KeY Proof Obligations

Andreas Roth

June 9, 2005

KeY Proof Obligations So Far

- Ad hoc
- No concept of global correctness, when finished?
- Bugs in (horizontal) POs

KeY Proof Obligations So Far

- Ad hoc
- No concept of global correctness, when finished?
- Bugs in (horizontal) POs

Example

1.
```
class A{ /*@instance invariant b==0 & a!=this*/
         private int b; A a;
         void m() {a.b=1;} }
```

PreservesInvariant for m() passes!

KeY Proof Obligations So Far

- Ad hoc
- No concept of global correctness, when finished?
- Bugs in (horizontal) POs

Example

1.
```
class A{ /*@instance invariant b==0 & a!=this*/
         private int b; A a;
         void m() {a.b=1;} }
```
PreservesInvariant for m() passes!

2.
```
class A{ /*@instance invariant b.c==0; */ private B b;}
class B{ private int c; setC(int c){this.c=c;} }
```
PreservesInvariant for setC(int) not available!

KeY Proof Obligations So Far

- Ad hoc
- No concept of global correctness, when finished?
- Bugs in (horizontal) POs

Example

1.
```
class A{ /*@instance invariant b==0 & a!=this*/
         private int b; A a;
         void m() {a.b=1;} }
```
PreservesInvariant for m() passes!

2.
```
class A{ /*@instance invariant b.c==0; */ private B b;}
class B{ private int c; setC(int c){this.c=c;} }
```
PreservesInvariant for setC(int) not available!

3.
```
class A{ B b; /*\result==0*/int m() {return b.getC();}}
class B{int c; /*instance invariant c==0;*/ int getC(){return c;}}
```
Not proveable with EnsuresPostcondition

The Task

Input from Translation OCL$\rightarrow$DL or JML$\rightarrow$DL

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$

  $\left.\begin{array}{l}\psi_{opct}(o, p_1, \ldots, p_n) \\ \varphi_{opct}(o, p_1, \ldots, p_n, r, exc)\end{array}\right\}$ as FOL formula

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface D,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface D,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\forall o{:}\, T\ (o.\texttt{<created>} \rightarrow \varphi(o))$

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface D,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\overline{\forall} o{:}\, T\ \varphi(o) := \forall o{:}\, T\ (o.\texttt{<created>} \rightarrow \varphi(o))$

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$

    } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\overline{\forall} o\colon T\ \varphi(o) := \forall o\colon T\ (o.\texttt{<created>} \rightarrow \varphi(o))$

The Task

Input from Translation OCL$\rightarrow$DL or JML$\rightarrow$DL:

- Operation contracts:
  - the method declaration *op* in class or interface D,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$

    $\Big\}$ as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\overline{\forall} o\colon T\ \varphi(o) := \forall o\colon T\ (o.\texttt{<created>} \rightarrow \varphi(o))$
- Static invariants Inv_{stat} as FOL formulas (not from OCL)

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\overline{\forall} o\colon T\ \varphi(o) := \forall o\colon T\ (o.\texttt{<created>} \rightarrow \varphi(o))$
- Static invariants Inv_{stat} as FOL formulas (not from OCL)

The Task

Input from Translation OCL→DL or JML→DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\overline{\forall} o{:}\,T\ \varphi(o) := \forall o{:}\,T\ (o.\texttt{<created>} \rightarrow \varphi(o))$
- Static invariants Inv_{stat} as FOL formulas (not from OCL)

Output: Proof Obligations, (sets of) JavaDL formulas whose validity ensures a “good” property of a model or program.

The Task

Input from Translation OCL$\rightarrow$DL or JML$\rightarrow$DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$

  $\Big\}$ as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\overline{\forall} o{:}\, T\ \varphi(o) := \forall o{:}\, T\ (o.\texttt{<created>} \rightarrow \varphi(o))$
- Static invariants Inv_{stat} as FOL formulas (not from OCL)

Output: Proof Obligations, (sets of) JavaDL formulas whose validity ensures a "good" property of a model or program.

The Task

Input from Translation OCL$\rightarrow$DL or JML$\rightarrow$DL:

- Operation contracts:
  - the method declaration *op* in class or interface *D*,
  - a precondition $\psi_{opct}(o, p_1, \ldots, p_n)$ } as FOL formula
  - a postcondition $\varphi_{opct}(o, p_1, \ldots, p_n, r, exc)$ } as FOL formula
  - an assignable clause $\{l_1, \ldots, l_n\}$ as FOL terms
  - a marker from $\{partial, total\}$
- Instance invariants Inv_{inst} as FOL formulas
  Shape: $\varphi(o)$. Treat as $\overline{\forall} o{:}\, T\ \varphi(o) := \forall o{:}\, T\ (o.\texttt{<created>} \rightarrow \varphi(o))$
- Static invariants Inv_{stat} as FOL formulas (not from OCL)

Output: Proof Obligations, (sets of) JavaDL formulas whose validity ensures a "good" property of a model or program.

