L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

Annotating UML with MOAL

Burkhart Wolff
wolff@lri.fr

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:

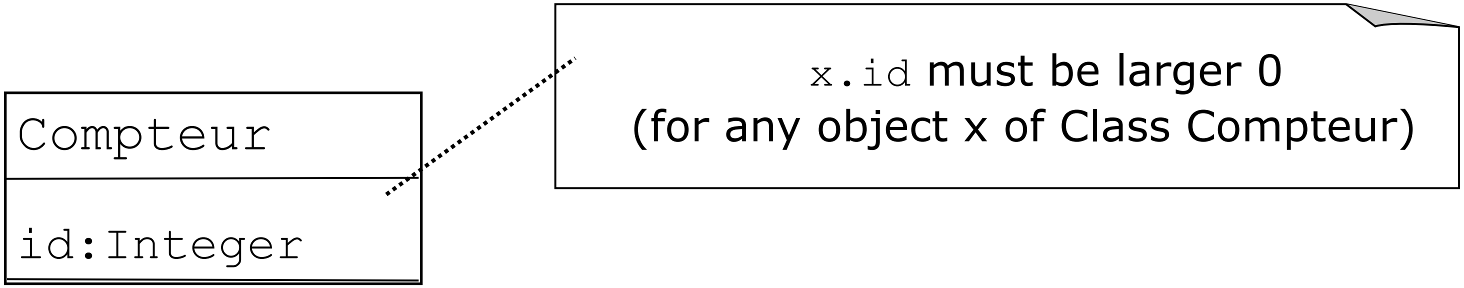
  Specify system components to improve analysis, design, test and verification activities

- . . . understanding how some analysis tools work . . .

- . . . understanding key concepts such as class invariants and contracts for analysis and design

