22c181:
Formal Methods in Software Engineering

The University of Iowa

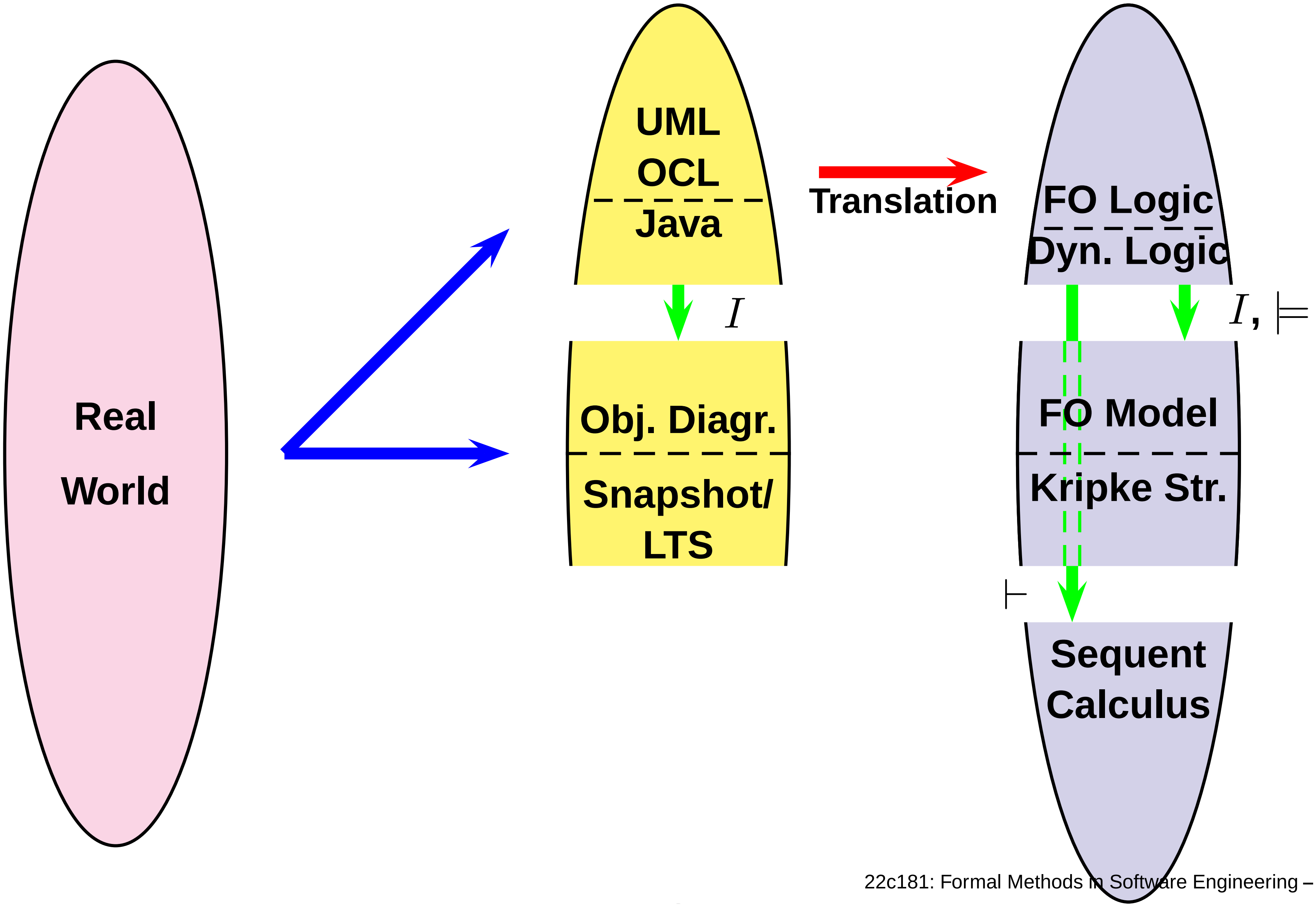
Spring 2008

From OCL to Typed First-order Logic

Contents

- Overview of KeY
- UML and its semantics
- Introduction to OCL
- Specifying requirements with OCL
- Modelling of Systems with Formal Semantics
- Propositional & First-order logic, sequent calculus
- OCL to Logic, horizontal proof obligations, using KeY
- Dynamic logic, proving program correctness
- Java Card DL
- Vertical proof obligations, using KeY
- Wrap-up, trends

