

Modelling System Invariants

Part 2: Usage of JRS in KeY

Bernhard Beckert, Steffen Schlager and Richard Bubel

7th June 2006

Preliminary

Definition (Object Repository)

For each *non-abstract class type* T , there is a rigid function symbol

$$T::\langle get \rangle : \text{Integer} \rightarrow T$$

evaluating to an element of dynamic type T .

Preliminary

Definition (Object Repository)

For each *non-abstract class type* T , there is a rigid function symbol

$$T::\langle get \rangle : \text{Integer} \rightarrow T$$

evaluating to an element of dynamic type T .

Definition (Object Index)

The index of an object o of dynamic type T is the integer i , so that

$$o \doteq T::\langle get \rangle(i)$$

holds.

System State Description in *JavaCardDL*

Implicit Static Fields Record System State; for each type T

- ▶ T.<classPrepared>,
T.<classInitialized>,
T.<classInitInProgress> and
T.<classErroneous>
- ▶ T.<nextToCreate>

System State Description in *JavaCardDL*

Implicit Static Fields Record System State; for each type T

- ▶ T.<classPrepared>,
T.<classInitialized>,
T.<classInitInProgress> and
T.<classErroneous>
- ▶ T.<nextToCreate>

Implicit Instance Fields State of an Instance

- ▶ o.<created>: auxiliary attribute (queries if an object is created)
- ▶ o.<initialized>: indicates, if object initialization terminated properly

Java Reachable States - As Formula?

Instead of a formula, a non-rigid predicate

JRS

is used.

JRS holds iff. a state is 'reachable' via a *Java* program.

System Invariants - Created Instances

An object o of dynamic type T is *created*, iff. its index i satisfies

$$0 \leq i < T.\langle \text{nextToCreate} \rangle$$

System Invariants - Created Instances

An object o of dynamic type T is *created*, iff. its index i satisfies

$$0 \leq i < T.\langle \text{nextToCreate} \rangle$$

In a Java Reachable State, we require for each non-abstract class type T

- ▶ static field $T.\langle \text{nextToCreate} \rangle$ is not negative
- ▶ instance field $o.\langle \text{created} \rangle$ is consistent

In *JavaCardDL*

$T.\langle \text{nextToCreate} \rangle \geq 0$ &

\forall int i ;

$(i \geq 0 \ \& \ i < T.\langle \text{nextToCreate} \rangle \ \leftrightarrow$

$T::\langle \text{get} \rangle(i).\langle \text{created} \rangle = \text{TRUE})$

System Invariants - Created Instances

An object o of dynamic type T is *created*, iff. its index i satisfies

$$0 \leq i < T.\langle \text{nextToCreate} \rangle$$

In a Java Reachable State, we require for each non-abstract class type T

- ▶ static field $T.\langle \text{nextToCreate} \rangle$ is not negative
- ▶ instance field $o.\langle \text{created} \rangle$ is consistent

In *JavaCardDL*

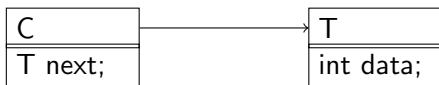
$T.\langle \text{nextToCreate} \rangle \geq 0$ &

\forall int i ;

$(i \geq 0 \ \& \ i < T.\langle \text{nextToCreate} \rangle \ \leftrightarrow$

$T::\langle \text{get} \rangle(i).\langle \text{created} \rangle = \text{TRUE})$

System Invariants - Legal Reference Structure



Situation

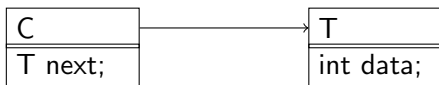
`c.<created> = TRUE,`

`\forall T o; (o.<created> = TRUE \rightarrow o.data \geq 0)`

\Rightarrow

`c.next = null, c.next.data \geq 0`

System Invariants - Legal Reference Structure



Situation

$c.<created> = \text{TRUE}$, **JRS**,

$\forall \text{ T } o; (o.<created> = \text{TRUE} \rightarrow o.\text{data} \geq 0)$