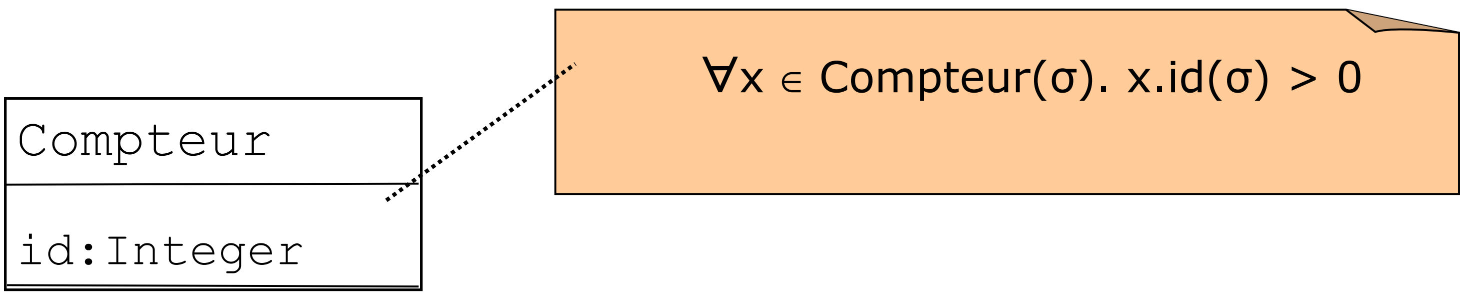
Motivation: Why Logical Annotations

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



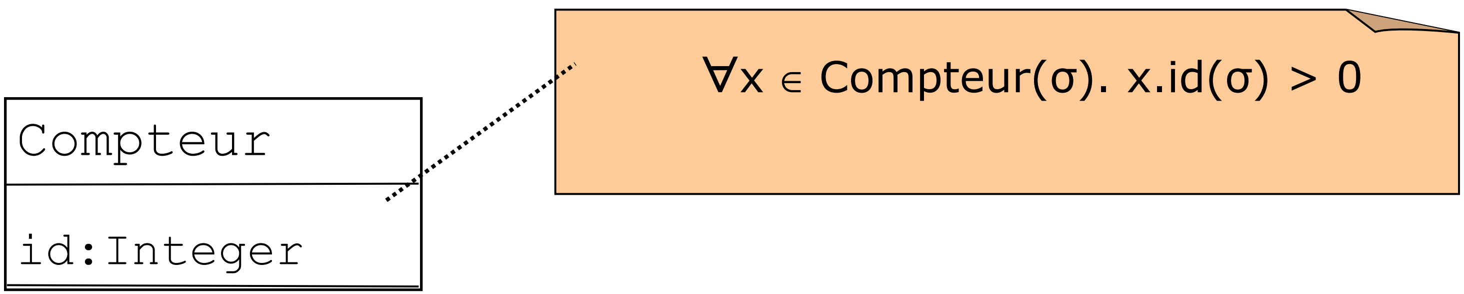
Motivation: Why Logical Annotations

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



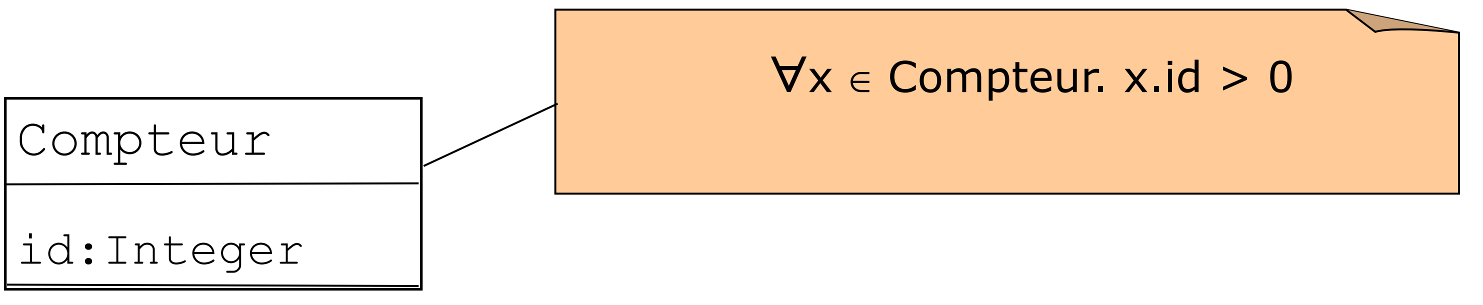
Motivation: Why Logical Annotations

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



Motivation: Why Logical Annotations

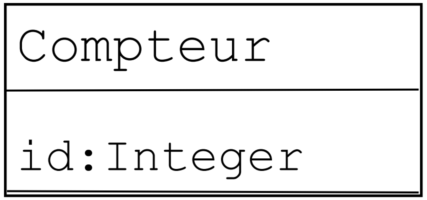
- More precision needed
  (like JML, VCC) that constrains an underlying state σ



... by abbreviation convention if no confusion arises.

Motivation: Why Logical Annotations

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



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

... or by convention

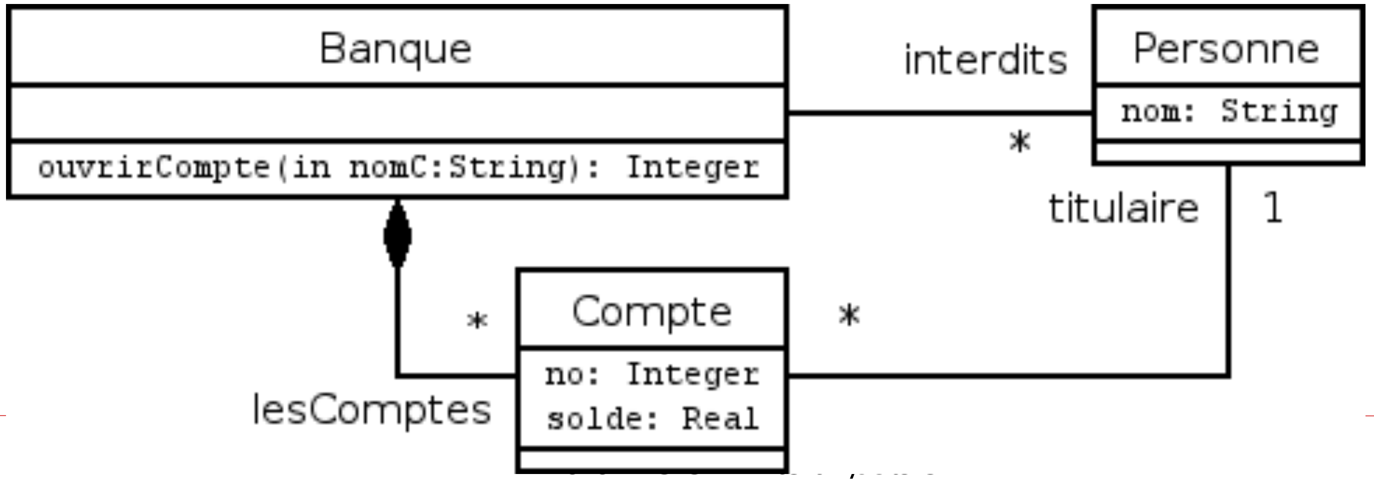
$$\text{definition inv}_{Compteur} \equiv \forall x \in Compteur.\ x.id > 0$$

... 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 unique ≡ isUnique(.no)(Compte)

definition noOverdraft ≡ ∀c ∈ Compte. c.id ≥ -200

definition $pre_{ouvrirCompte}$(b:Banque, nomC:String)≡
∀p ∈ Personne. p.nom ≠ nomC

definition $post_{ouvrirCompte}$(b:Banque,nomC:String,r::Integer)≡
|{p ∈ Personne | p.nom = nomC ∧ isNew(p)}| = 1
∧ |{c∈Compte | c.titulaire.nom = nomC}| = 1
∧ ∀c∈Compte. c.titulaire.nom = nomC ⟶ c.solde = 15
∧ isNew(c)

MOAL: a specification langage?

