Specifying Subtypes in SCJ Programs

Ghaith Haddad and Gary T. Leavens
Department of Electrical Engineering and Computer Science
University of Central Florida

This work is partially supported by NSF Grant CCF-0916350
SafeJML

Design Goals

- Support SCJ
- Specification of functionality as in JML
- Specification of execution time
- Support both static verification and dynamic checking

Tool based on JastAdd and JastAddJ Java Compiler
public class Vector2d {

    protected float x, y;

    public void scale(float factor) {
        this.x *= factor; this.y *= factor;
    }
}

public class Vector3d extends Vector2d {

    protected float z;

    public void scale(float factor) {
        super.scale(factor); this.z *= factor;
    }
}
```java
public class Vector3d extends Vector2d {

    protected /*@ spec_public @*/ float z;

    /** also */
    @ public normal_behavior
    @ requires !Float.isNaN(factor);
    @ assignable z;
    @ ensures z == \old(z) * factor; @*/

    public void scale(float factor) {
        super.scale(factor); this.z *= factor;
    }
}
```
public class Vector2d {

    protected /*@ spec_public */ float x, y;

    /*@ public normal_behavior */
    @ requires !Float.isNaN(factor);
    @ assignable x, y;
    @ ensures x == \old(x) * factor
    & y == \old(y) * factor; @*/

    public void scale(float factor) {
        this.x *= factor; this.y *= factor;
    }
}

SafeJML – Vector2d - functionality
SafeJML Example – Vector3d

public class Vector3d extends Vector2d {

    protected /*@ spec_public @*/ float z;

    /** also
    @  public normal_behavior
    @   requires !Float.isNaN(factor);
    @   assignable z;
    @  ensures z == \old(z) * factor; @*/
    public void scale(float factor) {
        super.scale(factor);
        this.z *= factor;
    }
}
public class Vector2d {

    protected /*@ spec_public @*/ float x, y;

    /**
     * public normal_behavior
     * @ . . .
     * @ duration 2 * (MultiplyTime + AssignTime); @*/

    public void scale(float factor) {
        this.x *= factor; this.y *= factor;
    }
}

Problem: Subtype Polymorphism

- Subtype objects often contain more information than supertype objects
  - `Vector3d <: Vector2d`
  - `FighterJet <: Aircraft`
- Overriding methods will often need more time than the methods they override
  - `scale()`
  - `takeoffChecks()`
- How to specify methods to allow overriding in subtypes and still do timing analysis?
Solutions to the Problem?

- Use different method names for subtypes
  - don’t use overriding
  - This is equivalent to declaring all methods to be **final**

```java
public final void scale3(float factor) {
    super.scale(factor); this.z *= factor;
}
```

- Pessimistic Underspecification
  - allow maximum conceivable time for overrides

```java
public class Vector2d {
    ...
    @ duration MAXDimension * (MultiplyTime + AssignTime);
    @*/
```
A better Solution: Supertype Abstraction

- Modular reasoning with subtype polymorphism
- Idea: Use specifications of static types in reasoning
- Example
  - To verify
    \[
    \{P\} \ o.\ m(); \ \{Q\}
    \]
  - Use the specification of \texttt{m} associated with the static type of \texttt{o}
- Soundness = Behavioral Subtyping
  - Types must be behavioral subtypes of their supertypes
  - I.e., all overriding methods must obey the specification of the method they override
Parkinson’s Abstract Predicates

- Parkinson uses predicate families that depend on the dynamic receiver’s types.
- In Vector2d
  - Instead of
    
    `duration 2*(MultiplyTime+AssignTime);`
  
  use

    `duration scaleTime();`

  - `scaleTime()` is a pure model method in SafeJML
  - Override `scaleTime()` in each concrete type
public class Vector2d { /* ... */
    /*@ ... @*/
    @  duration scaleTime() @*/
    public void scale(float factor) { /* ... */ }

    /*@*/
    public pure model long scaleTime() { 
        return this.getDimensions() *
            *(MultiplyTime+AssignTime);
    }

    ensures \result >= 2;
    public pure model int getDimensions() { return 2; }
    @*/
public class Vector3d extends Vector2d {
    ...
    // specification inherited
    public void scale(float factor) {
        ...
    }
    /*@
    public pure model int getDimensions() {
        return 3;
    }
    @*/
Related Work

- Parkinson *et al.*
  - Introduced the concept of abstract predicate families to modular reasoning of specifications

- Krone *et al.*
  - *duration* clause for timing constraints, adopted by JML
  - Supports modular verification of performance constraints

- PERC Pico product from Atego
  - Verifies space specifications of a predefined set of subclasses

- Schoeberl and Pedersen
  -describe a precise WCET for Java Systems based on the Java Optimized Processor (JOP)
Future Work

- Complete implementation of the tool
  - Proof of concept can be found at http://tinyurl.com/28zllux

- Evaluation and refinement of design
  - Case studies

- Linking duration specifications to platform
  - Through model variables?

Questions?
- Ghaith Haddad – haddad@ieee.org
- Skype ID: ghaith_on_skype
Backup Slides