Support for Supertype Abstraction in JML

Gary T. Leavens
University of Central Florida
Support from US National Science Foundation
HCSS, March 6, 2008

jmlspecs.org www.eecs.ucf.edu/~leavens

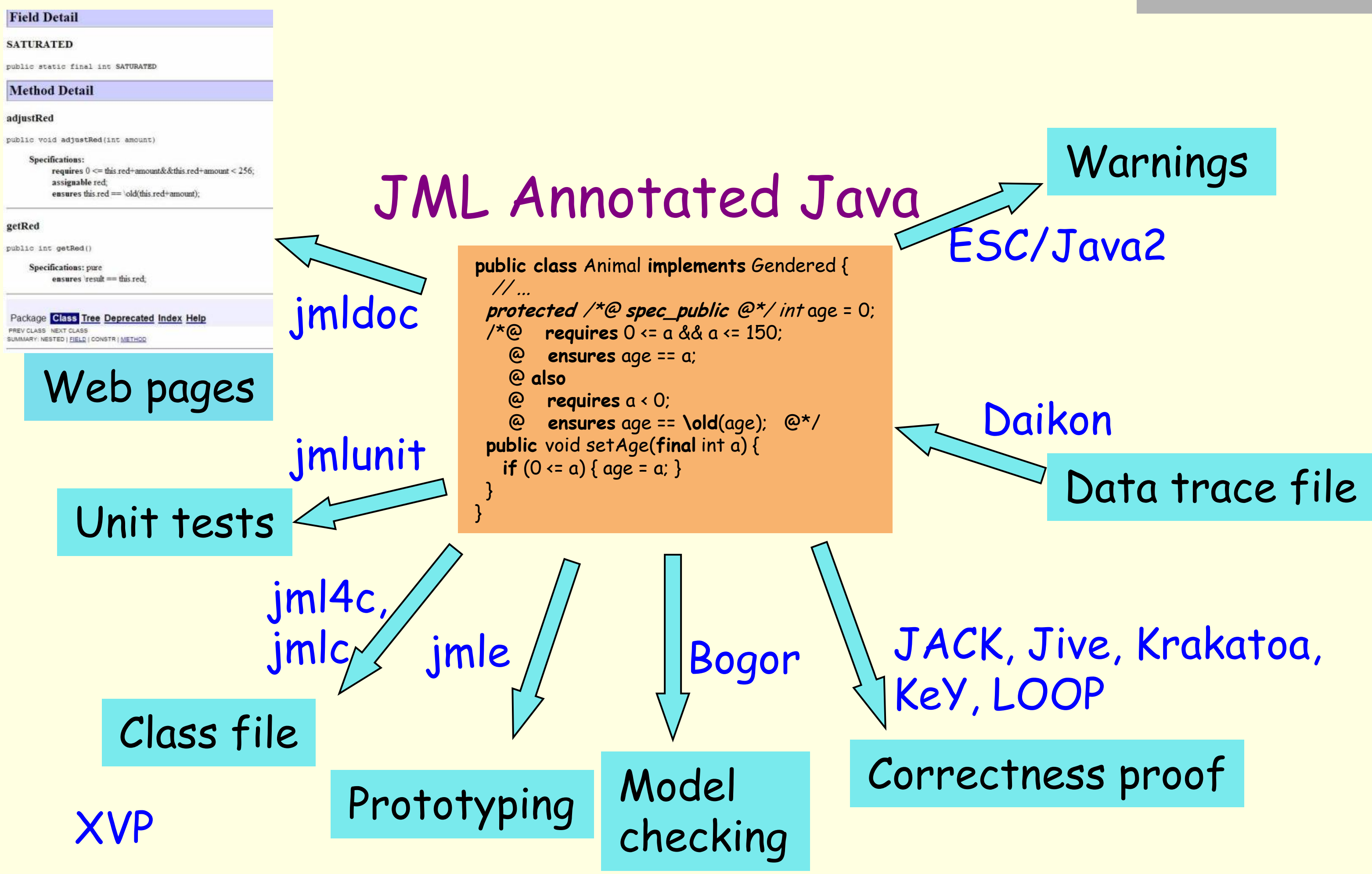
Outline

- JML Background
- Modular reasoning with supertype abstraction

Java Modeling Language—JML

- Formal specification language for Java
  - Functional behavior
  - Sequential
- Goals:
  - Practical, effective for detailed designs
  - Existing code
  - Wide range of tools
- Hoare-style (Contracts)
  - Method pre- and postconditions
  - Type invariants

