Transforming Event B Models into Verified C# Implementations

Dominique Méry (Université de Lorraine, France)
Rosemary Monahan (NUI Maynooth, Ireland)

presented by Geoff Hamilton (DCU, Ireland)

First International Workshop on
Verification and Program Transformation
July 13th and 14th, 2013, Saint Petersburg, Russia

Summary

Introduction

Modelling Languages

Case study : the binary search problem

Deriving the Recursive Program

Transforming the recursive algorithm

In the Spec# world

Conclusion

Topics

Topics

- Combining the efforts of program refinement as supported by EVENT B and program verification as supported by the Spec# programming system.

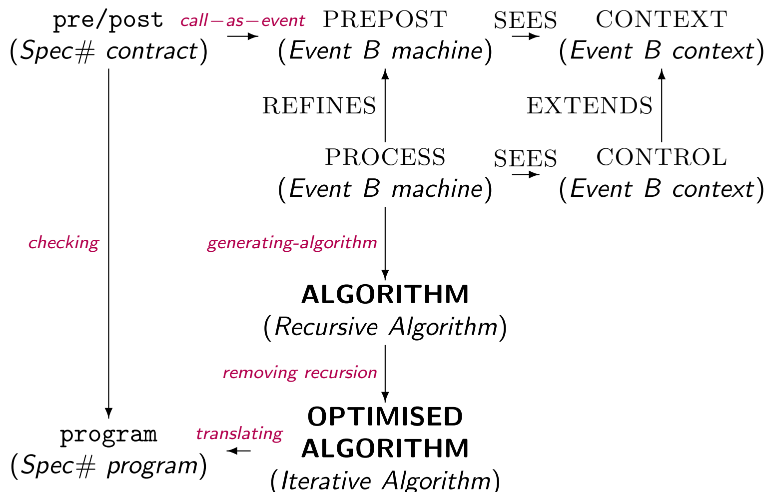
Topics

- Combining the efforts of program refinement as supported by EVENT B and program verification as supported by the Spec# programming system.
- Proposing an architecture Integrated Development Framework, which induces a methodology and which improves the usability of formal verification tools for the specification, the construction and the verification of correct sequential algorithms.

Topics

- Combining the efforts of program refinement as supported by Event B and program verification as supported by the Spec# programming system.
- Proposing an architecture Integrated Development Framework, which induces a methodology and which improves the usability of formal verification tools for the specification, the construction and the verification of correct sequential algorithms.
- In this paper : we focus on the transformation of the final concrete specification into an executable algorithm :
  1. transforming an Event B specification into a recursive algorithm
  2. transforming from that recursive program to an iterative version of the same program.

Integrated Development Framework



Event B Modelling

Event B Modelling

$$\textbf{any } t \textbf{ where } G(t,x) \textbf{ then } x :| (R(x,x',t)) \textbf{ end}$$

(t is a local parameter and the event actions establish $x :| (R(x,x',t))$)
$(\exists t \cdot (G(t,x) \ \wedge \ R(x,x',t)))$

Event B Modelling

$$\textbf{any } t \textbf{ where } G(t,x) \textbf{ then } x :| (R(x,x',t)) \textbf{ end}$$

(t is a local parameter and the event actions establish $x :| (R(x,x',t))$)
$(\exists t \cdot (G(t,x) \ \wedge \ R(x,x',t)))$

Event B Modelling

$$\textbf{any } t \textbf{ where } G(t,x) \textbf{ then } x :| (R(x,x',t)) \textbf{ end}$$

(t is a local parameter and the event actions establish $x :| (R(x,x',t))$)
$(\exists t \cdot (G(t,x) \ \wedge \ R(x,x',t)))$

```
MACHINE specquare
SEES square0
VARIABLES
  r
INVARIANTS
  inv1 : r ∈ ℕ
EVENTS
EVENT INITIALISATION
  begin
    act1 : r := 0
  end
EVENT square_computing
  begin
    act1 : r := n ∗ n
  end
END
```

