Synthesizing B specifications from EB^3 attribute definitions

Frédéric Gervais[1,2],
Marc Frappier[1],
Régine Laleau[3]

[1] GRIL, Département d'Informatique, Université de Sherbrooke
[2] CEDRIC, Institut d'Informatique d'Entreprise
[3] LACL, Université Paris 12

Specification of IS

- Main issues:
  - Consistency of the DB: data integrity constraints
  - Answer of the IS
  - Modeling of the IS behaviour
- Aim: use of formal notations and techniques
- Coupling of formal specifications?
  - B: state-based formal language
  - EB^3: event-based formal language

B

- State-based formal language
- Based on set theory and first order logic:
  - Static part: definition of state variables and invariant properties
  - Dynamic part: definition of operations (preconditions + substitutions)
  - Proof obligations: the invariant must be preserved by the operations
- Refinement
- Tools: Atelier B, B4Free, Click'n'Prove, B-Core

EB^3

- Trace-based formal language
- Inspired from process algebras (CSP, CCS, LOTOS)
- Specifically created for the design of information systems
- Specification of the functional behaviour of the system
- Interpreter (EB3PAI)

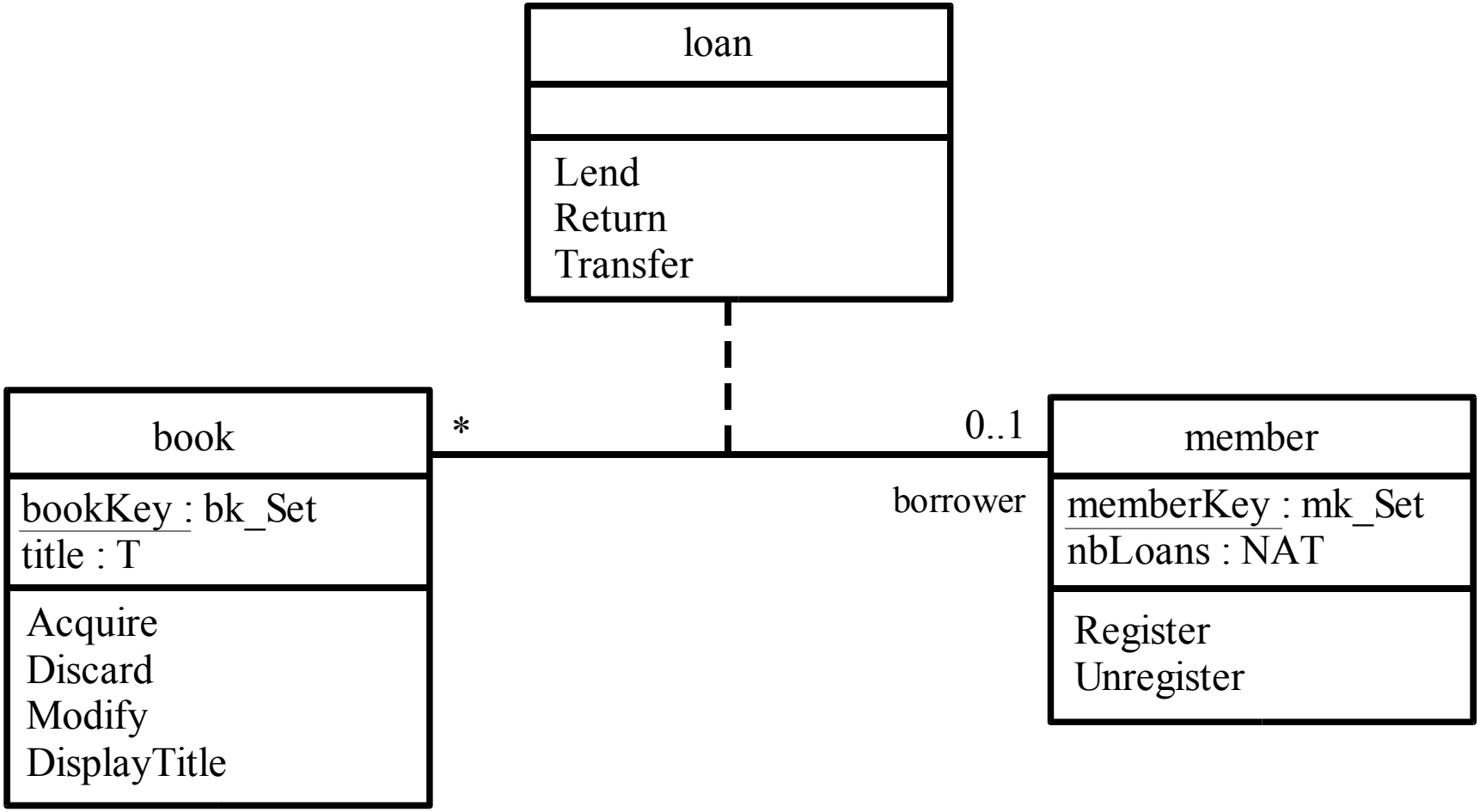
EB^3 specification

- User requirements class diagram (entity types, associations)
