Static Program Checking

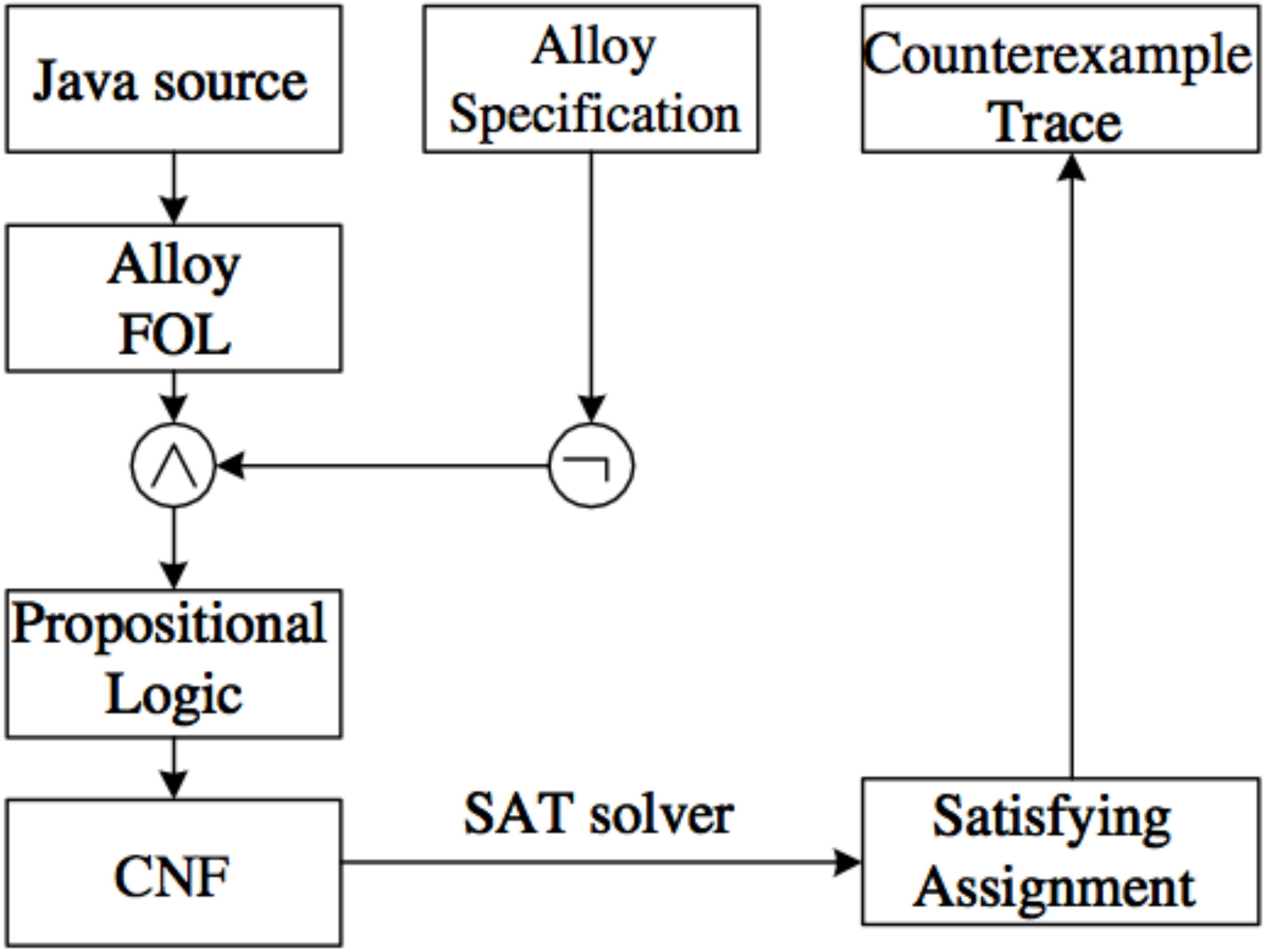
Bounded Verification – Forge

Automated Software Analysis Group, Institute of Theoretical Informatics

Juniorprof. Dr. Mana Taghdiri

Thursday – June 12, 2014

Jalloy architecture – review

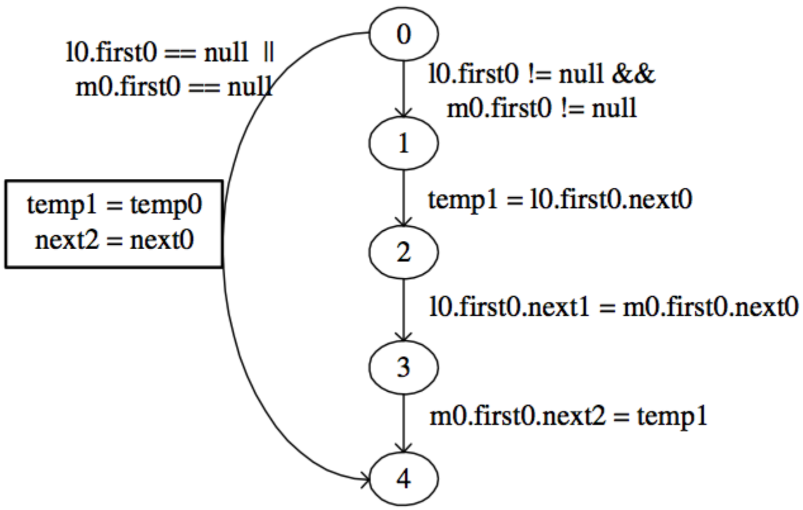


Jalloy encoding – example

Swaps the tails of the two given linked lists

```
class ListElem {
  int val;
  ListElem next;
}
class List {
  ListElem first;
  static void swapTail(List l, List m){
    if (l.first != null && m.first != null) {
      ListElem temp = l.first.next;
      l.first.next = m.first.next;
      m.first.next = temp;
    }
  }
}
```

Example – data & control flow constraints




```
E_01 || E_04 &&
E_01 => E_12 &&
E_12 => E_23 &&
E_23 => E_34
```

```
E_01 => some l0.first0 && some m0.first0
E_04 => no l0.first0 || no m0.first0
E_12 => temp1 = l0.first0.next0
E_23 => l0.first0.next1 = m0.first0.next0 &&
          all o: ListElem - l0.first0 | o.next1 = o.next0
E_34 => m0.first0.next2 = temp1 &&
          all o:ListElem-m0.first0 | o.next2 = o.next1
```

Frame codition →

Frame codition →

Forge

- A bounded verification tool following Jalloy
  - Requires a bound on the heap
  - Requires a bound on loop iterations
  - Produces sound counterexamples
- Uses kodkod rather than Alloy Analyzer
- Can handle abstract specifications
  - Requires abstraction functions to relate actual code to the abstract spec

Example – integer set implementation and spec

```
class LinkedIntSet {

/*
 * @specfield
 *  elems : set int
 *
 * @abstraction
 *  elems = (header.^next - header).element
 *
 * @invariant
 *  (header in header.^next) and
 *  (all e1, e2: header.^next - header |
