L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

UML with MOAL-Contracts

Burkhart Wolff
wolff@lri.fr

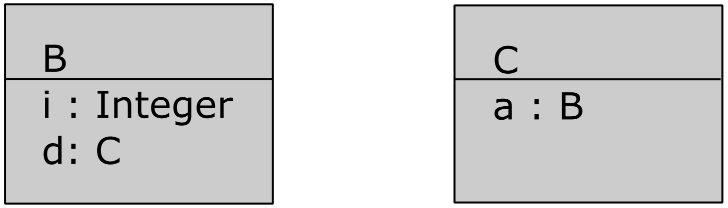
Recall:

- MOAL is a logic used to make UML diagrams more precise
- it comprises
  - typed sets, lists, and some base types
  - classes and objects from UML class diagrams
  - subtyping and casts
  - a semantics for path navigation and associations.

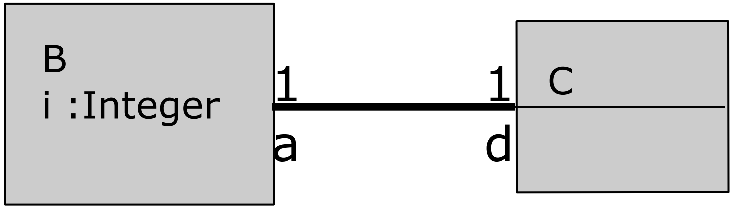
Recall: Object Attributes

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

  They can have class types.



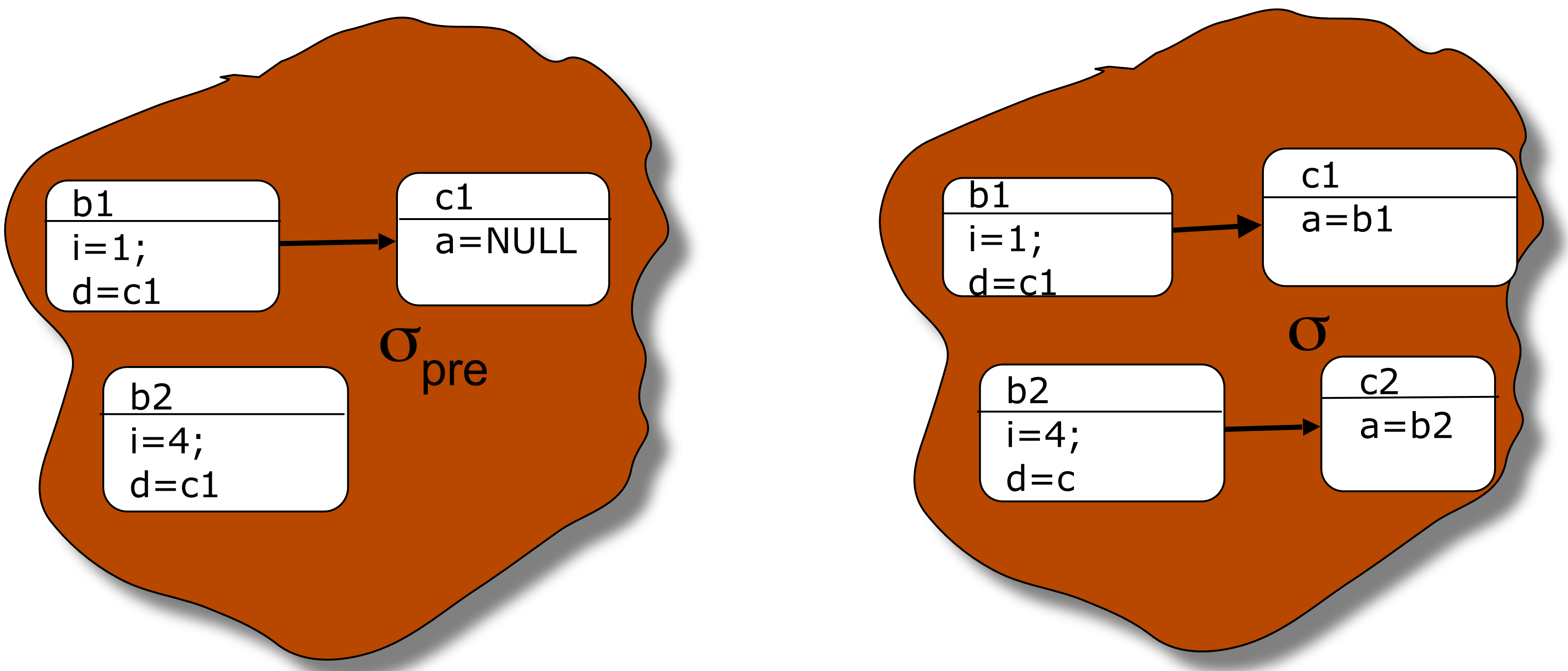
- Reminder: In class diagrams, this situation is represented traditionally by Associations (equivalent)



Syntax and Semantics of Object Attributes

- Example:

  attributes of class type in states σ_{pre} and σ.



