

*L3 Mention Informatique
Parcours Informatique et MIAE*

Génie Logiciel Avancé - Advanced Software Engineering

Annotating UML with MOAL

Burkhardt Wolff

(burkhardt.wolff@universite-paris-saclay.fr)

<https://usr.lmf.cnrs.fr/~wolff>

Plan of the Chapter

- Syntax & Semantics of our own language

MOAL

- mathematical
- object-oriented
- UML-annotation
- language

(conceived as the „essence“ of annotation
languages like OCL, JML, Spec#, ACSL, ...)

Plan of the Chapter

❑ Concepts of MOAL

- Basis: Logic and Set-theory
- MOAL is a Typed Language
- Basic Types, Sets, Pairs and Lists
- Object Types from UML
- Navigation along UML attributes and associations

(Idea from OCL and JML)

❑ Purpose :

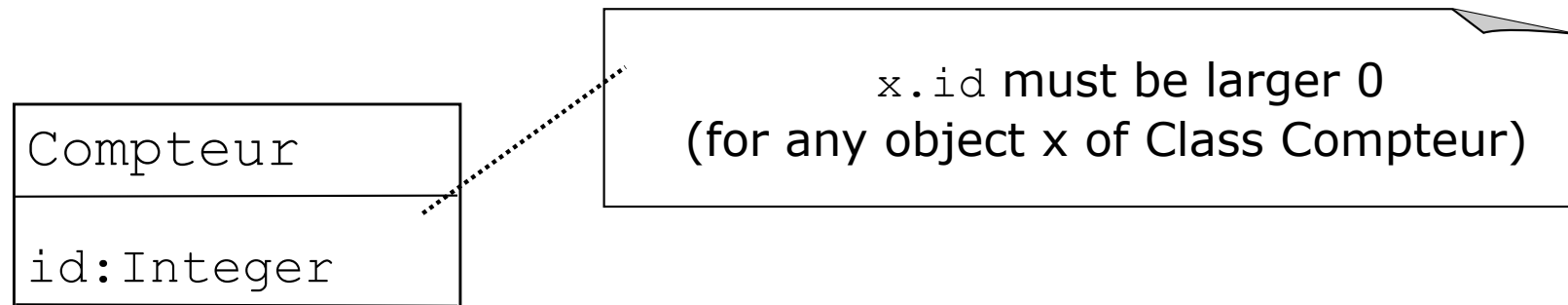
- Class Invariants
- Method Contracts with Pre- and Post-Conditions
- Annotated Sequence Diagrams for Scenarios, . . .

Plan of the Chapter

- ❑ Ultimate Goal:
Specify system components to improve analysis, design, test and verification activities
- ❑ . . . understanding how some analysis tools work . . .
- ❑ . . . understanding key concepts such as class invariants and contracts for analysis and design

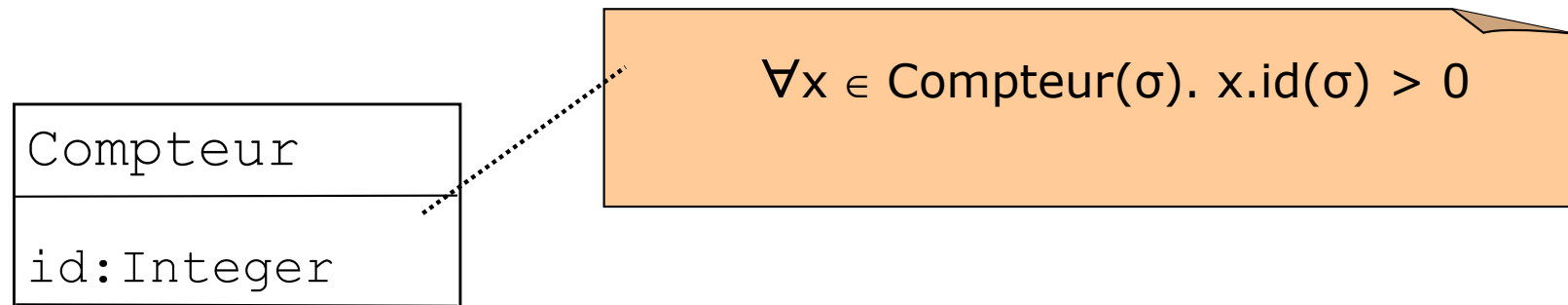
Motivation: Why Logical Annotations

- ❑ More precision needed
(like JML, VCC) that constrains an underlying **state σ**



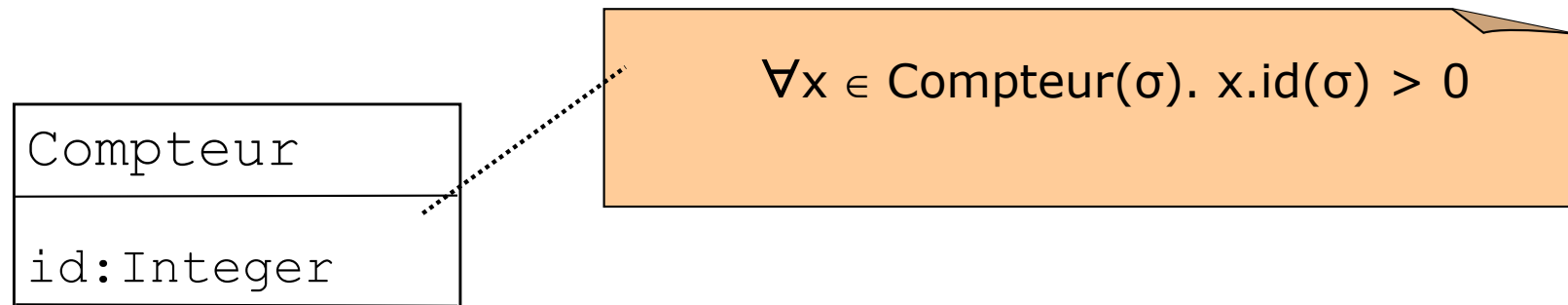
Motivation: Why Logical Annotations

- More precision needed
(like JML, VCC) that constrains an underlying **state** σ



