JML Framed!

PASTE'04

June 7, 2004

Gary T. Leavens

Iowa State University

JML is joint work with Clyde Ruby, Yoonsik Cheon, Curtis Clifton, Rustan Leino, Bart Jacobs, Erik Poll, David Cok, Patrice Chalin, Steve Edwards, Michael Ernst, Peter Müller, ...

Funding from the US National Science Foundation

jmlspecs.org

Outline

- Overview of JML
  - *Background and goals*
  - JML's key advances in language design
- Framing in JML
- Conclusions

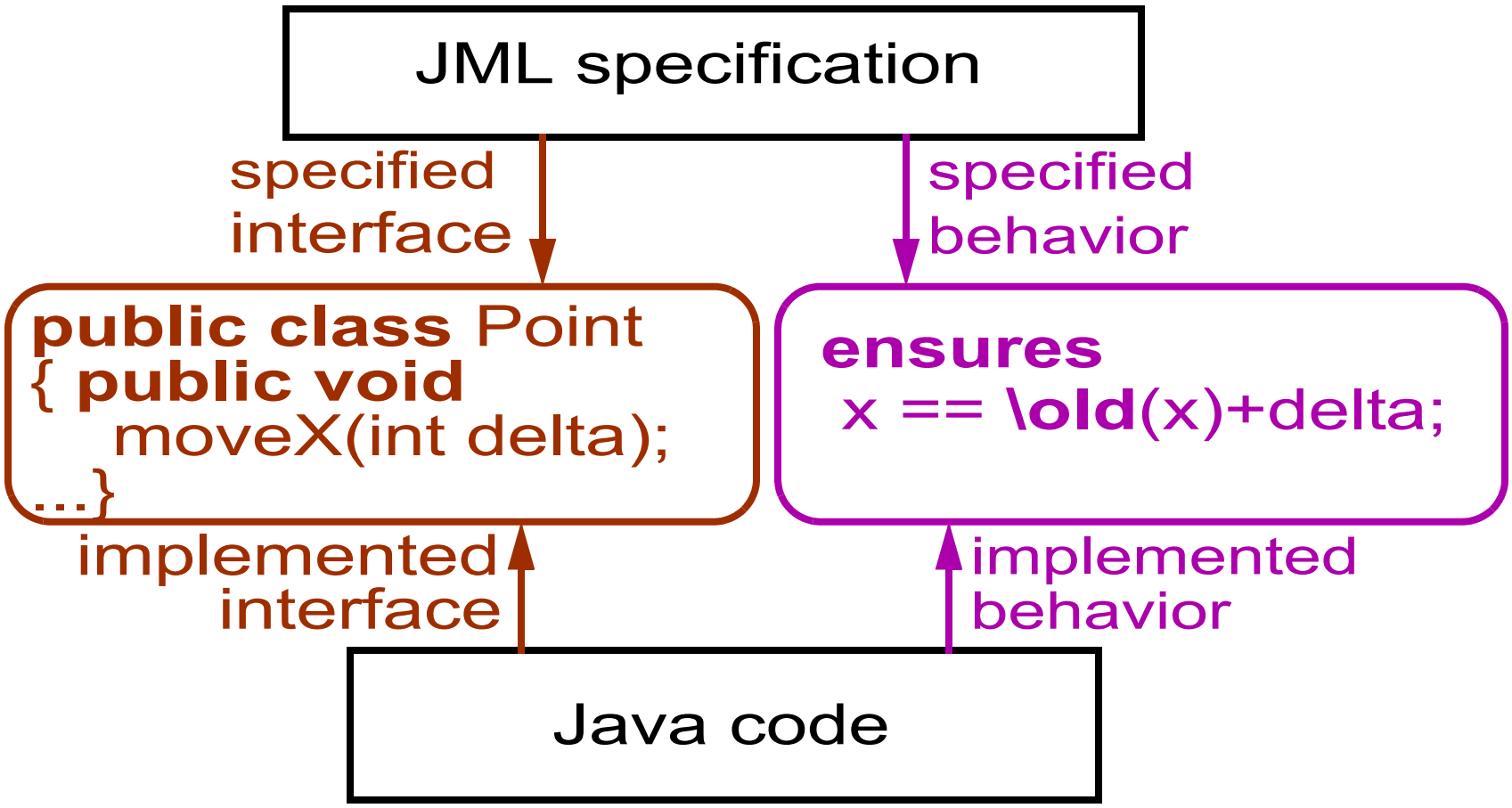
The Java Modeling Language (JML)