Recall Navigation

- Object assessor functions are „dereferentiations of pointers in a state"
- Accessor functions of class type are strict wrt. NULL.
  - NULL.d = NULL
    NULL.a = NULL
  - Recall that navigation expressions depend on their underlying state:
    $$b1.d(\sigma_{pre}).a(\sigma_{pre}).d(\sigma_{pre}).a(\sigma_{pre}) = NULL$$
    $$b1.d(\sigma).a(\sigma).d(\sigma).a(\sigma) = b1 \quad !!!$$
    (cf. Object Diagram pp 28)

Recall Object Attributes

- Object assessor functions are „dereferentiations of pointers in a 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

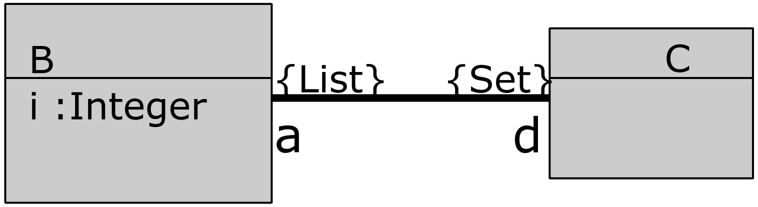
`b1.d.a.d.a` = b1 !!!

(cf. Object Diagram pp 28)

Recall Object Attributes

- Attibutes can be List or Sets of class types:

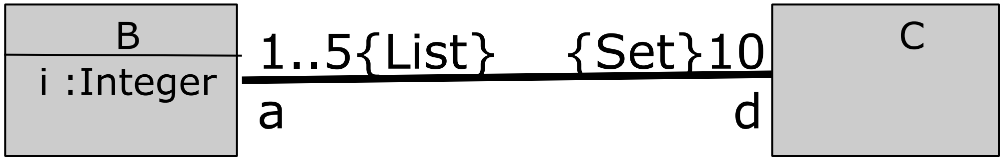
- Reminder: In class diagrams, this situation is represented traditionally by Associations (equivalent)



- 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 hurt ...

Recall Cardinalities vs Invariants

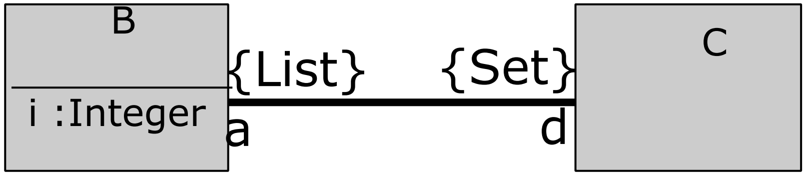
- Cardinalities in Associations can be translated canonically into MOCL invariants:



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

Strictness of Collection Attributes

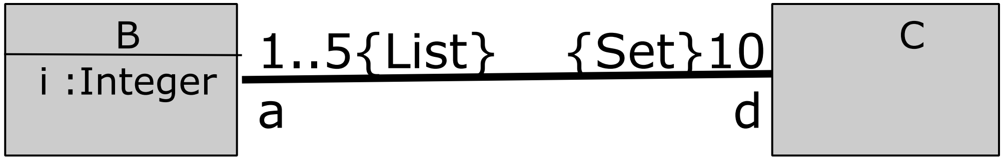
- Accessor functions are defined as follows for the case of NULL:



  - 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:

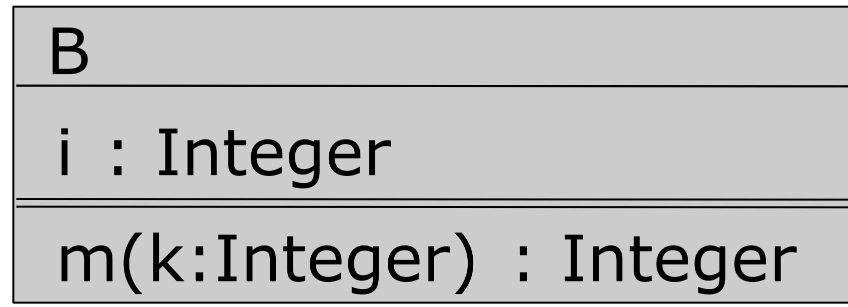


- definition $card_{B.d} \equiv \forall x \in B.\ |x.d| = 10$
- definition $card_{C.a} \equiv \forall x \in C.\ 1 \leq |x.a| \leq 5$

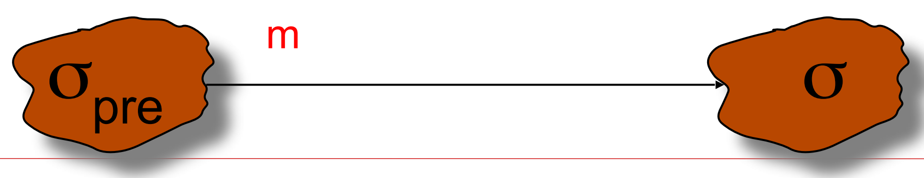
Operation Contracts

Operation Contracts

- Many UML diagrams talk over a sequence of states (not just individual global states)
- This appears for the first time in so-called contracts for (Class-model) methods:



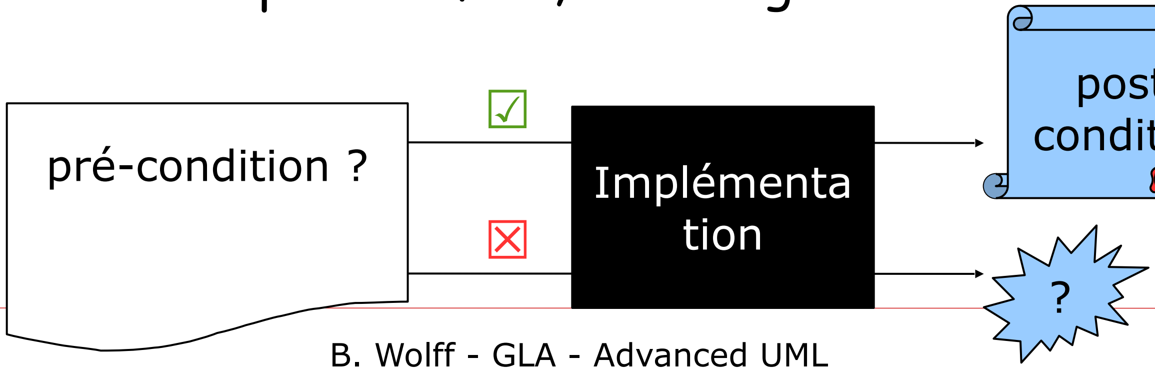
- The « method » m can be seen as a « transaction » of a B object transforming the underlying pre-state σ_{pre} in the state « after » m yielding a post-state σ.



Principe de la conception par contrats : contrat entre l'opération appelée et son appelant

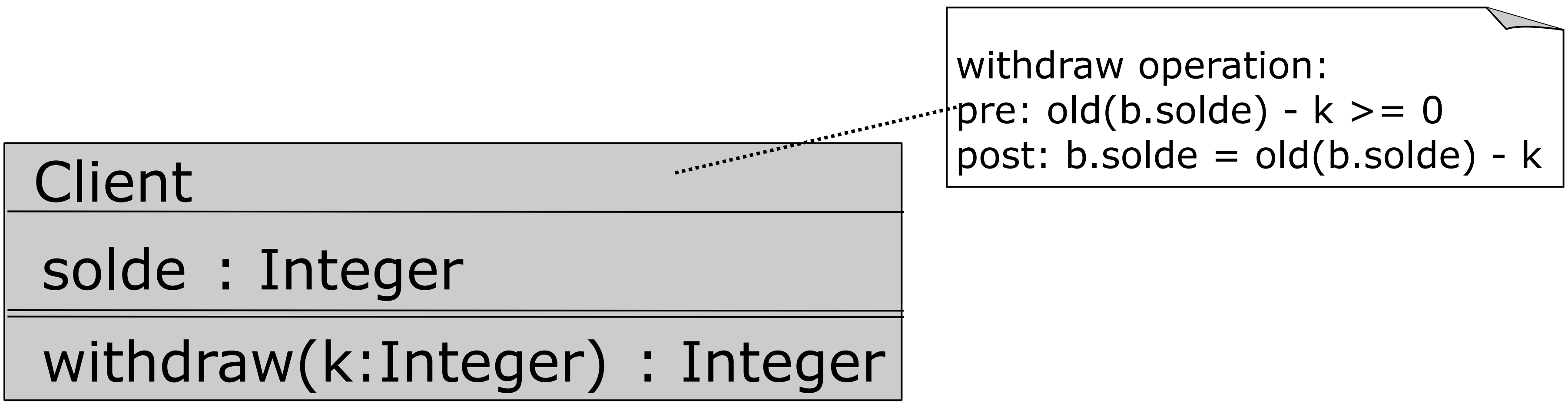
- Appelant responsable d'assurer que la pré-condition est vraie
- Implémentation de l'opération appelée responsable d'assurer la terminaison et la post-condition à la sortie, si la pré-condition est vérifiée à l'entrée

Si la pré-condition n'est pas vérifiée, aucune garantie sur l'exécution de l'opération



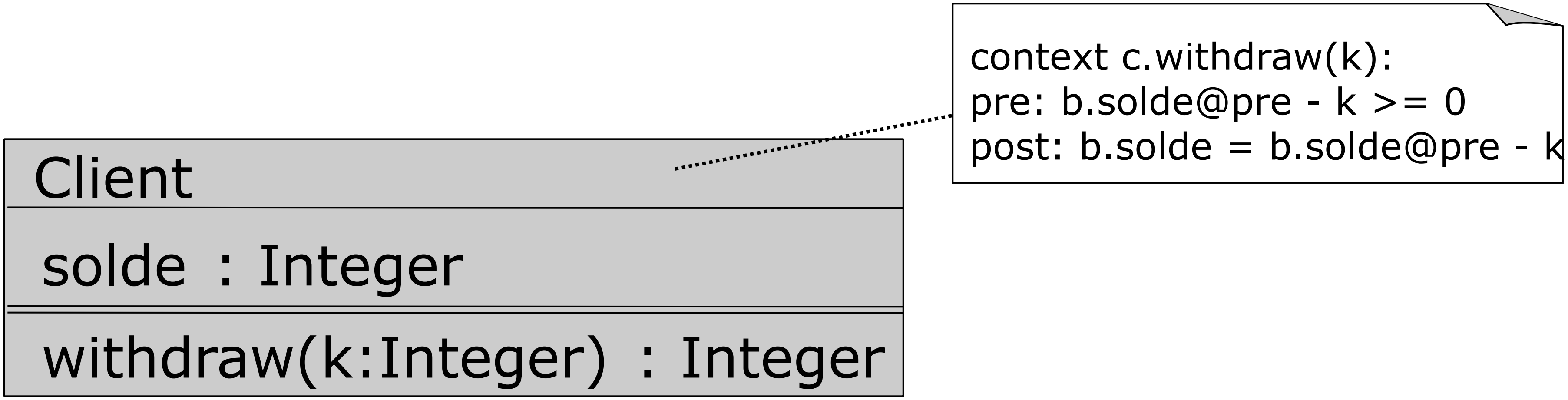
Operations in UML and MOAL

- Syntactically, contracts are annotated like this (in MOAL convention):



