Formal Methods for Java

Lecture 10: Ownership and Friendship

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

November 25, 2011
A Ghost Variable for Invariants

Idea of David A. Naumann and Mike Barnett:

- Make the places where an invariant does not hold explicit.
A Ghost Variable for Invariants

Idea of David A. Naumann and Mike Barnett:

- Make the places where an invariant does not hold explicit.
- Add a ghost variable $packed$ that indicates if the invariant should hold.
- Before modifying an object set this variable to $false$.
- When modification is finished, set it to $true$.
A Ghost Variable for Invariants

Idea of David A. Naumann and Mike Barnett:

- Make the places where an invariant does not hold explicit.
- Add a ghost variable $packed$ that indicates if the invariant should hold.
- Before modifying an object set this variable to $false$.
- When modification is finished, set it to $true$.
- The following invariant should always hold:
  
  $packed ==> \text{invariants of object}$
A Ghost Variable for Invariants

Idea of David A. Naumann and Mike Barnett:

- Make the places where an invariant does not hold explicit.
- Add a ghost variable \( packed \) that indicates if the invariant should hold.
- Before modifying an object set this variable to \( \text{false} \).
- When modification is finished, set it to \( \text{true} \).
- The following invariant should always hold:
  
  \[ \text{packed} \implies \text{invariants of object} \]

- The \textit{caller} has to ensure that the objects he uses are packed.
The pack/unpack Mechanism

- An object must be unpacked before fields may be accessed.
- The invariant has to hold only while object is packed.
- The invariant may only depend on fields of the object.
Adding Ownership

- The invariant may also depend on fields of other classes.
- The class must own a class to depend on its fields.
- A class can only be unpacked and changed if the owner is unpacked.
The owner must be unpacked before an owned object can be unpacked.

The invariant of owner may depend on owned objects.
Ownership vs. Friendship

The ownership discipline has a few restrictions.
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
- An object can have at most one owner.
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
- An object can have at most one owner.
- A field may only be changed by the owner, or if the owner is unpacked.

Sometimes too restrictive!
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
- An object can have at most one owner.
- A field may only be changed by the owner, or if the owner is unpacked.

Sometimes too restrictive!

Friendship offers another way to depend on other objects:
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
- An object can have at most one owner.
- A field may only be changed by the owner, or if the owner is unpacked.

Sometimes too restrictive!

Friendship offers another way to depend on other objects:

- An invariant of a friend can depend on fields of granters.
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
- An object can have at most one owner.
- A field may only be changed by the owner, or if the owner is unpacked.

Sometimes too restrictive!

Friendship offers another way to depend on other objects:

- An invariant of a friend can depend on fields of granters.
- The friend class must define update guards for the fields it depends on.
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
- An object can have at most one owner.
- A field may only be changed by the owner, or if the owner is unpacked.

Sometimes too restrictive!

Friendship offers another way to depend on other objects:

- An invariant of a friend can depend on fields of granters.
- The friend class must define update guards for the fields it depends on.
- The granter class has a list of friends that depend on fields.
Ownership vs. Friendship

The ownership discipline has a few restrictions.

- An object invariant can only depend on fields of owned objects.
- An object can have at most one owner.
- A field may only be changed by the owner, or if the owner is unpacked.

Sometimes too restrictive!

Friendship offers another way to depend on other objects:

- An invariant of a friend can depend on fields of granters.
- The friend class must define update guards for the fields it depends on.
- The granter class has a list of friends that depend on fields.
- A field may be changed if the update guards of all friends holds.
Friendship is not symmetric. The allies are:

- **Granter** $G$ that gives rights to depend on a field.
  ```java
class G {
    int f;
    friend C reads f
  }
```

- **Friend** $C$ whose invariant depends on a field.
  ```java
class C {
    invariant packed ==> ... g.deps.has(this) && g.f == ...
    guard g.f := val by ...
  }
```

Every class that changes a field of $G$ has to check the friend’s guard.
class FriendClass {
    //@ invariant packed ==> friendInvariant(granter.field)
    //@ guard granter.field := val by updateGuardForField(granter, val);
}