Event B Modelling

$$\textbf{any } t \textbf{ where } G(t,x) \textbf{ then } x :| (R(x,x',t)) \textbf{ end}$$

(t is a local parameter and the event actions establish $x :| (R(x,x',t))$)
$(\exists t \cdot (G(t,x) \ \wedge \ R(x,x',t)))$

```
MACHINE specquare
SEES square0
VARIABLES
  r
INVARIANTS
  inv1 : r ∈ ℕ
EVENTS
EVENT INITIALISATION
  begin
    act1 : r := 0
  end
EVENT square_computing
  begin
    act1 : r := n ∗ n
  end
END
```

```
CONTEXT square0
CONSTANTS
  n
AXIOMS
  axm1 : n ∈ ℕ
END
```

Event B Modelling

$$\textbf{any } t \textbf{ where } G(t,x) \textbf{ then } x :| (R(x,x',t)) \textbf{ end}$$

(t is a local parameter and the event actions establish $x :| (R(x,x',t))$)
$(\exists t \cdot (G(t,x) \wedge R(x,x',t)))$

```
MACHINE specquare
SEES square0
VARIABLES
  r
INVARIANTS
  inv1 : r ∈ ℕ
EVENTS
EVENT INITIALISATION
  begin
    act1 : r := 0
  end
EVENT square_computing
  begin
    act1 : r := n ∗ n
  end
END
```

```
CONTEXT square0
CONSTANTS
  n
AXIOMS
  axm1 : n ∈ ℕ
END
```

- *square0* is a context defining properties of a natural number n

Event B Modelling

$$\textbf{any } t \textbf{ where } G(t,x) \textbf{ then } x :| (R(x,x',t)) \textbf{ end}$$

(t is a local parameter and the event actions establish $x :| (R(x,x',t)))$
$(\exists t \cdot (G(t,x) \ \wedge \ R(x,x',t)))$

```
MACHINE specquare
SEES square0
VARIABLES
   r
INVARIANTS
   inv1 : r ∈ ℕ
EVENTS
EVENT INITIALISATION
   begin
      act1 : r := 0
   end
EVENT square_computing
   begin
      act1 : r := n ∗ n
   end
END
```

```
CONTEXT square0
CONSTANTS
   n
AXIOMS
   axm1 : n ∈ ℕ
END
```

- *square0* is a context defining properties of a natural number n
- *specsquare* is a machine with an event square_computing computing the square function for n and assigning the value to r.

Event B Modelling

$$\textbf{any } t \textbf{ where } G(t,x) \textbf{ then } x :| (R(x,x',t)) \textbf{ end}$$

(t is a local parameter and the event actions establish $x :| (R(x,x',t))$)
$(\exists t \cdot (G(t,x) \wedge R(x,x',t)))$

```
MACHINE specquare
SEES square0
VARIABLES
  r
INVARIANTS
  inv1 : r ∈ ℕ
EVENTS
EVENT INITIALISATION
  begin
    act1 : r := 0
  end
EVENT square_computing
  begin
    act1 : r := n * n
  end
END
```

```
CONTEXT square0
CONSTANTS
  n
AXIOMS
  axm1 : n ∈ ℕ
END
```

- *square0* is a context defining properties of a natural number n
- *specsquare* is a machine with an event square_computing computing the square function for n and assigning the value to r.
- The **SEES** clause related the context and the machine.

Spec#

- The Spec# programming language extends C# 2.0 through a rich assertion language that allows the specification of objects through class invariants, field annotations, and method specifications.

Spec#

- The Spec# programming language extends C# 2.0 through a rich assertion language that allows the specification of objects through class invariants, field annotations, and method specifications.
- The Spec# compiler statically enforces non-null types, emits run-time checks for method contracts and invariants and records the contracts as metadata for consumption by downstream tools.

