Transformations of Event-B models into recursive programs

Milena Dyle
Supervisor: Dominique Méry

Université de Lorraine

June 28, 2015

Event-B

Event-B is a mathematical formal language. It aims at:

- Building correct-by-construction systems
- Understating system behavior
- System documentation

It achieves this by:

- Precisely formalizing the system behavior
- Building incremental models of the system
- Giving a certificate for model correctness

Overview

1. Background
2. Event-B to Recursive algorithms EB2RC
3. Connections
4. Case Study
5. Conclusions

Event-B, Concepts

```
MACHINE m_square
SEES c_square
VARIABLES r
INVARIANTS r ∈ ℕ
EVENTS
  INITIALISATION
   BEGIN
    act1: r:=0
   END
  EVENT compute_square
   BEGIN
    act1: r := n * n
   END
END
```

```
CONTEXT c_square
  CONSTANTS
    n
  AXIOMS
    axm1 : n ∈ ℕ
  END
```

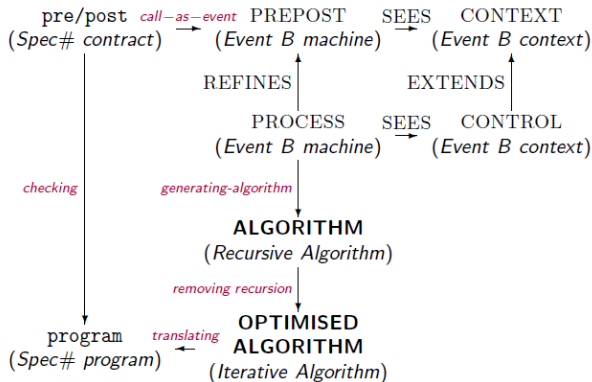
Event-B, Concepts

```
MACHINE m_square
SEES c_square
VARIABLES r
INVARIANTS r ∈ ℕ
EVENTS
  INITIALISATION
   BEGIN
    act1 :  r:=0
   END
  EVENT compute_square
   BEGIN
    act1 :  r := n ∗ n
   END
END
```

```
CONTEXT c_square
  CONSTANTS
    n
  AXIOMS
    axm1 : n ∈ ℕ
  END
```

Proof obligations will ensure the model is correct

Call-as-event principle



Translation

- Assertions

$Inv1 : c \implies A(x)$ — $/ * A(x) * /$

- Basic Events

EVENT e
WHEN
$l = l_1$
$g_{l_1,l_2}(x)$
THEN
$l := l_2$
$x := f_{l_1,l_2}(x)$
END

IF $g_{l_1,l_2}(x)$
THEN
$x := f_{l_1,l_2}(x)$

Translation

- Recursive Events

```
EVENT rec%PROC(h(x),y;Q(y))
  ANY y
  WHEN
    l = l1
    g_{l1,l2}(x)
  THEN
    l := l2
    x := f_{l1,l2}(x)
  END
```

$Q(y) := PROC(h(x), y)$

- Caller Events

```
EVENT call%APROC(h(x),y;Q(y))
  ANY y
  WHEN
    l = l1
    g_{l1,l2}(x)
  THEN
    l := l2
    x := f_{l1,l2}(x)
  END
```

$Q(y) := APROC(h(x), y)$

Connection

Context

- A caller(recursive) event encapsulates a specific sub-problem
- The caller event states the problem and it doesn't explain how to compute it
- The caller event assumes the problem is specified into another model

Connection

Context

- A caller(recursive) event encapsulates a specific sub-problem
- The caller event states the problem and it doesn't explain how to compute it
- The caller event assumes the problem is specified into another model

Problem

There is no link between the caller event and the callee model to assure successful connection.

Connection

Context

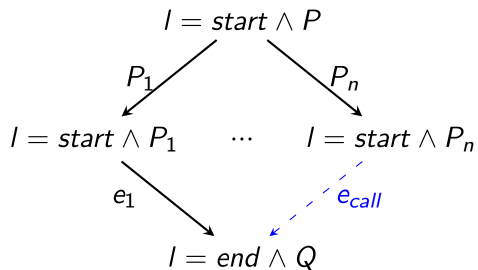
- A caller(recursive) event encapsulates a specific sub-problem
- The caller event states the problem and it doesn't explain how to compute it
- The caller event assumes the problem is specified into another model

Problem

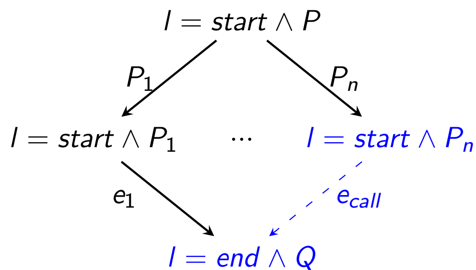
There is no link between the caller event and the callee model to assure successful connection.