Formal Verification



OCL Context Declarations as Universal Quantifiers

Classifier Context (Invariants)

context **typeName**

inv **‘Boolean OclExpression-with-self’**

OCL Context Declarations as Universal Quantifiers

Classifier Context (Invariants)

context **typeName**

inv **'Boolean OclExpression-with-self'**

Equivalent to universally quantified expression

inv **typeName.allInstances() –> forAll(x | OclExpression-with-x)**

OCL Context Declarations as Universal Quantifiers

Classifier Context (Invariants)

context typeName

  inv 'Boolean OclExpression-with-self'

Equivalent to universally quantified expression

  inv typeName.allInstances() –> forAll(x | OclExpression-with-x)

Example

context Person
  inv self.age >= 0

$\Rightarrow$

inv Person.allInstances() –>
    forAll(x | x.age >= 0)

Translating Universal Quantifiers from OCL to FOL

Universally quantified OCL expression

inv **typeName.allInstances –> forAll(x | OclExpression-with-x)**

Translating Universal Quantifiers from OCL to FOL

Universally quantified OCL expression

inv **typeName.allInstances –> forAll(x | OclExpression-with-x)**

Translation T to universal quantifier over variable x of type typeName

$$\forall x. T(\textbf{OclExpression-with-x})$$

Translating Universal Quantifiers from OCL to FOL

Universally quantified OCL expression

inv **typeName.allInstances –> forAll(x | OclExpression-with-x)**

Translation T to universal quantifier over variable x of type typeName

$$\forall x . T(\textbf{OclExpression-with-x})$$

Example

inv **Person.allInstances() –>**
forAll(x | x.age >= 0)

$$\stackrel{T}{\Rightarrow}$$

x : **Person**
$\forall x . (T(\textbf{x.age >= 0}))$

Quantification over Existing Objects

If x is variable of type C from UML context,
then $\forall x.\phi$ quantifies over all objects typeable with C

We want only the created objects in the current snapshot!

Quantification over Existing Objects

If x is variable of type C from UML context,
then $\forall x.\phi$ quantifies over all objects typeable with C

We want only the created objects in the current snapshot!

Assume that each class C has Boolean attribute $< \mathrm{created} >$

$\mathcal{I}(< \mathrm{created} >)(\mathbf{o})$ **is true iff $\mathbf{o}$ has been created in state described by $\mathcal{I}$**

Quantification over Existing Objects

If x is variable of type C from UML context,
then $\forall x.\phi$ quantifies over all objects typeable with C

We want only the created objects in the current snapshot!

Assume that each class C has Boolean attribute $<\mathrm{created}>$

$\mathcal{I}(<\mathrm{created}>)(\mathbf{o})$ **is true iff $\mathbf{o}$ has been created in state described by $\mathcal{I}$**

Instead of $\forall$ use quantifier $\dot{\forall}$ defined as:

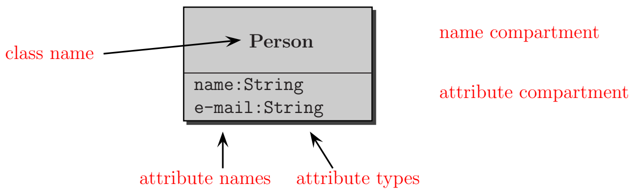
$$\dot{\forall} x.\phi \quad \textbf{<->} \quad \forall x.(x.<\mathrm{created}> \textbf{->} \phi)$$

Instead of $\exists$ use quantifier $\dot{\exists}$ defined as:

$$\dot{\exists} x.\phi \quad \textbf{<->} \quad \exists x.(x.<\mathrm{created}> \textbf{\&} \phi)$$

Translating OCL to FOL: Attributes

Attributes

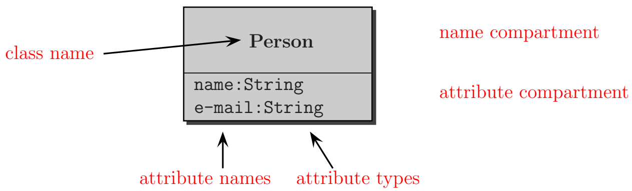


OCL constraint with attribute

x.age >= 0

Translating OCL to FOL: Attributes

Attributes



OCL constraint with attribute

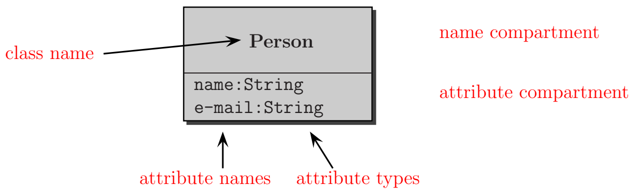
x.age >= 0

UML attribute semantics

$I(\texttt{age})$ **function from** $I(\textbf{Person})$ **to** $I(\texttt{int})$