- EB^3 process expressions
- Input-output rules
- Recursive functions defined on EB^3 traces (attribute definitions, auxiliary functions)

Behaviour

```
trace := [];
forever do
    receive input event σ;
    if main can accept trace::σ then
        trace := trace::σ;
        send output event o such that (trace,o) ∈ R ;
    else
        send error message ;
```

Class diagram




```
Acquire(bId : bk_Set, ttl : T) : void
Modify(bId : bk_Set, ttl : T) : void
...
```

EB^3 process expressions

```
book(bId : bk_Set) =
    Acquire(bId,_).
    (
        ( | mId : mk_Set : loan(bId,mId) )*
     |||
        Modify(bId,_)*
     |||
        DisplayTitle(bId)*
    ).
    Discard(bId)
```

EB^3 process expressions

```
book(bId : bk_Set) =
    Acquire(bId,_).
    (
        ( | mId : mk_Set : loan(bId,mId) )*
     |||
        Modify(bId,_)*
     |||
        DisplayTitle(bId)*
    ).
    Discard(bId)
```

```
RULE R1
INPUT DisplayTitle(bId)
OUTPUT title(trace,bId)
END;
```

EB^3 attribute definitions

- Recursive functions on the valid traces
- Two kinds of definitions:
  - Key definition: set of the existing key values for the key
  - Non-key attribute definition: the attribute value for a given key
- Specification: functional style with pattern matching on the last event of the trace

