JML Model Fields

Christian Engel

ITI, Universität Karlsruhe

08. Juni 2005

Outline

1. JML
2. What are model fields?
3. Translation to JavaDL
   - Axiomatic approach
   - Interpretation as model methods/queries
4. Demo

JML

JML . . .

- . . . is a specification language tailored to Java.

JML

JML . . .

- . . . is a specification language tailored to Java.
- . . . serves as an input language for KeY.

JML

JML ...

- ... is a specification language tailored to Java.
- ... serves as an input language for KeY.
- ... can be used for specifying method contracts and loop invariants.

JML

JML . . .

- . . . is a specification language tailored to Java.
- . . . serves as an input language for KeY.
- . . . can be used for specifying method contracts and loop invariants.
- . . . allows declaring model methods and model fields.

Model fields

Model fields are only visible on the level of specification.

Model fields

Model fields are only visible on the level of specification.

Example:

```
//@ public model int a;
```

Model fields

Model fields are only visible on the level of specification.

Example:

```
//@ public model int a;
```

The `represents` clause defines, how the value of a model field is related to the implementation.

Model fields

Model fields are only visible on the level of specification.

Example:

```
//@ public model int a;
```

The `represents` clause defines, how the value of a model field is related to the implementation.

```
/*@ public represents a \such_that 0<=a && a<size();
@*/
```

The `represents` clause

The `represents` clause defines a relation $\mathtt{R}(\mathtt{x}, \tilde{\mathtt{q}})$ between a model field `x` and a vector $\tilde{\mathtt{q}}$, consisting of fields and methods.

```
//@ model t x;
//@ represents x \such_that R(x,q̃);
```

The axiomatic approach – a first attempt

A first attempt:
We interpret $R(x, \tilde{q})$ as an axiom that holds in every state of the program.

The axiomatic approach – a first attempt

A first attempt:
We interpret $R(x, \tilde{q})$ as an axiom that holds in every state of the program.

But:
This is not possible, since there can be a state s, in which $R(x, \tilde{q})$ is equivalent to *false*.

The axiomatic approach – a first attempt

A first attempt:
We interpret $R(x, \tilde{q})$ as an axiom that holds in every state of the program.

But:
This is not possible, since there can be a state s, in which $R(x, \tilde{q})$ is equivalent to *false*.

Solution:
The axiom we have to use is:

$$(\exists a : t\ (R(a, \tilde{q}))) \rightarrow R(x, \tilde{q})$$

The axiomatic approach expressed in JavaDL

- Let $\phi(x)$ be a first order formula with occurences of x.

The axiomatic approach expressed in JavaDL

- Let $\phi(x)$ be a first order formula with occurences of x.
- x is the result of the translation of a model field `x` of type t.

The axiomatic approach expressed in JavaDL

- Let $\phi(x)$ be a first order formula with occurences of x.
- x is the result of the translation of a model field `x` of type t.
- $R(x, \tilde{q})$ is the formula provided by the represents clause.

The axiomatic approach expressed in JavaDL

- Let $\phi(x)$ be a first order formula with occurences of x.
- x is the result of the translation of a model field `x` of type t.
- $R(x, \tilde{q})$ is the formula provided by the represents clause.

Then we get the formula:

$$\forall x' : t \, (\{x := x'\}(A(x) \rightarrow \phi(x)))$$

with

$$A(x) := (\exists a : t \, (R(a, \tilde{q}))) \rightarrow R(x, \tilde{q})$$

Drawbacks of the axiomatic approach

- bigger, less readable formulas

Drawbacks of the axiomatic approach

- bigger, less readable formulas
- not applicable for recursively defined *represents* clauses

Drawbacks of the axiomatic approach

- bigger, less readable formulas
- not applicable for recursively defined *represents* clauses

One possible solution: Use Taclets

$$\frac{}{\Gamma \vdash \phi(x()), \Delta}$$

Drawbacks of the axiomatic approach

- bigger, less readable formulas
- not applicable for recursively defined *represents* clauses

One possible solution: Use Taclets

$$\frac{\Gamma \vdash \phi(x()), \exists x' : t\ (R(x', \tilde{q})), \Delta}{\Gamma \vdash \phi(x()), \Delta}$$