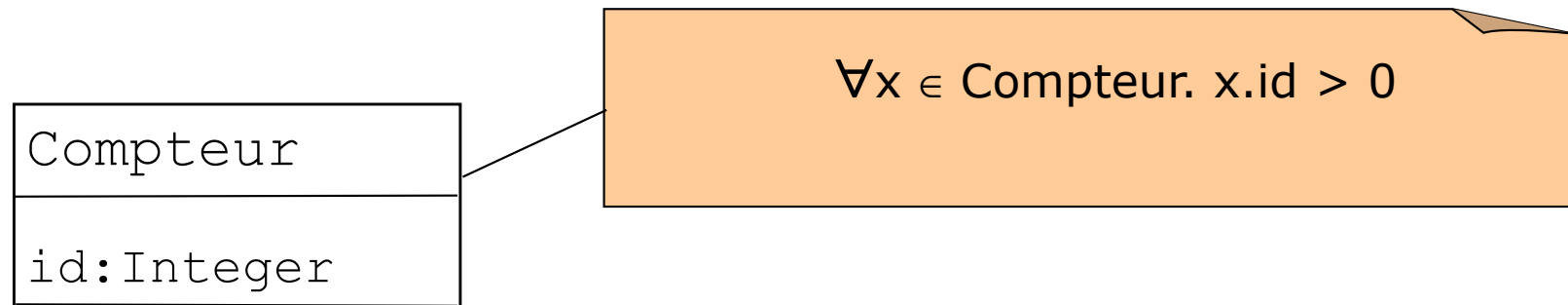
Motivation: Why Logical Annotations

- More precision needed
(like JML, VCC) that constrains an underlying **state σ**



Motivation: Why Logical Annotations

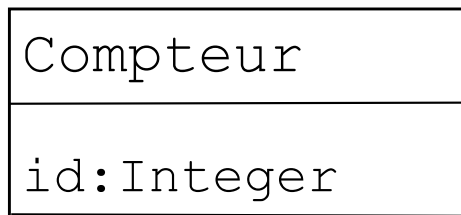
- More precision needed
(like JML, VCC) that constrains an underlying **state σ**



... by abbreviation convention if no confusion arises.

Motivation: Why Logical Annotations

- More precision needed
(like JML, VCC) that constrains an underlying **state σ**



definition $\text{inv}_{\text{Compteur}}(\sigma) \equiv \forall x \in \text{Compteur}(\sigma). x.\text{id}(\sigma) > 0$

... or by convention

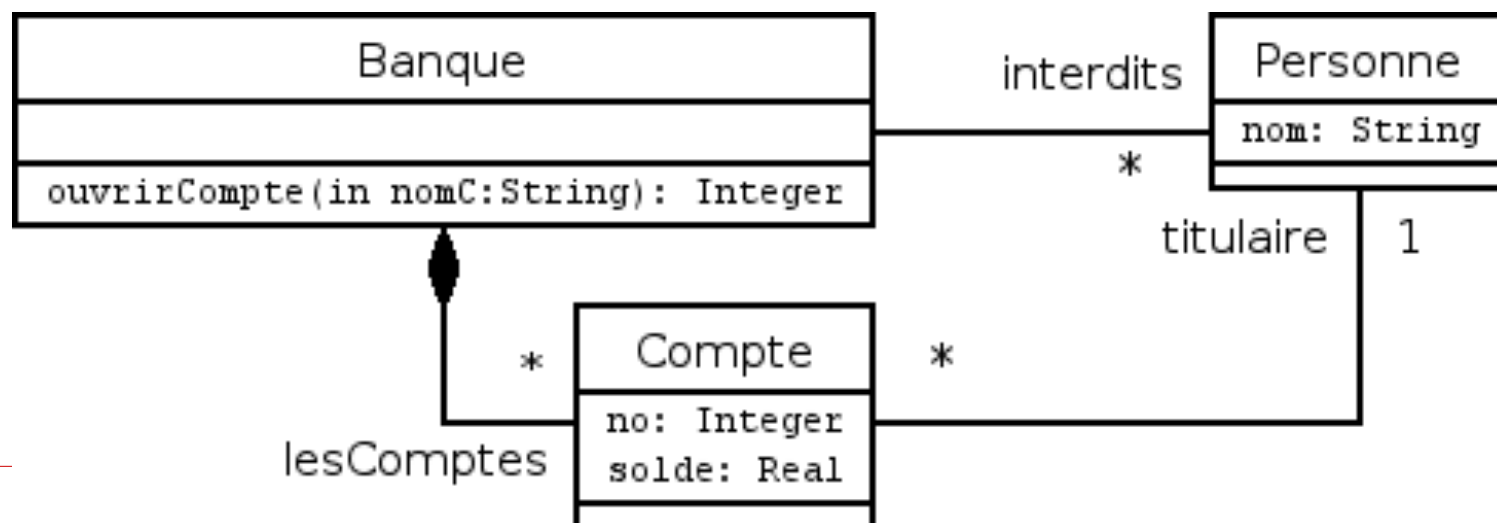
definition $\text{inv}_{\text{Compteur}} \equiv \forall x \in \text{Compteur}. x.\text{id} > 0$

... or as mathematical definition in a separate document or text ...

A first Glance to an Example: Bank

Opening a bank account. Constraints:

- ❑ there is a blacklist
- ❑ no more overdraft than 200 EUR
- ❑ there is a present of 15 euros in the initial account
- ❑ account numbers must be distinct.