Many Tools, One Language



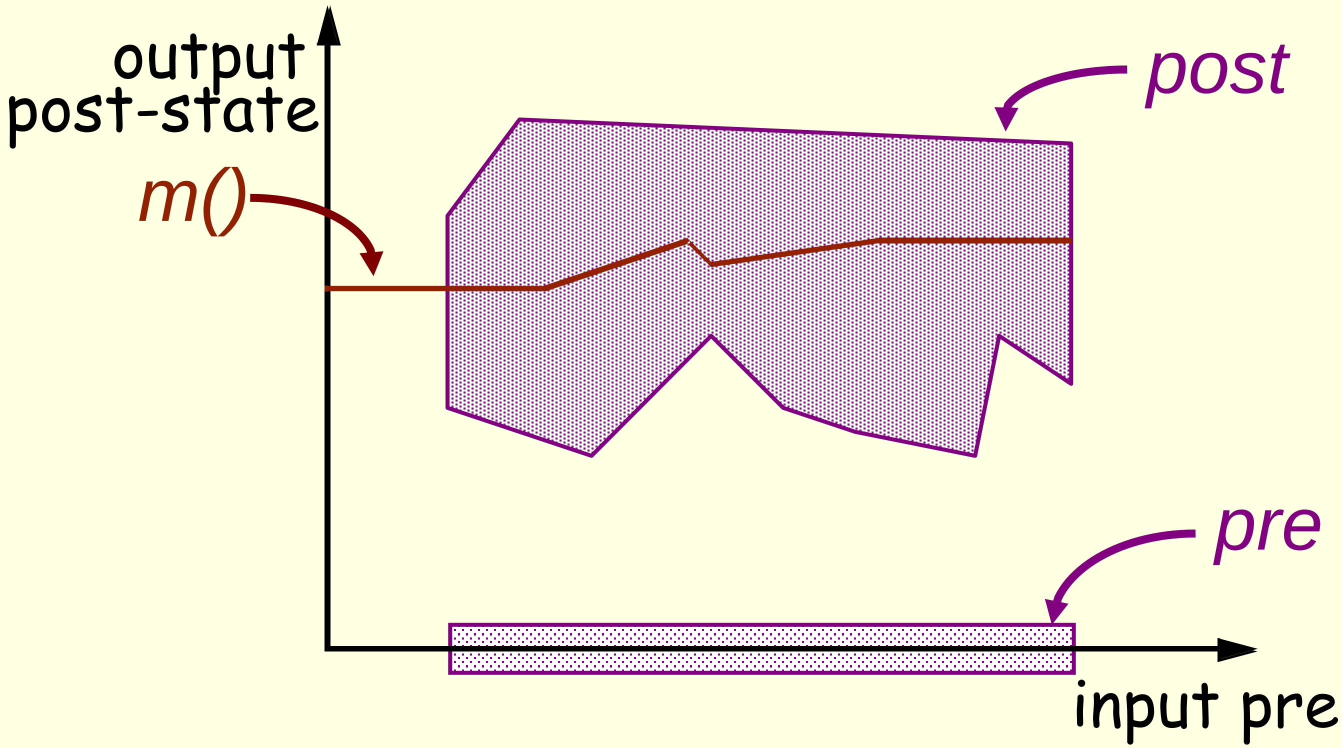
Example JML Specification

```
public interface Gendered {
  //@ model instance String gender;

  //@ requires true;
  //@ ensures \result == gender.equals("female");
  /*@ pure @*/ boolean isFemale();
}
```

model field specification

method behavior specification

Model of Method Specifications



T ▹ (*pre*, *post*)

Specification of Model Field

```
public interface Gendered {
  //@ model instance String gender;

  //@ requires true;
  //@ ensures \result == gender.equals("female");
  /*@ pure @*/ boolean isFemale();
}
```

Implementation of Model Field

```
public class Animal implements Gendered, /*…*/ {
  protected boolean gen; //@ in gender;
  /*@ protected represents gender
    @                <- (gen ? "female" : "male");
    @*/
   public /*@ pure @*/ boolean isFemale()
   { return gen; }
   // …
```

correctness

Problem: Modular Reasoning with Subtyping and Dynamic Dispatch

Reasoning about dynamic dispatch:

```
Gendered e = (Gendered)elems.next();
if (e.isFemale()) {
    //@ assert e.gender.equals("female");
    // ...
}
```

- e could be any subtype (Animal, ...)
- Different implementations
- Different specifications

Problem: Modularity

Reasoning about dynamic dispatch:

```
Gendered e = (Gendered)elems.next();
if (e.isFemale()) {
    //@ assert e.gender.equals("female");
    // ...
}
```

- Verify for each subtype?
- Subtypes may be added later!

Methodology: Supertype Abstraction

Reason using static type information:

```
Gendered e = (Gendered)elems.next();
if (e.isFemale()) {
    //@ assert e.gender.equals("female");
    // ...
}
```

- Use specification from Gendered
- As if no subtyping

Modularity of Supertype Abstraction

- Client reasoning ignores subtyping
- Implementations must be behavioral subtypes

More Details: Supertype Absraction in General

Use static type's specifications to reason about:

- Method calls,
- Invariants,
- History constraints,
- Initially predicates

Supertype Abstraction in General

T o = */* create a new object */;*

//@ **assume** o.ext_inv^T;

/ … */*

//@ **assert** o.$ext\_pre^T_m$;
o.m();
//@ **assume** o.$ext\_post^T_m$ && o.ext_inv^T ;

Supertype Abstraction's Soundness

Valid if:

- Invariants etc. hold as needed (in pre-states), and
- Each subtype is a behavioral subtype

Validity of Supertype Abstraction: Client (Supertype) view

T o = */* create a new object */*;

//@ **assume** o.*ext_inv*^{T};

/ … */*

//@ **assert** *o.ext_pre*^T_m ;

o.m();

//@ **assume** *o.ext_post*^T_m && o.*ext_inv*^{T};

Validity of Supertype Abstraction: Implementation (Subtype) View

/* *body of constructor of* T' */

//@ **assert** o.$ext\_inv^{T'}$;

/* … */

//@ **assume** o.$ext\_pre_m^{T'}$ && o.$ext\_inv^{T'}$;
/* *body of* o.m(); */
//@ **assert** o.$ext\_post_m^{T'}$ && o.$ext\_inv^{T'}$;