- 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: $E \lor E'$, and: $E \land E'$, implies: $E \longrightarrow E'$
  - $E = E'$, $E \neq E'$,
  - if C then E else E' endif
  - let x = E in E′
  - Quantifiers on sets and lists:

$\forall x \in Set.\ P(x)$ $\exists x \in Set.\ P(x)$

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 ...   resp. 1.0, 2.3, pi.`
  - `- `*E*`, `*E*` + `*E*`',`
  - *E*` * `*E*`', `*E*` / `*E*`',`
  - `abs(E), `*E*` div E', E mod E'...`

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
- $isUnique(f)(S) \equiv \forall x,y \in S.\ f(x)=f(y) \longrightarrow x=y$
- {}, {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
- `fst(X,Y) = X` projection
- `snd(X,Y) = Y` projection

Syntax and Semantics of MOAL Lists

Lists *S* have the following operations:

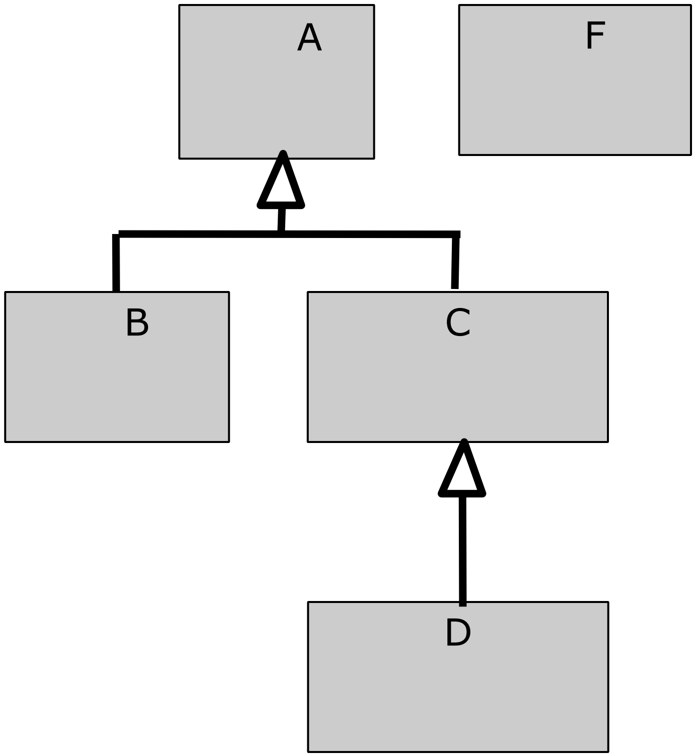
- x ∈ L -- is element (overload!)
- |S| -- length as Integer
- head(L),last(L)
- *nth*(L,*i*) -- for i between 0 et |S|-1
- *L@L'* -- concatenate
- *e#S* -- append at the beginning
- ∀x ∈ List. P(x) -- quantifiers :
- [x∈L | P(x)] -- filter
- Finally, denotations of lists: [1,2,3], ...

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:

    $$A(\sigma)$$

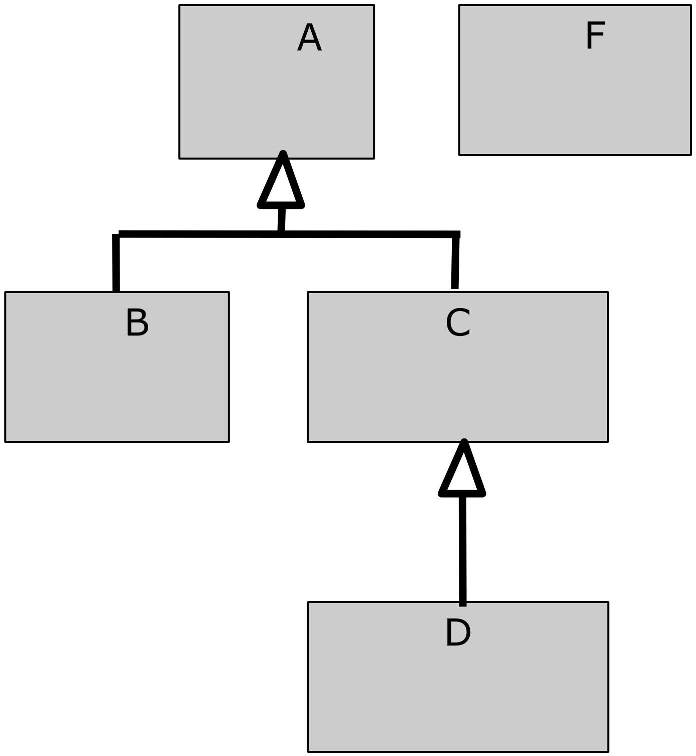
    which returns the set of objects of class A in state σ (the « instances » in σ).