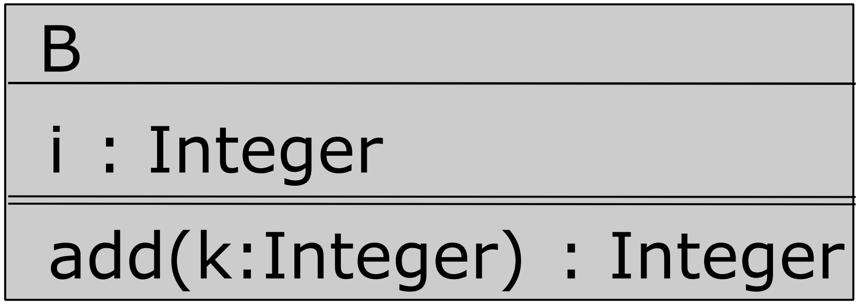
Operations in UML and MOAL

- ... or like this (OCL-ish):



Operations in UML and MOAL Contracts

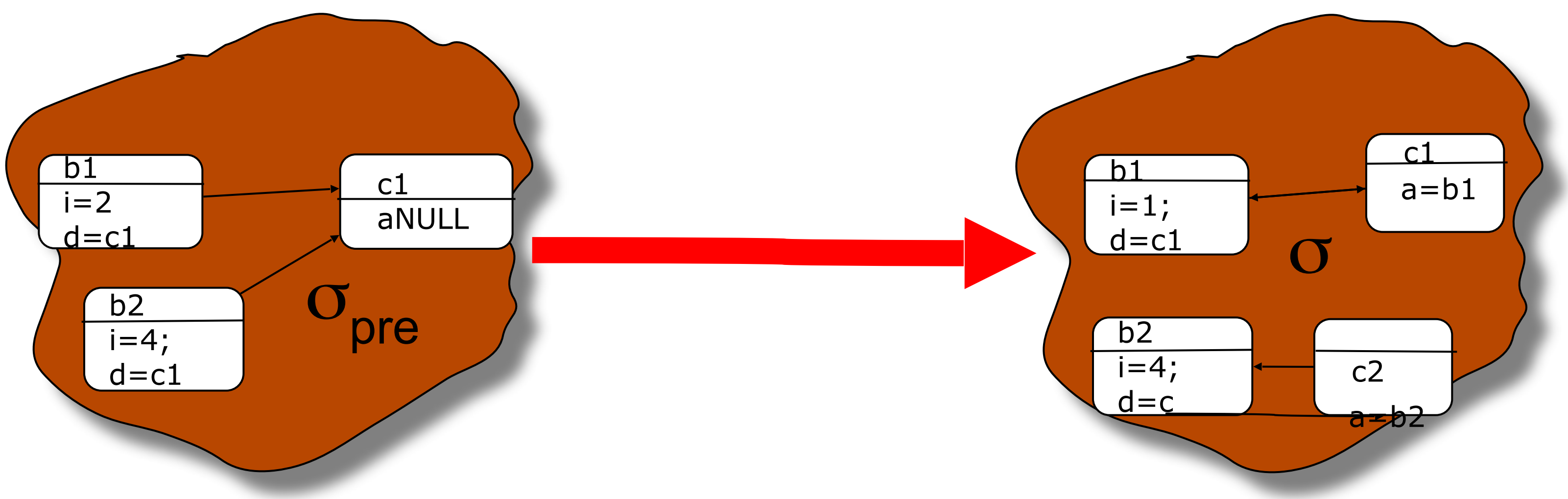
- This appears for the first time in so-called contracts for (Class-model) methods:



- The « method » add can be seen as a « transaction » of a B object transforming the underlying pre-state σ_{pre} in the state « after » add yielding a post-state σ.

Syntax and Semantics of MOAL Contracts

- Again: This is the view of a transaction (like in a data-base), it completely abstracts away intermediate states or time. (This possible in other models/calculi, like the Hoare-calculus, though).



Syntax and Semantics of MOAL Contracts

- Consequence:
  - The pre-condition is a formula referring to the σ_{pre} and the method arguments b1, a_1, ..., a_n only.
  - the post-condition is only assured if the pre-condition is satisfied
  - otherwise the method
    - ...may do anything on the state and the result, may even behave correctly , may non-terminate !
    - raise an exception (recommended in Java Programmer Guides for public methods to increase robustness)

Syntax and Semantics of MOAL Contracts

- Consequence:
  - The post-condition is a formula referring to both σ_{pre} and σ, the method arguments b1, a_1, ..., a_n and the return value captured by the variable result.