Overview

Lightweight
Design Validation
Properties

Lightweight
Program Correctness
Properties

Heavyweight
Program Correctness
Properties

Overview

Lightweight
Design Validation
Properties

Lightweight
Program Correctness
Properties

Heavyweight
Program Correctness
Properties

Model only,
No Program

Horizontal Verification

Overview

Lightweight Design Validation Properties	Lightweight Program Correctness Properties
	Heavyweight Program Correctness Properties
Model only, No Program	(Correctness) Properties of Programs
Horizontal Verification	Vertical Verification

Overview

Lightweight Design Validation Properties	Lightweight Program Correctness Properties	single properties of interest
	Heavyweight Program Correctness Properties	
Model only, No Program	(Correctness) Properties of Programs	
Horizontal Verification	Vertical Verification	

Overview

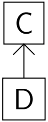
Lightweight Design Validation Properties	Lightweight Program Correctness Properties	single properties of interest
	Heavyweight Program Correctness Properties	statements about whole program
Model only, No Program	(Correctness) Properties of Programs	
Horizontal Verification	Vertical Verification	

Overview

Lightweight Design Validation Properties	Lightweight Program Correctness Properties	single properties of interest
	Heavyweight Program Correctness Properties	statements about whole program
Model only, No Program	(Correctness) Properties of Programs	
Horizontal Verification	Vertical Verification	

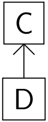
Structural Subtyping

Original Idea: Invariants of D at least as strong as those of C.

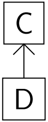


Structural Subtyping

Original Idea: Invariants of D at least as strong as those of C.
Translation of instance invariant of C: $\overline{\forall} o{:}\, C\ \varphi_C(o)$

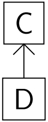


Structural Subtyping



Original Idea: Invariants of D at least as strong as those of C.
Translation of instance invariant of C: $\overline{\forall} o{:}\, C\ \varphi_C(o)$
FOL inclusion semantics: Quantification covers instances of D
$\overline{\forall} o{:}\, C\ \varphi_C(o) \rightarrow \overline{\forall} o{:}\, D\ \varphi_C(o)$.

Structural Subtyping

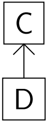


Original Idea: Invariants of D at least as strong as those of C.
Translation of instance invariant of C: $\overline{\forall} o{:}\, C\ \varphi_C(o)$
FOL inclusion semantics: Quantification covers instances of D
$\overline{\forall} o{:}\, C\ \varphi_C(o) \rightarrow \overline{\forall} o{:}\, D\ \varphi_C(o)$.

Invariant of C is already (implicit) invariant of D.
Thus: invariants of D at least as strong as that of C by definition.

No need for PO?

Structural Subtyping



Original Idea: Invariants of D at least as strong as those of C.
Translation of instance invariant of C: $\overline{\forall} o{:}C\ \varphi_C(o)$
FOL inclusion semantics: Quantification covers instances of D
$\overline{\forall} o{:}C\ \varphi_C(o) \rightarrow \overline{\forall} o{:}D\ \varphi_C(o)$.

Invariant of C is already (implicit) invariant of D.
Thus: invariants of D at least as strong as that of C by definition.

No need for PO?
Alternatives:

- Yes, no need

Structural Subtyping

$$\begin{tikzcd} \boxed{C} \\ \boxed{D} \arrow[u] \end{tikzcd}$$

Original Idea: Invariants of D at least as strong as those of C.
Translation of instance invariant of C: $\overline{\forall} o{:}\, C\ \varphi_C(o)$
FOL inclusion semantics: Quantification covers instances of D
$\overline{\forall} o{:}\, C\ \varphi_C(o) \rightarrow \overline{\forall} o{:}\, D\ \varphi_C(o)$.

Invariant of C is already (implicit) invariant of D.
Thus: invariants of D at least as strong as that of C by definition.

No need for PO?
Alternatives:

- Yes, no need
- Use *exact* quantification in translation unnatural

Structural Subtyping

$$\begin{tikzcd} \boxed{C} \\ \boxed{D} \arrow[u] \end{tikzcd}$$

Original Idea: Invariants of D at least as strong as those of C.
Translation of instance invariant of C: $\overline{\forall} o{:}\, C \ \varphi_C(o)$
FOL inclusion semantics: Quantification covers instances of D
$\overline{\forall} o{:}\, C \ \varphi_C(o) \rightarrow \overline{\forall} o{:}\, D \ \varphi_C(o)$.