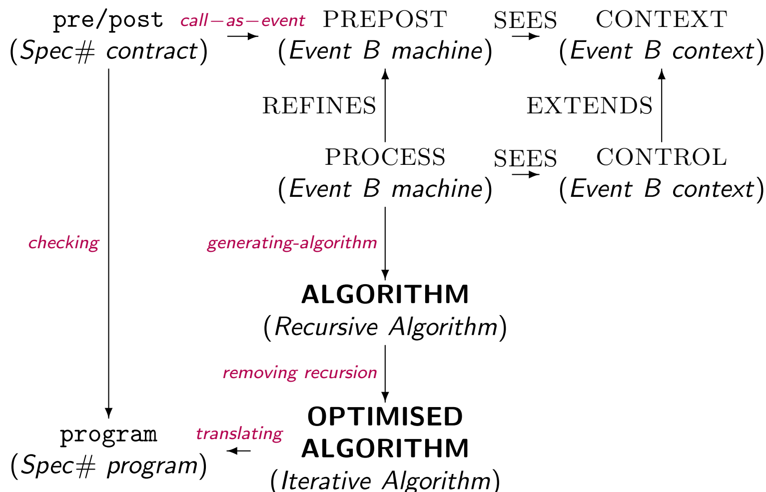
Spec#

- The Spec# programming language extends C# 2.0 through a rich assertion language that allows the specification of objects through class invariants, field annotations, and method specifications.
- The Spec# compiler statically enforces non-null types, emits run-time checks for method contracts and invariants and records the contracts as metadata for consumption by downstream tools.
- The Spec# static program verifier (SscBoogie) : generates logical verification conditions from a Spec# program uses an automatic reasoning engine (Z3) to analyse the verification conditions proving the correctness of the program or finding errors.

Spec# Specification : Sorting an Array

```
public static void sortArray(int[] !st, int[] !pi)
 requires st != pi && st.Length == pi.Length;
 requires forall{int i in (0:st.Length); st[i] == pi[i]};
 modifies st[*];
 ensures forall{int j in (1:st.Length);(st[j-1] <= st[j])};
 ensures forall{int w in (0:st.Length);
        (count{int v in (0:st.Length);st[v] == pi[w]}
        == count{int u in (0:st.Length); pi[u] == pi[w]})};
```

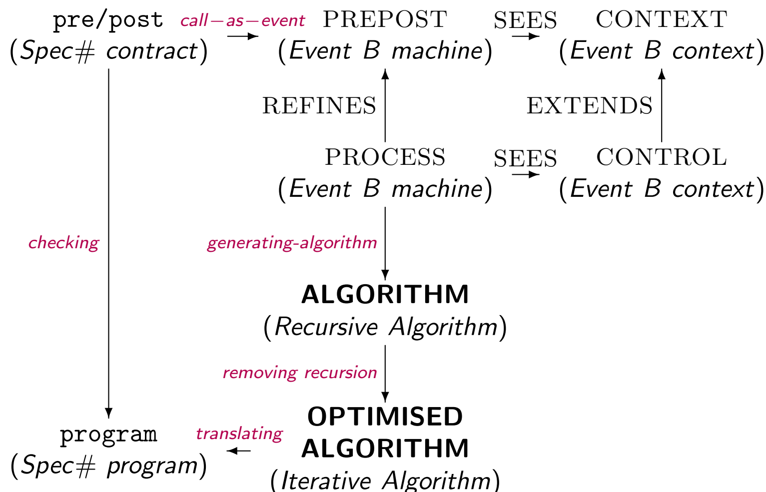
Integrated Development Framework



Integrated Development Framework

- The Event B machine PREPOST contains events, which have the same contract as that expressed in the original pre/post contract. This machine SEES the Event B CONTEXT, which expresses static information about the machine.
- The Event B machine PROCESS refines PREPOST generating a concrete specification that satisfies the contract. This machine SEES the Event B context CONTROL, which adds control information for the new machine.
- The labelled actions REFINES, SEES and EXTENDS, are supported by the RODIN platform and are checked *completely* using the proof assistant provided by RODIN.
- Transformation of an Event B machine into a concrete recursive algorithm (represented by the arrow labelled *generating-algorithm*).
- Transformation of this recursive algorithm into its equivalent partially annotated and iterative algorithm (represented by the arrow labelled *removing recursion*).