How can we force the link between two models in terms of caller events?

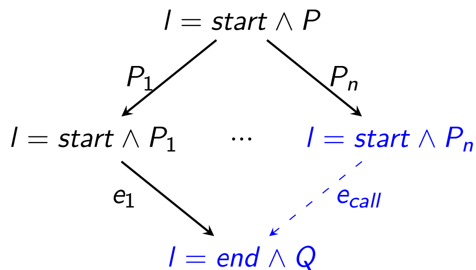
Connection schema

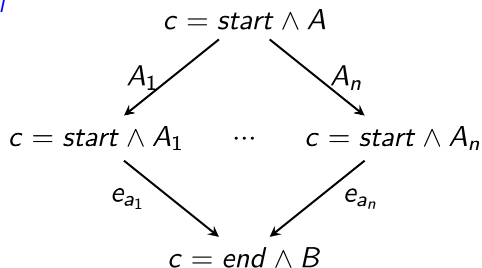


Connection schema

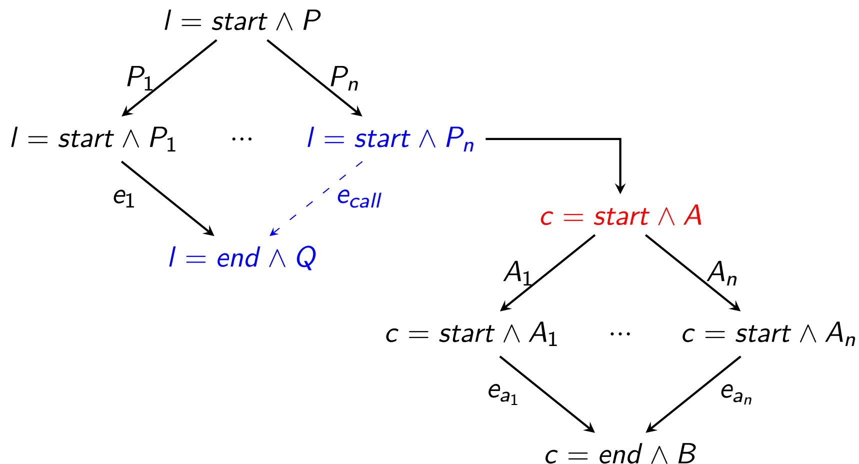


Connection schema

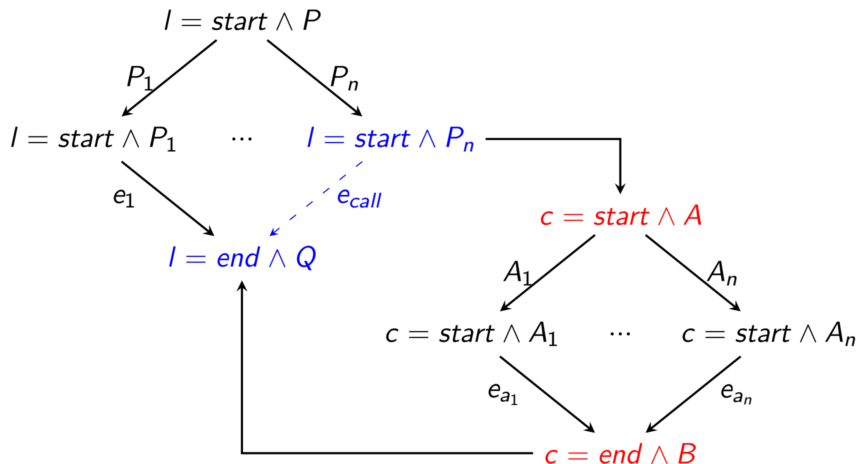




Connection schema



Connection schema



Definition

Let model M specify a sequential problem. Let this model state a caller event e_{call}. Let model N design another sequential algorithm. Connection in Event-B is the process of:

- Configuring model M and N for a possible connection
- Applying substitutions rule to force inter-model communication
- Applying prof obligations rules to ensure the connection is correct
  - Axioms preservation: $Axm(c,x)(M) \wedge R(l_1,x) \wedge Inv(c,x)(M) \wedge config(c_m,x_m,y_m,c_n,x_n,y_n) \implies Axm(c,x)(N)$
  - Guards preservation: $Axm(c,x)(M) \wedge R(l_1,x) \wedge Inv(c,x)(M) \wedge config(c_m,x_m,y_m,c_n,x_n,y_n) \wedge A(c,x)(N) \wedge B(c,x)(N) \implies Grd(x,y)(M)$

Insertion-Sort

Description

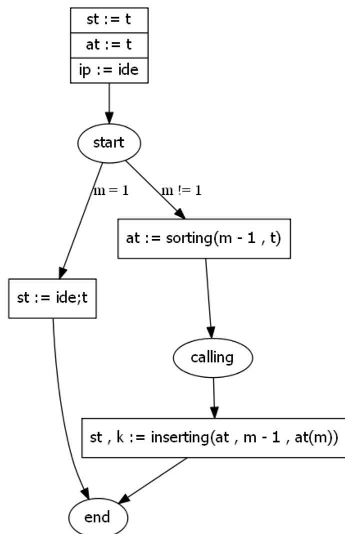
Insertion-sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. At each step, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

PROCEDURE sorting(m,t; VAR st)

$$PRE \begin{pmatrix} t \in 1\,..\,n \rightarrow \mathbb{N} \\ n \in \mathbb{N}_1 \\ m \in \mathbb{N}_1 \\ m \leq n \end{pmatrix}$$

$$POST \begin{pmatrix} \exists pi.(pi \in 1\,..\,n \rightarrowtail 1\,..\,n \\ \wedge \forall i \cdot i \in 1\,..\,m-1 \Rightarrow pi;t(i) \leq pi;t(i+1) \\ \wedge st = pi;t) \end{pmatrix}$$

Refinement



- Introduction of controls points
- Control points control the state flow for insertion-sort
- Recursive event sorting(m-1,t)
- Caller event inserting(at, m-1, at(m))

Refinement, Recursive event

```
rec@sorting(m−1,t;at)@m≠1   ≙
    STATUS
  ordinary
ANY
  pi
WHERE
  grd5   :   c=start
  grd6   :   m > 1
  grd1   :   pi ∈ 1‥n ⤖ 1‥n
  grd2   :   ∀k· k ∈ dom(pi) ∧ k ∈ 1‥m−2 ⇒ t(pi(k)) ≤ t(pi(k+1))
  grd3   :   ∀j· j ∈ dom(pi) ∧ j ∈ m‥n ⇒ pi(j)=j
  grd4   :   ∀ i · i ∈ 1‥m−1 ⇒ pi(i) ∈ 1‥m−1
THEN
  act2   :   c ≔ calling
  act1   :   at≔ pi;t
  act3   :   ip≔ pi
END
```

Refinement, Caller event

```
call@inserting(at, m-1, at(m);st,k)@NULL   ≙
   STATUS
 ordinary
REFINES
 sorting
ANY
 qi
 k
WHERE
 grd9   :   c=calling
 grd1   :   qi ∈ 1‥n ⤖ 1‥n
 grd2   :   k ∈ 1‥m
 grd3   :   ∀l· l∈1‥m−2 ⇒ at(l)≤at(l+1)
 grd6   :   ∀u· u∈1‥k−1 ⇒ qi(u)=u
 grd7   :   ∀u· u∈k+1‥m ⇒ qi(u)=u−1
 grd15   :   ∀ j · j ∈ m+1‥n ⇒ qi(j)=j
 grd8   :   qi(k)=m
 grd13   :   ∀i· i ∈ dom(qi) ∧ i ∈ 1‥m−1 ⇒ at(qi(i)) ≤ at(qi(i+1))
WITH
 pi   :   pi=qi;ip
THEN
 act2   :   c≔end
 act1   :   st≔qi;at
END
```

Translation

Data: $n \in \mathbb{N}_1 \wedge m \in \mathbb{N}_1 \wedge m \leq n \wedge t \in 1 \mathinner{..} n \rightarrow \mathbb{N}$
Result: $st \in 1 \mathinner{..} n \wedge \forall i \cdot i \in 1 \mathinner{..} m-1 \Rightarrow st(i) \leq st(i+1))$

$st := t;$
$at := t;$
$ip := ide;$
if $m = 1$ **then**
  $st := ide; t;$
else
  $at := sorting(m-1, t);$
  $st, k := inserting(at, m-1, at(m));$
end

Inserting

Description

Insertion is a simple algorithm that given a sorted sequence [1,m] in an array (or list) it will insert the $m+1^{th}$ element by keeping the array sorted . At each step it will shift until it finds the location it belongs within the sorted list, and inserts it there.