Syntax and Semantics of Objects

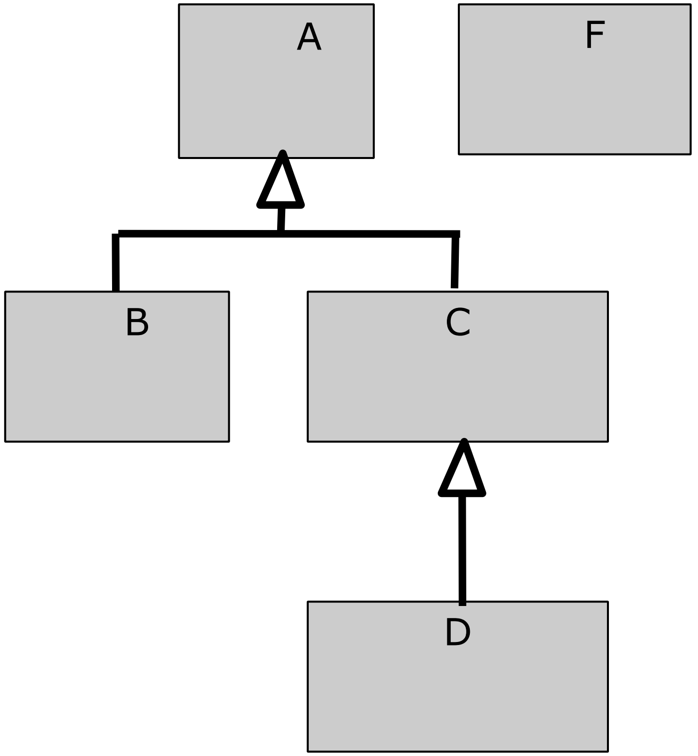
- Objects and Classes follow the semantics of UML

  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

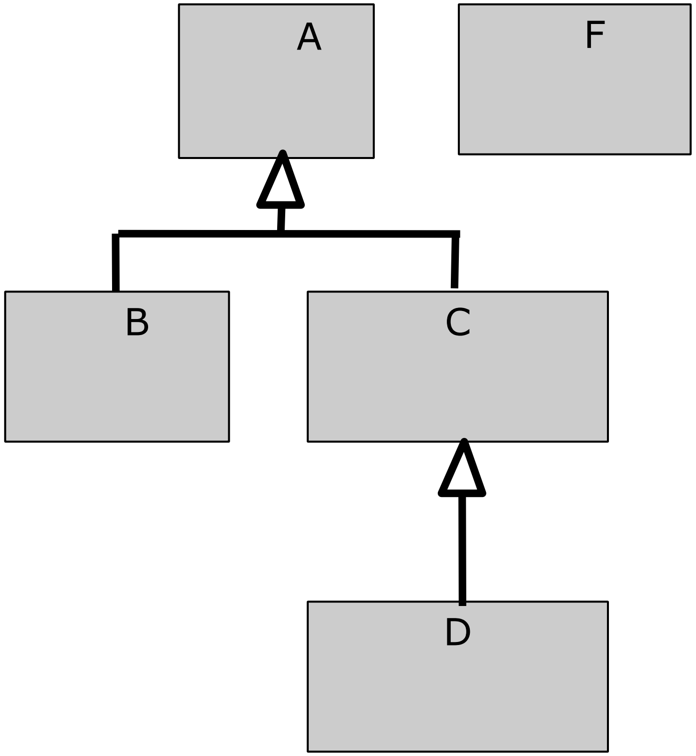
  `a in class A,b in class B,`
  `c in class C,d in class D,`

- 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

  ⟨A⟩B(σ) (or just: ⟨A⟩B)
  is the set of instances
  of B casted to A.