Behavioral Subtyping for JML

Suppose $T' \leq T$. Then

T' is a strong behavioral subtype of T

if and only if

whenever **this** has type T':

$$ext\_inv^{T'} \Rightarrow ext\_inv^{T},$$

and for all instance methods *m* of T

$$ext\_spec_m^{T'} \sqsupseteq^{T'} ext\_spec_m^{T}$$

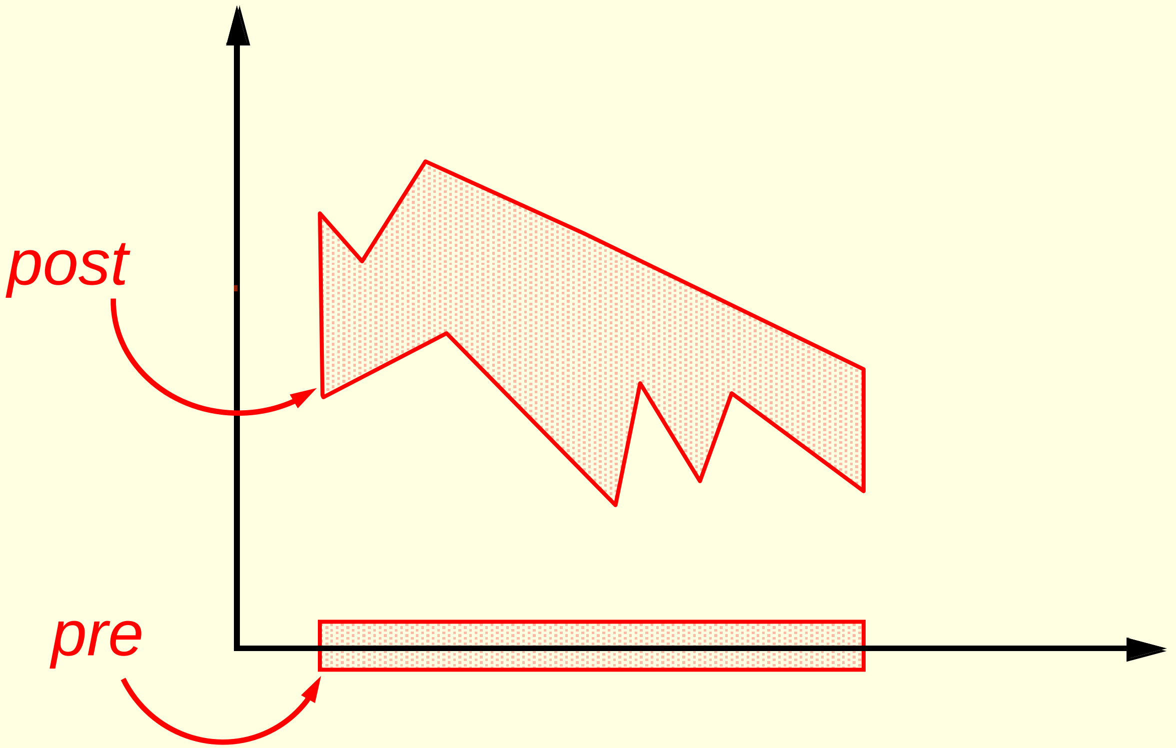
Method Specification Refinement with respect to T'

Notation:

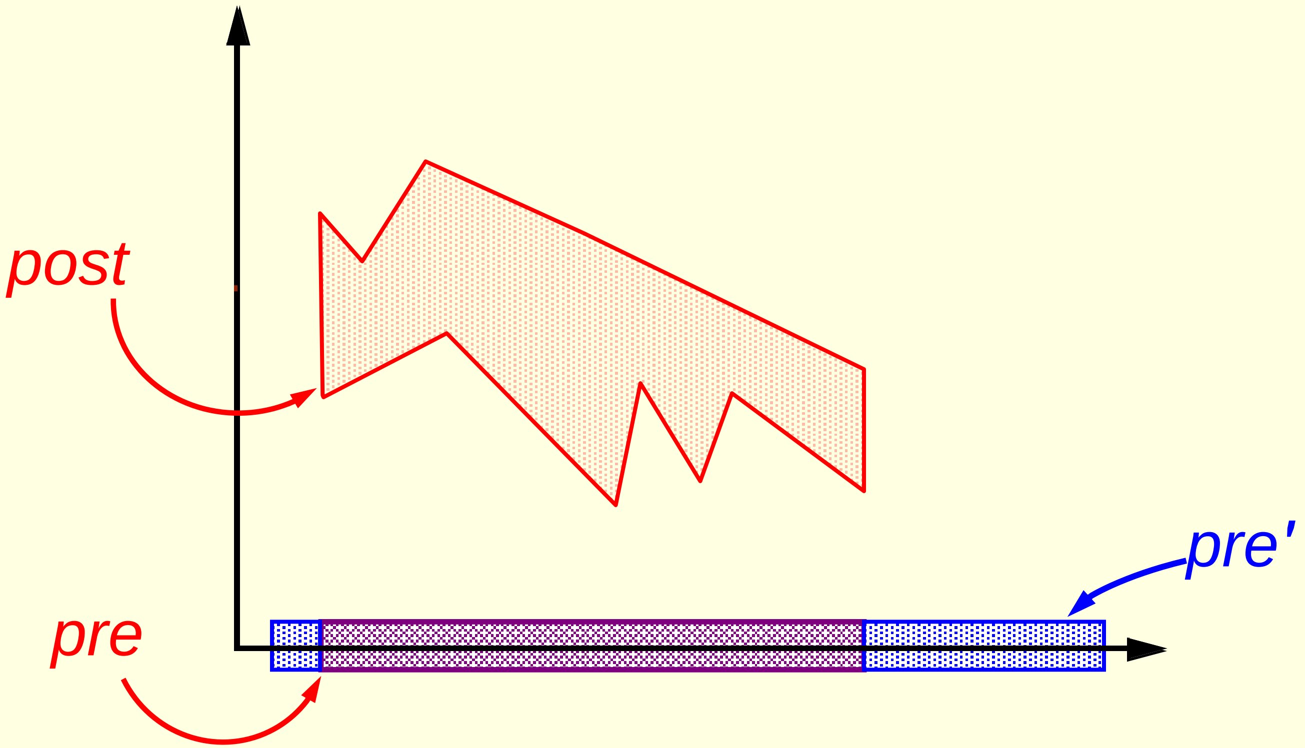
$$(pre', post') \sqsupseteq^{T'} (pre, post)$$