Integrated Development Framework



Implementing EVENT B models

Our integrated development framework for implementing abstract EVENT B models brings together the strengths of the refinement based approaches and verification based approaches to software development :

1. Splitting the abstract specification to be solved into its component specifications.
2. Refining these specifications into a concrete model using EVENT B and the RODIN platform.
3. Transforming the concrete model into recursive and iterative algorithms that can be directly implemented as real source code.
4. Verifying the iterative algorithm in the automatic program verification environment of Spec#.

Specifying the binary search problem

$$
\begin{array}{|ll|}
\hline
\text{PROCEDURE} & \mathit{binsearch}(t, val, lo, hi, ok, result) \\
\text{PRE} & \left(\begin{array}{l} t \in 0..t.Length \longrightarrow \mathbb{N} \\ \forall k.k \in lo..hi-1 \Rightarrow t(k) \leq t(k+1) \\ val \in \mathbb{N} \\ l, h \in 0..t.Length \\ lo \leq hi \end{array}\right) \\
\text{POST} & \left(\begin{array}{l} ok = TRUE \Rightarrow \ \ t(result) = val \\ ok = FALSE \Rightarrow (\forall i.i \in lo..hi \Rightarrow t(i) \neq val \end{array}\right) \\
\hline
\end{array}
$$

The two possible resulting *calls* to the procedure $binsearch(t, val, lo, hi; ok, result)$:

- **EVENT** find is $binsearch(t, val, lo, hi; ok, result)$ with $ok = TRUE$
- **EVENT** fail is $binsearch(t, val, lo, hi; ok, result)$: with $ok = FALSE$

Events find and fail

EVENT find
  any j
  where
    $grd1 : j \in lo\,..\,hi$
    $grd2 : t(j) = val$
  then
    $act1 : ok := TRUE$
    $act2 : i := j$
  end

EVENT fail
  when
    $grd1 : \forall k \cdot k \in lo\,..\,hi \Rightarrow t(k) \neq val$
  then
    $act1 : ok := FALSE$
END

- The two events form the machine called *binsearch1* (which corresponds to the PREPOST machine).
- The machine is refined to obtain *binsearch2* (which corresponds to PROCESS).
- This refined machine contains a new control variable, l, which *simulates* how the binary search is achieved.

Refinement for Computation

1. $$\begin{pmatrix} l = start \\ lo = hi \\ t(lo) = val \end{pmatrix} \xrightarrow{m_1} \begin{pmatrix} l = end \\ lo = hi \\ ok = TRUE \ \wedge \ result = lo \end{pmatrix}$$

2. $$\begin{pmatrix} l = start \\ lo = hi \\ t(lo) \neq val \end{pmatrix} \xrightarrow{m_2} \begin{pmatrix} l = end \\ lo = hi \\ ok = FALSE \end{pmatrix}$$

3. $$\begin{pmatrix} l = start \\ lo < hi \end{pmatrix} \xrightarrow{split} \begin{pmatrix} l = middle \\ lo < hi \\ mi = (lo + hi)/2 \end{pmatrix}$$

4. $$\begin{pmatrix} l = middle \\ lo < hi \\ mi = (lo + hi)/2 \\ val < t(mi) \end{pmatrix} \xrightarrow{rec(lo, mi-1, val, ok, result)} \begin{pmatrix} l = end \\ ok = TRUE \wedge t(result) = val \end{pmatrix}$$