  - any transition is permitted that satisfies the post-condition (provided that the pre-condition is true)

Syntax and Semantics of MOAL Contracts

- Consequence:
  - The semantics of a method call:

$$b1.m(a_1, ..., a_n)$$

is thus:

$$pre_m(b1,a_1, ..., a_n) (\sigma_{pre})$$

$$\longrightarrow$$

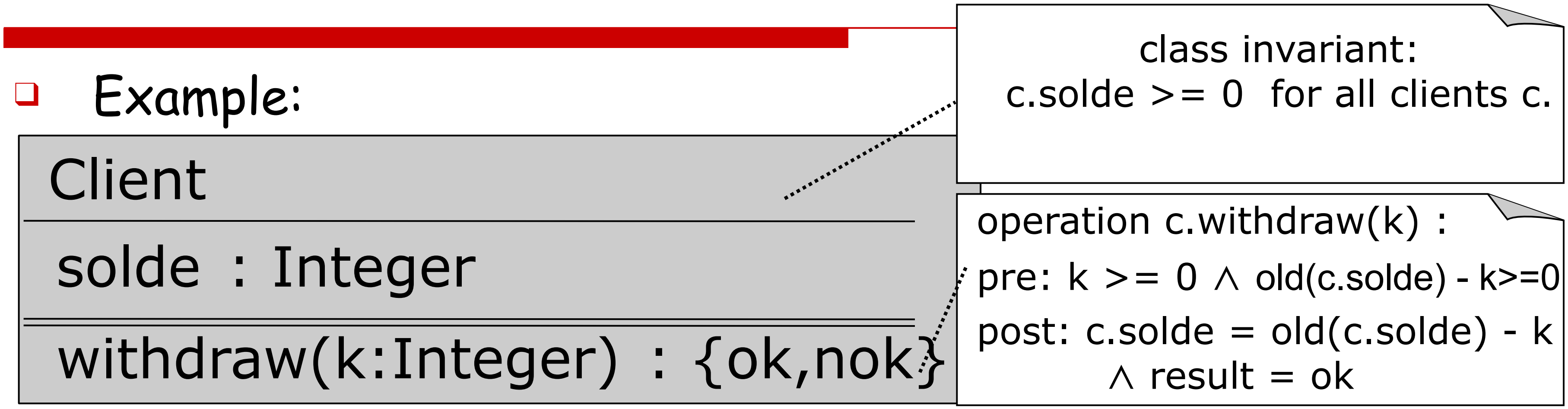
$$post_m(b1,a_1, ..., a_n,result)(\sigma_{pre},\sigma)$$

  - Note that moreover all global class invarants have to be added for both pre-state σ_{pre} and post-state σ !
  - For a successful transition, the following must hold:

$$Inv(\sigma_{pre}) \wedge pre_m ... (\sigma_{pre}) \wedge post ... (\sigma_{pre},\sigma) \wedge Inv(\sigma)$$

Syntax and Semantics of MOAL Contracts

- Example:



- definition $inv_{Client}(\sigma) \equiv$

  $\forall c \in Client(\sigma).\ 0 \leq c.solde(\sigma)$

- definition $pre_{withdraw}(c, k)(\sigma) \equiv$

  $c \in Client(\sigma) \wedge 0 \leq k \wedge 0 \leq c.solde(\sigma) - k$

- definition $post_{withdraw}(c, k, result)(\sigma_{pre}, \sigma) \equiv$

  $c \in Client(\sigma_{pre}) \wedge result = ok$

  $\wedge\ c.solde(\sigma) = c.solde(\sigma_{pre}) - k$

Syntax and Semantics of MOAL Contracts

- Notation:
  - In order to relax notation, we will use for applications to σ_{pre} the old-notation:

    - Client(σ_{pre}) **becomes** old(Client)

    - `c.solde`(σ_{pre}) **becomes** old(`c.solde`)

Syntax and Semantics of MOAL Contracts

- Example (revised):

Client
solde : Integer
withdraw(k:Integer) : {ok,nok}

class invariant:
c.solde >= 0 for all clients c.

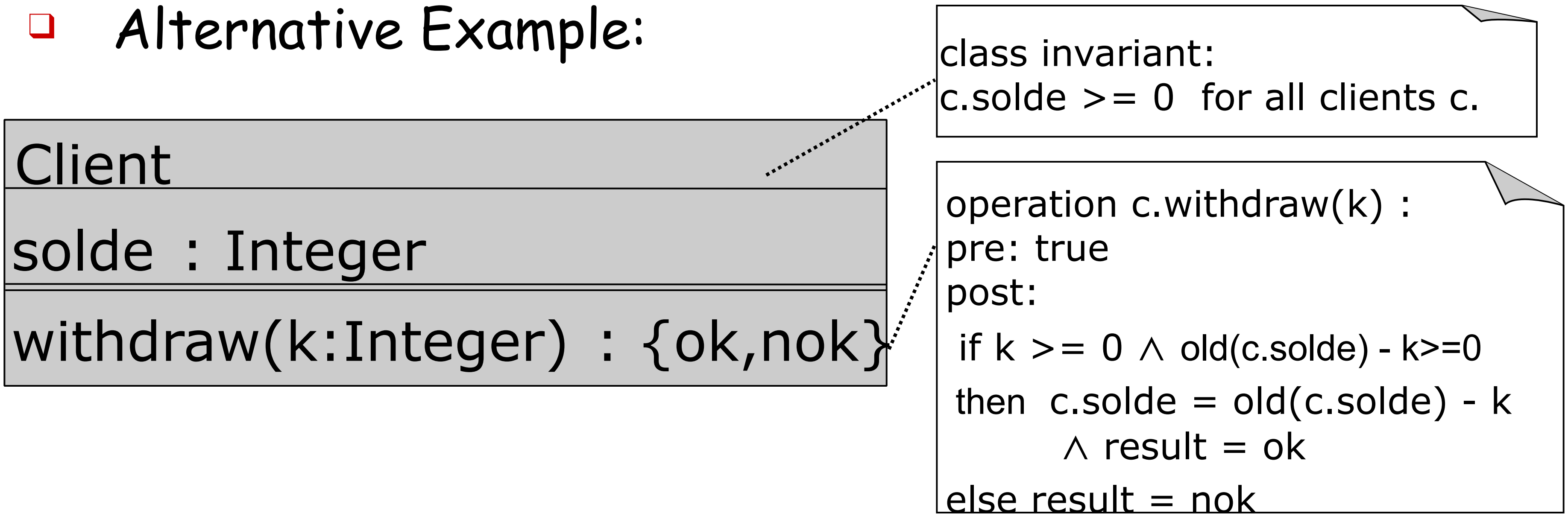
operation c.withdraw(k) :
pre: k >= 0 ∧ old(c.solde) - k>=0
post: c.solde = old(c.solde) - k
∧ result = ok