details

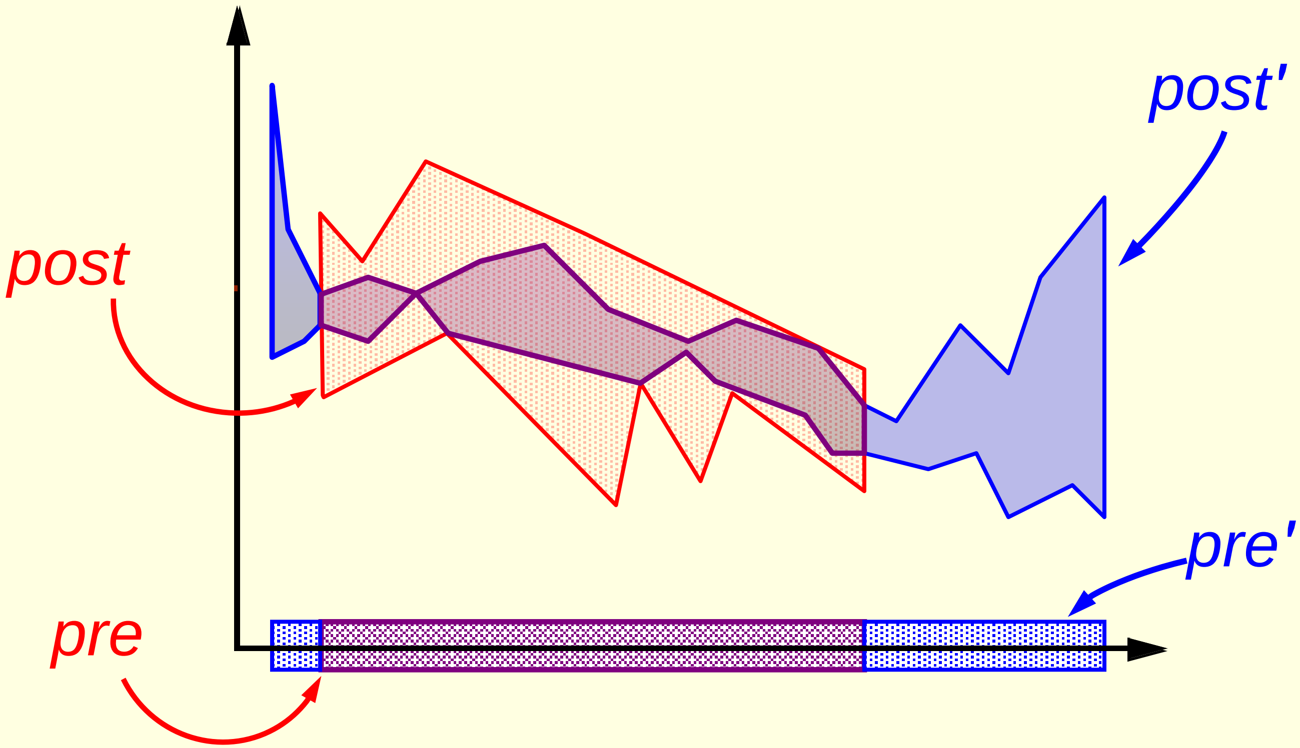
Refinement with respect to T'



Refinement with respect to *T′*



Refinement with respect to T'



Proving Method Refinements

Theorem 1. Suppose $T' \leq T$, and $T' \triangleright (pre', post')$, $T \triangleright (pre, post)$ specify m.