Behavioral interface specification language, tailored to Java

- Records detailed designs
- Pre- and postconditions, invariants, ...
- Synthesizes ideas in sequential specification

Used in Java Smart Card industry, Fulcrum,...

Behavioral Interface Specification



Like: Eiffel, Larch

Unlike: Z, VDM-SL, OCL

Example

```
public class Point {

   private /*@ spec_public @*/ int x, y;

   //@ requires delta > 0;
   //@ assignable x;
   //@ ensures x == \old(x + delta);
   public void moveRight(int delta) { x += delta; }

   //@ assignable \nothing;
   //@ ensures \result == x;
   public /*@ pure @*/ int getX() { return x; }

   // ...
}
```

JML is an Open Cooperative Project

- Leavens's group at Iowa State, Cheon at UTEP
- Hatcliff at Kansas State Univ. (Dwyer in Nebraska)
- Jacob's LOOP group at U. Nijmegen (Poll, Kiniry,...)
- Ernst's Daikon group, and others at MIT
- Chalin's group at Concordia, Montreal
- David Cok
- Edwards's group at Virginia Tech.
- Müller's group at ETH Zürich
- Poetzsch-Heffter's group at Kaiserslautern
- Logical group at INRIA Futurs & Université Paris-Sud
- Huisman's group at INRIA Sophia-Antipolis

Why Cooperate?

Common syntax and semantics

- Users benefit from multiple tools
- Tool builders share customers
- Helps exchange of ideas

The Overall Research Problem

Formal specification language good at:

- Documenting detailed designs, existing code
- Supporting a range of tools:
  - Documentation generators
  - Runtime assertion checking, unit testing
  - Static analysis, verification

Also

- Readable, Expressive, Modular

Approach

- Focus on sequential subset of Java
- For readability:
  - Java expressions for assertions (à la Eiffel)
  - Hide math behind class library
- For expressiveness:
  - Interface specifications (à la Larch)
  - Model-based (à la VDM, Larch)
  - Synthesize other ideas
- For use by verification tools
  - Statically check purity in assertions

Outline

- Overview of JML
  - Background and goals
  - *JML's key advances in language design*
- Framing in JML
- Conclusions

Key to Range of Tools: Purity Checking

If assertions might have side effects

- could cause hard-to-find errors
- mathematical models must be complex

Prohibit side-effects in assertions:

- No side-effecting expressions (e.g., ++)
- Assertions can only call "pure" methods

```
public /*@ pure @*/ int getX() { return x; }
```

- Pure methods may not:
  - assign (non-local) existing locations
  - perform I/O

Semantics

A method annotated as “pure” must satisfy:

```
assignable \nothing;
```

Purity inherited by method overrides.

Questions

Enforcement too conservative?

- Called methods must be pure.

Can synchronized methods be pure?

Purity annotations for Java’s libraries?