Invariant of C is already (implicit) invariant of D.
Thus: invariants of D at least as strong as that of C by definition.

No need for PO?
Alternatives:

- Yes, no need
- Use *exact* quantification in translation — unnatural
- No problem. Specs. are better if they occur again in subclasses...
  PO: $\overline{\forall} o{:}\, D \ (\varphi_D(o) \rightarrow \varphi_C(o))$

Behavioural Subtyping

m_D overrides m_C.

Pre-conditions: ψ_D as weak as ψ_C, Post-conditions: φ_D as strong as φ_C

Behavioural Subtyping

m_D overrides m_C.
Pre-conditions: ψ_D as weak as ψ_C, Post-conditions: φ_D as strong as φ_C

Can be enforced by inheriting operation contracts (as in JML)

Behavioural Subtyping

m_D overrides m_C.
Pre-conditions: ψ_D as weak as ψ_C, Post-conditions: φ_D as strong as φ_C

Can be enforced by inheriting operation contracts (as in JML)

If inheritance of specifications not desired:

$$\psi_C \rightarrow \psi_D \tag{1}$$

$$\varphi_D \rightarrow \varphi_C \tag{2}$$

with ψ_C, ψ_D pre-conditions
with φ_C, φ_D post-conditions
with $\{l_1, \ldots, l_n\}$ modifies clause

Behavioural Subtyping

m_D overrides m_C.
Pre-conditions: ψ_D as weak as ψ_C, Post-conditions: φ_D as strong as φ_C

Can be enforced by inheriting operation contracts (as in JML)

If inheritance of specifications not desired:

$$\psi_C \rightarrow \psi_D \tag{1}$$

$$\forall x_1, \ldots, x_n(\psi_D \rightarrow \{l_1 := x_1, \ldots, l_n := x_n\}(\varphi_D \rightarrow \varphi_C)) \tag{2}$$

with ψ_C, ψ_D pre-conditions
with φ_C, φ_D post-conditions
with $\{l_1, \ldots, l_n\}$ modifies clause

Behavioural Subtyping

m_D overrides m_C.
Pre-conditions: ψ_D as weak as ψ_C, Post-conditions: φ_D as strong as φ_C

Can be enforced by inheriting operation contracts (as in JML)

If inheritance of specifications not desired:

$$\psi_C \rightarrow \psi_D \tag{1}$$
$$\forall x_1, \ldots, x_n(\psi_D \rightarrow \{l_1 := x_1, \ldots, l_n := x_n\}(\varphi_D \rightarrow \varphi_C)) \tag{2}$$

with ψ_C, ψ_D pre-conditions
with φ_C, φ_D post-conditions
with $\{l_1, \ldots, l_n\}$ modifies clause

or no modifies information: $$\varphi_D \rightarrow \varphi_C \tag{2'}$$

Strong Postcondition

Does the post condition on its own establish own instance invariant (after operation call)?

Strong Postcondition

Does the post condition on its own establish own instance invariant (after operation call)?

Otherwise: Operation contract not "functionally complete".

Strong Postcondition

Does the post condition on its own establish own instance invariant (after operation call)?

Otherwise: Operation contract not "functionally complete".

$$\left(\varphi_{post} \longrightarrow \bigwedge_{\varphi \in I} \varphi\right)$$

with I instance invariants
φ_{post} post-conditions

Strong Postcondition

Does the post condition on its own establish own instance invariant (after operation call)?

Otherwise: Operation contract not “functionally complete”.

$$\forall x_1, \ldots, x_n \Big(\varphi_{pre} \wedge \bigwedge_{\varphi \in I} \varphi \rightarrow \{ l_1 := x_1, \ldots, l_n := x_n \} \big(\varphi_{post} \rightarrow \bigwedge_{\varphi \in I} \varphi \big) \Big)$$

with I instance invariants
φ_{post} post-conditions
with φ_{pre} pre-conditions
$\{l_1, \ldots, l_n\}$ modifies clause

Distinct Preconditions

Given two operation contracts $opct_1, opct_2$. Do the pre-conditions not overlap?

Distinct Preconditions

Given two operation contracts $opct_1, opct_2$. Do the pre-conditions not overlap?

Important for JML normal_behavior / exceptional_behavior

Example

```
/* @          normal_behavior
   @            requires true //...
   @    also  exceptional_behavior
   @            requires p==0 //...
   @*/
public void m(int p) {//...
```

Distinct Preconditions

Given two operation contracts $opct_1, opct_2$. Do the pre-conditions not overlap?