 *     e1 != e2 => e1.element != e2.element)
 */

  Entry header;

  /*
   * @ensures no elems'
   * @modifies elems
   */
  void clear() {
    this.header.next = this.header;
  }
```

Approach

- P(s, s') represents the translation of code
- S(s, s') is a user-provided specification
- Find counterexamples by solving P(s, s') and not S(s, s')

- If the spec contains abstract data,
  - User should provide an abstraction function A(c, a)
    - Relates concrete and abstract states
    - Must be written for every implementation
      - But the specification is written once

- R(c) Representation invariant on concrete representation

- Solve

  R(c) and P(c, c') and A(c, a) and A(c', a') and not S(a, a')

Forge encoding

Karlsruhe Institute of Technology

- Performs a symbolic execution
  - Starts from symbolic constants
  - Collects the expressions for all variables and relations
  - Collects all loop termination conditions

- Relational view of the heap
  - Field dereference becomes relational join
    - x.f encoded as (X.F)
  - Field update becomes relational override
    - x.f = y encoded as (F++(X→Y))
    - Jalloy couldn't do that due to Alloy 3 inefficiencies

Swaptail revisited

```
static void swapTail(List l, List m){
  if (l.first != null && m.first != null) {
    ListElem temp = l.first.next;
    l.first.next = m.first.next;
    m.first.next = temp;
  }
}
```

```
pred swapTail(l, m, first, next) {
   let c = (l.first = NULL) && (m.first = NULL) |{
   let temp1 = l.first.next |{
   let next1 = next ++ l.first → m.first.next |{
   let next2 = next1 ++ m.first → temp1 |{
       c => next' = next2
       else next' = next
   }}}}
}
```