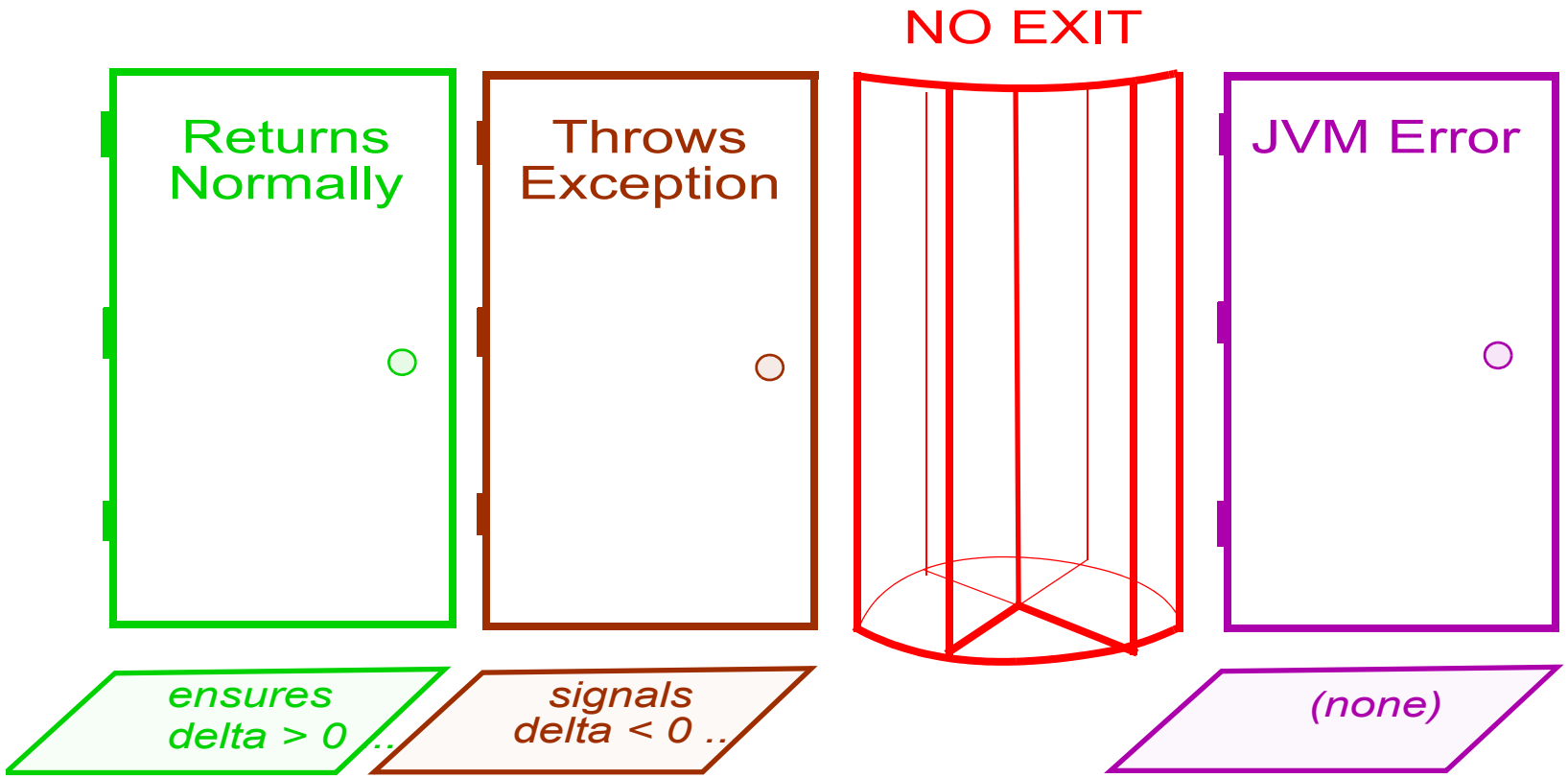
Specification of Exceptional Behavior

```
public class Point {

   private /*@ spec_public @*/ int x, y;

   /*@ requires x >= 0;
     @ assignable y;
     @ ensures delta > 0 && y == \old(y + delta)
     @         && \result == \old(y);
     @ signals (IllegalArgumentException e)
     @         delta < 0 && e != null && y == \old(y);
     @*/
   public int moveUp(int delta);
   // ...
}
```

Four “Outcomes”



No obligation if:

- JVM encounters an error [PH97]
- Precondition not satisfiable in pre-state

Case Analysis ("also"), Defaults

```
/*@  public normal_behavior
  @    requires x >= 0 && delta > 0;
  @    assignable y;
  @    ensures y == \old(y + delta)
  @          && \result == \old(y);
  @also
  @  public exceptional_behavior
  @    requires x >= 0 && delta < 0;
  @    assignable \nothing;
  @    signals (IllegalArgumentException e)
  @              e != null;
  @*/
public int moveUp(int delta);
```

Desugaring of Defaults

```
/*@  public behavior
  @    requires x >= 0 && delta > 0;
  @    assignable y;
  @    ensures y == \old(y + delta)
  @          && \result == \old(y);
  @    signals (Exception e) false;
  @also
  @  public behavior
  @    requires x >= 0 && delta < 0;
  @    assignable \nothing;
  @    ensures false;
  @    signals (IllegalArgumentException e)
  @                  e != null;
  @*/
public int moveUp(int delta);
```

Standardization of Frames

```
/*@  public behavior
  @    requires x >= 0 && delta > 0;
  @    assignable y;
  @    ensures y == \old(y + delta)
  @          && \result == \old(y);
  @    signals (Exception e) false;
  @also
  @  public behavior
  @    requires x >= 0 && delta < 0;
  @    assignable y;
  @    ensures false;
  @    signals (IllegalArgumentException e)
  @                  e != null && y == \old(y);
  @*/
public int moveUp(int delta);
```

Desugaring of “also” [Win83, Wil94]

```
/*@  public behavior
  @    requires (x >= 0 && delta > 0)
  @                || (x >= 0 && delta < 0);
  @    assignable y;
  @    ensures \old(x >= 0 && delta > 0)
  @                ==> (y == \old(y + delta)
  @                        && \result == \old(y));
  @    signals (IllegalArgumentException e)
  @       \old(x >= 0 && delta < 0)
  @        ==> e != null && y == \old(y);
  @*/
public int moveUp(int delta);
```

Specification Inheritance [DL96]

```
public class Super {
   /*@ public behavior B1 @*/ public T m();
      // ...
}

public class Sub extends Super {
   /*@ also public behavior B2 @*/ public T m();
      // ...
}
```

Completed Specification of Subclass

```
public class Sub extends Super {
   /*@        public behavior B1
     @ also public behavior B2 @*/ public T m();
      // ...
}
```

Less Restrictive than Eiffel

If superclass precondition doesn't apply, then its postcondition also doesn't apply.

```
public class Unchecked {
   /*@ public normal_behavior
     @    requires x > 0;
     @       ensures \result > 5; @*/
   public int f(int x);
      // ...
}

public class Checked extends Unchecked {
   /*@ also
     @    public exceptional_behavior
     @       requires x <= 0;
     @       signals (IllegalArgumentException e)
     @                  true;  @*/
   public int f(int x);
}
```

What Gets Inherited

- Specifications of instance methods (with "**also**")
- Fields and associated specifications
- Instance invariants and constraints.

Forces Behavioral Subtyping [DL96]

- Subtypes are behavioral subtypes.
- If try to avoid it, get unimplementable specification.
- Thus, need to underspecify supertypes

Other Features

- Model fields allow more complete specification of collections
- Quantifiers in expressions
- Redundancy: **implies_that**, **for_example**, ...

Eiffel Features not in JML

- Labels for assertions
- Specifications are statements in Eiffel

Outline

- Overview of JML
- Framing in JML
  (with Müller and Poetzsch-Heffter)
  - *Problems*
  - Approach
  - Related work
- Conclusions

Goal

Clearly specify what locations a method:

- may assign, and
- *cannot assign*

in a way that preserves information hiding.

Information Hiding Problem [Lei95, Lei98]

```
public abstract class LinkedList {

   protected Node first, last;

   /*@ public normal_behavior
     @     assignable first; @*/
   public void initializeFirst() {
      first = null;
   }

   /* ... */

}
```

What does “first” mean to a client?