PROCEDURE insert(table, m, value;VAR tbl, k)

$$\left(\begin{array}{c} n \in \mathbb{N} \\ table \in 1 \mathinner{..} n \rightarrow \mathbb{N} \\ m \in 1 \mathinner{..} n \\ n \geq 1 \\ m < n \\ value \in \mathbb{N} \\ \forall i \cdot i \in 1 \mathinner{..} m-1 \Rightarrow \\ table(i) \leq table(i+1) \\ table(m+1) = value \end{array}\right) \left(\begin{array}{c} \exists k, ipi.(\forall i \cdot i \in 1 \mathinner{..} m \Rightarrow tbl(i) \leq tbl(i+1)) \\ k \in 1 \mathinner{..} m+1 \\ tbl = ipi; table \\ ipi \in 1 \mathinner{..} n \rightarrow 1 \mathinner{..} n \\ \forall i \cdot i \in 1 \mathinner{..} k-1 \Rightarrow ipi(i) = i \\ \forall i \cdot i \in k+1 \mathinner{..} m+1 \Rightarrow ipi(i) = i-1 \\ ipi(k) = m+1 \\ \forall j \cdot j \in m+2 \mathinner{..} n \Rightarrow ipi(j) = j \end{array}\right)$$

Connection

Table: Proof obligation example

$$
\begin{array}{l}
\text{inv5: } c = calling \Rightarrow (at = ip;t \\
\qquad\qquad\qquad \wedge(\forall i \cdot i \in 1 \,..\, m0-1 \Rightarrow ip(i) \in 1 \,..\, m0-1) \\
\qquad\qquad\qquad \wedge(\forall j \cdot j \in 1 \,..\, m0-2 \Rightarrow at(j) \leq at(j+1)) \\
\qquad\qquad\qquad \wedge(\forall k \cdot k \in m0 \,..\, n0 \Rightarrow ip(k) = k)) \\
\text{conf: } table = at \\
\text{conf: } m = m0 - 1 \\
\text{conf: } n = n0 \\
\vdash \\
\text{axm10: } \forall i \cdot i \in 1 \,..\, m-1 \Rightarrow table(i) \leq table(i+1)
\end{array}
$$

Translation

Data: $n \in \mathbb{N}_1 \wedge m \in \mathbb{N}_1 \wedge m \leq n \wedge t \in 1 \,..\, n \rightarrow \mathbb{N}$

Result:

$$st \in 1..n \wedge (\forall i \cdot i \in dom(pi) \wedge i \in 1..m-1 \Rightarrow st(i) \leq st(i+1))$$

$st := t$;
$at := t$;
$ip := ide$;
if $m = 1$ **then**
  $st := ide; t$;
else
  $at := sorting(m-1, t)$;
  $n \in \mathbb{N} \wedge at \in 1..n \rightarrow \mathbb{N} \wedge m-1 \in 1..n \wedge n \geq 1 \wedge m-1 < n \wedge value \in \mathbb{N} \wedge (\forall i \cdot i \in 1 \,..\, m-2 \Rightarrow at(i) \leq at(i+1)) \wedge at(m) = value$;
  $st, k := inserting(at, m-1, at(m))$;
  $\forall i \cdot i \in dom(pi) \wedge i \in 1 \,..\, m-1 \Rightarrow st(i) \leq st(i+1) \wedge k \in 1 \,..\, m+1$
  ;
end

Conclusions & Future work

Conclusions

- Investigation of modeling in Event-B, the call-as-event principle and code transformation
- Introduced *connections* to ensure correctness of transformation process
- Illustration of the overall idea by insertion-sort case study, CYK case study.

Conclusions & Future work

Conclusions

- Investigation of modeling in Event-B, the call-as-event principle and code transformation
- Introduced *connections* to ensure correctness of transformation process
- Illustration of the overall idea by insertion-sort case study, CYK case study.

Future work

- Implementation of connections in EB2RC
- Connections can be adapted to other domains
- Improvement of EB2RC to generate Spec# OO types
- Generating the iterative version of the recursive algorithm

The End