Refinement for Computation

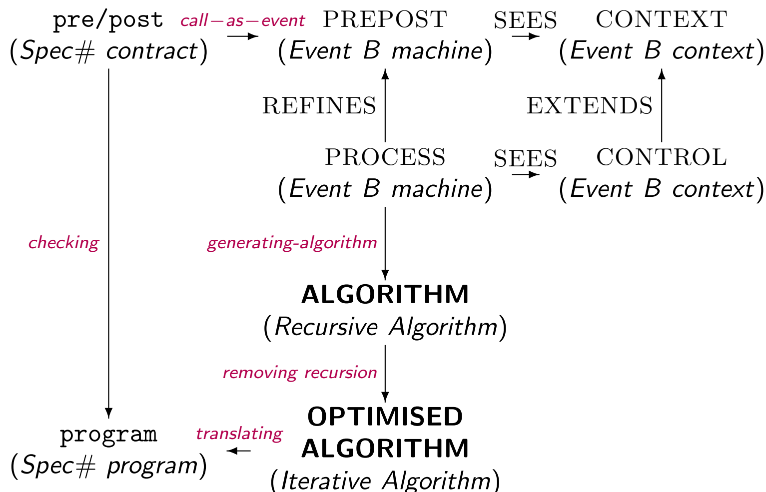
1. $$\begin{pmatrix} l = middle \\ lo < hi \\ val < t(mi) \end{pmatrix} \overset{rec(lo,mi-1,val,ok,result)}{\longrightarrow} \begin{pmatrix} l = end \\ \wedge\ ok = FALSE \wedge (\forall i.i \in lo..hi \Rightarrow t(i) \neq val \end{pmatrix}$$

2. $$\begin{pmatrix} l = middle \\ lo < hi \\ mi = (lo+hi)/2 \\ val = t(mi) \end{pmatrix} \overset{m_3}{\longrightarrow} \begin{pmatrix} l = end \\ ok = TRUE\ \wedge\ result = mi \end{pmatrix}$$

3. $$\begin{pmatrix} l = middle \\ lo < hi \\ mi = (lo+hi)/2 \\ val > t(mi) \end{pmatrix} \overset{rec(mi+1,hi,val,ok,result)}{\longrightarrow} \begin{pmatrix} l = end \\ ok = TRUE \wedge\ t(result) = val \end{pmatrix}$$

4. $$\begin{pmatrix} l = middle \\ lo < hi \\ mi = (lo+hi)/2 \\ val > t(mi) \end{pmatrix} \overset{rec(mi+1,hi,val,ok,result)}{\longrightarrow} \begin{pmatrix} l = end \\ \wedge\ ok = FALSE \wedge (\forall i.i \in lo..hi \Rightarrow t(i) \neq val \end{pmatrix}$$

Integrated Development Framework



Case 1 :Basic Events

```
EVENT e
  when
    ℓ = ℓ1
    g_{ℓ1,ℓ2}(x)
  then
    ℓ := ℓ2
    x := f_{ℓ1,ℓ2}(x)
  end1
```

- If the event e is a basic event controlling the state of the variable x, guarded by $g_{\ell_1,\ell_2}(x)$ and modified by the assignment $x := f_{\ell_1,\ell_2}$ where f is a function, the event e takes the form below.
- the function f_{ℓ_1,ℓ_2} is implementable.
- If the event e labels the link $\ell_1 \xrightarrow{e} \ell_2$ then the statement act_{ℓ_2} is defined as **when** $g_{\ell_1,\ell_2}(x)$ **then** $x := f_{\ell_1,\ell_2}(x)$.