\Rightarrow

$c.\text{next} = \text{null}, c.\text{next}.\text{data} \geq 0$

Legal Reference Structure

Objects referenced by created instances are `null` or must be created. (similar for arrays)

System Invariants - Legal Reference Structure



Situation

$c.\langle\text{created}\rangle = \text{TRUE}, \text{JRS},$
 $\backslash\text{forall } T \text{ } o; (o.\langle\text{created}\rangle = \text{TRUE} \rightarrow o.\text{data} \geq 0)$
 \Rightarrow
 $c.\text{next} = \text{null}, c.\text{next}.\text{data} \geq 0$

Legal Reference Structure

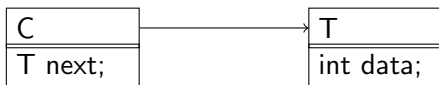
Objects referenced by created instances are null or must be created. (similar for arrays)

In *JavaCardDL*

For each reference type T , for each attribute a_i of reference type declared in T

$\backslash\text{forall } T \text{ } o; (o.\langle\text{created}\rangle = \text{TRUE} \rightarrow$
 $(o.a_i = \text{null} \mid o.a_i.\langle\text{created}\rangle = \text{TRUE}))$

System Invariants - Legal Reference Structure



Situation

$c.\langle\text{created}\rangle = \text{TRUE}, \text{JRS},$
 $\backslash\text{forall } T \text{ } o; (o.\langle\text{created}\rangle = \text{TRUE} \rightarrow o.\text{data} \geq 0)$
 \Rightarrow
 $c.\text{next} = \text{null}, c.\text{next}.\text{data} \geq 0$

Legal Reference Structure

Objects referenced by created instances **are null or must be created**. (similar for arrays)

In *JavaCardDL*

For each reference type T , for each attribute a_i of reference type declared in T

$\backslash\text{forall } T \text{ } o; (o.\langle\text{created}\rangle = \text{TRUE} \rightarrow$
 $(o.a_i = \text{null} \mid o.a_i.\langle\text{created}\rangle = \text{TRUE}))$

System Invariants - Others

Further system invariants required to hold are, e.g.

System Invariants - Others

Further system invariants required to hold are, e.g.

B	
<clInit>	= TRUE
<clPrep>	= TRUE
<clInInit>	= FALSE
<classErr>	= FALSE

If a **class or interface is initialized**, it

- ▶ has been prepared,
- ▶ has finished its initialization phase and
- ▶ is not marked as erroneous

... and several other properties.

No dynamic class loading.

System Invariants - Others

Further system invariants required to hold are, e.g.

B		
<clInit>	=	TRUE
<clPrep>	=	TRUE
<clInInit>	=	FALSE
<classErr>	=	FALSE

If a class or interface is initialized, it

- ▶ has been prepared,
- ▶ has finished its initialization phase and
- ▶ is not marked as erroneous

... and several other properties.

No dynamic class loading.

System Invariants - Others

Further system invariants required to hold are, e.g.

B		
<clInit>	=	TRUE
<clPrep>	=	TRUE
<clInInit>	=	FALSE
<classErr>	=	FALSE

If a class or interface is initialized, it

- ▶ has been prepared,
- ▶ has finished its initialization phase and
- ▶ is not marked as erroneous

... and several other properties.

No dynamic class loading.

System Invariants - Others

Further system invariants required to hold are, e.g.

B		
<clInit>	=	TRUE
<clPrep>	=	TRUE
<clInInit>	=	FALSE
<classErr>	=	FALSE

If a class or interface is initialized, it

- ▶ has been prepared,
- ▶ has finished its initialization phase and
- ▶ is not marked as erroneous

... and several other properties.

No dynamic class loading.

