Models for generation of verification conditions

GECCOO Final meeting

January 15th, 2007

Outline

Outline

General goal: Deductive Verification of imperative and object-oriented programs

- Goal: prove behavioral properties of Java (Card) source code
- Behaviors specified by JML annotations
- 2 platforms in GECCOO:
  - Jack (Everest)
  - Why/Krakatoa/Caduceus (ProVal)
- Common approach: Verification Conditions generated by a weakest precondition calculus
- Modeling of memory heap in first-order logic
- VCs discharged by standard first-order theorem provers (Automatic or interactive)

Evolution

- Beginning of GECCOO:
  - Jack: output for the B prover
  - Krakatoa: output for the Coq proof assistant
- End of GECCOO:
  - Jack: output for B, Simplify, haRVey, Coq
  - Jack: handles Java Bytecode, annotations in BML
  - Krakatoa: output for the Coq, PVS, Simplify, Ergo, SMT Provers (Yices, CVS Lite, haRVey, etc.)
  - Caduceus: tool for C source code, with a JML-like annotation language
  - Caduceus/Krakatoa:
    - share the common intermediate language Why
    - have similar modelings of memory heap

Models: contributions

- Jack: modeling Bytecode, BML [Pavlova, PhD, january 2007]
- Krakatoa: a specific modeling of Java/JML, in particular for `assigns` clauses [TPHOLs'05]
- Krakatoa: Java Card transactions [SEFM'06] (model+case study)
- Krakatoa case study: Demoney Applet, verification of security property (access authorization levels), based on the B modeling by N. Stouls (LSR) (one bug found)
- Abstract models:
  - Native specifications in Jack+Coq [Charles, FTfJP'06]
  - Algebraic models in Krakatoa [Preliminary work, Challenges in Java Program Verification, 2006]

Contributions (case of C language)

- New tool: Caduceus
- An original modeling of heap memory and pointer arithmetic [ICFEM'04]
- Model incorporating separation analysis [to appear]
  - application to Java in progress
- Model for floating-point programs [to appear]
  - easily applicable to Java
- Case studies:
  - Schorr-Waite graph-marking algorithm [SEFM'05],
  - Avionics embedded code provided by Dassault aviation

Outline

Algebraic Models in Krakatoa

- Underlying Why logic:
  - multi-sorted first order logic
  - one may declare sorts, logical functions, predicates, axioms.
- Idea:
  - use this logic for describing models of programs
  - $\rightsquigarrow$ algebraic specifications of models
- In C:
  - available since the beginning of Caduceus
  - used for linked lists [ICFEM'04], reachability in a graph [SEFM'05]
- In Java:
  - JML models are OO, not algebraic
  - so Krakatoa now diverges from JML: allows algebraic models

Toy example

- General-purpose class for finite sets of integers
- Basic interface :

```
class IntSet {

    // checks whether n belongs to this
   public boolean mem(int n);

    // adds n to this
   public void add(int n);
}
```

Algebraic model (1)

- Sort, functions, predicates:

```
/*@ logic type intset;
  @ logic intset emptyset();
  @ logic intset singleton(int n);
  @ logic intset union(intset s1, intset s2);
  @ predicate in(int n,intset s);
  @*/
```

Algebraic model (2)

- Axiomatization:

```
/*@ axiom in_empty :
  @  (\forall int n; !in(n,emptyset()));
  @
  @ axiom in_singleton :
  @  (\forall int n,k;
  @   in(n,singleton(k)) <==> n==k;
  @
  @ axiom in_union :
  @  (\forall int n; \forall intset s1,s2 ;
  @   in(n,union(s1,s2)) <==>
  @       in(n,s1) || in(n,s2);
  @
  @ axiom intset_ext :
  @  (\forall intset s1,s2 ;
  @   (\forall int n ; in(n,s1) <==> in(n,s2))
  @   ==> s1==s2) ;
  @*/
```

Specification of IntSet class

Introducing a model field:

```
class IntSet {
 //@ model intset my_model;
```

Methods' behaviors:

```
 //@ ensures \result <==> in(n,my_model)) ;
 public boolean mem(int n);

 /*@ modifiable my_model;
   @ model ensures
   @  my_model == union(\old(my_model),singleton(n)) ;
   @*/
 public void add(int n);
}
```

An implementation

By a sorted array

```
class IntSet {

  int size;
  int t[];

   /*@ invariant
    @    t != null &&
    @    0 <= size && size <= t.length &&
    @    (\forall int i,j;
    @       0 <= i && i <= j && j < size;
    @       t[i] <= t[j]);
    @*/

}
```

Relating models and implementation

```
/*@ predicate IntSet_models(intset s, IntSet this) {
  @    this != null &&
  @     array_models(s,this.t,0,this.size-1)
  @ }
  @
  @ predicate array_models(intset s, int t[],
  @                        int i, int j) {
  @   t != null &&
  @   0 <= i && j < t.length &&
  @   (\forall int n; in(n,s) <==>
  @      (\exists int k;
  @         i <= k && k <= j ; n==t[k]))
  @ }
  @*/

//@ invariant IntSet_models(my_model,this);
```

Demo