Important for JML normal_behavior / exceptional_behavior

Example

```
/* @          normal_behavior
   @            requires true //...
   @    also  exceptional_behavior
   @            requires p==0 //...
   @*/
public void m(int p) {//...
```

$$\forall o \forall r \forall p_1, \ldots, p_n (\neg\psi_1(o, r, p_1, \ldots, p_n) \vee \neg\psi_2(o, r, p_1, \ldots, p_n))$$

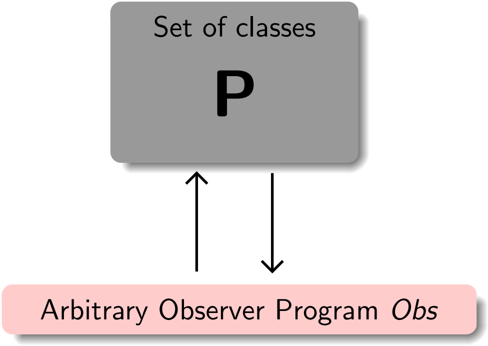
with $\psi_i(o, r, p_1, \ldots, p_n)$ precondition of $opct_i(o, r, p_1, \ldots, p_n)$

Overview

Lightweight Design Validation Properties	Lightweight Program Correctness Properties	single properties of interest
	Heavyweight Program Correctness Properties	statements about whole program
Model only, No Program	(Correctness) Properties of Programs	
Horizontal Verification	Vertical Verification	

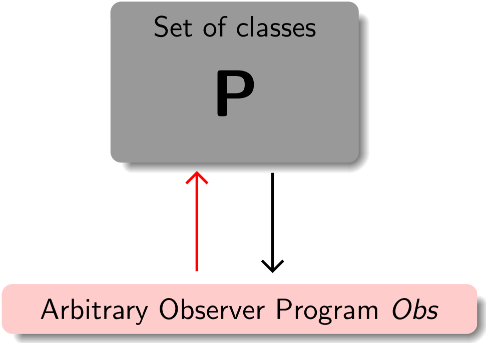
Observational Correctness (single-threaded Java)

Observational Correctness (single-threaded Java)



Observer program: arbitrary calls to P.
Judges: Does call of P correspond to specification S of P?

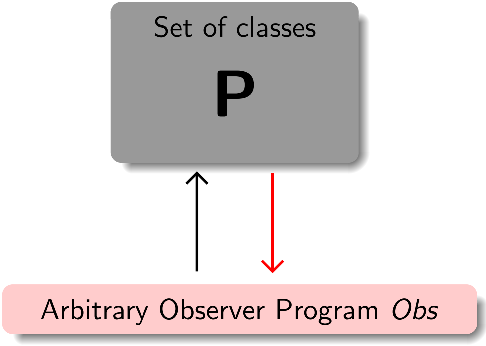
Observational Correctness (single-threaded Java)



Observer program: arbitrary calls to P.

Judges: Does call of P correspond to specification S of P?

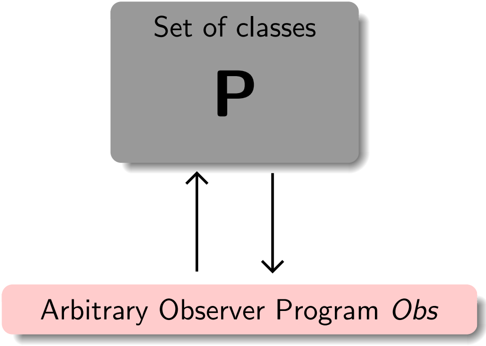
Observational Correctness (single-threaded Java)



Observer program: arbitrary calls to P.

Judges: Does call of P correspond to specification S of P?

Observational Correctness (single-threaded Java)

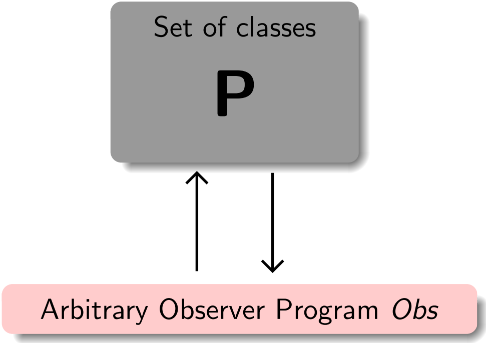


Observer program: arbitrary calls to P.

Judges: Does call of P correspond to specification S of P?

- What are requirements on *Obs* (Assumptions on calls to P)?
- How is judgement made?

Observational Correctness (single-threaded Java)



Observer program: arbitrary calls to P.

Judges: Does call of P correspond to specification S of P?

- What are requirements on *Obs* (Assumptions on calls to P)?
- How is judgement made?

Two variants: *call correctness* and *persistent correctness*

Definition of Call Correctness

Given program P, observer Obs.

Definition of Call Correctness

Given program P, observer Obs.

Obs calls operations in P only if

1. one of the preconditions holds

Definition of Call Correctness

Given program P, observer Obs.