Then $(pre', post') \sqsupseteq^{T'} (pre, post)$

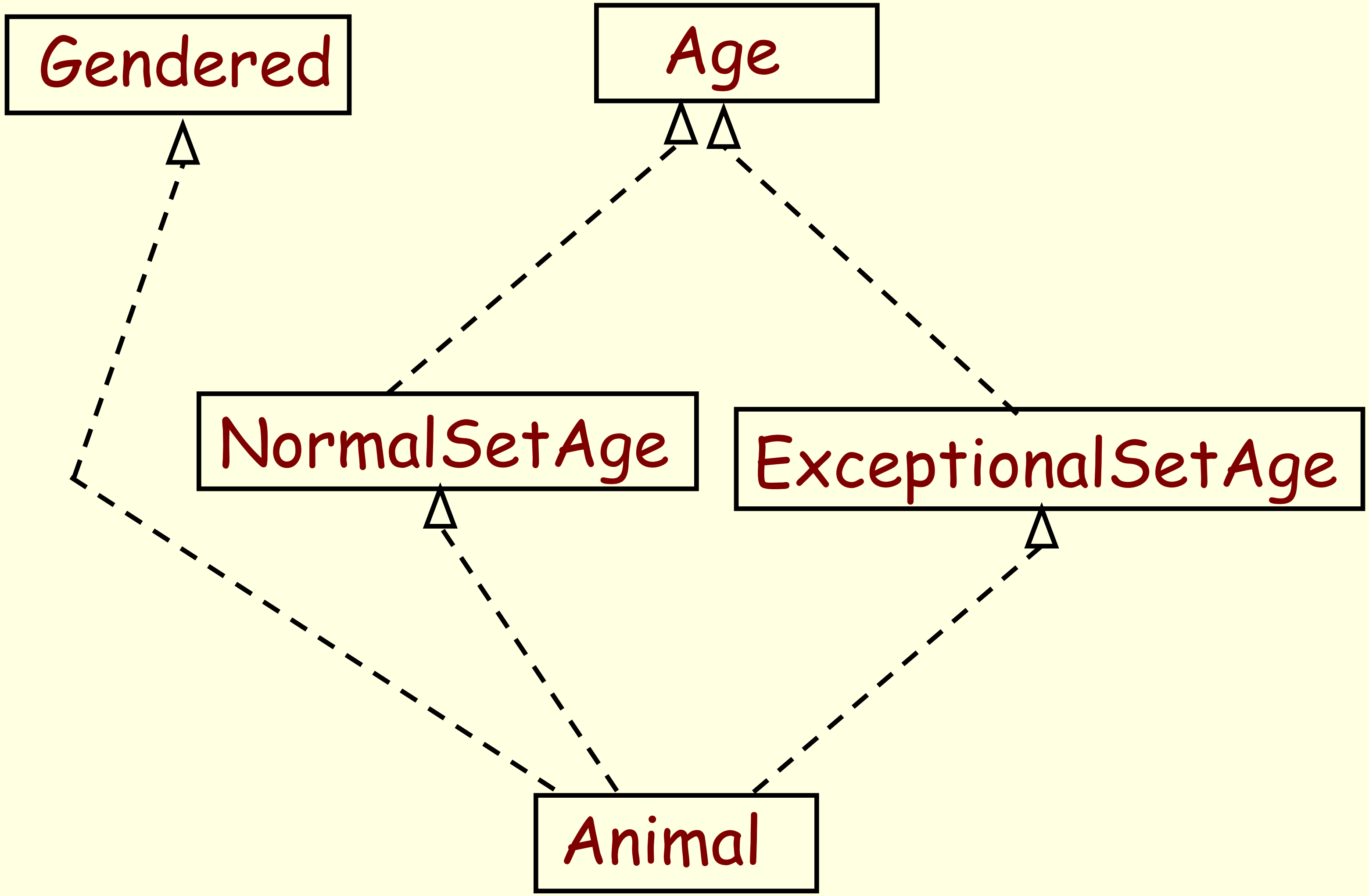
if and only if:

$$Spec(T') \vdash pre \;\&\&\; (\textbf{this instanceof } T') \Rightarrow pre',$$

and

$$Spec(T') \vdash \textbf{\textbackslash old}(pre \;\&\&\; (\textbf{this instanceof } T')) \Rightarrow (post' \Rightarrow post).$$

Subproblem: Avoiding Proofs by Specification Inheritance



Age and NormalSetAge

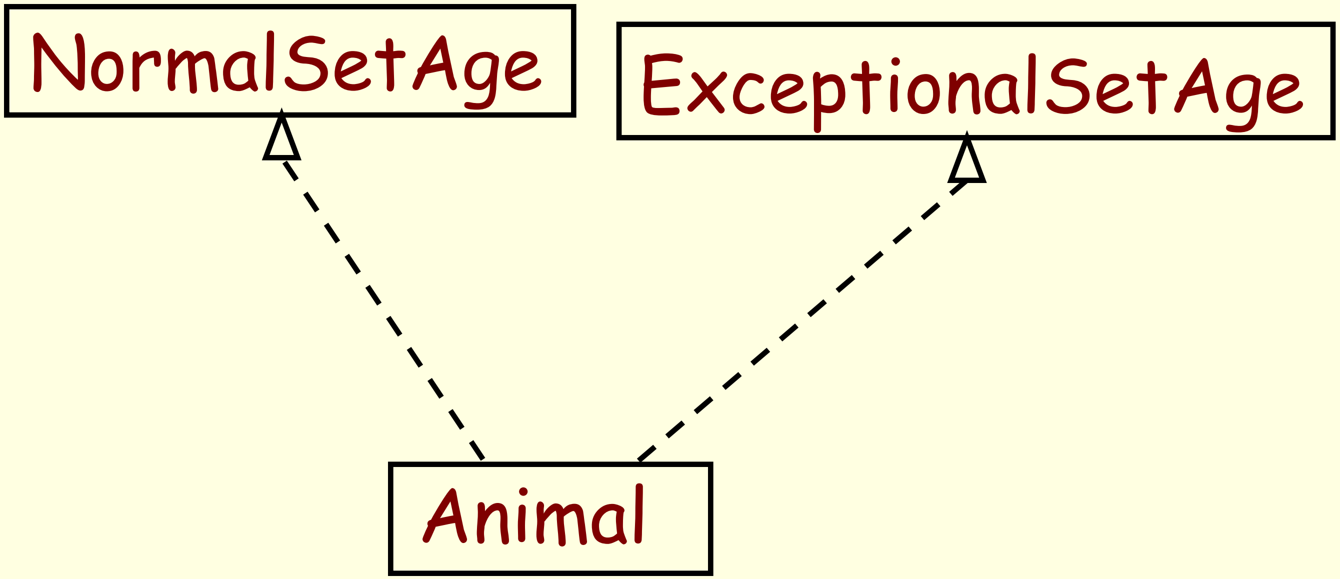
```
public interface Age {
  //@ model instance int age;
}

public interface NormalSetAge implements Age {
  /*@ requires 0 <= a && a <= 150;
    @ ensures age == a;   @*/
  public void setAge(final int a);
}
```