Examples

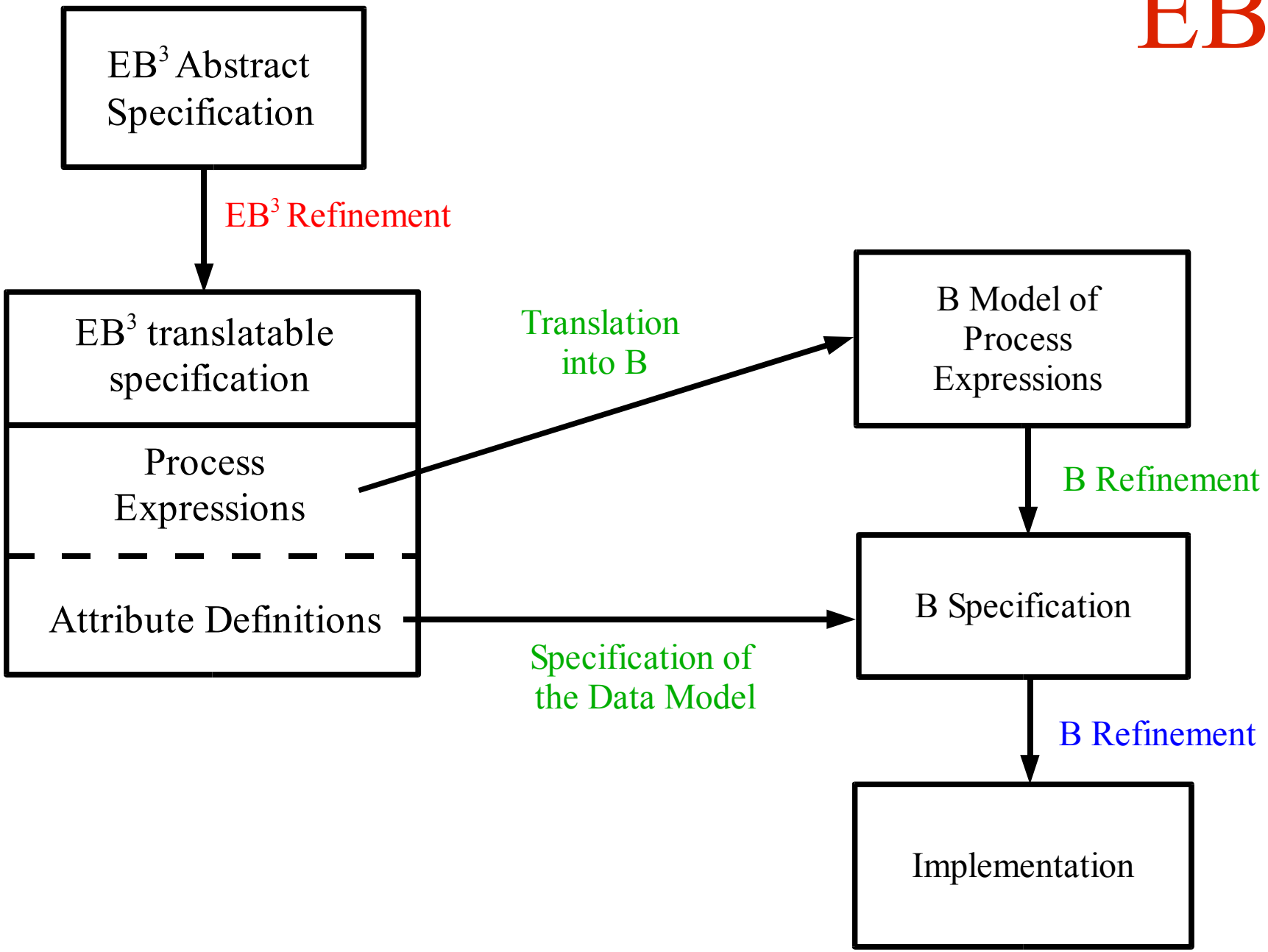
```
bookKey(s : τ(main)) : 𝔽(bk_Set) =
match last(s) with
    ⊥                   : ∅,
    Acquire(bId,ttl)    : bookKey(front(s)) ∪ { bId },
    Discard(bId)        : bookKey(front(s)) - { bId },
    _                   : bookKey(front(s));
```

```
title(s : τ(main), bId : bk_Set) : T =
match last(s) with
    ⊥                   : ⊥,
    Acquire(bId,ttl)    : ttl,
    Modify(bId,ttl)     : ttl,
    Discard(bId)        : ⊥,
    _                   : title(front(s),bId);
```