Translating OCL to FOL: Attributes

Attributes



OCL constraint with attribute

x.age >= 0

UML attribute semantics

$I(\mathtt{age})$ **function from** $I(\mathbf{Person})$ **to** $I(\mathtt{int})$

FOL type hierarchy & signature **(fragment)**

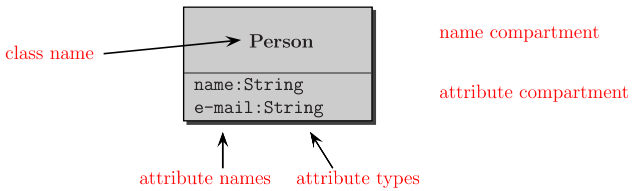
$\mathcal{T} = \{\mathbf{Person}, \ldots, \mathtt{int}, \ldots\}$

FSym $= \{\mathtt{age}\}$ **with** $\mathtt{age} : \mathbf{Person} \to \mathtt{int}$

PSym $= \{>=, <=, >, <, \ldots\}$

Translating OCL to FOL: Attributes

Attributes



OCL constraint with attribute

x.age >= 0

UML attribute semantics

$I(\mathtt{age})$ **function from** $I(\mathbf{Person})$ **to** $I(\mathtt{int})$

FOL type hierarchy & signature (fragment)

$\mathcal{T} = \{\mathbf{Person}, \ldots, \mathtt{int}, \ldots\}$

FSym $= \{\mathtt{age}\}$ **with** $\mathtt{age} : \mathbf{Person} \to \mathtt{int}$

PSym $= \{>=, <=, >, <, \ldots\}$

FOL translation

T**(x.age >= 0)** **=** $\mathtt{age}(x) >= 0$

Notational Conventions

Allow postfix-dot notation for functions that model attributes

Example

$$\mathtt{age}(x) >= 0 \quad \stackrel{T}{\Longrightarrow} \quad x.\mathtt{age} >= 0$$

In simple cases FOL translation looks exactly like OCL:

OCL expressions w/o iterators are alternative concrete syntax of FOL

Notational Conventions

Allow postfix-dot notation for functions that model attributes

Example

$$\mathtt{age}(x) >= 0 \quad \stackrel{T}{\Rightarrow} \quad x.\mathtt{age} >= 0$$

In simple cases FOL translation looks exactly like OCL:

OCL expressions w/o iterators are alternative concrete syntax of FOL

No generic types in Java Card and FOL (such as $\mathrm{Set}(\mathrm{Person})$**)**

Translation generates suitable flat types on-the-fly

$\mathrm{SetOfPerson}$, $\mathrm{SequenceOfPerson}$, **etc.**

Shorthand for sets of objects: $\mathrm{Vehicle}\{\}$, $\mathrm{Person}\{\}$, $\mathrm{int}\{\}$

Assorted Remarks

- **FOL translation of OCL attribute interpreted as total function**

  Value of an attribute might be `null`

- **Symbols with fixed interpretation for many OCL properties**

  $<=$**,** size**,** $\mathrm{includes}$**,** $+$**,** 17**,** self**,** result**, etc.**

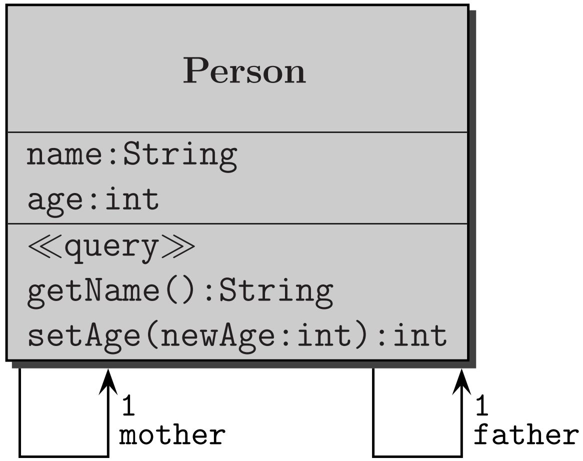
  Correct intended semantics guaranteed by sound calculus rules (automatically loaded)

- **If owner type of functions that model attributes and operations is required to resolve overloading, then write it in front:**

  $\mathrm{Person}::\mathrm{age}(\mathrm{x})$**,** $\mathrm{Person}\{\}::\mathrm{includes}(\mathrm{siblings}(\mathrm{self}), \mathrm{p})$

Translating OCL to FOL: Associations

Associations



Multiplicity 1: like attributes, but no dot notation

Function $\langle$**supplier-role-name**$\rangle : \langle$**client-type**$\rangle \to \langle$**supplier-type**$\rangle$