The update guard must guarantee that the invariant is not invalidated:
friends.packed && friendInvariant(granter.field)
&& updateGuardForField(granter, val) ==> friendInvariant(val)
Field update on Friends

- Friend’s invariant can depend on granted fields.
- Access to granted fields is checked against update guards.
- A granter can have many friends.
- All current friends must be checked.
- The friend objects can be packed or unpacked.
Friend’s invariant can depend on granted fields.
Access to granted fields is checked against update guards.
A granter can have many friends.
All current friends must be checked.
The friend objects can be packed or unpacked.
Guard is not checked for unpacked friends.
static class Node {
    Node next, prev;
    Object value;
    //friend Node reads next,prev,deps

    //guard next.next = val by next != prev;
    //guard prev.prev = val by prev != next;

    /*@ invariant packed ==> 
       (next != null && prev != null && 
        deps.equals(new JMLObjectSet().insert(next).insert(prev)) && 
        next.deps.has(this) && next.prev == this && 
        prev.deps.has(this) && prev.next == this); 
    */
}

static class Node {
    //@requires n.prev == n.next == n;
    public void add(/*@non_null*/ Node n) {
        //@unpack n
        //@unpack this
        //@unpack this.prev
        n.prev = this.prev;
        n.next = this;
        this.prev.next = n;
        this.prev = n;
        //@set n.deps = new JMLObjectSet().insert(this).insert(this.prev);
        //@set this.deps = this.deps.remove(prev).add(n);
        //@set prev.deps = prev.deps.remove(this).add(n);
        //@pack this.prev
        //@pack this
        //@pack n
    }
}
What May Appear in an Invariant

Only the following field accesses are allowed in an invariant:

- A field of the current class:
  
  `this.field` for all fields.

- A field of a (transitively) owned class:
  
  `x.field` if `x.owner == this` can be proven.

- A field of a granter class:
  
  `x.field` if `x.deps.has(this)` can be proven.
What May Appear in an Invariant

Only the following field accesses are allowed in an invariant:

- A field of the current class:
  
  ```java
  this.field for all fields.
  ```

- A field of a (transitively) owned class:
  
  ```java
  x.field if x.owner...owner == this can be proven.
  ```
Only the following field accesses are allowed in an invariant:

- A field of the current class:
  \( \texttt{this.field} \) for all fields.

- A field of a (transitively) owned class:
  \( \texttt{x.field} \) if \( \texttt{x.owner...owner == this} \) can be proven.

- A field of a granter class:
  \( \texttt{x.field} \) if \( \texttt{x.deps.has(this)} \) can be proven.
Why Is This Sound?

We need to show the following invariant holds for each instance `this` at every time:

\[
\text{this.packed} \implies \text{this.invariant}
\]

A field access `obj.f=val` can change the truth of invariant if:

1. `obj == this` is the current class:
   
   Then `this` is unpacked, formula holds trivially.
Why Is This Sound?

We need to show the following invariant holds for each instance \texttt{this} at every time:

\[
\texttt{this.packed} \implies \texttt{this.invariant}
\]

A field access \texttt{obj.f=val} can change the truth of invariant if:

- \texttt{obj == this} is the current class:
  Then \texttt{this} is unpacked, formula holds trivially.

- \texttt{obj.owner...owner == this} (a field of an owned class):
  Then \texttt{obj} is unpacked, hence \texttt{this} must also be unpacked. The formula holds trivially.
Why Is This Sound?

We need to show the following invariant holds for each instance `this` at every time:

\[ \text{this.packed} \Rightarrow \text{this.invariant} \]

A field access `obj.f=val` can change the truth of invariant if:

- `obj == this` is the current class:
  Then `this` is unpacked, formula holds trivially.

- `obj.owner...owner == this` (a field of an owned class):
  Then `obj` is unpacked, hence `this` must also be unpacked. The formula holds trivially.

- `obj.deps.has(this)` (a field of a granter class):
  Then the update guard `this.guard(f, val)` is true. If `this.packed` is true, the invariant held before. Hence it must hold afterwards.