A first Glance to an Example: Bank (2)

definition $\text{unique} \equiv \text{isUnique}(.no) (\text{Compte})$

definition $\text{noOverdraft} \equiv \forall c \in \text{Compte}. c.id \geq -200$

definition $\text{pre}_{\text{ouvrirCompte}}(b:\text{Banque}, \text{nomC}:\text{String}) \equiv$
 $\forall p \in \text{Personne}. p.nom \neq \text{nomC}$

definition $\text{post}_{\text{ouvrirCompte}}(b:\text{Banque}, \text{nomC}:\text{String}, r::\text{Integer}) \equiv$
 $|\{p \in \text{Personne} \mid p.nom = \text{nomC} \wedge p.isNew()\}| = 1$
 $\wedge |\{c \in \text{Compte} \mid c.titulaire.nom = \text{nomC}\}| = 1$
 $\wedge \forall c \in \text{Compte}. c.titulaire.nom = \text{nomC} \rightarrow c.solde = 15$
 $\wedge isNew(c)$

MOAL: a specification language?

- In the following, we will discuss the

MOAL Language in more detail ...

Syntax and Semantics of MOAL

□ The usual logical language:

- True, False
- negation : $\neg E$,
- or: $E \vee E'$, and: $E \wedge E'$, implies: $E \rightarrow E'$
- $E = E'$, $E \neq E'$,
- if C then E else E' endif
- let $x = E$ in E'

- Quantifiers on sets and lists:

$\forall x \in \text{Set}. P(x)$

$\exists x \in \text{Set}. P(x)$

Syntax and Semantics of MOAL

- ❑ MOAL is (like OCL or JML) a typed language.
 - Basic Types:
Boolean, Integer, Real, String
 - Pairs: $X \times Y$
 - Lists: List(X)
 - Sets: Set(X)

Syntax and Semantics of MOAL

□ The arithmetic core language.

expressions of type Integer or Real:

➤ $1, 2, 3 \dots$ resp. $1.0, 2.3, \pi$.

➤ $-E, E + E',$

➤ $E * E', E / E',$

➤ $\text{abs}(E), E \text{ div } E', E \text{ mod } E' \dots$

Syntax and Semantics of MOAL

❑ The expressions of type `String`:

- `S concat S'`
- `size(S)`
- `substring(i, j, S)`
- `'Hello'`

Syntax and Semantics of MOAL Sets

- $| S |$ size as Integer
- $\text{isUnique}(f)(S) \equiv \forall x, y \in S. f(x)=f(y) \rightarrow x=y$
- $\{\}, \{a, b, c\}$ empty and finite sets
- $e \in S, e \notin S$ is element, not element
- $S \subseteq S'$ is subset
- $\{x \in S \mid P(x)\}$ filter
- $S \cup S', S \cap S'$ union, intersect
between sets of same type
- Integer, Real, String ...
are symbols for the set
of all Integers, Reals, ...

Syntax and Semantics of MOAL Pairs

- (X, Y) pairing
- $\text{fst}(X, Y) = X$ projection
- $\text{snd}(X, Y) = Y$ projection

Syntax and Semantics of MOAL Lists

Lists S have the following operations:

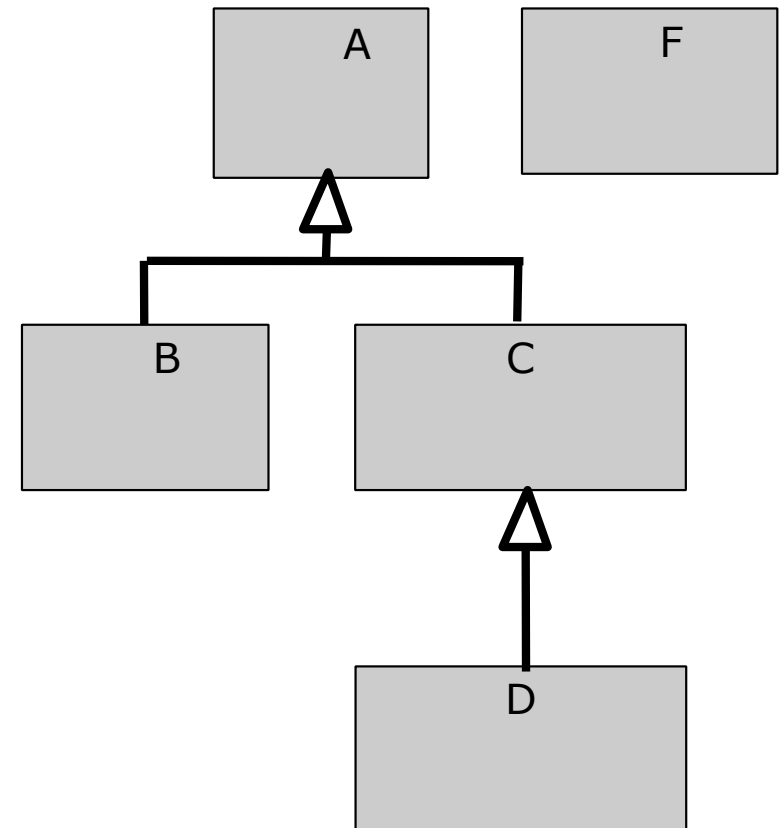
- $x \in L$ -- is element (overload!)
- $|S|$ -- length as Integer
- $\text{head}(L), \text{last}(L)$
- $\text{nth}(L, i)$ -- for i between 0 et $|S| - 1$
- $L @ L'$ -- concatenate
- $e \# S$ -- append at the beginning
- $\forall x \in \text{List}. P(x)$ -- quantifiers :
- $[x \in L \mid P(x)]$ -- filter
- Finally, denotations of lists: $[1, 2, 3], \dots$