ExceptionalSetAge

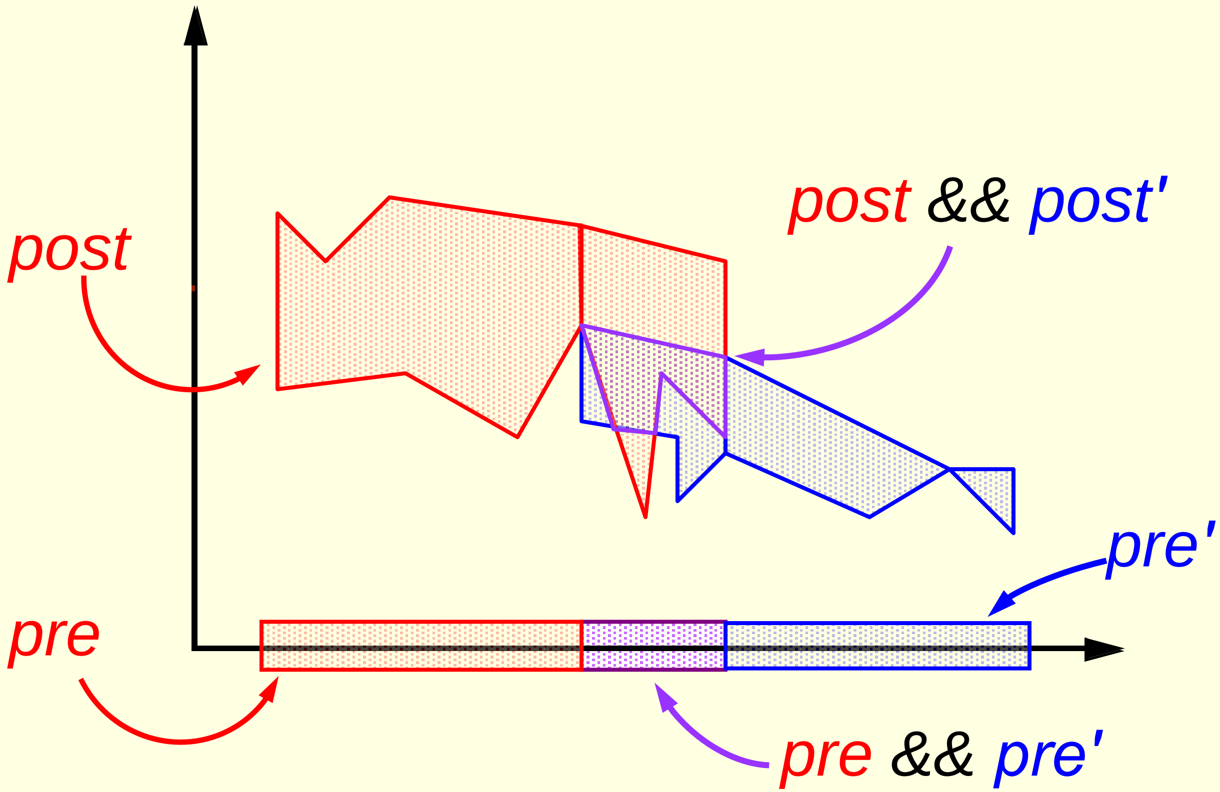
```
public interface ExceptionalSetAge
                                    implements Age {
  /*@  requires a < 0;
     @  ensures age == \old(age);  @*/
  void setAge(final int a);
}
```

What about Animal?

- It's both
- Should obey both specifications



Join of Specification Cases



Join of Specification Cases, 'also'

```
requires 0 <= a && a <= 150;
ensures age == a;
also
requires a < 0;
ensures age == \old(age);
```

means

```
requires (0 <= a && a <= 150) || a < 0;
ensures (\old(0 <= a && a <= 150) ==> age == a)
     && (\old(a < 0) ==> age == \old(age));
```

Join of Specification Cases, $\sqcup^S$

If $T' \triangleright (pre', post')$, $T \triangleright (pre, post)$, $S \leq T'$, $S \leq T$,

then

$$(pre', post') \sqcup^S (pre, post)$$
$$= (p, q)$$

where $p = pre' \mathbin{||} pre$

and q = (\old(pre') ==> $post'$)
&& (\old(pre) ==> $post$)

and $S \triangleright (p, q)$

Model of Inheritance
T's Added Specifications

Declared in *T* (without inheritance):

$added\_inv^T$ invariant

$added\_spec_m^T$ *m*'s specification

Other Notations

$supers(T) = \{U \mid T \leq U\}$

$methods(\mathcal{T}) = \{\, m \mid m \text{ declared in } T \in \mathcal{T}\,\}$