Solution: Model Fields

```
//@ model import org.jmlspecs.models.*;
public class LinkedList {
   //@ public model JMLObjectSequence listValue;

   protected Node first, last;



   /*@ public normal_behavior
     @   assignable listValue;
     @   ensures listValue != null && listValue.isEmpty();
     @*/
   public void initializeFirst() {
      first = null;
   }
   /* ... */
}
```

Represents Clauses Connect Concrete and Abstract Fields [Hoa72]

```
//@ model import org.jmlspecs.models.*;
public class LinkedList {
   //@ public model JMLObjectSequence listValue;

   protected Node first, last;

   /*@ protected represents listValue <-
     @       (first==null ? new JMLObjectSequence()
     @                    : first.values); @*/

   /*@ public normal_behavior
     @   assignable listValue;
     @   ensures listValue != null && listValue.isEmpty();
     @*/
   public void initializeFirst() {
      first = null;
   }
   /* ... */
}
```

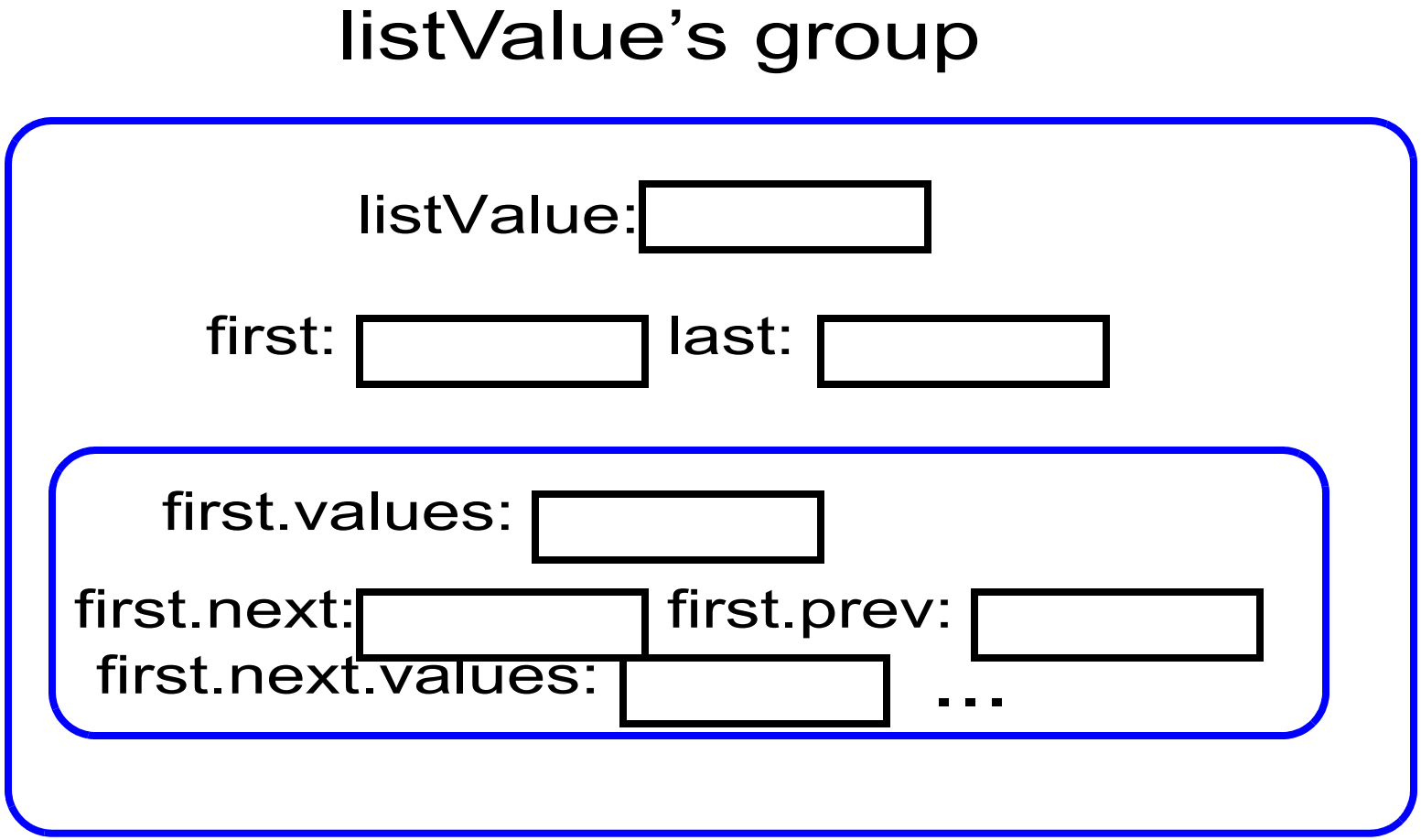
Modification of Concrete Locations

How can initializeFirst assign to first?

```
/*@ public normal_behavior
  @   assignable listValue;
  @   ensures listValue != null && listValue.isEmpty();
  @*/
public void initializeFirst() {
   first = null;
}
```

Solution Idea: Data Groups [Lei98]

A set of locations



associated to each model field

Data Group Declarations, Maps Clauses

```
//@ model import org.jmlspecs.models.*;
public class LinkedList {
   //@ public model JMLObjectSequence listValue;

   protected Node first, last; //@ in listValue;
   //@                 maps first.values \into listValue;

   /*@ protected represents listValue <-
     @       (first==null ? new JMLObjectSequence()
     @                    : first.values); @*/

   /*@ public normal_behavior
     @   assignable listValue;
     @   ensures listValue != null && listValue.isEmpty();
     @*/
   public void initializeFirst() {
      first = null;
   }
   /* ... */
}
```

```