Obs calls operations in P only if

1. one of the preconditions holds
2. all invariants are satisfied

Definition of Call Correctness

Given program P, observer Obs.

Obs calls operations in P only if

1. one of the preconditions holds
2. all invariants are satisfied

Definition of Call Correctness

Given program P, observer Obs.

Obs calls operations in P only if

1. one of the preconditions holds
2. all invariants are satisfied

P is *call correct* if

1. every operation call to P in Obs fulfils the operation contracts attached to that operation

Definition of Call Correctness

Given program *P*, observer *Obs*.

Obs calls operations in *P* only if

1. one of the preconditions holds
2. all invariants are satisfied

P is *call correct* if

1. every operation call to *P* in *Obs* fulfils the operation contracts attached to that operation
2. all invariants of all objects and classes hold after operation call to *P*

Definition of Call Correctness

Given program P, observer Obs.

Obs calls operations in P only if

1. one of the preconditions holds
2. all invariants are satisfied

P is *call correct* if

1. every operation call to P in Obs fulfils the operation contracts attached to that operation
2. all invariants of all objects and classes hold after operation call to P

Definition of Call Correctness

Given program P, observer *Obs*.

Obs calls operations in P only if

1. one of the preconditions holds
2. all invariants are satisfied

P is *call correct* if

1. every operation call to P in *Obs* fulfils the operation contracts attached to that operation
2. all invariants of all objects and classes hold after operation call to P

Example

Instance invariants: $\forall o{:}\,T\ (o.\texttt{<created>} \rightarrow \varphi(o))$

Before constructor call $\surd$ After constructor call $\surd$

Proof Obligations for Call Correctness

Proof Obligation: Simulate observer call generically

Proof Obligations for Call Correctness

Proof Obligation: Simulate observer call generically

General shape of proof obligations:

$$\psi \rightarrow \langle \alpha \rangle \varphi$$

Proof Obligations for Call Correctness

Proof Obligation: Simulate observer call generically

General shape of proof obligations:

$$\psi \rightarrow \langle \alpha \rangle \varphi$$

- Assumptions

Proof Obligations for Call Correctness

Proof Obligation: Simulate observer call generically

General shape of proof obligations:

$$\psi \rightarrow \langle \alpha \rangle \varphi$$

- Assumptions
- Programs

Proof Obligations for Call Correctness

Proof Obligation: Simulate observer call generically

General shape of proof obligations:

$$\psi \rightarrow \langle \alpha \rangle \varphi$$

- Assumptions
- Programs
- Assertions

Proof Obligations for Call Correctness

Proof Obligation: Simulate observer call generically

General shape of proof obligations:

$$\psi \rightarrow \langle \alpha \rangle \varphi$$

- Assumptions
- Programs
- Assertions
- Modalities

Call Correctness: Assumptions

In Observer: Which conditions hold before a call to P?

Call Correctness: Assumptions

In Observer: Which conditions hold before a call to P?
The general assumption $\mathcal{A}(op, o, p_1, \ldots, p_n)$ before call to method or constructor *op* on object described by *o*:

$$\bigwedge_{\varphi \in Inv_{cl}} \varphi \wedge \bigvee_{\substack{\varphi_{pre}(\overrightarrow{x}) \\ \text{precondition} \\ \text{of } OpCt_{op}}} \varphi_{pre}(o, p_1, \ldots, p_n) \wedge o.\texttt{<created>} = TRUE$$

Call Correctness: Assumptions

In Observer: Which conditions hold before a call to P?
The general assumption $\mathcal{A}(op, o, p_1, \ldots, p_n)$ before call to method or constructor op on object described by o:

$$\bigwedge_{\varphi \in Inv_{cl}} \varphi \wedge \bigvee_{\substack{\varphi_{pre}(\overrightarrow{x}) \\ \text{precondition} \\ \text{of } OpCt_{op}}} \varphi_{pre}(o, p_1, \ldots, p_n) \wedge o.\texttt{<created>} = TRUE$$

- all invariants of all classes hold,

Call Correctness: Assumptions

In Observer: Which conditions hold before a call to P?
The general assumption $\mathcal{A}(op, o, p_1, \ldots, p_n)$ before call to method or constructor *op* on object described by *o*:

$$\bigwedge_{\varphi \in Inv_{cl}} \varphi \wedge \bigvee_{\substack{\varphi_{pre}(\overrightarrow{x}) \\ \text{precondition} \\ \text{of } OpCt_{op}}} \varphi_{pre}(o, p_1, \ldots, p_n) \wedge o.\texttt{<created>} = TRUE$$

- all invariants of all classes hold,
- at least one of the preconditions of *op* holds,