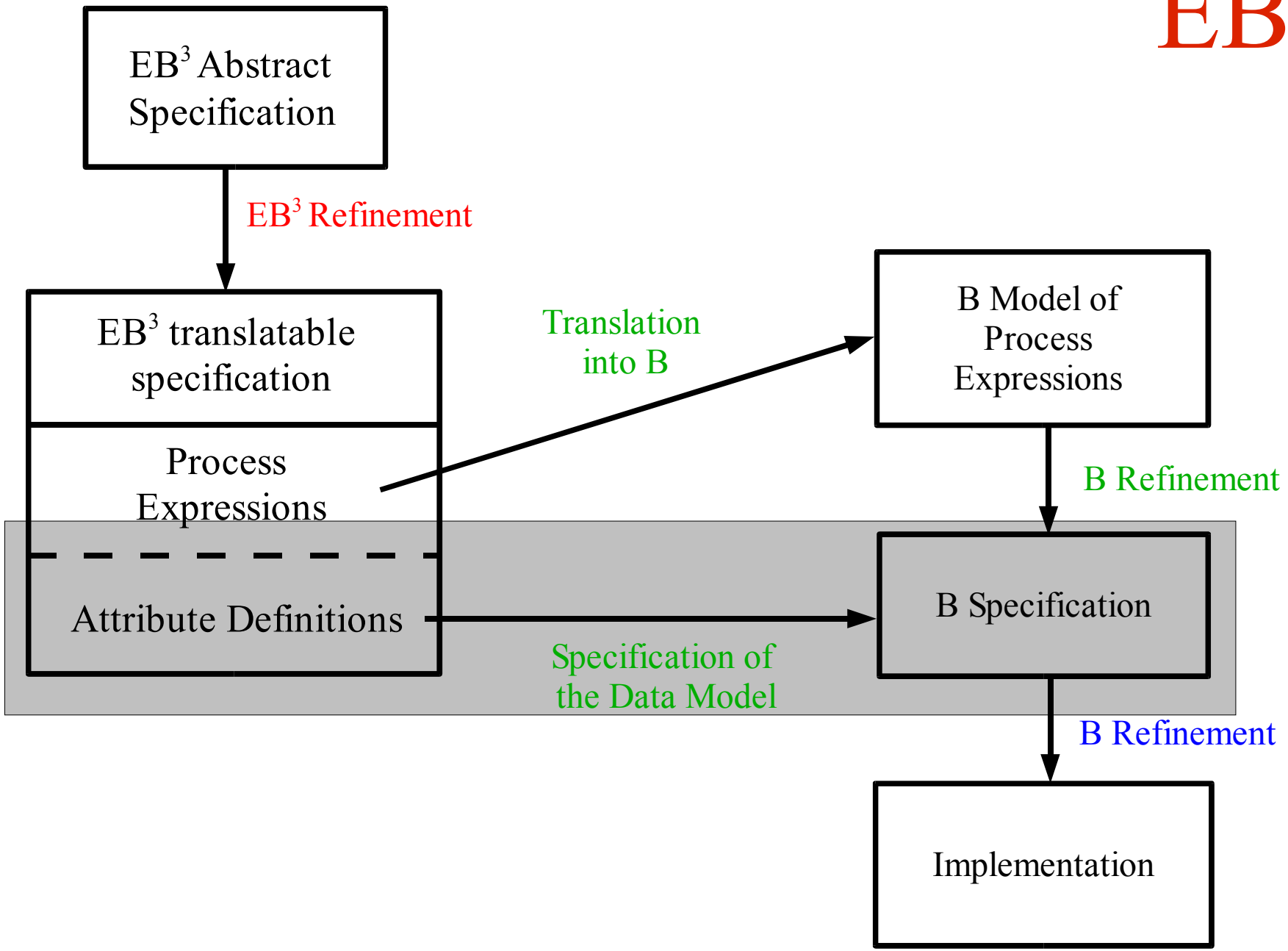
EB^4

- Aim: define a formal approach coupling state-based and event-based specifications for IS
- EB^3:
  - Trace-based formal language for IS specification
  - Description of the functional behaviour
- B:
  - State-based formal language
  - Description of static data integrity constraints
  - Theorem prover

EB^4


EB3 Abstract Specification
EB3 Refinement
EB3 translatable specification
Process Expressions
Attribute Definitions
Translation into B
B Model of Process Expressions
B Refinement
B Specification
Specification of the Data Model
B Refinement
Implementation

EB^4


EB3 Abstract Specification
EB3 Refinement
EB3 translatable specification
Process Expressions
Attribute Definitions
Translation into B
B Model of Process Expressions
B Refinement
B Specification
Specification of the Data Model
B Refinement
Implementation

B specification in 4 steps

- Generation of the static part from the user requirements class diagram
- Generation of the substitutions for the initialization
- Generation of the substitutions for the operations
- Computation of the preconditions for the operations

B static part

MACHINE *B_Library*
SETS *mk_Set; bk_Set; T*
VARIABLES *memberKey, bookKey, loan, title, nbLoans*
INVARIANT

$memberKey \subseteq mk\_Set \wedge$
$bookKey \subseteq bk\_Set \wedge$
$loan \in bookKey \rightarrowtail\!\!\!\!\!\rightarrow memberKey \wedge$
$title \in bookKey \longrightarrow T \wedge$
$nbLoans \in memberKey \longrightarrow NAT$

DEFINITIONS

$borrower(x) == loan(x)$

B substitutions

- Orthogonal specification styles:
  - EB^3 attribute definitions: for each attribute, effects of the actions
  - B specifications: for each operation, substitutions on the attribute variables
- Analysis of the input clauses:
  - Determine each occurrence of the action
  - Determine the relevant key values (pattern matching + predicates in conditional terms)
  - Determine the effects on the attributes

Example: Acquire

```
bookKey(s : τ(main)) : 𝔽(bk_Set) =
match last(s) with
    ⊥                   : ∅,
    Acquire(bId,ttl)    : bookKey(front(s)) ∪ { bId },
    Discard(bId)        : bookKey(front(s)) - { bId },
    _                   : bookKey(front(s));


title(s : τ(main), bId : bk_Set) : T =
match last(s) with
    ⊥                   : ⊥,
    Acquire(bId,ttl)    : ttl,
    Modify(bId,ttl)     : ttl,
    Discard(bId)        : ⊥,
    _                   : title(front(s),bId);
```

