The Design of SafeJML,

a Specification Language for SCJ with Support for Timing Constraints

Ghaith Haddad, Faraz Hussain and **Gary T. Leavens**

School of Electrical Engineering and Computer Science

University of Central Florida

This work is partially supported by NSF Grant CCF-0916350

Goals of SafeJML's design

- Support SCJ (+ C code for drivers)
  - Working with the oSCJ team from Purdue
- Specification of timing constraints for methods, etc.
  - Modular division of timing budget
  - Isolation of code causing timing problems
- Support both static verification and dynamic checking
  - aiT for static verification(WCET)
  - RapiTime for detecting violations dynamically

Basic Decisions

- Use JML style annotation comments,

  ```
  //@ duration 10 * MICROSEC;
  ```

  not Java annotations

  ```
  @Duration("10 * MICROSEC")
  ```

- Allow specifier to communicate with analysis tools (RapiTime and aiT) with new JML syntax

Duration Clauses for Methods

duration-clause ::= **duration** *spec-expression* ;

```
/*@  public behavior
  @     requires position.x >= 0.0f && position.y >= 0.0f;
  @     duration 3 * MILLISEC;
  @ also
  @   public behavior
  @     requires position.x < 0.0f ^ position.y < 0.0f;
  @     duration 4 * MILLISEC;
  @ also
  @   public behavior
  @     requires position.x < 0.0f && position.y < 0.0f;
  @     duration 5 * MILLISEC;
  @*/
protected void voxelHash(Vector3d position, Vector2d voxel)
```

(Duration) Annotations on Individual Statements

refining-statement ::= **refining** *spec-statement statement*
| **refining** *generic-spec-statement-case statement*
generic-spec-statement-case ::= ... | *simple-spec-statement-body*
simple-spec-statement-body ::=
*simple-spec-statement-clause simple-spec-statement-clause**

```
//@ refining
//@    duration 3 * MILLISEC;
{ m(); }
```

Problem: Subtype Polymorphism

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

Approaches to Subtype Polymorphism

- Use different method names for subtypes
  - don't use overriding
- Underspecification
  - allow maximum conceivable time for method
- Abstract Predicate Families
  - time depends on dynamic type

Assumptions to give type bounds

- To facilitate abstract predicate families, **assume** statements can give type bounds

```
assume SafeJML.type_bound(S,E,T);
```

Example

```
//@ assume SafeJML.type_bound(Vector3d, vo, Vector2d);
```

Communicating with Tools

- Features to pass information to RapiTime (or aiT)
  - When to use splitting (context-sensitive analysis) for a method
  - maximum loop iterations
  - maximum executions of a conditionally guarded block per loop execution

Violation Reporting

Timing contracts

- *duration-clause*,
  notify user after program finishes

Tool communication features

- *max-loop-iter-stmt*, *local-worst-case-stmt*:
  throw `JMLAssertionError` when detected

Implementation and Evaluation

- Implementation
  - Built on the JAJML compiler, a JML implementation based on JastAdd and JastAddJ Java Compiler
- Evaluation
  - MiniCDj, a SCJ rewrite of the CDx benchmark suite
  - More evaluation needed!

See http://tinyurl.com/28zllux

Related Work

- Krone *et al.*
  - **duration** clause for timing constraints, adopted by JML
  - Supports modular verification of performance constraints
- RapiTime
  - Hybrid dynamic analysis of execution times
  - No specification of the times allowed.
- AbsInt's aiT
  - Static analysis for WCET times
  - Uses annotation files and binaries generated from C or Ada compilers

Future Work

- Evalution and refinement of design
  - Case studies

Thank you…

Questions?