//@ model import org.jmlspecs.models.*;
public class Node {
   //@public model non_null JMLObjectSequence values;

   public Node next, prev; //@ in values;
     //@                  maps next.values \into values;

   public /*@ readonly @*/ Object  val; //@ in values;

   /*@ protected represents values <-
     @       (next==null ? new JMLObjectSequence(val)
     @                   : next.values.insertFront(val)); @*/

   /* ... */
}
```

Data Groups and Frame Axioms

Better semantics: a method can assign to all locations in data groups mentioned in its assignable clause.

Modification of Extended State

```
public abstract class LengthCachedList
        extends LinkedList
 {

   protected int len;

   public void initializeFirst() {
      first = null;
      len = 0;        // illegal!
   }
   /* ... */
}
```

Data Groups also fix this Problem

```
public class LengthCachedList extends LinkedList {

   //@ public model int length; in listValue;

   protected int len; //@ in length;
   //@ protected represents length <- len;

   /*@ also
     @   assignable listValue;
     @   assignable_redundantly length;
     @   ensures length == 0;
     @*/
   public void initializeFirst() {
      first = null;
      len = 0;
      //@ assert length == 0;
   }
   /* ... */
}
```

Modularity (Layering) Problem [Mül02]

```
//@ model import org.jmlspecs.models.*;
public class Set {

   //@ public model non_null JMLObjectSet setValue;

   protected /*@ non_null @*/  LinkedList theList;
    //@ in setValue; maps theList.listValue \into setValue;
    /*@ protected represents setValue \such_that
      @       (\forall Object o; o != null;
      @            theList.listValue.has(o)
      @             <==> setValue.has(o));   @*/

   /*@ public normal_behavior
     @   assignable setValue;
     @   ensures setValue.isEmpty();
     @*/
   public void emptyOut() {
      theList.initializeFirst();
   }

   /* ... */
}
```

Modularity Problem

Set's call to List.initializeFirst:

