Event-B Decomposition for Parallel Programs

Thai Son Hoang and Jean-Raymond Abrial

Department of Computer Science
Swiss Federal Institute of Technology Zürich (ETH Zürich)

ABZ2010, 22nd-25th February, 2010
Orford, Québec, Canada

Motivation

- Parallel programs.
- Event-B for discrete transition systems.
- Formal reasoning about parallel programs.
  - Work on "interference-free" (by S. Owicki and D. Gries).
  - Work on Rely/Guarantee (by C. Jones)
  - Conjoining specifications (M. Abadi and L. Lamport)
  - Parallel programs with Action Systems (R-J. Back and K. Sere)
  - etc.
- Example: the FindP program.

Outline

The FindP Program. Overview

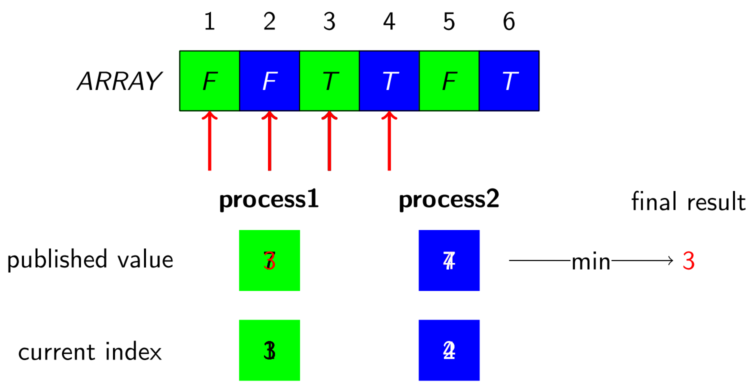
	1	2	3	$\cdots$	M
$ARRAY$	F	F	T	$\cdots$	T

Purpose of the FindP Program

Finding the first index k of a boolean array $ARRAY$, if there is one, such that $ARRAY(k) = T$. Otherwise, return $M + 1$.

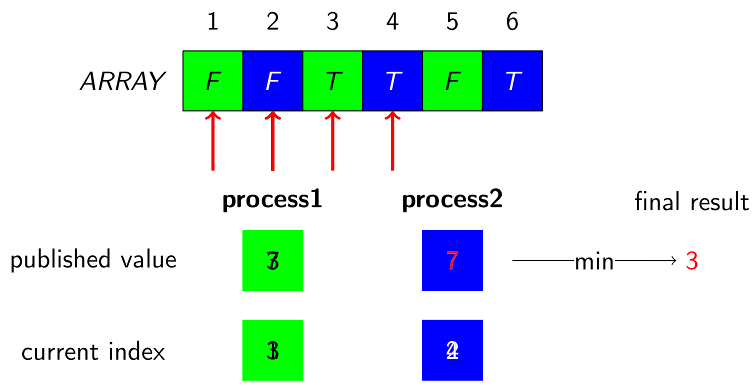
- The program use two parallel processes
  to check two parts $PART1$ and $PART2$ of the array separately.
- Each process publishes the first index that it finds.

FindP. First Animation



Motivation
Example. The "FindP" Program
Decomposition
Formal Development
Conclusions

FindP. Second Animation



Motivation
Example. The "FindP" Program
Decomposition
Formal Development
Conclusions

FindP with Parallel Processes

Main programs

$index1, index2 := min(PART1), min(PART2);$
$publish1, publish2 := M + 1, M + 1;$
process1 || **process2**;
$k := min(\{publish1, publish2\})$

Process: process1

while $index1 < min(\{publish1, publish2\})$ **do**
  if $ARRAY(index1) = T$ **then**
    $publish1 := index1$
  else
    $index1 :=$ `the-next-index-in-PART1`
  end
end

Motivation
Example. The "FindP" Program
Decomposition
Formal Development
Conclusions

A Detour. Atomicity Assumptions

- Shared variables: written by one process, read by the other process.
- Local variables: written and read by only one process.
- Statements involving only local tests and actions
  can be performed concurrently.
- Elementary atomic action:

  $local\_variable := shared\_variable$.