- definition $inv_{Client} \equiv \forall c \in Client.\ 0 \leq c.solde$
- definition $pre_{withdraw}(c, k) \equiv$
  $c \in Client \wedge 0 \leq k \wedge 0 \leq c.solde - k$
- definition $post_{withdraw}(c, k, result) \equiv$
  $c \in old(Client) \wedge result = ok$
  $c.solde = old(c.solde) - k \wedge$

MOAL σ convention!

Syntax and Semantics of MOAL Contracts

- Alternative Example:



What are the differences between these contracts?

Syntax and Semantics of MOAL Contracts

- Answer:

```
operation c.withdraw(k) :
pre: true
post:
 if k >= 0 ∧ old(c.solde) - k>=0
 then  c.solde = old(c.solde) - k
          ∧ result = ok
else result = nok
```

"withdraw" is now always defined; in case of illegal arguments it yields an error

Semantics of MOAL Contracts

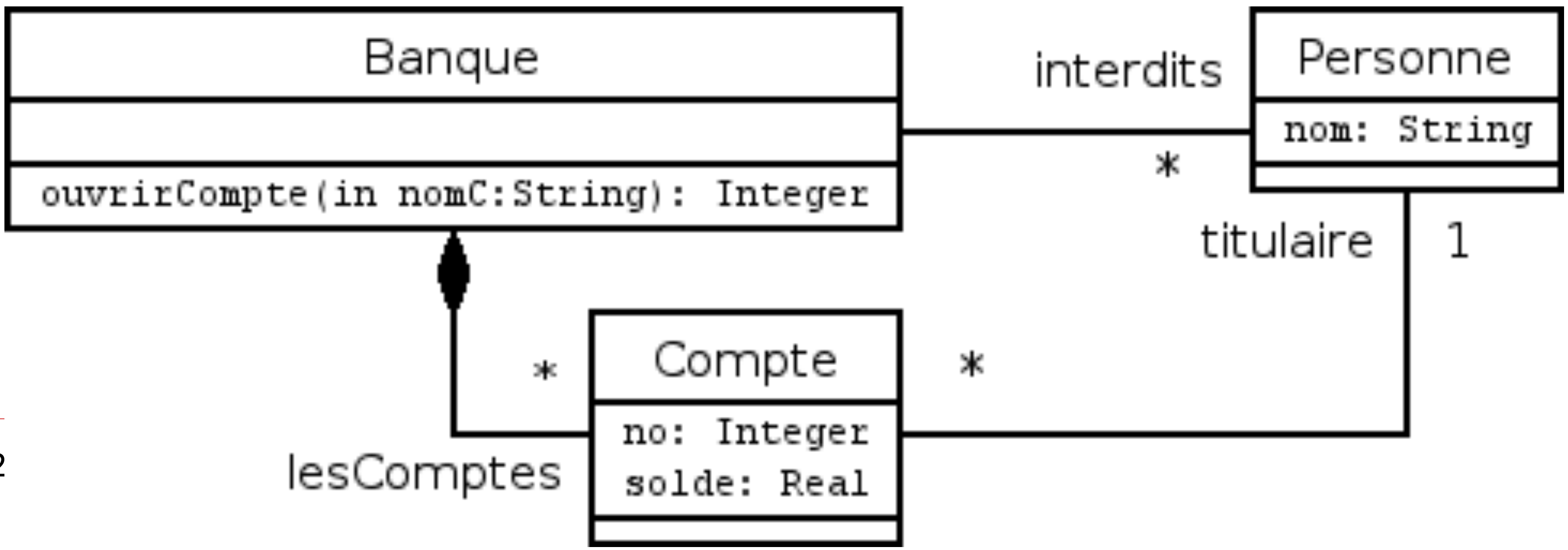
- Two predicates are helpful when defining contracts. They exceptionally refer to both (σ_{pre},σ)
  - isNew(p)(σ_{pre},σ) is true only if object p of class C does not exist in σ_{pre} but exists in σ
  - modifiesOnly(S)(σ_{pre},σ) is only true iff
    - all objects in σ_{pre} are except those in S identical in σ
    - all objects exist either in σ are or are contained in S

    With this predicate, one can express : „and nothing else changes". It is also called «framing condition»

A Revision of the 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 Revision of the Example: Bank (2)