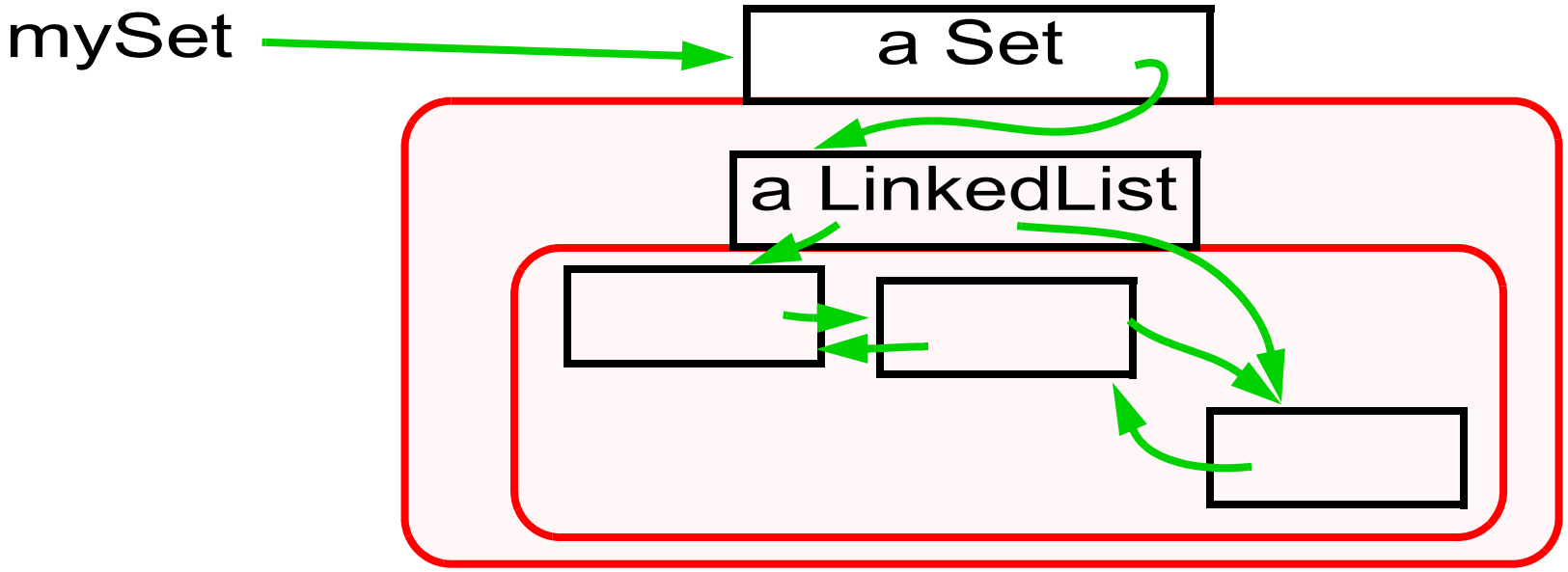
- changes theList.listValue,
- hence changes setValue.

So initializeFirst changes a location that it can't change!

```
/*@ public normal_behavior
  @   assignable listValue;
  @   ensures listValue != null && listValue.isEmpty();
  @*/
public void initializeFirst() {
   first = null;
}
```

Approach to Modular Framing [Mül02]

- Use an alias-controlling type system; gives a hierarchy of *contexts,* each with an *owner*



- Underspecify frames for modularity:
  - Only "relevant" locations are in receiver's context
- Use semantic restrictions for soundness

Using the Universe Type System

```
//@ model import org.jmlspecs.models.*;
public class Set {

   //@ public model non_null JMLObjectSet setValue;

   protected /*@ rep non_null @*/  LinkedList theList;
    //@ in setValue; maps theList.listValue \into setValue;
    /*@ protected represents setValue \such_that
      @       (\forall Object o; o != null;
      @            theList.listValue.has(o)
      @             <==> setValue.has(o));   @*/

   /*@ public normal_behavior
     @   assignable setValue;
     @   ensures setValue.isEmpty();
     @*/
   public void emptyOut() {
      theList.initializeFirst();
   }

   /* ... */
}
```

Modularity for Data Group Information

Data group membership declared when fields are declared.