Specification Inheritance's Meaning: Extended Specification of *T*

- **Methods**: for all $m \in methods(supers(T))$

$$ext\_spec_m^T = \sqcup^T \{ added\_spec_m^U \mid U \in supers(T) \}$$

- **Invariant:**

$$ext\_inv^T = \wedge \{ added\_inv^U \mid U \in supers(T) \}$$

also Makes Refinements

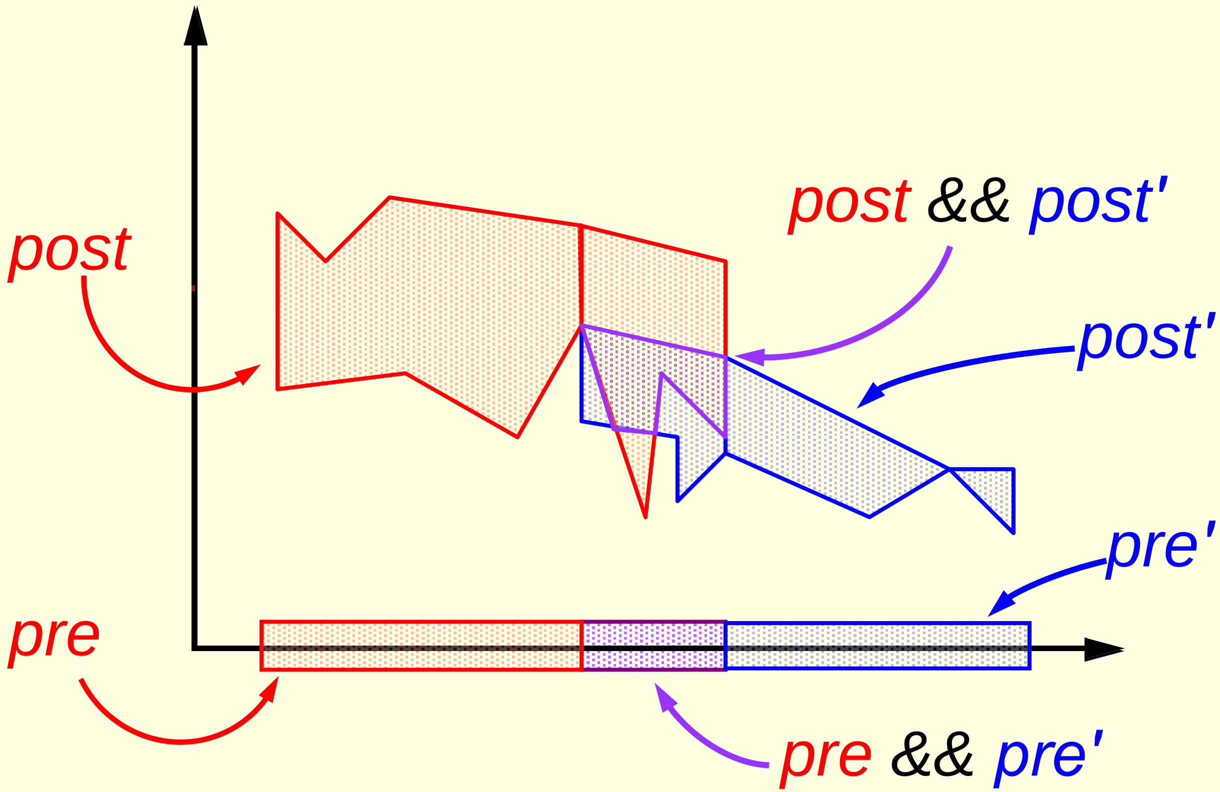
Theorem 2. Suppose **\old** is monotonic.
Suppose $T' \leq T$, and
$T' \triangleright (pre', post')$, $T \triangleright (pre, post)$ specify m.

Then

$$(pre', post') \sqcup^{T'} (pre, post) \sqsupseteq^{T'} (pre, post).$$

Proof: use Theorem 1.

also Makes Refinements



Specification Inheritance Forces Behavioral Subtyping

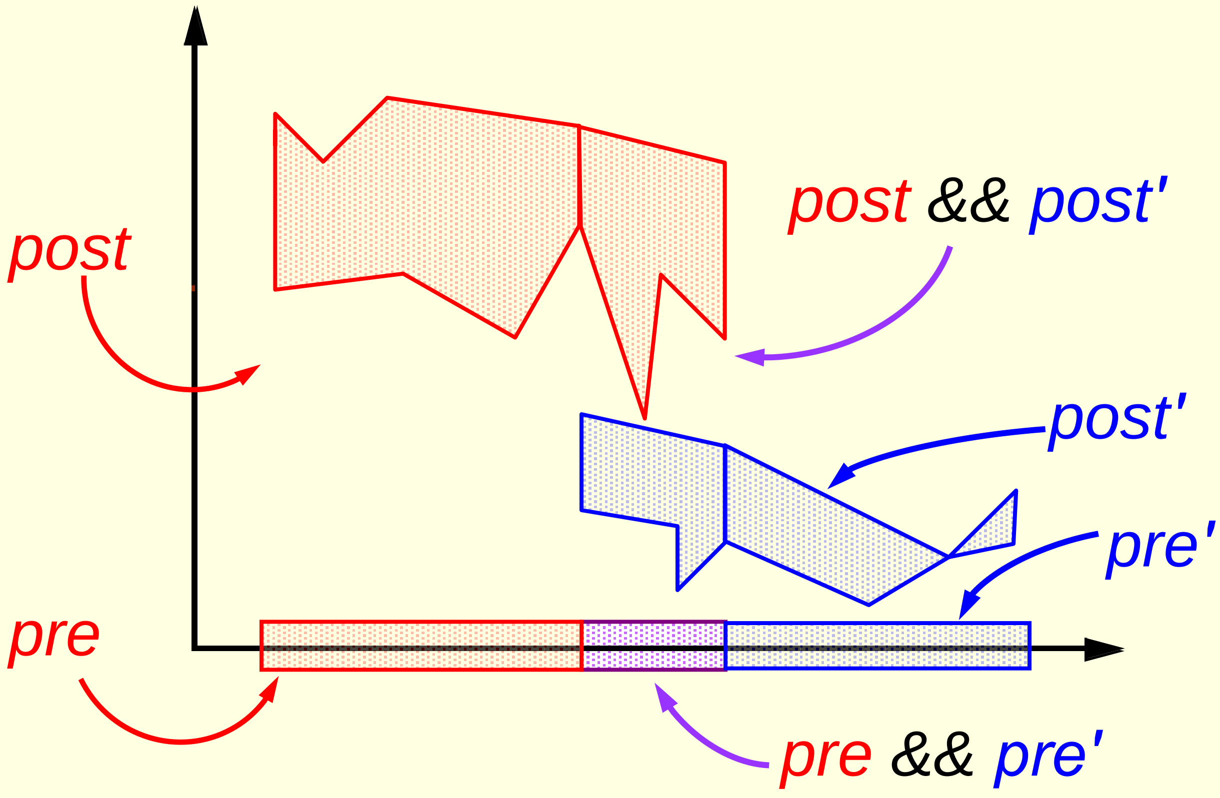
Theorem 3. Suppose $T' \leq T$. Then the extended specification of T' is a strong behavioral subtype of the extended specification of T.