Example: Acquire

Acquire(bld,ttl) =
pre $bld \in bk\_Set \wedge ttl \in T$
then
$bookKey := bookKey \cup \{ bld \} \parallel$
$title := title \lhd\!\!+ \{ bld \mapsto ttl \}$
end;

Example: Discard

```
bookKey(s : τ(main)) : 𝔽(bk_Set) =
match last(s) with
    ⊥                   : ∅,
    Acquire(bId,ttl)    : bookKey(front(s)) ∪ { bId },
    Discard(bId)        : bookKey(front(s)) - { bId },
    _                   : bookKey(front(s));


title(s : τ(main), bId : bk_Set) : T =
match last(s) with
    ⊥                   : ⊥,
    Acquire(bId,ttl)    : ttl,
    Modify(bId,ttl)     : ttl,
    Discard(bId)        : ⊥,
    _                   : title(front(s),bId);
```

Example: Discard

Discard(bld) =
pre *bld* $\in$ *bk_Set*
then
    bookKey := *bookKey* − { *bld* } ||
        title := { *bld* } $\lhd$ *title*
end;

Conditional terms

- The key values to delete, to update or to insert are determined by the pattern matching condition and the predicates in the **if** parts
- Generation and analysis of a decision tree

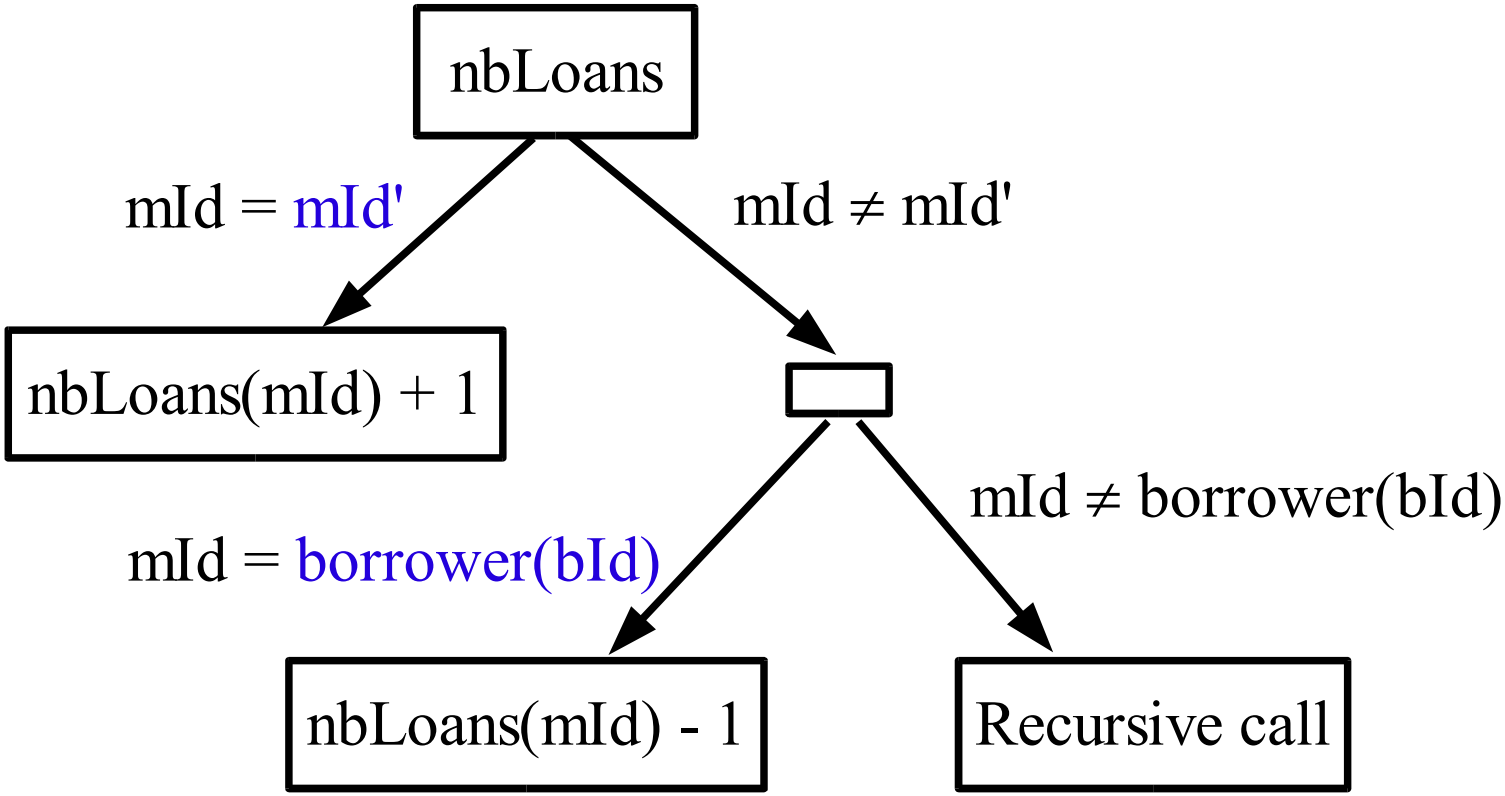
Transfer

```
nbLoans(s : τ(main), mId : mk_Set) : NAT =
match last(s) with
    ⊥                    : ⊥,
    Register(mId)        : 0,
    Unregister(mId)      : ⊥,
    Lend(_,mId)          : 1 + nbLoans(front(s),mId),
    Return(bId)          : if mId = borrower(front(s),bId)
                           then nbLoans(front(s),mId) – 1
                           end,
    Transfer(bId,mId')   : if mId = mId'
                           then nbLoans(front(s),mId) + 1
                           else if mId = borrower(front(s),bId)
                                then nbLoans(front(s),mId) – 1
                                end
                           end,
    _                    : nbLoans(front(s),mId);
```