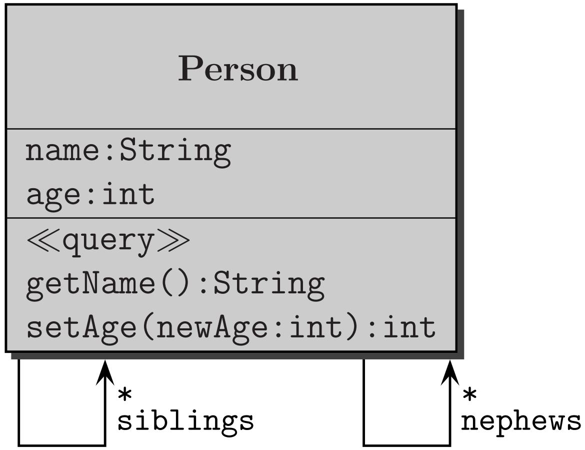
Example: $\mathbf{father} : \mathbf{Person} \to \mathbf{Person}$

Use explicit role name if present, otherwise default role name

$$\text{not}(\text{self.father} = \text{self.mother}) \quad \stackrel{T}{\Rightarrow} \quad \mathbf{!}(\mathbf{father}(self) \doteq \mathbf{mother}(self))$$

Translating OCL to FOL: Associations

Associations

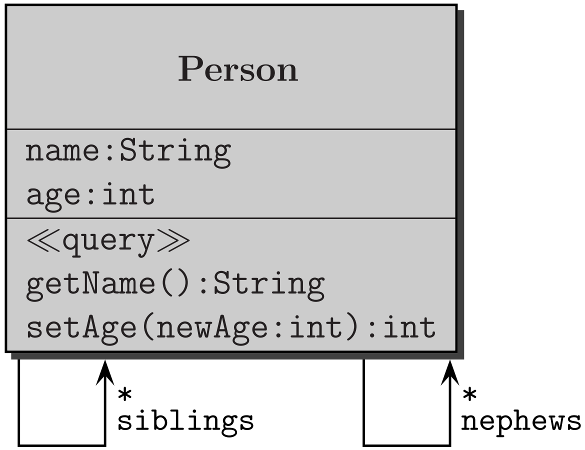


Other multiplicity than 1:

Function $\langle$**supplier-role-name**$\rangle$: $\langle$**client-type**$\rangle \rightarrow \langle$**Supplier-type**$\{\}\rangle$

Translating OCL to FOL: Associations

Associations



Other multiplicity than 1:

Function $\langle$**supplier-role-name**$\rangle$: $\langle$**client-type**$\rangle \rightarrow \langle$**Supplier-type**$\{\}\rangle$

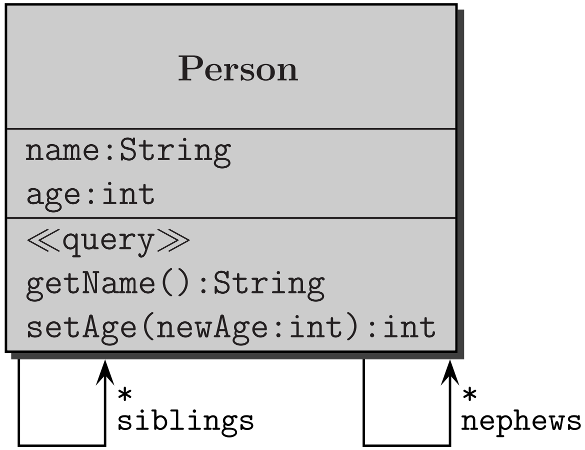
Example: $\textbf{siblings} : \textbf{Person} \rightarrow \textbf{Person}\{\}$

$$\text{self.siblings} = \text{self.nephews} \quad \overset{T}{\Rightarrow} \quad \textbf{siblings}(self) \doteq \textbf{nephews}(self)$$

Problem: **no rules for equality of sets of objects** $\Rightarrow$ **extensionality**

Translating OCL to FOL: Associations

Associations



Other multiplicity than 1:

Function $\langle$**supplier-role-name**$\rangle$: $\langle$**client-type**$\rangle \rightarrow \langle$**Supplier-type**$\{\}\rangle$