Call Correctness: Assumptions

In Observer: Which conditions hold before a call to P?
The general assumption $\mathcal{A}(op, o, p_1, \ldots, p_n)$ before call to method or constructor *op* on object described by *o*:

$$\bigwedge_{\varphi \in Inv_{cl}} \varphi \wedge \bigvee_{\substack{\varphi_{pre}(\overrightarrow{x}) \\ \text{precondition} \\ \text{of } OpCt_{op}}} \varphi_{pre}(o, p_1, \ldots, p_n) \wedge o.\texttt{<created>} = TRUE$$

- all invariants of all classes hold,
- at least one of the preconditions of *op* holds,
- called object is created

Programs 1

“Generic” Observer contains statement $\alpha_{op_D}(\mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n), \mathsf{r})$:

Programs 1

"Generic" Observer contains statement $\alpha_{op_D}(\mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n), \mathsf{r})$:

Instance methods op_D:

self.m@D(p_1,...,p_n):r;

self.m@D(p_1,...,p_n);

Programs 1

"Generic" Observer contains statement $\alpha_{op_D}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r})$:

Instance methods op_D:

self.m@D(p_1,. . . ,p_n):r;

self.m@D(p_1,. . . ,p_n);

Static methods:

r=$D.m$(p_1,. . . ,p_n);

$D.m$(p_1,. . . ,p_n);

Programs 1

"Generic" Observer contains statement $\alpha_{op_D}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r})$:

Instance methods op_D:
self.m@D(p_1,. . . ,p_n):r;
self.m@D(p_1,. . . ,p_n);

Static methods:
r=$D.m$(p_1,. . . ,p_n);
$D.m$(p_1,. . . ,p_n);

Constructors:
new D(p_1,. . . ,p_n);

Programs 2

Gather information on abrupt termination behaviour:

$$\widetilde{\alpha}_{op_D}(\mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n), \mathsf{r}, \mathsf{exc}) := \left\{ \begin{array}{l} \mathsf{exc}{=}\textbf{null};\\ \textbf{try}\{\\ \quad \alpha_{op_D}(\mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n), \mathsf{r})\\ \}\ \textbf{catch}\ (\mathsf{Throwable\ e})\ \{\\ \quad \mathsf{exc} = \mathsf{e};\\ \} \end{array} \right.$$

$$\widetilde{\alpha}_{op_D}(\mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n), \mathsf{r}) := \left\{ \begin{array}{l} \textbf{try}\{\\ \quad \alpha_{op_D}(\mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n), \mathsf{r})\\ \}\ \textbf{catch}\ (\mathsf{Throwable\ e})\ \{\} \end{array} \right.$$

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*:

If op is called

1. on object described by o with parameters $p_1, \ldots, p_n$ and the returned value is contained in r and the thrown exception in *exc*,

then

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*:

If op is called

1. on object described by o with parameters $p_1, \ldots, p_n$ and the returned value is contained in r and the thrown exception in exc,
2. in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$, and

then

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*:

If op is called

1. on object described by *o* with parameters $p_1, \ldots, p_n$ and the returned value is contained in *r* and the thrown exception in *exc*,
2. in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$, and
3. in a state which satisfies the precondition $\psi_{opct}(o, (p_1, \ldots, p_n))$

then

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*:

If op is called

1. on object described by o with parameters $p_1, \ldots, p_n$ and the returned value is contained in r and the thrown exception in exc,
2. in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$, and
3. in a state which satisfies the precondition $\psi_{opct}(o, (p_1, \ldots, p_n))$

then

1. if the *total* marker of *opct* is set, the call terminates in a post-state

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*:

If op is called

1. on object described by o with parameters $p_1, \ldots, p_n$ and the returned value is contained in r and the thrown exception in *exc*,
2. in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$, and
3. in a state which satisfies the precondition $\psi_{opct}(o, (p_1, \ldots, p_n))$

then

1. if the *total* marker of *opct* is set, the call terminates in a post-state
2. if there is a post-state, then in the post-state

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*:

If op is called

1. on object described by o with parameters $p_1, \ldots, p_n$ and the returned value is contained in r and the thrown exception in *exc*,
2. in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$, and
3. in a state which satisfies the precondition $\psi_{opct}(o, (p_1, \ldots, p_n))$

then

1. if the *total* marker of *opct* is set, the call terminates in a post-state
2. if there is a post-state, then in the post-state
    - the postcondition $\varphi_{opct}(o, (p_1, \ldots, p_n), r, exc)$ of *opct* holds and

Call Correctness: Assertions 1

An operation *op* fulfils operation contract *opct*:

If op is called

1. on object described by o with parameters $p_1, \ldots, p_n$ and the returned value is contained in r and the thrown exception in *exc*,
2. in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$, and
3. in a state which satisfies the precondition $\psi_{opct}(o, (p_1, \ldots, p_n))$

then

1. if the *total* marker of *opct* is set, the call terminates in a post-state
2. if there is a post-state, then in the post-state
   - the postcondition $\varphi_{opct}(o, (p_1, \ldots, p_n), r, exc)$ of *opct* holds and
   - only the elements $l_i(o, p_1, \ldots, p_n)$ of the modifies clauses in the operation contracts are (permanently) modified.