Forge encoding

- There are exactly two relations for each field:

  r (in pre-state) and r' (in post-state)

  - No intermediate relations

- The expressions are large with a lot of shared subexpressions
  - Kodkod can handle that efficiently

- Null is a proper atom
  - <A_i, null> is added to the upper bound of every relation F: A → B
  - Type of null is not important – kodkod relations and atoms are untyped

Integers in Forge

- Forge predefines following relations at the beginning of the analysis
  - a relation representing the set of all integers, size = scope(Int)
  - inc, a binary relation that totally orders the integers: for all i except the last, i.inc equals i + 1
  - add, a ternary relation mapping the two integer operands to their sum, so that the addition of i and j can be written j.(i.add)

- Inequalities:
  - i > j is encoded as (i in j.ˆ inc)

- Now we can exploit partial instances in Kodkod:
  - Pre-compute all values of add, subtract, etc.
  - Use those tuples as both the upper and lower bounds of relations

Discussion

- Hard to compare forge with jalloy
  - One uses kodkod, the other Alloy
  - Hard to tell where the performance improvement comes from
- Applied to 10 implementations of linked list
  - Max scope = 6, loops = 5, for Sun add method takes 20 minutes
  - Found 2 errors in JML specifications of add and indexOf
  - Found 1 bug in the add method of GNU Trove library (off by one error)
- Smallest scope needed to find these bugs:
  - A single loop unrolling
  - All but one required scope = 2 and integer bit-width = 3
  - One error required scope = 3 and bit-width=4
  - Supports small scope hypothesis

Jforge Experiments

```
public class LinkedList {
  class ListElem {
     int val;
     ListElem next;
   }
  ListElem first;
  public void swapTail(LinkedList l, LinkedList m) {
    if (l.first != null && m.first != null) {
      ListElem temp = l.first.next;
      l.first.next = m.first.next;
      m.first.next = temp;
    }
  }
}
```

Jforge Experiments

- Check that m.first.next in post-state equals l.first.next in pre-state

- Keywords:
  - @Ensures(“..”)
  - @Requires(“..”)
  - @Returns(“..”)
  - @old()
  - @Modifies(“..”)
  - @Invariant(“..”)

- Check that if m and l are acyclic in the pre-state, m is acyclic in the post-state

Jforge Experiments – solutions

Check that m.first.next in post-state equals l.first.next in pre-state

```
@Requires("l != null && m != null")
@Ensures("
    (l.first != null && m.first != null) => m.first.next = @old(l.first.next)")
@Modifies("l.first.next, m.first.next")
```

Remarks:

@Requires("l.first != null") states that l.first is not null in the pre-state

@Ensures("l.first != null") asserts that l.first is not null in the post-state

@Ensures("l.first = @old(m.first)") asserts that l.first in the post-state equals m.first in the pre-state

@Modifies("..") lists all the fields that *may* be modified by the method

Jforge Experiments – solutions

Karlsruhe Institute of Technology

Check that if m and l are acyclic in the pre-state, m is acyclic in the post-state

```
@Requires("l != null && m != null &&
    (all x: m.first.*next | x !in x.^next) && (all x: l.first.*next | x !in x.^next)”)
@Ensures("all x: m.first.*next | x !in x.^next")
@Modifies("l.first.next, m.first.next")
```

OR

```
@Requires("l != null && m != null”)
@Ensures(”@old((all x: m.first.*next | x !in x.^next) &&
                (all x: l.first.*next | x !in x.^next)) =>
                  (all x: m.first.*next | x !in x.^next)")
@Modifies("l.first.next, m.first.next")
```

Jforge Experiments – specfield

```
public boolean contains(int x) {
  ListElem p = this.first;
  while (p != null) {
    if (p.val == x) return true;
    p = p.next;
  }
  return false;
}
```

Write the spec once without abstract specification and once by using @SpecField("values: set int from this.first | this.values = ..")

Solution

@Ensures(“return = (x in this.first.*next.val)")

OR

@Returns(“x in this.first.*next.val”)

OR

@SpecField(

”values: set int from this.first | (this.values = this.first.*next.val)")

@Ensures(“return = (x in this.values)")