Transfer

```
nbLoans(s : τ(main), mId : mk_Set) : NAT =
match last(s) with
    ⊥                    : ⊥,
    Register(mId)        : 0,
    Unregister(mId)      : ⊥,
    Lend(_,mId)          : 1 + nbLoans(front(s),mId),
    Return(bId)          : if mId = borrower(front(s),bId)
                           then nbLoans(front(s),mId) – 1
                           end,
    Transfer(bId,mId')   : if mId = mId'
                           then nbLoans(front(s),mId) + 1
                           else if mId = borrower(front(s),bId)
                                then nbLoans(front(s),mId) – 1
                                end
                           end,
    _                    : nbLoans(front(s),mId);
```

Decision tree



Precondition

- Typing constraints for input parameters
- The weakest precondition required to preserve the invariant of the machine
- Condition required to impose ordering constraints on operations

Invariant

Discard(bId) =
pre $bId \in bk\_Set$
then
$bookKey := bookKey - \{ bId \} \parallel$
$title := \{ bId \} \lhd title$
end;

$$\boxed{loan \in bookKey \rightarrowtail\!\!\!\!\!\rightarrow memberKey}$$

Invariant

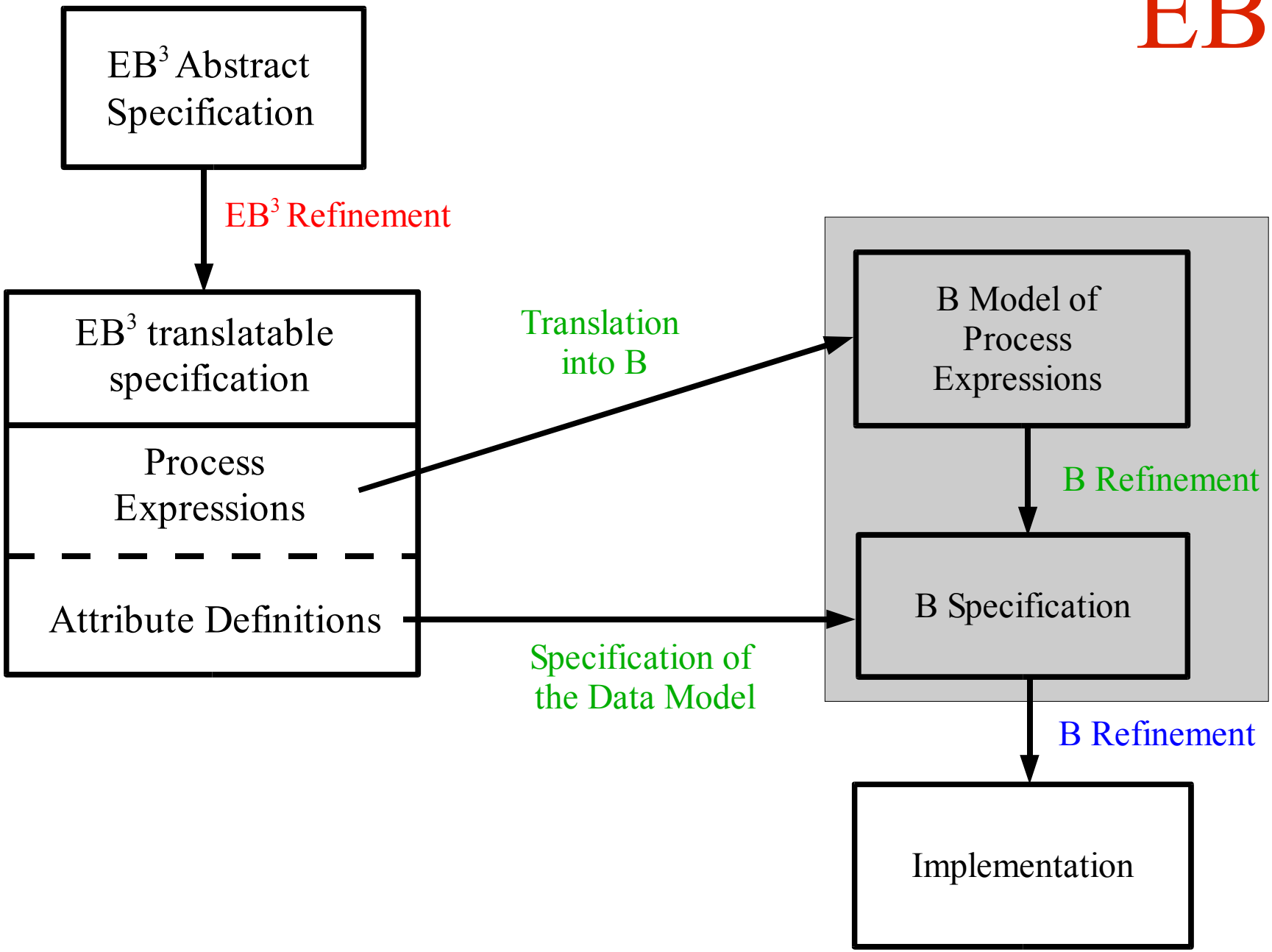
Discard(bld) =
pre $bld \in bk\_Set \wedge bld \notin dom(loan)$
then
$bookKey := bookKey - \{ bld \} \parallel$
$title := \{ bld \} \lhd title$
end;