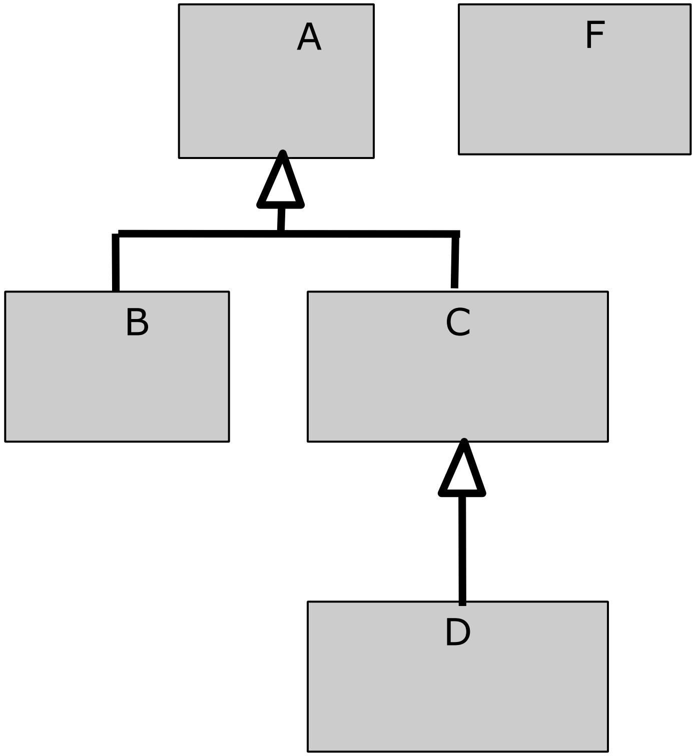
  We have:

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

  but:

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

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



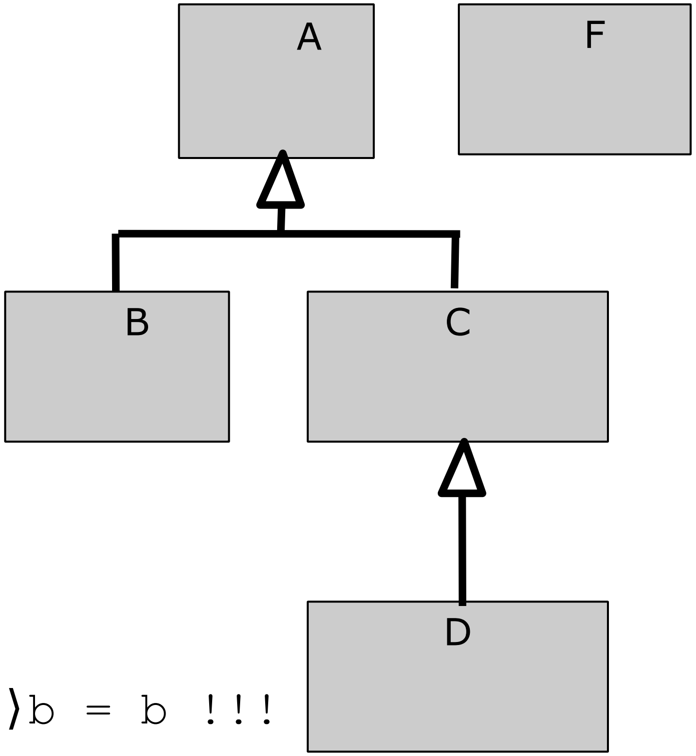
Syntax and Semantics of Objects

- 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

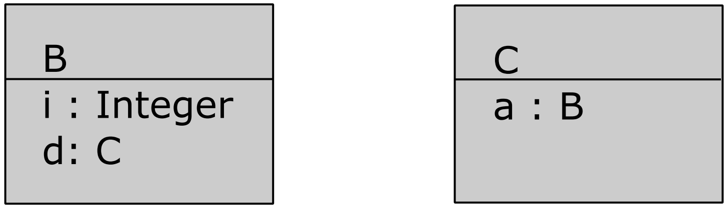
- Summary:
  - there is the concept of **actual** and **apparent** type (anywhere outside of Java: **dynamic** and **static** type)
  - type tests check the former
  - type casts influence the latter,

  but not the former
  - up-casts possible
  - down-casts invalid
  - consequence:
    up-down casts are identities.

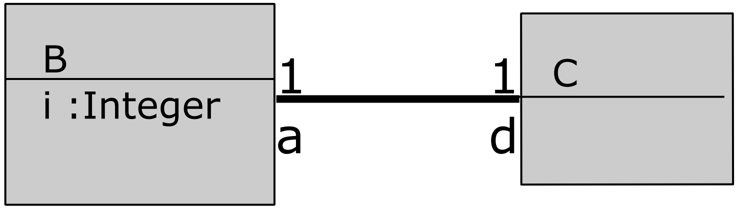
Syntax and Semantics of Object Attributes

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

  They can have class types.