So new code:

- can't change membership of existing field
- knows membership for each field it sees

Modularity for Verification for Frames

Verification for frames is divided:

- Method implementer verifies frame only for relevant locations
- Method caller uses:
  - the called method's frame
  - the data groups and represents clauses

  to accumulate the effect of the call.

```
/*@ public normal_behavior
  @   assignable setValue;
  @   ensures setValue.isEmpty();
  @*/
public void emptyOut() {
   theList.initializeFirst();
}
```

Current work [MPHL03]: Extension to Invariants

Problem: what invariants must hold when?

Approach: use ownership contexts...

- Invariants can only depend on owned locations
- Verification is only concerned with invariants of relevant objects

Related Work

- VDM-SL [And97]
  - has reads and writes clauses
  - doesn't solve info hiding or extended state
- Leino and Nelson [LN02]
  - More complex language
  - Scope dependent semantics: hard to prove sound
- Spec# (Boogie)
  - Better ownership transfer
  - Directly handles reentrance
  - More dynamic, so more expressive
  - But harder for programmers to use?

Future Work on Framing

- Checking more of JML's semantics statically
- Implementing the Universe type system (ongoing by Müller)
- Evaluating the design in practice
- Fixing the Universe type system's problems
- Blending the Spec# approach?

Framing Conclusions

- Alias control or ownership seems to help:
  - Prevent rep exposure
  - Prevent arg exposure
    (what invariant dependencies are allowed?)
  - Allows modular verification of frames
- Data groups easy to work with
  - Easier than explicit dependencies
  - Automatically guarantee crucial properties

JML Conclusions

- JML supports both run time checking and static verification
- JML combines Eiffel, Larch
  - Eiffel-style syntax: readability
  - Larch-style semantics: well-defined
- More expressive than Eiffel-style
- More practical than Larch-style
- Hiding model types behind Java classes
- We're open to new collaborators

See jmlspecs.org

References

[And97] Derek Andrews. *A Theory and Practice of Program Development*. FACIT. Springer-Verlag, London, UK, 1997.

[DL96] Krishna Kishore Dhara and Gary T. Leavens. Forcing behavioral subtyping through specification inheritance. In *Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany*, pages 258–267. IEEE Computer Society Press, March 1996. A corrected version is Iowa State University, Dept. of Computer Science TR #95-20c.

[Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281, 1972.

[Lei95] K. Rustan M. Leino. *Toward Reliable Modular Programs*. PhD thesis, California Institute of Technology, 1995. Available as Technical Report Caltech-CS-TR-95-03.

[Lei98] K. Rustan M. Leino. Data groups: Specifying the modification of extended state. In *OOPSLA '98 Conference Proceedings*, volume 33(10) of *ACM SIGPLAN Notices*, pages 144–153. ACM, October 1998.

[LH94] K. Lano and H. Haughton, editors. *Object-Oriented Specification Case Studies*. The Object-Oriented Series. Prentice Hall, New York, NY, 1994.

[LN02] K. Rustan M. Leino and Greg Nelson. Data abstraction and information hiding. *ACM Transactions on Programming Languages and Systems*, 24(5):491–553, September 2002.

Peter Müller, Arnd Poetzsch-Heffter, and Gary T. Leavens. Modular invariants for object structures. Technical Report 424, ETH Zurich, October 2003.

[Mül02] Peter Müller. *Modular Specification and Verification of Object-Oriented Programs*, volume 2262 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002. The author's Ph.D. Thesis. Available from rlhttp://www.informatik.fernuni-hagen.de/import/pi5/publications.html.

[PH97] Arnd Poetzsch-Heffter. Specification and verification of object-oriented programs. Habilitation thesis, Technical University of Munich, January 1997.

[Wil94] Alan Wills. Refinement in Fresco. In Lano and Houghton [LH94], chapter 9, pages 184–201.

[Win83] Jeannette Marie Wing. A two-tiered approach to specifying programs. Technical Report TR-299, Massachusetts Institute of Technology, Laboratory for Computer Science, 1983.

Need to cite [LH94] for bibtex