$$\boxed{loan \in bookKey \rightarrowtail memberKey}$$

Weakest precondition

- Issue: determining the weakest precondition from the invariant and the operation substitutions
- Solution: using the logic solver of Atelier B
- Weakness: interactive use of the prover
- Work in progress: definition of systematic rules

EB^4


EB3 Abstract Specification
EB3 Refinement
EB3 translatable specification
Process Expressions
Attribute Definitions
Translation into B
Specification of the Data Model
B Model of Process Expressions
B Refinement
B Specification
B Refinement
Implementation

It's almost finished!

Discard(bId) =
pre $bId \in bk\_Set \wedge bId \notin dom(loan) \wedge bId \in bookKey$
then
$bookKey := bookKey - \{ bId \} \parallel$
$title := \{ bId \} \lhd title$
end;

```
book(bId : bk_Set) =
(
   Acquire(bId,_).
   ( | mId : mk_Set : loan(bId,mId) )*.
   Discard(bId)
)*
```

Conclusion

- Generation of B substitutions from EB^3 attribute definitions
- Generation of the operation preconditions:
  - Weakest preconditions
  - Proof of static data integrity constraints
- EB^4: IS specification by coupling EB^3 and B

Perspectives

- Definition of systematic rules for the weakest preconditions
- Definition of refinement patterns
- EB^4:
  - Definition of a methodology
  - Definition of verification techniques
  - Implementation of tools
  - Integration between EB^3 and B

Questions?