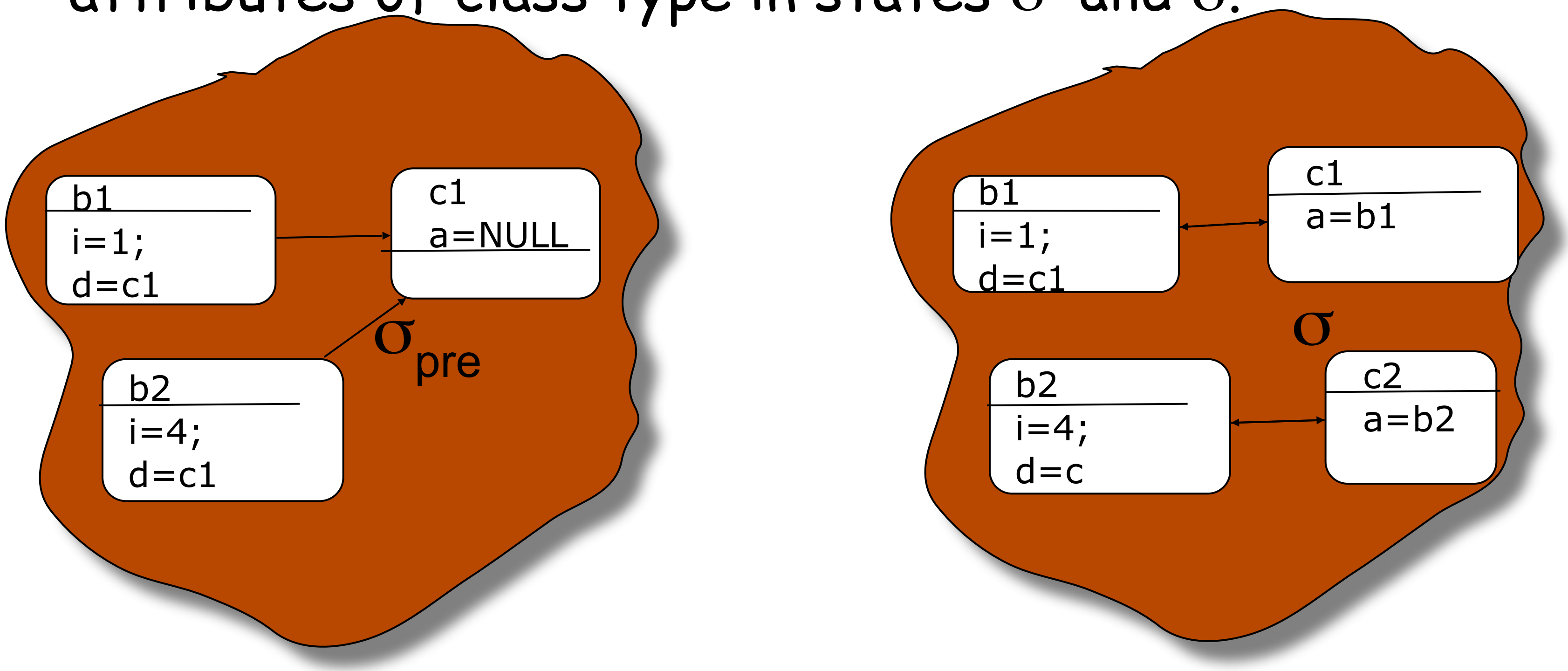
- Reminder: In class diagrams, this situation is represented traditionally by Aggregations (somewhat sloppily: Associations)



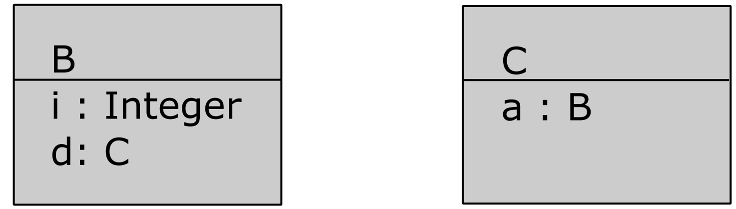
Syntax and Semantics of Object Attributes

- 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(σ) :: B -> Integer

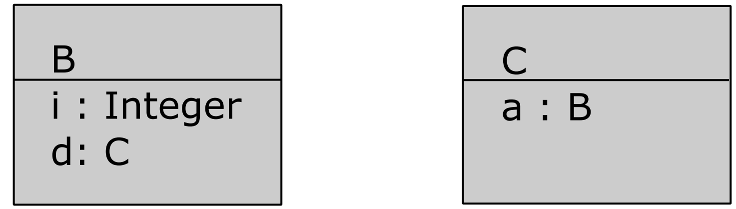
  .a(σ) :: C -> B

  .d(σ) :: B -> C

- This makes navigation expressions possible:
  - ```
    b1.d(σ) :: C
    c1.a(σ) :: B
    ```