Syntax and Semantics of Objects

- ❑ Objects and Classes follow the semantics of UML
 - inheritance / subtyping
 - casting
 - objects have an id
 - NULL is a possible value in each class-type
 - for any class A, we assume a function:

$$A(\sigma)$$

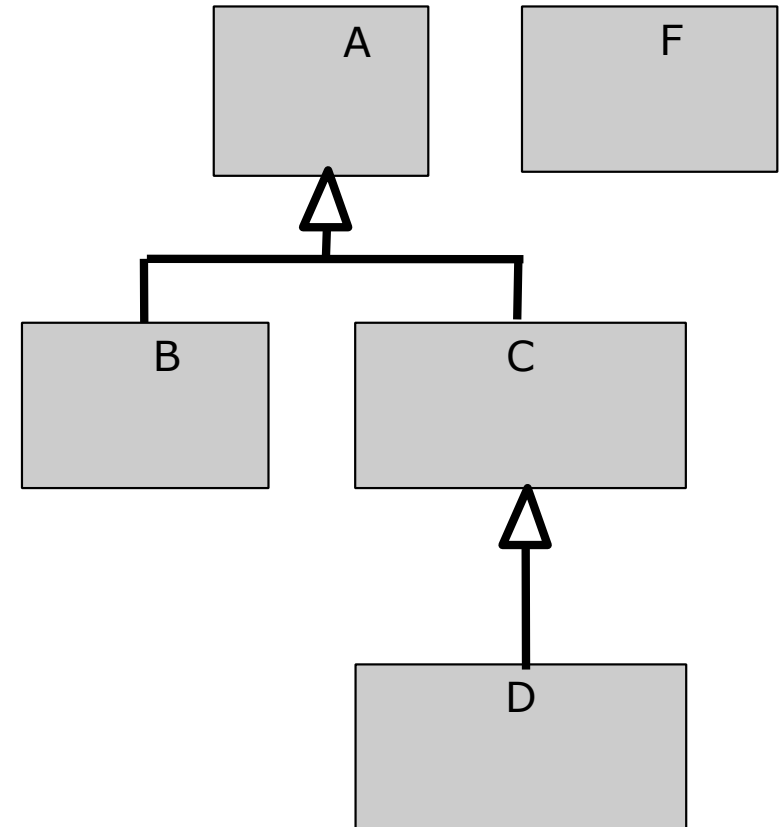
which returns the set of objects of class A
in state σ (the « instances » in σ).



Syntax and Semantics of Objects

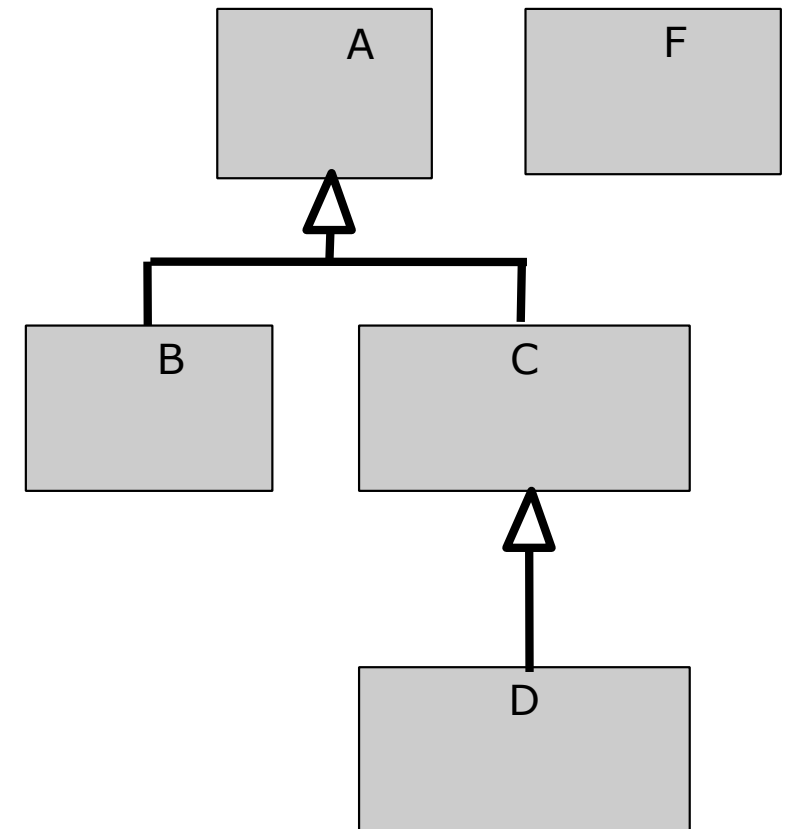
- ❑ Objects and Classes follow the semantics of UML

Recall that we will drop the index (σ) whenever it is clear from the context



Syntax and Semantics of Objects

- ❑ As in all typed object-oriented languages casting allows for converting objects.
- ❑ Objects have two types:
 - the « apparent type »
(also called static type)
 - the « actual type »
(the type in which an object was created)
 - casting changes the apparent type along the class hierarchy, but not the actual type



Syntax and Semantics of Objects

➤ Assume the creation of objects

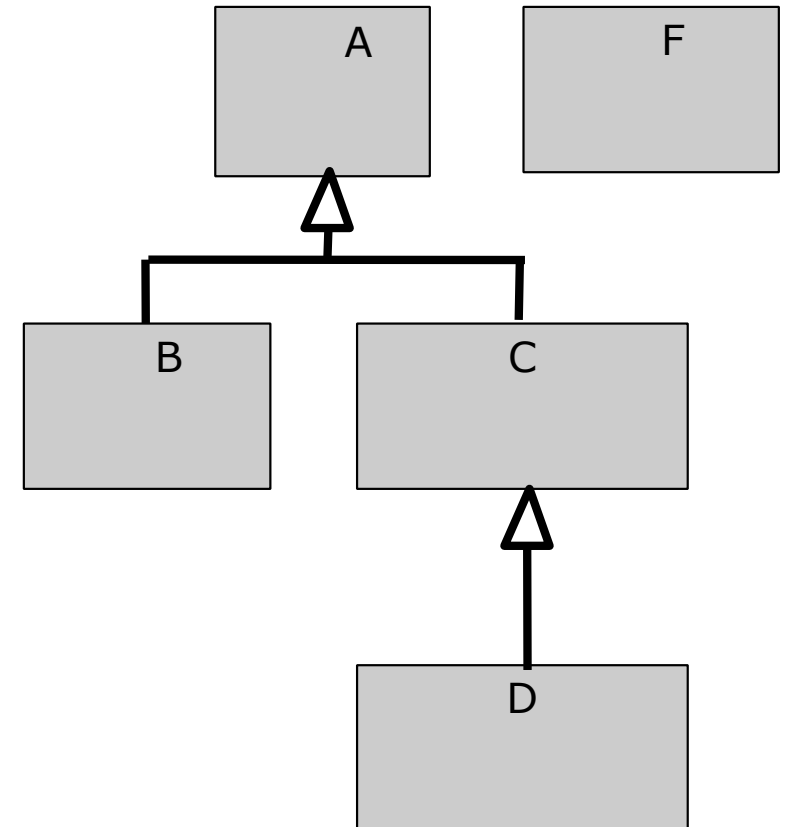
a in class A, b in class B,
c in class C, d in class D,

