( M_1 \) (abstract system).
--- A trace in \( M_1 \) may not have a corresponding trace with same statement sequence in \( M \).
--- Does the counter-example trace \( \sigma \) in \( M_1 \) have a corresponding trace \( \sigma \) with same statement sequence in \( M \) ?
--- If not, then \( \sigma \) is a spurious counter-example.

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

--- But how do we ...
--- ... compute the new preds \( p_{(k+1)}, \ldots, p_n \)?
--- No satisfactory answer, still somewhat active topic of research.
--- All existing approaches are based on analysis of the spurious counter-example trace \( \sigma \).
--- Concretize the abstract states of \( \sigma \) 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
--- ```
    pc = L0, p = false
    pc = L1 \land x = 0
    While(1)
    pc = L1, p = false
    x++
    pc = L2, p = false
    pc = L2 \land x = 1
    pc = L3 \land x = 0
    ```
--- Clearly, such states should be unreachable in the concrete system.
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
  - $p = (x = 1)$
  - $q = (x = 0)$

Final result

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

Constructing Explanations

- Start from the end (or beginning of the trace)
  - Strongest post condition ($\text{SP}$), [next slide]
  - Or Weakest Pre condition ($\text{WP}$) [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?

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

- \(a := b\)
- \(a := a - 1\)
- \(\text{assume}(a \geq b)\)

If we choose \(a = b - 1\), \(a \geq b\) 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 \((a \geq b)\) \{ error\}

Using the predicates
- \(\{a \geq b\} \)
- \(\{ a \geq b, a = b - 1\} \)

Feel free to use forwards or backwards counter-example analysis ...

Additional: Dealing with pointers

```c
int *p, *q;
void main()
{
  if (*p == 3){
    *q = 2;
    if (*p == 2){
      if (*q == 2){
        *p = 3;
        if (*q == 2){
          ERROR
        } Is the ERROR state ever reachable ?
      } ERROR
    } p may or may not be aliased to q
  } else if (*q == 3){
    *p = 3;
    if (*q == 3){
      ERROR
    } ERROR
  } else if (*p == 2){
    *p = 3;
  }
}
```

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
      - \((p = q) \land \ldots \land \ldots \) \lor \((\neg(p = q) \land \ldots \land \ldots \) \)

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)
c = 2*b
a = b
a = a - 1
assume(a<b)
assume(c=a)
```
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.