Proof: Use Theorem 2 and definition of extended specification.

Discussion

- Every subtype inherits
- Every subtype is a behavioral subtype
  - Not all satisfiable
  - Supertype must allow refinement

Unsatisfiable Refinements



example

Older Related Work

- Wills's Fresco [Wil92] introduced specification inheritance.
- Wing's dissertation [Win83] combined specification cases like **also**.
- Eiffel [Mey97] has behavioral subtyping and a form of specification inheritance.
- America [Ame87] [Ame91] first proved soundness with behavioral subtyping.
- See survey with Dhara [LD00].

Conclusions

- Supertype abstraction allows modular reasoning.
- Supertype abstraction is valid if:
  - methodology enforced, and
  - subtypes are behavioral subtypes.
- JML's **also** makes refinements.
- Specification inheritance in JML forces behavioral subtyping.
- Supertype abstraction automatically valid in JML.

Acknowledgments

Thanks to David Naumann, William Weihl, Krishna Kishore Dhara, Cesare Tinelli, Don Pigiozzi, Barbara Liskov, Jeannette Wing, Yoonsik Cheon, Al Baker, Clyde Ruby, Tim Wahls, Patrice Chalin, Curtis Clifton, David Cok, Joseph Kiniry, Rustan Leino, Peter Müller, Arnd Poetzsch-Heffter, Erik Poll, and the rest of the JML community.

Join us at...

jmlspecs.org

References

[Ame87] Pierre America. Inheritance and subtyping in a parallel object-oriented language. In Jean Bezivin et al., editors, *ECOOP '87, European Conference on Object-Oriented Programming, Paris, France*, pages 234–242, New York, NY, June 1987. Springer-Verlag. Lecture Notes in Computer Science, volume 276.

[Ame91] Pierre America. Designing an object-oriented programming language with behavioural subtyping. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, volume 489 of *Lecture Notes in Computer Science*, pages 60–90. Springer-Verlag, New York, NY, 1991.

[BCC+05] Lilian Burdy, Yoonsik Cheon, David R. Cok, Michael D. Ernst, Joeseph R. Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer*, 7(3):212–232, June 2005.

[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 ISU CS TR #95-20c, rlhttp://tinyurl.com/s2krg.

[FF01] Robert Bruce Findler and Matthias Felleisen. Contract soundness for object-oriented languages. In *OOPSLA '01 Conference Proceedings, Object-Oriented Programming, Systems, Languages, and Applications, October 14-18, 2001, Tampa Bay, Florida, USA*, pages 1–15, October 2001.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580,583, October 1969.

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

[LD00] Gary T. Leavens and Krishna Kishore Dhara. Concepts of behavioral subtyping and a sketch of their extension to component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, chapter 6, pages 113–135. Cambridge University Press, 2000.

[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.

[LN06] Gary T. Leavens and David A. Naumann. Behavioral subtyping, specification inheritance, and modular reasoning. Technical Report 06-20b, Department of Computer Science, Iowa State University, Ames, Iowa, 50011, September 2006.

[LW94] Barbara H. Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, November 1994.

[Mey97] Bertrand Meyer. *Object-oriented Software Construction*. Prentice Hall, New York, NY, second edition, 1997.

[MPHL06] Peter Müller, Arnd Poetzsch-Heffter, and Gary T. Leavens. Modular invariants for layered object structures. *Science of Computer Programming*, 62(3):253– 286, October 2006.

[Mül02] Peter Müller. *Modular Specification and Verification of Object-Oriented Programs*, volume 2262 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[Par05] Matthew J. Parkinson. Local reasoning for Java. Technical Report 654, University of Cambridge Computer Laboratory, November 2005. The author's Ph.D. dissertation.

[PHM99] A. Poetzsch-Heffter and P. Müller. A programming logic for sequential Java. In S. D. Swierstra, editor, *European Symposium on Programming (ESOP '99)*, volume 1576 of *Lecture Notes in Computer Science*, pages 162–176. Springer-Verlag, 1999.

[Pie06] Cees Pierik. *Validation Techniques for Object-Oriented Proof Outlines*. PhD thesis, Universiteit Utrecht, 2006.

[SBC92] Susan Stepney, Rosalind Barden, and David Cooper, editors. *Object Orientation in Z*. Workshops in Computing. Springer-Verlag, Cambridge CB2 1LQ, UK, 1992.

[Wil92] Alan Wills. Specification in Fresco. In Stepney et al. [SBC92], chapter 11, pages 127–135.

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