➤ Then casting:

`<F>b` is illtyped

`<A>b` has apparent type A,
but actual type B

`<A>d` has apparent type A,
but actual type D



Syntax and Semantics of OCL / UML

- We will also apply cast-operators to an entire set: So

$\langle A \rangle B(\sigma)$ (or just: $\langle A \rangle B$)
is the set of instances
of B casted to A .

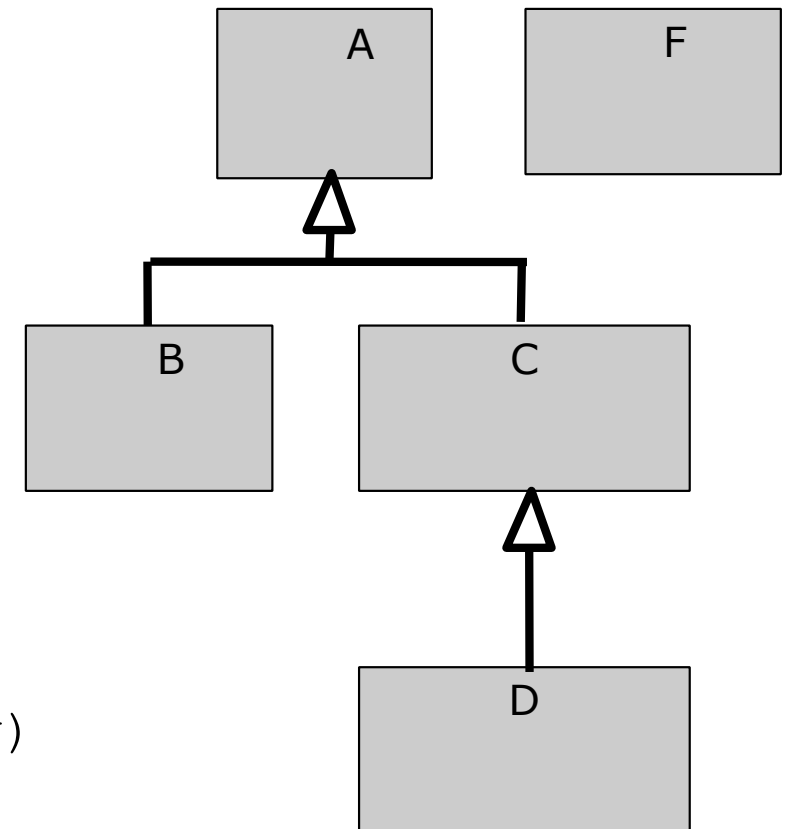
We have:

$$\langle A \rangle B \cup \langle A \rangle C \subseteq A$$

but:

$$\langle A \rangle B \cap \langle A \rangle C = \{\}$$

and also: $\langle A \rangle D \subseteq A$ (for all σ)



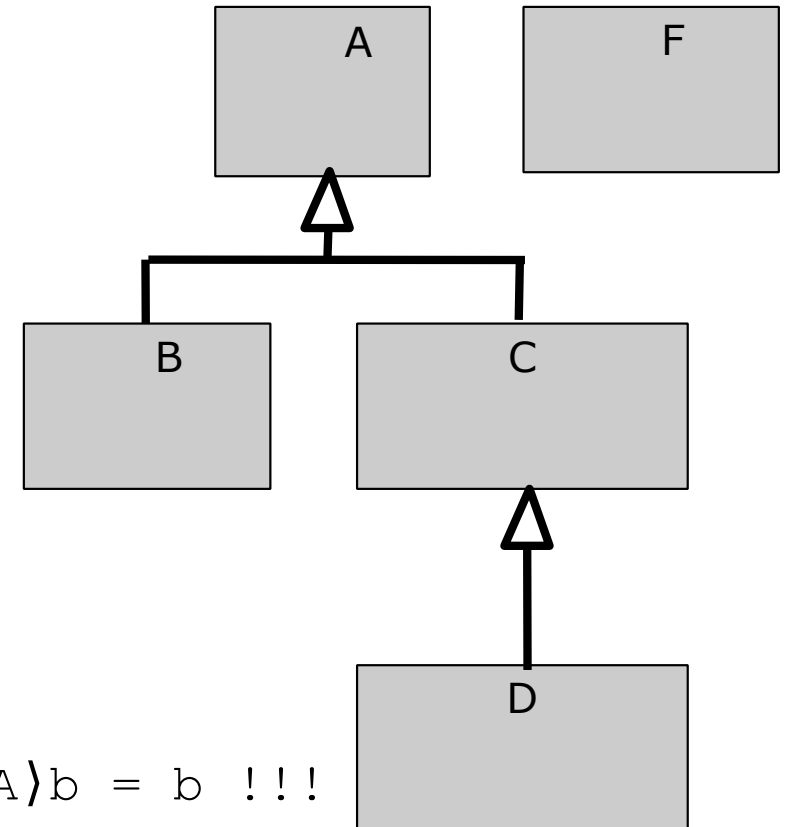
Syntax and Semantics of Objects

- Instance sets can be used to determine the actual type of an object:

$x \in B$

corresponds to Java's
`instanceof` or OCL's
`isKindOf`. Note that
casting does NOT change
the actual type:

$\langle A \rangle b \in B$, and $\langle B \rangle \langle A \rangle b = b$!!!