How to prove *with* JRS

Adding JRS as formula inefficient. Exploit JRS ensured properties using taclets:

```
only_created_object_are_referenced {
  \assumes (obj.<created> = TRUE, JRS ==>)
  \find ( obj.#a ) \sameUpdateLevel
  \varcond(\isReference(\typeof(#a)))
  \add (obj.#a.<created> = TRUE | obj.#a = null ==>)
  \displayname "referenced object is created"
};
```

Modifies/Assignable Clauses preserve JRS

Loop Invariant Rule - Translation of Modifies Clause

```
/*@  
  * loop_invariant <some_jml_boolean_expression>;  
  * assignable o.a;  
  */  
while (b) { ... }
```

Assignable: $\langle o.a := c_a(o) \rangle$ (represented as anonymizing update)

Preserves Invariant:

$$\langle o.a := c_a(o) \rangle (inv \ \& \ b = \text{TRUE} \rightarrow \langle \alpha \rangle inv)$$

Modifies/Assignable Clauses preserve JRS

Loop Invariant Rule - Translation of Modifies Clause

```
/*@  
  * loop_invariant <some_jml_boolean_expression>;  
  * assignable o.a;  
  */  
while (b) { ... }
```

Assignable: $\langle o.a := c_a(o) \rangle$ (represented as anonymizing update)

Preserves Invariant:

$$\langle o.a := c_a(o) \rangle (inv \ \& \ b = \text{TRUE} \rightarrow \langle \alpha \rangle inv)$$

Specification languages assume implicitly: Modifies respects JRS

Modifies/Assignable Clauses preserve JRS

Loop Invariant Rule - Translation of Modifies Clause

```
/*@  
  * loop_invariant <some_jml_boolean_expression>;  
  * assignable o.a;  
  */  
while (b) { ... }
```

Assignable: $\langle o.a := c_a(o) \rangle$ (represented as anonymizing update)

Preserves Invariant:

$$\langle o.a := c_a(o) \rangle (inv \ \& \ b = \text{TRUE} \rightarrow \langle \alpha \rangle inv)$$

Specification languages assume implicitly: Modifies respects JRS

$$\rightsquigarrow \langle o.a := c_a(o) \rangle (inv \ \& \ b = \text{TRUE} \ \& \ \text{JRS} \rightarrow \langle \alpha \rangle inv)$$

How to prove *validity of JRS*?

Situation

$\text{JRS} \Rightarrow \langle l_1 := v_1, \dots, l_m := v_m \rangle \text{JRS}$

How to prove *validity of JRS*?

Situation

$\text{JRS} \Rightarrow \langle l_1 := v_1, \dots, l_m := v_m \rangle \text{JRS}$

Problem

Naive replacement of JRS with its axioms is inefficient (too huge formula)

How to prove *validity of JRS*?

Situation

$JRS \Rightarrow \langle l_1 := v_1, \dots, l_m := v_m \rangle JRS$

Problem

Naive replacement of JRS with its axioms is inefficient (too huge formula)

Solution: Optimize

Generate optimized formula po_{JRS} , so that

$$\Gamma \Rightarrow \mathcal{U}JRS \text{ if } \Gamma \Rightarrow JRS \ \& \ \mathcal{U}po_{JRS}$$

- ▶ Analyze locations of update \mathcal{U}
- ▶ Use stronger formula po'_{JRS} (reachable from 'outer' state)

Optimization: Example

Example

$$\underbrace{\langle o.a := u, T.\langle \text{nextToCreate} \rangle := \text{ival} \rangle}_{\mathcal{U}}$$

Generation of the JRS proof obligation formula ρ_o the check of

- ▶ legal reference structure is only necessary for o.a.
- ▶ system invariant of $\langle \text{nextToCreate} \rangle$ is only necessary for type T.