```
b1.d(σ).a(σ).d(σ).a(σ) ...
```
```

Syntax and Semantics of Object Attributes



- each attribute is represen-ted by a function in MOAL. The class diagram right corresponds to delaration of accessor functions:

  .i(σ) :: B -> Integer

  .a(σ) :: C -> B

  .d(σ) :: B -> C

- Applying the σ–convention, this makes the following navigation expression syntax possible:
  - ```
    b1.d :: C
    c1.a :: B                b1.d.a.d.a  ...
    ```
    ```

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
  - Note that navigation expressions depend on their underlying state:
    b1.d(σ_{pre}) .a(σ_{pre}) .d(σ_{pre}) .a(σ_{pre}) = NULL
    b1.d(σ).a(σ).d(σ).a(σ) = b1 !!!
    (cf. Object Diagram pp 28)

Syntax and Semantics of Object Attributes

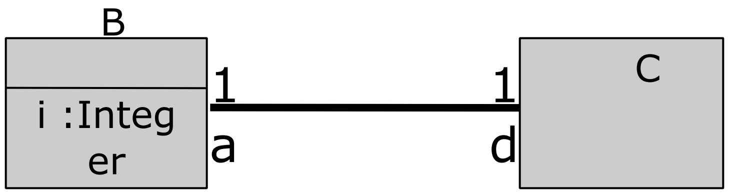
- Assessor functions "dereferentiate" pointers in a given state
- Accessor functions of class type are strict wrt. NULL.
  - NULL.d = NULL
NULL.a = NULL
  - The σ convention allows to write :

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

b1.d.a.d.a = b1 !!!

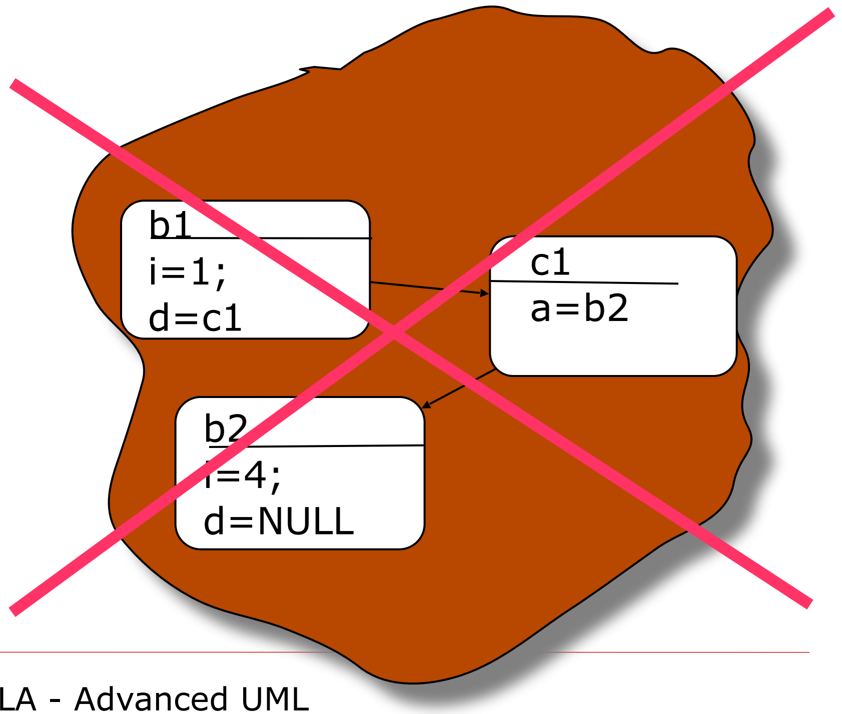
(cf. Object Diagram pp 28)

Syntax and Semantics of Object Attributes

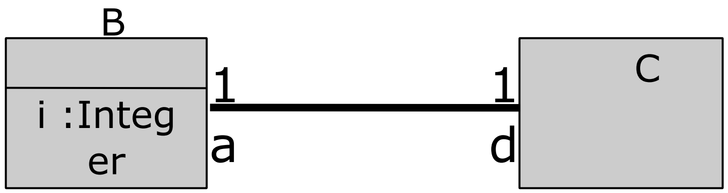


- Note that associations are meant to be « relations » in the mathematical sense.