Case 2 : Recursive Call of the Procedure

```
EVENT rec%PROC(h(x),y)%P(y)
any y
when
  ℓ = ℓ1
  g_{ℓ1,ℓ2}(x, y)
then
  ℓ := ℓ2
  x := f_{ℓ1,ℓ2}(x, y)
end1
```

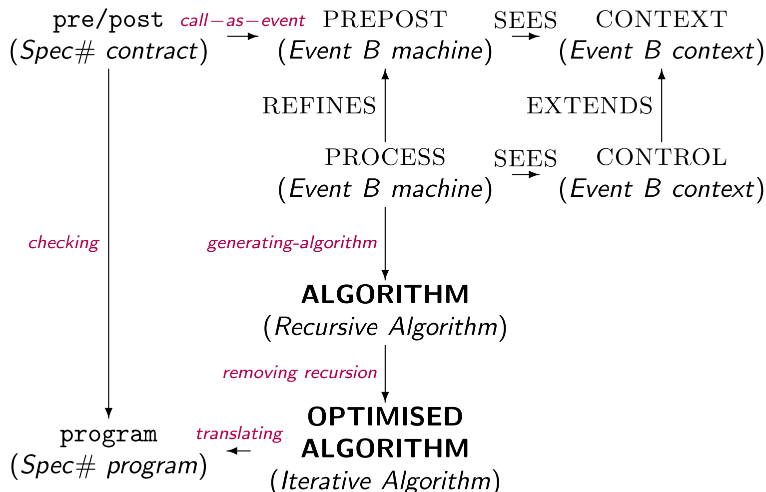
- The definition of the event e is not executable and the translation is driven by instances of the control variable ℓ in the guard (as $\ell = \ell_1$) and in the assignment ($\ell := \ell_2$).
- The statement act_{ℓ_2} is therefore defined as : $PROC(h(x), y)$.
- The choice of the event name is the responsibility of the writer of the EVENT B models, who must identify the case corresponding to a recursive call.

Case 3 : Non Recursive Call

```
EVENT call%APROC(h(x),y)%P(y)
any y
when
  ℓ = ℓ1
  g_{ℓ1,ℓ2}(x, y)
then
  ℓ := ℓ2
  x := f_{ℓ1,ℓ2}(x, y)
end1
```

- the event e can be transformed into a call of another procedure.
- The call is expressed by an event e, which we name $call\%\mathrm{APROC}(h(x), y)\%P(y)$ and the statement act_{ℓ_2} is defined as $\mathrm{APROC}(h(x), y)$.
- APROC is defined or to be defined in another framework.

Integrated Development Framework



Problems with the recursive algorithm

- Correct by construction according to the Event B side
- Limits of the Spec# tools with verifying recursive program
- Transforming recursive algorithms into iterative algorithm
- Applying the Spec# tools

Transforming the recursive algorithm into an iterative one

Theorem

The transformation is sound with respect to the pre and post specification.

```
procedure APROC(x; var y)
precondition P(x)
postcondition Q(x, y)
begin
local variables z
if C(x) then
  y := g(x);
else
  z := h(x, z);
  if D(x, z) then
    y := f(x, z)
  elseif E(x, z) then
    APROC(f1(x), y)
  else
    APROC(f2(x), y)
  endif
end
```

```
procedure BPROC(x; var y)
precondition P(x)
postcondition Q(x, y)
begin
local variables z
while not C(x) ∧ not D(x, z) do
  z := h(x, z);
  if E(x, z) then
    x := f1(x);
  else
    x := f2(x);
  endif
enddo
if C(x) then
  y := g(x);
elseif D(x, z) then
  y := f(x, z);
elseif E(x, z) then
  y := f1(x);
else
  y := f2(x);
  endif
end
```

PROCEDURE binsearch(t,val,lo,hi,ok,result)

procedure *binsearch*$(t, val, lo, hi, ok, result)$

precondition $\left(\begin{array}{l} t \in 0..t.Length \longrightarrow \mathbb{N} \\ \forall k.k \in lo..hi-1 \Rightarrow t(k) \leq t(k+1) \\ val \in \mathbb{N} \wedge lo, hi \in 0..t.Length \wedge lo \leq hi \end{array}\right)$

postcondition $\left(\begin{array}{l} ok = TRUE \Rightarrow \ t(result) = val \\ ok = FALSE \Rightarrow (\forall i.i \in lo..hi \Rightarrow t(i) \neq val \end{array}\right)$

begin

while *not* $\left(\begin{array}{l} lo = hi \wedge t(lo) = val \\ \vee \ lo = hi \wedge t(lo) \neq val \\ \vee \ lo < hi \wedge mi = (lo+hi)/2 \wedge t(mi) = val \end{array}\right)$ **do**