Drawbacks of the axiomatic approach

- bigger, less readable formulas
- not applicable for recursively defined *represents* clauses

One possible solution: Use Taclets

$$\frac{\Gamma \vdash \phi(x()), \exists x' : t\, (R(x', \tilde{q})), \Delta \quad \Gamma, \textcolor{red}{R(x(), \tilde{q})} \vdash \phi(x()), \Delta}{\Gamma \vdash \phi(x()), \Delta}$$

Representing model fields by model methods

Another approach:
Model fields are represented by model method that are free of side effects and have a “suitable” specification.

Representing model fields by model methods

Another approach:
Model fields are represented by model method that are free of side effects and have a "suitable" specification.

Let $\phi(x)$ and $R(x,\vec{q})$ be defined as on the previous slides. We get the formula:

$$\phi(m())$$

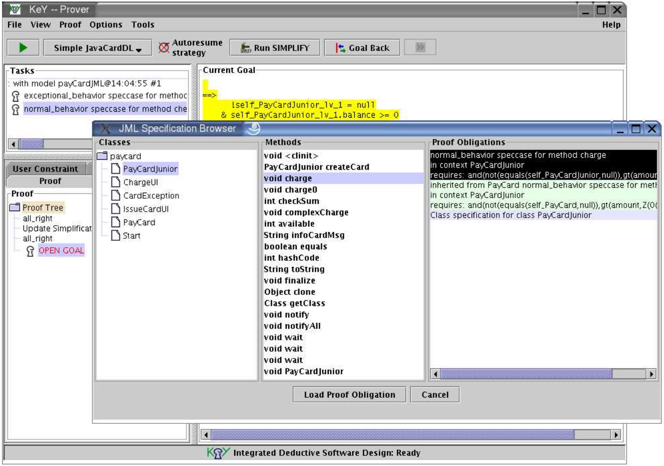
where $m()$ is the model method associated with $R(x,\vec{q})$.

Representing model fields by model methods

The specification of *m*():

```
/*@ public normal_behavior
  @  requires (\exists t x; R(x,q));
  @  assignable \nothing;
  @  ensures R(\result, q);
  @*/
```

Demo


KeY -- Prover
File View Proof Options Tools Help
Simple JavaCardDL
Autoresume strategy
Run SIMPLIFY
Goal Back
Tasks
: with model payCardJML@14:04:55 #1
exceptional_behavior speccase for methoc
normal_behavior speccase for method che
Current Goal
==>
!self_PayCardJunior_lv_1 = null
& self_PayCardJunior_lv_1.balance >= 0
JML Specification Browser
Classes
paycard
PayCardJunior
ChargeUI
CardException
IssueCardUI
PayCard
Start
Methods
void <clinit>
PayCardJunior createCard
void charge
void charge0
int checkSum
void complexCharge
int available
String infoCardMsg
boolean equals
int hashCode
String toString
void finalize
Object clone
Class getClass
void notify
void notifyAll
void wait
void wait
void wait
void PayCardJunior
Proof Obligations
normal_behavior speccase for method charge
in context PayCardJunior
inherited from PayCard normal_behavior speccase for meth
in context PayCardJunior
Class specification for class PayCardJunior
Load Proof Obligation
Cancel
User Constraint
Proof
Proof
Proof Tree
all_right
Update Simplificat
all_right
OPEN GOAL
Integrated Deductive Software Design: Ready

The interface LimitedIntContainer

```
public interface LimitedIntContainer{
    /*@
      @ public model int value;
      @ public model boolean regularState;
      @*/

    /*@ public normal_behavior
      @   ensures regularState ==> \result == value;
      @*/
    int /*@ pure @*/ available();
}
```

The class PayCard

```
public class PayCard implements LimitedIntContainer{

 /*@ public represents value <- balance;
   @ public represents regularState  <-
   @                (unsuccessfulOperations <= 3);
   @*/

  public /*@pure@*/ int available() {
      if (unsuccessfulOperations<=3) return balance;
      return 0;
  }

  ...
}
```