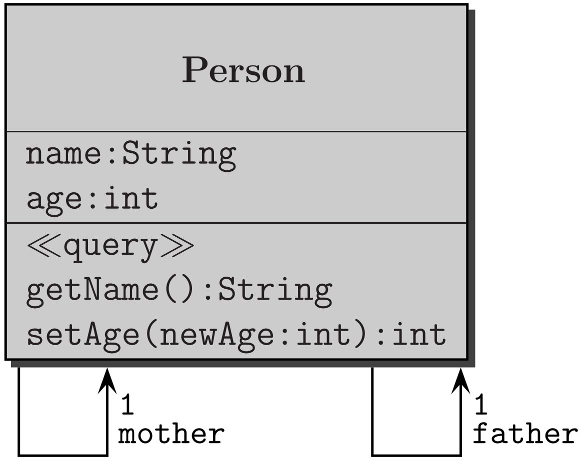
$\mathbf{siblings}(self) \doteq \mathbf{nephews}(self)$ **expanded into:**

$\dot{\forall} p.(\mathbf{Person}\{\}::\mathbf{includes}(\mathbf{siblings}(self), p)$

<->

$\mathbf{Person}\{\}::\mathbf{includes}(\mathbf{nephews}(self), p))$

Translating OCL to FOL: allInstances()



allInstances()

Argument of OCL quantifier forAll**,** exists

Analogous treatment to class context declaration

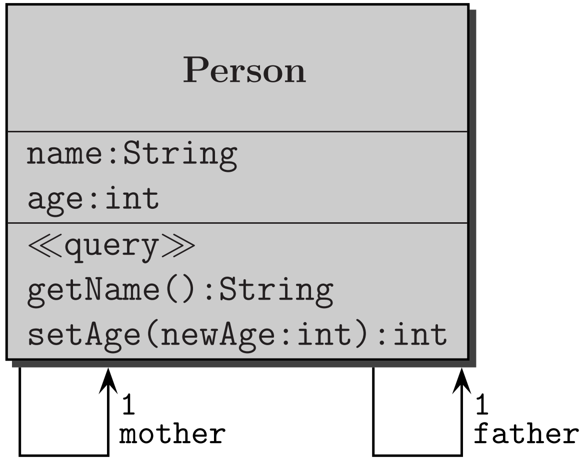
Example

Person.allInstances() -> forAll(age >= 0) $\stackrel{T}{\Rightarrow}$

$$\dot{\forall}\, x.(x.\mathsf{age} \!>\!=\! 0)$$

Translating OCL to FOL: allInstances()

allInstances()



Other collection property than quantifier

For $\mathrm{T.allInstances()}$ **create constant** $\mathbf{T\{\}::allInstances} : \rightarrow \mathbf{T\{\}}$

Add "definition" of $\mathbf{T\{\}::allInstances}$ **to goal antecedent:**

$$\dot{\forall} x.\ \mathbf{T\{\}::includes}(\mathbf{T\{\}::allInstances}, x)$$

Example for translation of allInstances()

Person.allInstances() -> size() = 1 $\overset{T}{\Rightarrow}$

$$\mathbf{Person\{\}}::size(\mathbf{Person\{\}::allInstances}) \doteq 1$$

Translating OCL to FOL: Important Issues

- **In many cases FOL translation follows OCL closely**
- **Some collection properties have complicated translations (select, reject)**

  Translator optimizes whenever possible
- **Sometimes, translation declares new function symbols**

  Definitions placed in antecedent (ie, left) of sequent arrow ==>
- **Details of translation (see also course web page):**

  B. Beckert, U. Keller, P Schmitt:
  Translating the OCL into First-order Predicate Logic

  A. Roth & P. Schmitt
  Formal Specification, Section 5.2.3

Horizontal Verification: Behavioural Subtyping

Substitution principle (Liskov, 1993)

Let ϕ be a property provable about objects x of type T.
Then ϕ should be true for objects y of type S where $S \sqsubseteq T$.

Horizontal Verification: Behavioural Subtyping

Substitution principle **(Liskov, 1993)**

Let ϕ be a property provable about objects x of type T.
Then ϕ should be true for objects y of type S where $S \sqsubseteq T$.

Consequence is invariant subtyping property:

Invariant of a class must imply invariant of all parent classes

Horizontal Verification: Behavioural Subtyping

Substitution principle (Liskov, 1993)

Let ϕ be a property provable about objects x of type T.
Then ϕ should be true for objects y of type S where $S \sqsubseteq T$.

Consequence is invariant subtyping property:

Invariant of a class must imply invariant of all parent classes

$\mathbf{inv}_S$ **is (FOL translation of) OCL invariant constraint of a class** S

$T_1, \ldots, T_n$ **parent classes and interfaces of** S

Proof obligation: $\dot{\forall}\, self.(\mathbf{inv}_S \text{ -> } (\mathbf{inv}_{T_1} \& \cdots \& \mathbf{inv}_{T_n}))$