Syntax and Semantics of Objects

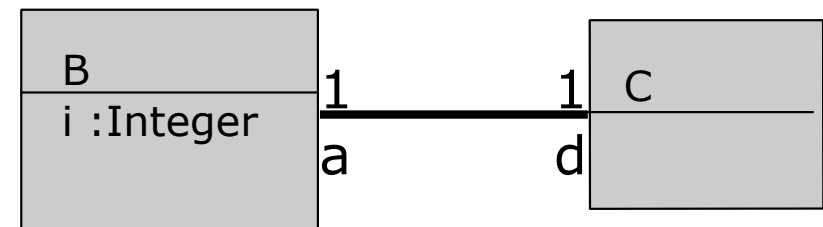
- ❑ Summary:
 - there is the concept of **actual** and **apparent** type
(anywhere outside of Java: **dynamic** and **static** type)
 - type tests check the former
 - type casts influence the latter,
but not the former
 - up-casts possible
 - down-casts invalid
 - consequence:
up-down casts are identities.

Syntax and Semantics of Object Attributes

- Objects represent structured, typed memory in a state σ . They have **attributes**.

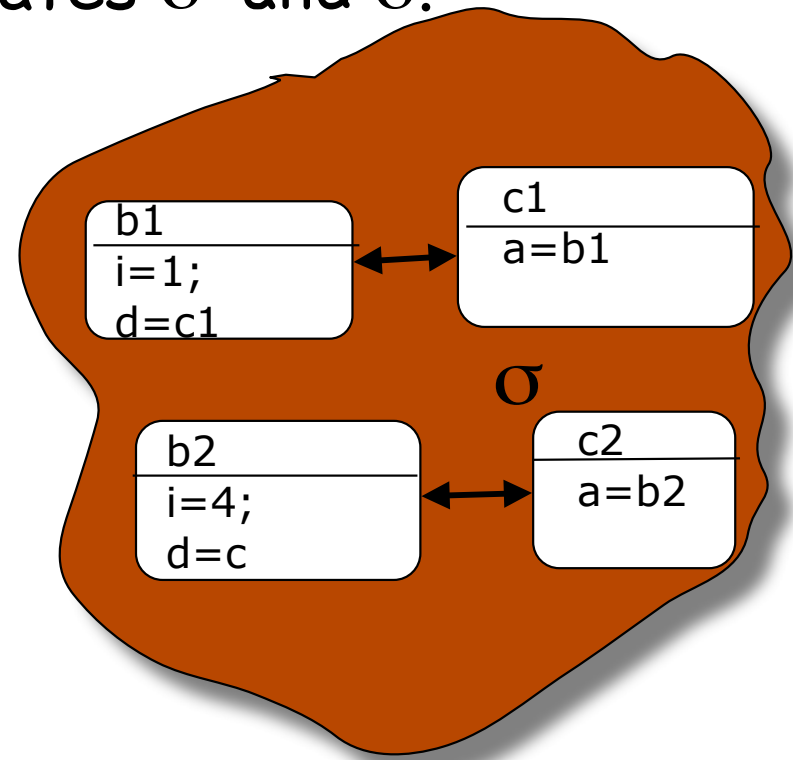
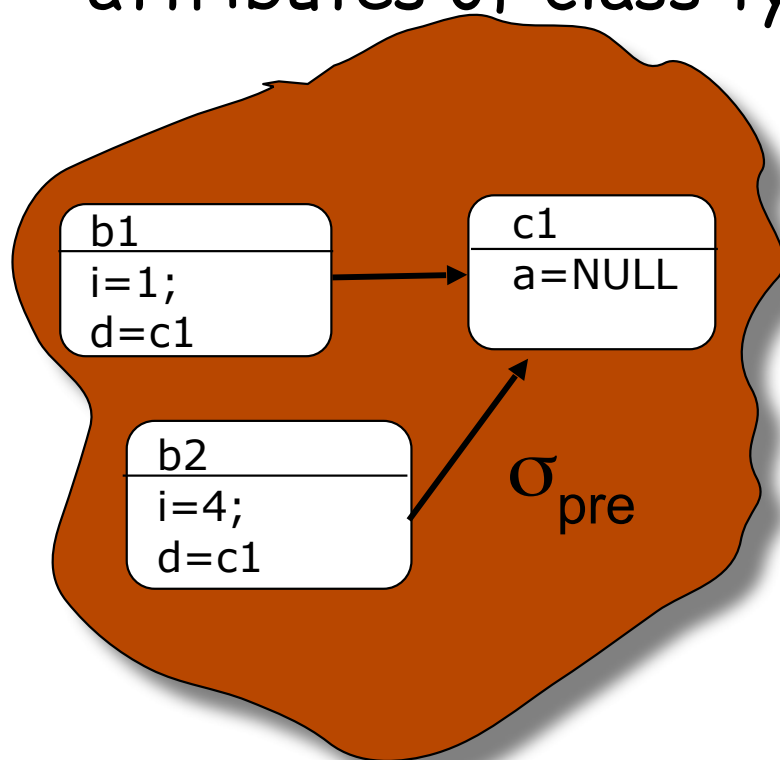
They can have class types.

- Reminder: In class diagrams, this situation is represented traditionally by Aggregations (somewhat sloppily: Associations)



Syntax and Semantics of Object Attributes

- Example:
attributes of class type in states σ' and σ .