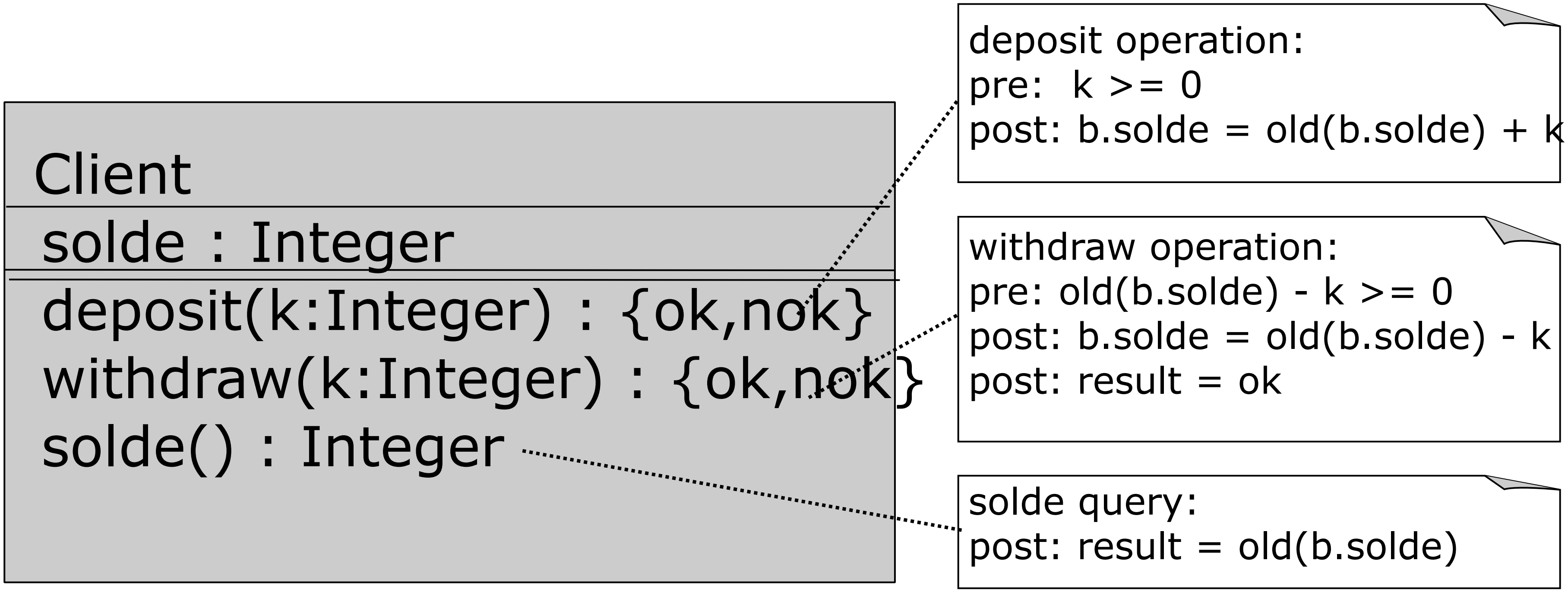
Now we can understand the complex looking contract of part III intro for the Bank:

```
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} | = 1
      ∧ ∀p ∈ Personne. p.nom = nomC ⟶ isNew(p)
      ∧ |{c∈Compte | c.titulaire.nom = nomC}| = 1
      ∧ ∀c∈Compte. c.titulaire.nom = nomC ⟶ c.solde = 15
                                              ∧ isNew(c)
      ∧ b.lesComptes=old(b.lesComptes)∪
                     {c∈Compte | c.titulaire.nom = nomC}
      ∧ b.interdits=old(b.interdits)∪
                     {c∈Compte | c.titulaire.nom = nomC}
      ∧ modifiesOnly({b}∪{c∈Compte c.titulaire.nom = nomC}
                     ∪ {p ∈ Personne | p.nom = nomC})
```