$mi := (lo+hi)/2;$

middle : $\left\{\left(\begin{array}{l} mi = (lo+hi)/2 \\ val < t(mi) \Rightarrow \forall k.k \in mi..hi \Rightarrow t(k) \neq val \\ val > t(mi) \Rightarrow \forall k.k \in lo..mi \Rightarrow t(k) \neq val \end{array}\right)\right\}$

if $mi + 1 \leq hi \wedge val > t(mi)$ **then**

$lo := mi + 1$

elseif $lo \leq mi - 1 \wedge val < t(mi)$ **then**

$hi := mi - 1$

enddo

if $lo = hi \wedge t(lo) = val$ **then**

$result := lo; ok := true$

elseif $lo = hi \wedge t(lo) \neq val$ **then**

$ok := false$

elseif $lo < hi \wedge t(mi) = val$ **then**

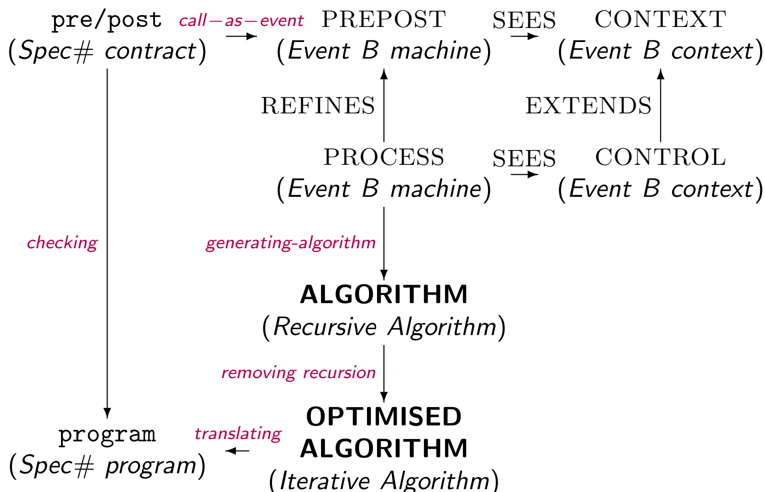
$result := mi; ok := true$

else $ok := false$

endif

end

General Framework Integrated Development Framework



Interpreting the algorithms within Spec#

- This is almost a one-to-one mapping : returning a value of -1 when our iterative algorithm sets *OK* to *false* and returning the index where the value is found when our iterative algorithm sets *OK* to *true*.
- The algorithm verified as correct, in less than 2 seconds using the Spec# programming system (version 2011-10-03).
- No user interaction is required in the verification as all assertions required (preconditions, postconditions and loop invariants) have been generated as part of the refinement and transformation of the initial abstract specification into the final iterative algorithm.
- Prior to formalising our transformation rules, our initial attempt at writing this iterative C# program contained an error.
- This error in the loop body, was due to our omission to check that the values of $mi + 1$ and $mi - 1$ were within the array bounds before narrowing the search space.
- This error was immediately detected by the Spec# programming system. The automatic verification of the final program is available online at `http://www.rise4fun.com/SpecSharp/psP4`.

Conclusion

- Our integrated development framework indicates where our transformations are used for producing a program that is *correct-by-construction*.
- The translation of the PROCESS machine into a recursive algorithm is straightforward and removes the control variable used to relate events when generating the code.
- Our experience shows that our approach assists students in developing and understanding the tasks of software specification and verification.
- It also makes different forms of formal software development more accessible to the Software Engineers, helping them to build correct and reliable software systems.
- Future work will include the development of adequate plugins, which will integrate and facilitate the co-operation between Spec# tools and RODIN tools.