Proof obligation $\rho_{o_{JRS}}$ (Legal Reference Structure)

```
\forall T o@pre;  
(o@pre = o  $\rightarrow$   
   $\mathcal{U}(o@pre.\langle \text{created} \rangle = \text{TRUE} \rightarrow$   
     $o@pre.a.\langle \text{created} \rangle = \text{TRUE} \mid o@pre.a = \text{null}))$ 
```

Optimization: Example

Example

$$\underbrace{\langle o.a := u, T.\langle nextToCreate \rangle := ival \rangle}_{\mathcal{U}}$$

Generation of the JRS proof obligation formula ρ_o the check of

- ▶ legal reference structure is only necessary for o.a.
- ▶ system invariant of $\langle nextToCreate \rangle$ is only necessary for type T.

Proof obligation ρ_{OJRS} (Createdness)

$\mathcal{U}(T.\langle nextToCreate \rangle) \geq 0 \ \&$

\backslash forall int i;

$(i \geq 0 \ \& \ i < \mathcal{U}(T.\langle nextToCreate \rangle)) \Leftrightarrow$

$\mathcal{U}(T::\langle get \rangle(i).\langle created \rangle) = \text{TRUE})$

Optimization: Example

Example

$$\underbrace{\langle o.a := u, T.\langle nextToCreate \rangle := ival \rangle}_{\mathcal{U}}$$

Generation of the JRS proof obligation formula ρ_o the check of

- ▶ legal reference structure is only necessary for o.a.
- ▶ system invariant of $\langle nextToCreate \rangle$ is only necessary for type T.

Proof obligation $\rho_{O_{JRS}}$ (Createdness)

$\mathcal{U}(T.\langle nextToCreate \rangle) \geq 0 \ \&$

$\backslash \text{forall int } i;$

$(i \geq 0 \ \& \ i < \mathcal{U}(T.\langle nextToCreate \rangle) \leftrightarrow$

$\mathcal{U}(T::\langle get \rangle(i).\langle created \rangle) = \text{TRUE})$

But: Decreasing $T.\langle nextToCreate \rangle$ effects also locations not occurring in the update.

Optimization: Example

Example

$$\underbrace{\langle o.a := u, T.\langle \text{nextToCreate} \rangle := \text{ival} \rangle}_{u}$$

Generation of the JRS proof obligation formula po the check of

- ▶ legal reference structure is only necessary for o.a.
- ▶ system invariant of $\langle \text{nextToCreate} \rangle$ is only necessary for type T.

Proof obligation po'_{JRS} (Createdness)

$\mathcal{U}(T.\langle \text{nextToCreate} \rangle) \geq 0 \ \&$

$\backslash \text{forall int } i;$

$(i \geq T.\langle \text{nextToCreate} \rangle \ \& \ i < \mathcal{U}(T.\langle \text{nextToCreate} \rangle) \rightarrow$
 $\mathcal{U}(T::\langle \text{get} \rangle(i).\langle \text{created} \rangle) = \text{TRUE})$

$\ \& \ T.\langle \text{nextToCreate} \rangle \leq \mathcal{U}(T.\langle \text{nextToCreate} \rangle)$

Therefore: Stronger proof obligation po'_{JRS} - Java reachable from the pre-state of the update.

Proving JRS - Restrictions

Optimizing not sufficient

Even in the optimized case some formulas would become too huge.

Example

$$\langle o.\langle \text{created} \rangle := \text{TRUE} \rangle$$

In this case o 's dynamic type cannot be derived (in general)

\rightsquigarrow all subtypes of the static type of o must be considered.

Therefore: Support only for $o = T::\langle \text{get} \rangle(j)$

Proving JRS - Restrictions

Optimizing not sufficient

Even in the optimized case some formulas would become too huge.

Example

$$\langle o.\langle \text{created} \rangle := \text{TRUE} \rangle$$

In this case o 's dynamic type cannot be derived (in general)

\rightsquigarrow all subtypes of the static type of o must be considered.

Therefore: Support only for $o = T::\langle \text{get} \rangle(j)$

Proving JRS without pre leading update

No rule for insertion of the definition is offered.