Call Correctness: Assertions 2

Operation *op* recovers a set of invariants I

Call Correctness: Assertions 2

Operation *op* recovers a set of invariants I:

If

1. *op* is called on object described by o with parameters $p_1, \ldots, p_n$,

Call Correctness: Assertions 2

Operation *op* recovers a set of invariants I:

If

1. *op* is called on object described by o with parameters $p_1, \ldots, p_n$,
2. *op* is called in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$,

Call Correctness: Assertions 2

Operation *op* recovers a set of invariants I:

If

1. *op* is called on object described by o with parameters $p_1, \ldots, p_n$,
2. *op* is called in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$,
3. the call terminates

Call Correctness: Assertions 2

Operation *op* recovers a set of invariants I:

If

1. *op* is called on object described by o with parameters $p_1, \ldots, p_n$,
2. *op* is called in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$,
3. the call terminates

Call Correctness: Assertions 2

Operation *op* recovers a set of invariants I:

If

1. *op* is called on object described by o with parameters $p_1, \ldots, p_n$,
2. *op* is called in a state which satisfies the general assumption $\mathcal{A}(op, o, (p_1, \ldots, p_n))$,
3. the call terminates

then in the post-state all $\varphi \in I$ are valid.

Program Correctness Proof Obligations

To prove entire call correctness, for all operations *op*:

- EnsuresPost

$$\begin{aligned}&\mathcal{A}(op, \mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n)) \land \psi_{opct} \\ &\qquad \to \langle\widetilde{\alpha}_{op}(\mathsf{self}, (\mathsf{p}_1, \ldots, \mathsf{p}_n), \mathsf{r}, \mathsf{exc})\rangle\varphi_{opct}\end{aligned}$$

Program Correctness Proof Obligations

To prove entire call correctness, for all operations *op*:

- EnsuresPost

$$\mathcal{A}(op, \mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n)) \land \psi_{opct} \\ \rightarrow \langle\widetilde{\alpha}_{op}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r}, \mathrm{exc})\rangle\varphi_{opct}$$

equivalent to

$$\bigwedge_{\varphi \in Inv_{cl}} \varphi \land \psi_{opct} \land \mathrm{self}.\texttt{<created>} = \mathit{TRUE} \\ \rightarrow \langle\widetilde{\alpha}_{op}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r}, \mathrm{exc})\rangle\varphi_{opct}$$

Program Correctness Proof Obligations

To prove entire call correctness, for all operations *op*:

- EnsuresPost

$$\mathcal{A}(op, \mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n)) \land \psi_{opct} \\ \to \langle \widetilde{\alpha}_{op}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r}, \mathrm{exc}) \rangle \varphi_{opct}$$

  equivalent to

$$\bigwedge_{\varphi \in Inv_{cl}} \varphi \land \psi_{opct} \land \mathrm{self}.\texttt{<created>} = TRUE \\ \to \langle \widetilde{\alpha}_{op}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r}, \mathrm{exc}) \rangle \varphi_{opct}$$

- RespectsModifies
  (see Ralf Sasse's minor thesis or alternative in my draft)

Program Correctness Proof Obligations

To prove entire call correctness, for all operations *op*:

- EnsuresPost

$$\mathcal{A}(op, \mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n)) \wedge \psi_{opct} \\ \rightarrow \langle \widetilde{\alpha}_{op}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r}, \mathrm{exc}) \rangle \varphi_{opct}$$

  equivalent to

$$\bigwedge_{\varphi \in Inv_{cl}} \varphi \wedge \psi_{opct} \wedge \mathrm{self}.\texttt{<created>} = TRUE \\ \rightarrow \langle \widetilde{\alpha}_{op}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r}, \mathrm{exc}) \rangle \varphi_{opct}$$

- RespectsModifies
  (see Ralf Sasse's minor thesis or alternative in my draft)
- for set I of all invariants: RecoverInv

$$\mathcal{A}(op, \mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n)) \rightarrow [\widetilde{\alpha}_{op_D}(\mathrm{self}, (\mathrm{p}_1, \ldots, \mathrm{p}_n), \mathrm{r})] \bigwedge_{\varphi \in I} \varphi$$

Refinement: Reducing the PO Explosion

For all operations and for all invariants: RecoverInv

Consequences:

- PO Explosion, to a high degree non-modular!

