Runtime Checking for Program Verification Systems

Karen Zee, Viktor Kuncak, and Martin Rinard
MIT CSAIL

Tuesday, March 13, 2007
Workshop on Runtime Verification

Background

- Jahob program verification system
  - Statically show program corresponds to specification
  - Specification
    - Higher-order logic (HOL) using Isabelle syntax
  - Implementation
    - Sequential, memory-safe subset of Java
    - Compile (run) under standard Java compilers (runtimes)
    - Not full Java (currently not supported: exceptions, inheritance, concurrency, Java 1.5 features)
    - Supports dynamic allocation and arrays
    - Sufficient to express data structures and client programs

Data Structures Verified using Jahob

- Data Structures
  - Singly- and doubly-linked lists
  - Array list
  - Association list
  - Binary heap
  - Binary search tree
  - Hash table
- Functional and imperative implementations
- Various interfaces
  - set, relation, list, map, priority queue

- "Using First-Order Theorem Provers in the Jahob Verification System" [VMCAI07]

Motivation

- Verifying programs is difficult
- What does it mean when the prover fails?
  - Lemma is too difficult for prover
  - Error in specification and/or implementation
- Runtime checking helps find problems due to incompleteness of theorem provers
- How to check logic formulas?
- Similar to executing declarative program

Outline

- Background
- Quantifiers and Set Comprehensions
- Specification Variables (Model Fields)
- Old Expressions
- Related Work
- Conclusion

Quantifiers and Sets

- Universal quantification: ALL (x : int). x > 0 → P(x)
- Existential quantification: EX (x : int). x > 0 → P(x)
- Set comprehension: { x. x > 0 & P(x) }
- FO quantification over bounded domain

  - Not: ALL (j : int). x[j] : content
  - ALL (j : int). 0 ≤ j & j < x.length → x[j] : content

  - Not: ALL (x : obj). x.next ≠ head
  - ALL (x : obj).
    x : allocatedObjects & x : Node → x.next ≠ head

However…

ALL (x : obj).

x : allocatedObjects & x : Node → x.next ≠ head

- Do we really want to look at every Node object in the heap?
- What if we had (doubly-linked list):

ALL (x : obj) (y : obj).

x : allocatedObjects & x : Node &

y : allocatedObjects & y : Node & x.next = y → y.prev = x

Optimizations

ALL (x : obj) (y : obj).

x : allocatedObjects & x : Node &

y : allocatedObjects & y : Node & x.next = y → y.prev = x

- Notice:
  - If we know x, we know y.
  - Quantification over y is for the purposes of naming
- Conclusion:

  If we have an equality defining the quantified variable, we can avoid enumerating over the domain
- Other opportunities:

  ALL (x : obj).

  x : allocatedObjects & x : content → P(x)

Outline

- Background
- Quantifiers and Set Comprehensions
- Specification Variables (Model Fields)
- Old Expressions
- Related Work
- Conclusion

Specification variables

- Specification variables and ghost variables
- JML terminology: model fields and ghost fields
- Specification variables
  - Defined by an HOL formula
- Ghost variables
  - Updated by the programmer
  - Can have types other than standard Java types
    - Sets, tuples, sets of tuples, etc.
  - Support for infinite sets

Deferred Evaluation Example

```
//: private ghost specvar InfSet :: int set = {};
int x = 0;
//: InfSet := { y . y > 0 };
//: assert "x ~: InfSet";
x = x + 1;
//: assert "x : InfSet";
```

- Use deferred evaluation + formula simplification
- x : { y . P(y) } rewritten into P(x)
- Formula simplification can also evaluate

  x : { y . y > 0 }

Outline

- Background
- Quantifiers and Set Comprehensions
- Specification Variables (Model Fields)
- Old Expressions
- Related Work
- Conclusion

Old Expressions

- Used in postconditions and assertions
- Refer to the value of expression in pre-state
- In JML, fully evaluated in pre-state
  - Restricted syntactically
  - Illegal: (\forall int i; 0 <= i && i < 7; \old(i < y))
- Unrestricted in Jahob
  - ALL (x : obj). P(x) → x.f = (old x.f)
- More flexible, but need to track pre-state
- Recovery (recursive) cache

Recovery Cache

- Horning et al. [1974] (fault-tolerant computing)
- Stack of frames: push on entry
- On first write: record location of write + original value
- On subsequent writes: no update to cache needed
- To access pre-state:
  - Look up original value in cache, if any
  - If not in cache, then heap holds current value
- On procedure exit, merge frames
- Implications
  - No overhead on reads except of old values
  - Greatest overhead on initial write
  - Smaller overhead on subsequent writes

Extension for Labels

- JML: \old(*expr*, *label*)
- Syntactic restriction to evaluate *expr* at *label*
- Extend recovery cache mechanism for labels
  - Use global clock (counter)
  - Increment time at each label
  - Cache entries contain time of write
  - Add new entry if value in cache is older
  - To read value, find entry with same or later time
  - Merge frames by taking earliest entry in top frame

Outline

- Background
- Quantifiers and Set Comprehensions
- Specification Variables (Model Fields)
- Old Expressions
- Related Work
- Conclusion

Related Work

- High-level
  - Specifications similar to implementations
  - Specifications in logic
    - More difficult to execute
    - Easier to understand semantics, proofs
    - More expressive
- JML (tool-dependent)
  - Quantifiers: domain restricted using range predicate
  - Set comprehension: function of an existing set
  - Old expressions: evaluated in pre-state
- Spec#
  - Quantifiers and comprehension: restricted syntactically
  - Old expressions: evaluated in pre-state

Conclusion

- Runtime checker for logic formulas
  - Debugging programs and specifications
  - Loop invariant inference
- Quantifiers and set comprehensions
  - Optimizations to avoid enumeration
- Specification variables
  - Deferred evaluation of some formulas
  - Can talk about infinite sets
- Old expressions
  - Supported using recovery (recursive) cache
  - Extension to support labels
- Prototype implementation
  - Interpreter

Future Work

- Compile checks
- Modular checking using constraint solving
- Higher-order quantification

The End

HO Quantification

- Currently not supported, but...
- 120 classes with quantification, none HO
- When might someone use HO quantification?
  - Isomorphism: EX (f : obj → obj). f x = y
  - Shortest path:
    ALL (r : obj → obj). path(r) → dist(sp) ≤ dist (r)
- Why don't we see it?
  - Different types of programs

More Jahob Background

- Expected usage scenario
  - Verify, using shape analysis and theorem proving, that an implementation conforms to its specification in HOL
- Specifications
  - Specification variables (model fields)
    - HOL formula definitions
    - Ghost variables (ghost fields) updated by programmer
      `//: gv := {x . g(x)}`
  - Class invariants
  - Requires clause (precondition)
  - Ensures clause (postcondition)
  - Modifies clause (frame condition)
  - Assertions