Software Abstractions (II)

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

Program $P$

Finite state
Model $M$

Model Checker

Temporal
Property $\varphi$

Refinement

In practice, provides preds.

Real Counter-example, $\varphi$ disproved

Impossible

\[
\begin{align*}
\text{if } (v<0) \text{ then } & s_1 \\
\text{if } (v>0) \text{ then } & s_2
\end{align*}
\]
An example program

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

Fragment of Concrete Transition System

Unreachable in actual executions

Abstract Transition System

$\neg p \equiv (y = 5)$

Abstract counter-example

- The following can be a counter-example trace returned by model checking
  - $<L0, p>, <L1, p>, <L2, \neg p>$
  - 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: \( AG( pc == L2 \implies x == 1) \)

A locational invariant

Initial Abstraction

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

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

Counter-example

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

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

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

- the constraint to hold initially is unsatisfiable.

One step of exact reasoning

\( L2, x =1 \)

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.
So, what do we know?

- We are verifying an invariant $\phi$ 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 $M_1 \models \phi$ to:
  - Case 1: Success. We have proved $M \models \phi$.
  - 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 $M_1$ (abstract system).
- A trace in $M_1$ may not have a corresponding trace with same statement sequence in $M$.
- Does the counter-example trace $\sigma_1$ in $M_1$ have a corresponding trace $\sigma$ 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 $\phi$.
- If $\sigma_1$ 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, 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

```
Pc = L0, p = false
```

```
While(1){
    x++
    pc = L2, p = false
}
```

```
Pc = L1, p = false
```

```
Pc = L2, p = false
```

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
  - \( p = (x = 1) \)
  - \( q = (x = 0) \)

New abstract transition system

```
While(1){
    x++
    pc = L1, not p, q
}
```

```
Pc = L2, p, not q
```

```
Pc = L3, not p, q
```

```
End of while loop
```

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
  - \( M |\models AG( pc == L2 \Rightarrow x == 1) \)
- Where M is the transition system corresponding to original program.

Constructing Explanations

- Start from the end (or beginning of the trace)
- Strongest post condition (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

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

- 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 is unsatisfiable?

**Exercise:** Try to work out the backwards traversal and investigate choices of predicates.

Choosing predicates

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

**Exercise:**

- Try verifying absence of error in
  - \( a := b; a := a - 1; \text{if } (a \geq b) \{ \text{error} \} \)
- **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)
        {
            *p = 3;
            if (*q == 2)
            {
                ERROR
            }
        }
    }
}
```

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

Other stuff

- Counter-example guided Abstraction refinement (additional reading)
  - [http://www-2.cs.cmu.edu/~emc/papers.htm](http://www-2.cs.cmu.edu/~emc/papers.htm)
  - One of the first papers to develop abstraction refinement. Try summarizing it if you are interested.
  - Regular reading appears in Lesson Plan.
Try it out – (1)

- Consider the program
  \[ x = 0; x = x + 1; x = x + 1; \]
  \[ \text{if} \ (x > 2) \ {\text{error}} \]

- 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.