Refinement: Reducing the PO Explosion

For all operations and for all invariants: RecoverInv

Consequences:

- PO Explosion, to a high degree non-modular!
- Coarse approximation to call correctness

Refinement: Reducing the PO Explosion

For all operations and for all invariants: RecoverInv

Consequences:

- PO Explosion, to a high degree non-modular!
- Coarse approximation to call correctness

Example

class A{ /*@instance invariant $\overbrace{\text{b.c=0;}}^{\varphi}$ */ private B b;}

class B{ private int c; setC(int c){this.c=c;} }

Invariant φ_{enc}: A is the only class that holds references to object in b

Refinement: Reducing the PO Explosion

For all operations and for all invariants: RecoverInv

Consequences:

- PO Explosion, to a high degree non-modular!
- Coarse approximation to call correctness

Example

class A{ /*@instance invariant $\overbrace{\text{b.c=0;}}^{\varphi}$ */ private B b;}

class B{ private int c; setC(int c){this.c=c;} }

Invariant φ_{enc}: A is the only class that holds references to object in b

Refinement: Reducing the PO Explosion

For all operations and for all invariants: RecoverInv

Consequences:

- PO Explosion, to a high degree non-modular!
- Coarse approximation to call correctness

Example

class A{ /*@instance invariant $\overbrace{\text{b.c=0;}}^{\varphi}$ */ private B b;}

class B{ private int c; setC(int c){this.c=c;} }

Invariant φ_{enc}: A is the only class that holds references to object in b

System not provable correct with naive approach.

Refinement: Reducing the PO Explosion

For all operations and for all invariants: RecoverInv
Consequences:

- PO Explosion, to a high degree non-modular!
- Coarse approximation to call correctness

Example

class A{ /*@instance invariant $\overbrace{\text{b.c=0;}}^{\varphi}$ */ private B b;}
class B{ private int c; setC(int c){this.c=c;} }
Invariant φ_{enc}: A is the only class that holds references to object in b

System not provable correct with naive approach.
Advanced approach:
Only prove RecoverInv(φ_{enc}) and RecoverInv(φ) for A.

Work still in progress.

Outlook: Persistent Correctness

For open programs call correctness is not enough.

Obs calls operations in P only if

- one of the preconditions holds
- all invariants are satisfied

P is call correct if

1. every operation call to P in *Obs* fulfils the operation contracts attached to that operation
2. all invariants of all objects and classes hold after operation call to P

Outlook: Persistent Correctness

For open programs call correctness is not enough.

Obs calls operations in *P* only if

- one of the preconditions holds
- ~~all invariants are satisfied~~

P is persistently correct if

1. every operation call to *P* in *Obs* fulfils the operation contracts attached to that operation
2. ~~all invariants of all objects and classes hold after operation call to *P*~~

Outlook: Persistent Correctness

For open programs call correctness is not enough.

Obs calls operations in *P* only if

- one of the preconditions holds
- ~~all invariants are satisfied~~

P is persistently correct if

1. every operation call to *P* in *Obs* fulfils the operation contracts attached to that operation
2. ~~all invariants of all objects and classes hold after operation call to *P*~~
3. all invariants hold in every intermediate state of *Obs*

Outlook: Persistent Correctness

For open programs call correctness is not enough.

Obs calls operations in *P* only if

- one of the preconditions holds
- ~~all invariants are satisfied~~

P is persistently correct if

1. every operation call to *P* in *Obs* fulfils the operation contracts attached to that operation
2. ~~all invariants of all objects and classes hold after operation call to *P*~~
3. all invariants hold in every intermediate state of *Obs*

For closed programs: Call correctness $\Longleftrightarrow$ Persistent Correctness

Conclusions

- Current proof obligations of KeY are insufficient, need re-design (already started)

Conclusions

- Current proof obligations of KeY are insufficient, need re-design (already started)
- Purpose of structural/behavioural subtyping POs considered doubtful

Conclusions

- Current proof obligations of KeY are insufficient, need re-design (already started)
- Purpose of structural/behavioural subtyping POs considered doubtful
- Number of new design validation POs useful

Conclusions

- Current proof obligations of KeY are insufficient, need re-design (already started)
- Purpose of structural/behavioural subtyping POs considered doubtful
- Number of new design validation POs useful
- Notion of entire correctness of sequential Java programs (observational call / persistent correctness)

Conclusions

- Current proof obligations of KeY are insufficient, need re-design (already started)
- Purpose of structural/behavioural subtyping POs considered doubtful
- Number of new design validation POs useful
- Notion of entire correctness of sequential Java programs (observational call / persistent correctness)
- POs ensure these kinds of correctness