  (Here, we treat them like aggregations, which is strict-ly speaking a design step)

  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 ».

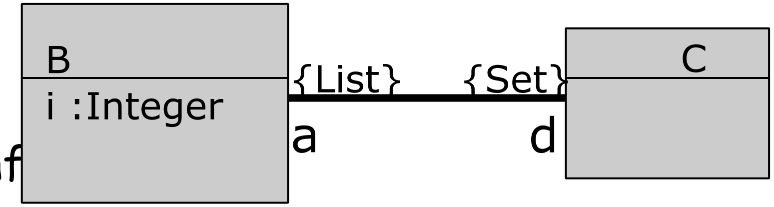
  For the 1-1-case, they are:

> - definition $ass_{B.d.a} \equiv \forall x \in B.\ x.d.a = x$
> - definition $ass_{C.a.d} \equiv \forall x \in C.\ x.a.d = x$

Syntax and Semantics of Object Attributes

- Attibutes can be Lists or Sets of class types:

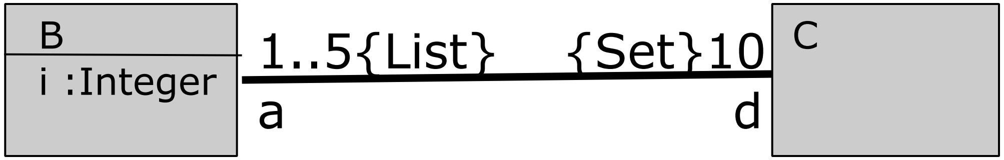
- Reminder: In class diagrams, this situation is represented traditionally by Associations (equivalent)



- In analysis-level Class Diagrams, the type inf 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

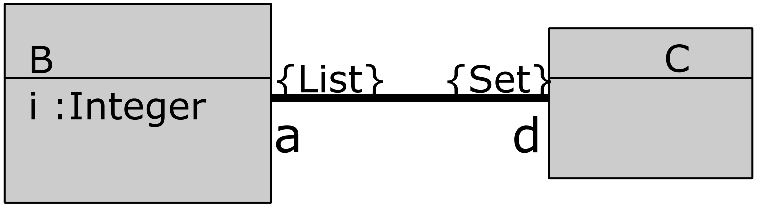
- Cardinalities in Associations can be translated canonically into MOCL invariants:



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

Syntax and Semantics of Object Attributes

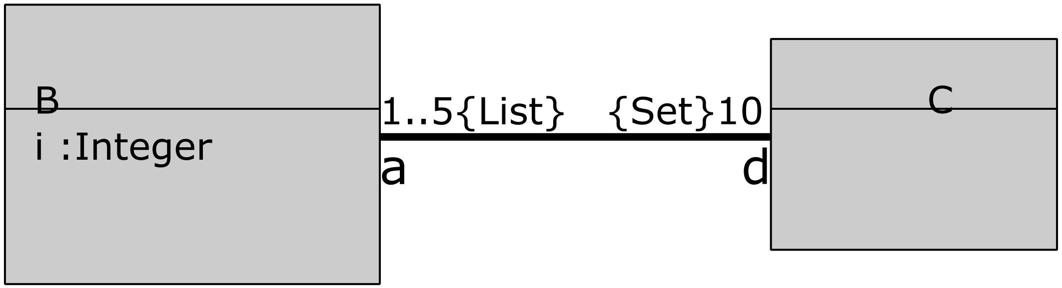
- Accessor functions are defined as follows for the case of NULL:



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

Syntax and Semantics of Object Attributes

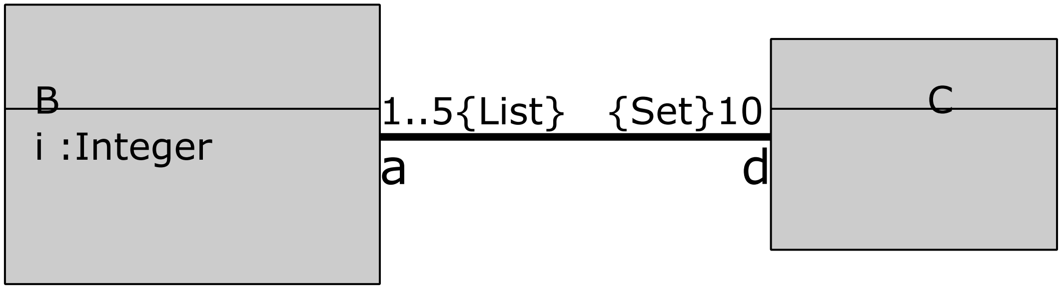
- Cardinalities in Associations can be translated canonically into MOCL invariants:



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

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.