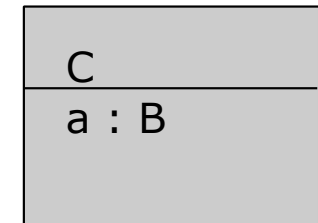
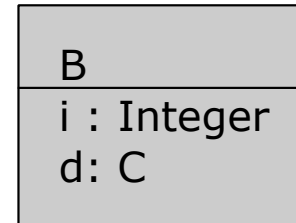
Syntax and Semantics of Object Attributes

- each attribute is represented by an *accessor-function* in MOAL. The class diagram right corresponds to the declaration of them:

$.i(\sigma) :: B \rightarrow \text{Integer}$

$.a(\sigma) :: C \rightarrow B$

$.d(\sigma) :: B \rightarrow C$



- This makes navigation expressions possible:

➤ $b1.d(\sigma) :: C$
 $c1.a(\sigma) :: B$

$b1.d(\sigma).a(\sigma).d(\sigma).a(\sigma) \dots$

Syntax and Semantics of Object Attributes

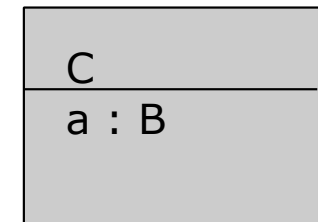
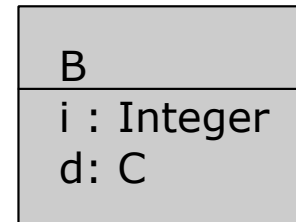
- each attribute is represented by a function in MOAL.

The class diagram right corresponds to declaration of accessor functions:

$.i(\sigma) :: B \rightarrow \text{Integer}$

$.a(\sigma) :: C \rightarrow B$

$.d(\sigma) :: B \rightarrow C$



- Applying the σ -convention, this makes the following navigation expression syntax possible:

➤ $b1.d :: C$

$c1.a :: B$

$b1.d.a.d.a \dots$

Syntax and Semantics of Object Attributes

- ❑ Assessor functions “dereferentiate” pointers in a given state
- ❑ Accessor functions of class type are

strict wrt. NULL.

➤ $\text{NULL}.d = \text{NULL}$
 $\text{NULL}.a = \text{NULL}$

- Note that navigation expressions depend on their underlying state:

$b1.d(\sigma_{\text{pre}}).a(\sigma_{\text{pre}}).d(\sigma_{\text{pre}}).a(\sigma_{\text{pre}}) = \text{NULL}$

$b1.d(\sigma).a(\sigma).d(\sigma).a(\sigma) = b1$!!!

(cf. Object Diagram pp 28)

Syntax and Semantics of Object Attributes

- ❑ Assessor functions “dereferentiate” pointers in a given state
- ❑ Accessor functions of class type are **strict** wrt. NULL.

- `NULL.d = NULL`
`NULL.a = NULL`

- The σ convention allows to write :

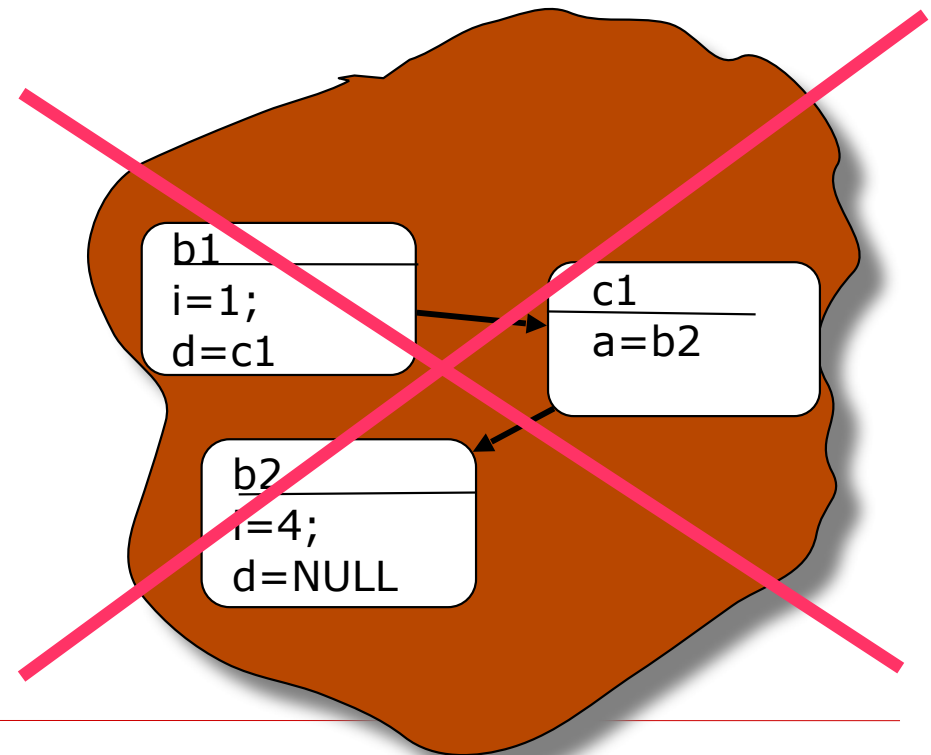
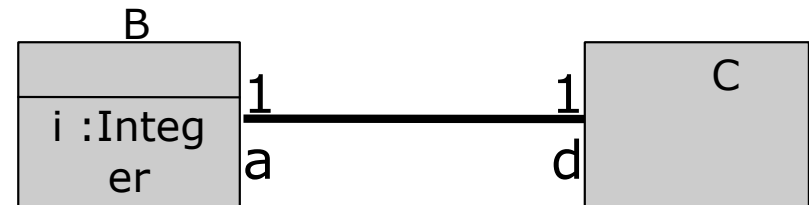
`old(b1.d.a.d.a) = NULL`
`b1.d.a.d.a = b1` !!!

(cf. Object Diagram pp 28)

Syntax and Semantics of Object Attributes

- Note that associations are meant to be « relations » in the mathematical sense.
(Here, we treat them like aggregations, which is strictly speaking a design step)

Thus, states (object-graphs) of this form do not represent an association of the cardinality 1 - 1:



Syntax and Semantics of Object Attributes

- ❑ This is reflected by 2
« association integrity
constraints ».