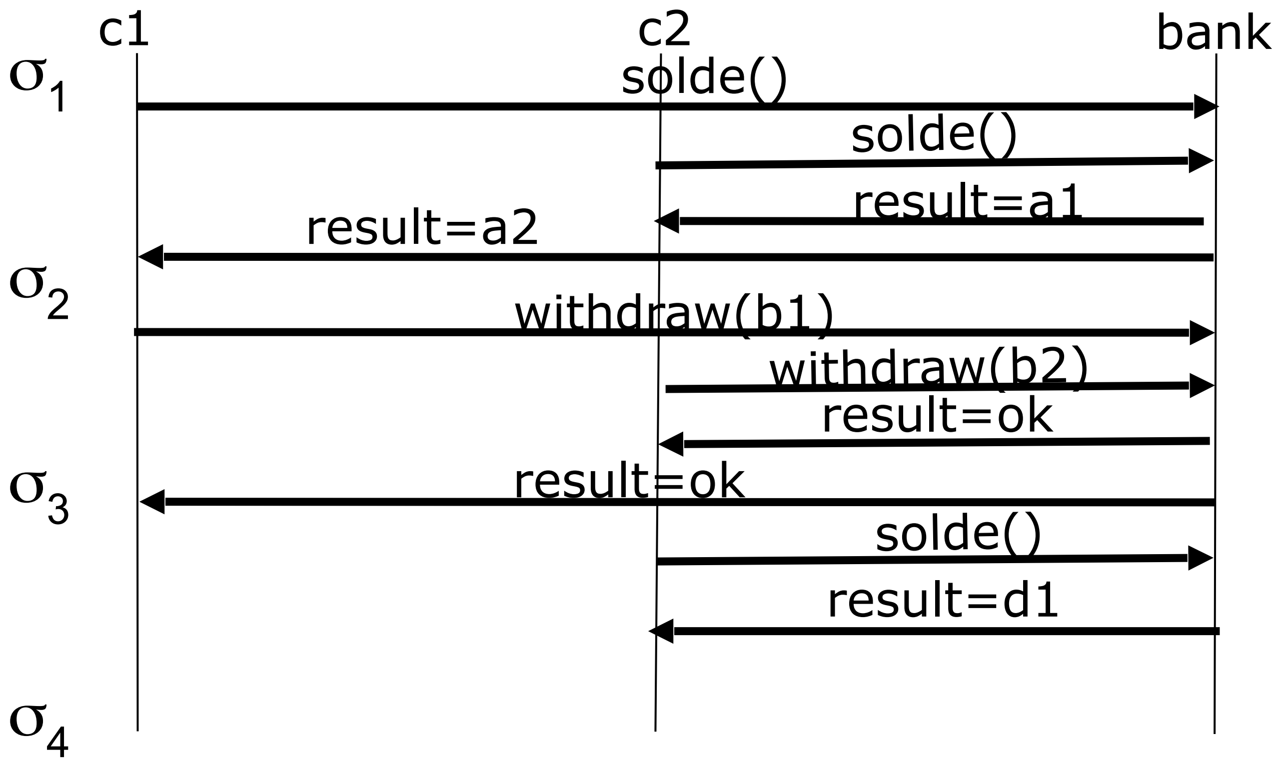
Operations in UML and MOAL

- A more complete example at a glance:



Operations in UML and MOAL

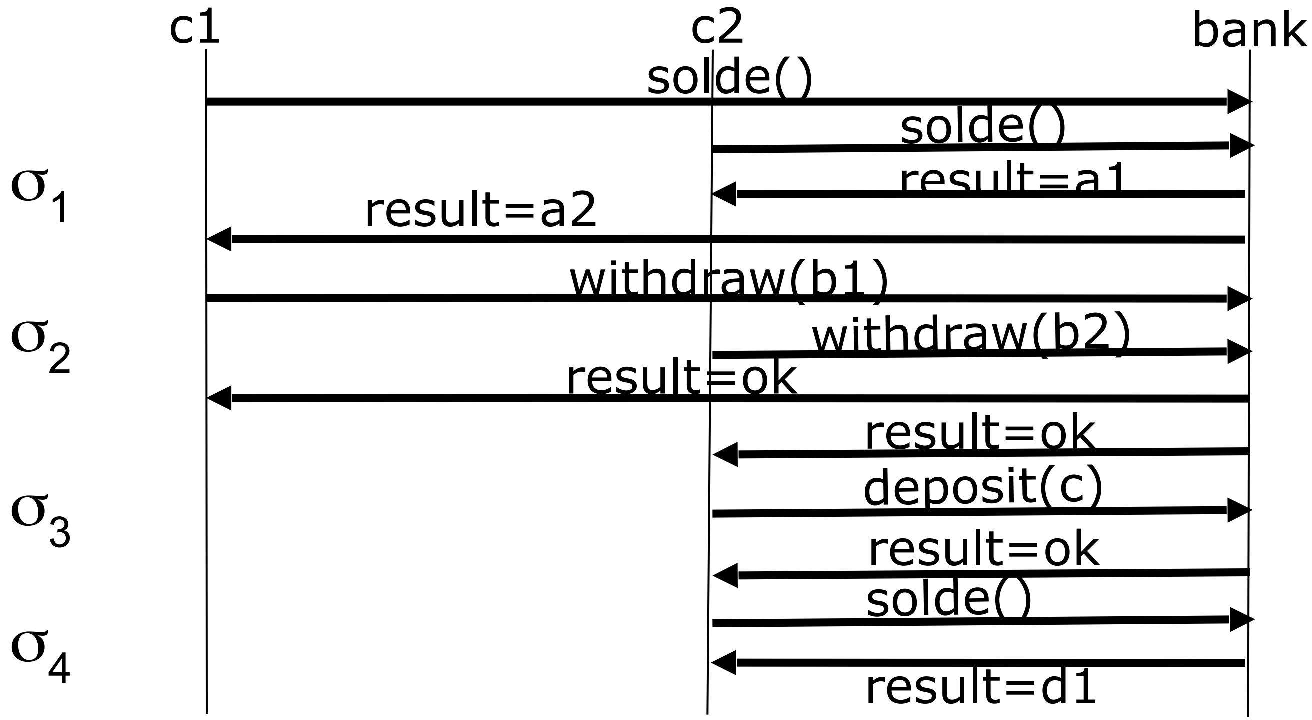
- Abstract Concurrent Test Scenario:



Assume that this scenario was valid, i.e. all conditions were satisfied: what do we know in σ_4 ?

Operations in UML and MOAL

- Abstract Concurrent Test Scenario:



Any instance of b1 and a1 is a test ! This is a „Test Schema" !
Note: b1 can be chosen dynamically during the test !

Summary

- MOAL makes the UML to a "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
  - Cahier de charge to Analysis
  - From Analysis to Designs and Tests.