For the 1-1-case, they are:

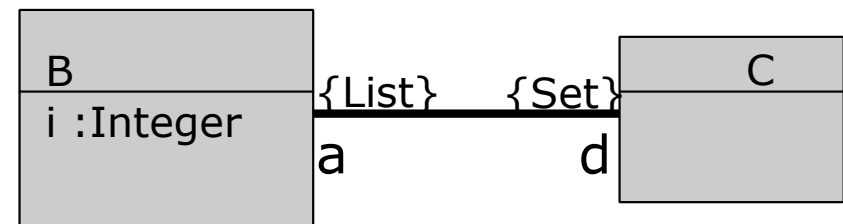
- definition $\text{ass}_{B.d.a} \equiv \forall x \in B. x.d.a = x$
- definition $\text{ass}_{C.a.d} \equiv \forall x \in C. x.a.d = x$

Syntax and Semantics of Object Attributes

- Attributes can be Lists or Sets of class types:



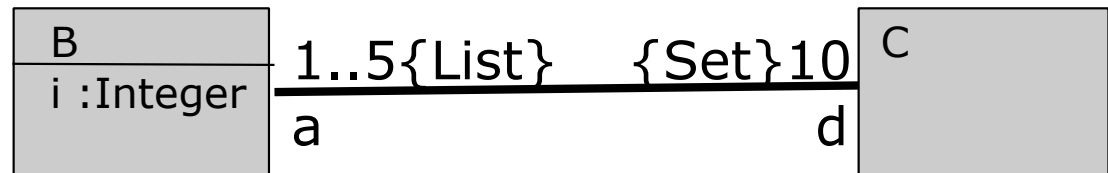
- Reminder: In class diagrams, this situation is represented traditionally by Associations (equivalent)



- In analysis-level Class Diagrams, the type information is still omitted; due to overloading of $\forall x \in X. P(x)$ etc. this will not hamper us to specify ...

Syntax and Semantics of Object Attributes

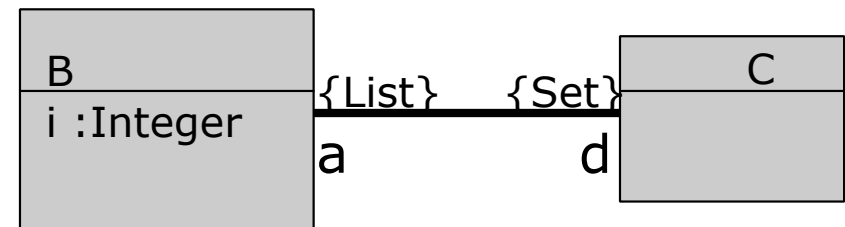
- Cardinalities in Associations can be translated canonically into MOCL invariants:



- > definition $\text{card}_{B.d} \equiv \forall x \in B. |x.d| = 10$
- > definition $\text{card}_{C.a} \equiv \forall x \in C. 1 \leq |x.a| \leq 5$

Syntax and Semantics of Object Attributes

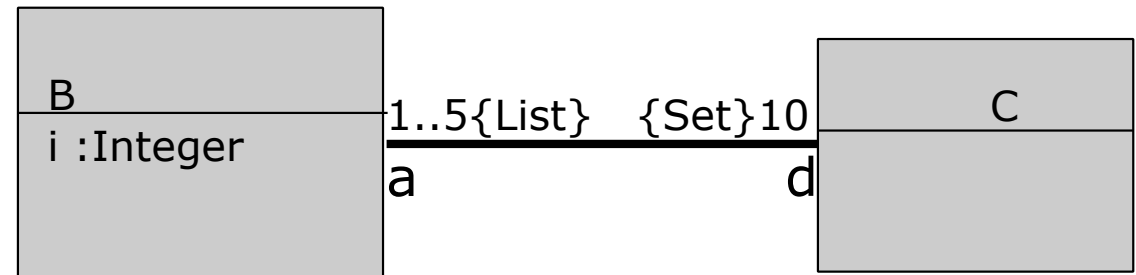
- ❑ Accessor functions are defined as follows for the case of NULL:



- `NULL.d = {}` -- mapping to the neutral element
- `NULL.a = []` -- mapping to the neutral element.

Syntax and Semantics of Object Attributes

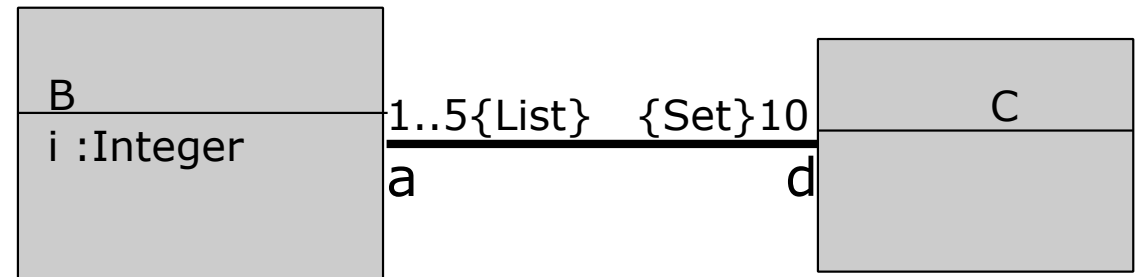
- Cardinalities in Associations can be translated canonically into MOCL invariants:



- > definition $\text{card}_{B.d} \equiv \forall x \in B. |x.d| = 10$
- > definition $\text{card}_{C.a} \equiv \forall x \in C. 1 \leq |x.a| \leq 5$

Syntax and Semantics of Object Attributes

- ❑ The corresponding association integrity constraints for the $*-*$ -case are:



➤ definition $ass_{B.d.a} \equiv \forall x \in B. x \in x.d.a$

➤ definition $ass_{C.a.d} \equiv \forall x \in C. x \in x.a.d$

Summary

- ❑ MOAL makes the UML to a real, formal specification language
- ❑ MOAL can be used to annotate Class Models, Sequence Diagrams and State Machines
- ❑ Working out, making explicit the constraints of these Diagrams is an important technique in the transition from Analysis documents to Designs.