- Extended atomic action:

  if $local\_tests$ **then**
    $local\_variable := shared\_variable$
    $local\_actions$
  end

Unfolding process1 (1/2)

Original process1

```
while index1 < min({publish1, publish2}) do
  if ARRAY(index1) = T then
    publish1 := index1
  else
    index1 := the-next-index-in-PART1
  end
end
```


Unfolding process1 (2/2)

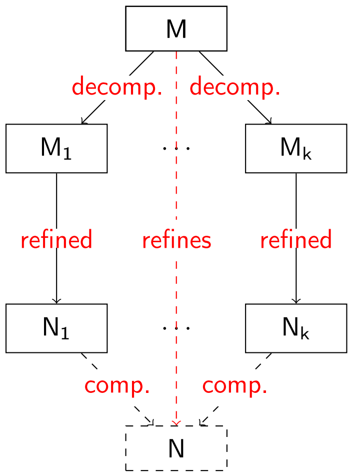
Original process1

```
while index1 < min({publish1, publish2}) do
  if ARRAY(index1) = T then
    publish1 := index1
  else
    index1 := the-next-index-in-PART1
  end
end
```

Unfold process1

```
1 : (read)        read1 := publish2;
2 :               if index1 < min({publish1, read1}) then
                    if ARRAY(index1) = T then
(found)               publish1 := index1 ; goto 3(end);
                    else
(inc)                 index1 := the-next-index-in-PART1 ; goto 1(read);
                    end
                  else
(not_found)         goto 3(end)
                  end
3 : (end)
```


Decomposition. An Overview



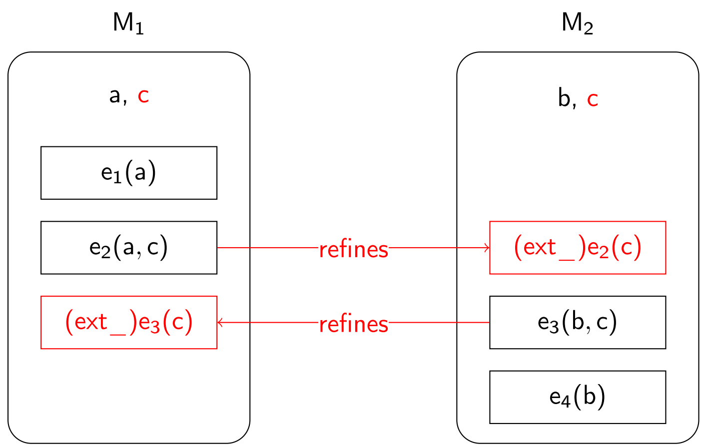
Shared Variables Decomposition in Event-B

- Sub-models share variables.
- The set of internal events of sub-models are disjoint.
- Each models having a set of external events
  to model the effect of these events on shared variables.

An Example (1/2)

- Assume model M has the following events:
  $e_1(a)$, $e_2(a,c)$, $e_3(b,c)$, $e_4(b)$.
- Events partition (chosen by the developer):
  - M_1: e_1, e_2.
  - M_2: e_3, e_4.
- Variables distribution (derived from events partition):
  - M_1: Private variable a, shared variable c.
  - M_2: Private variable b, shared variable c.
- Result:
  - M_1: Internal events $e_1(a)$, $e_2(a,c)$, external event $(ext\_)e_3(c)$.
  - M_2: Internal events $e_3(b,c)$, $e_4(c)$, external event $(ext\_)e_2(c)$.

An Example (2/2)



Constructing External Events

Informally ...

$(ext\_)e_2$ is the projection of e_2
on the state containing only external variables c.

More precisely ...

$M_1(a,c)$	$M_2(b,c)$
e_2 **any** t **where** $G(t,a,c)$ **then** $a,c :\mid Q(t,a,c,a',c')$ **end**	$(ext\_)e_2$ **any** t, a **where** $G(t,a,c)$ **then** $c :\mid \exists a' \cdot Q(t,a,c,a',c')$ **end**

Our Approach

A FORMAL approach combining refinement and decomposition

1. Specify *in-one-shot* to give the purpose of the program.
2. Refine the above specification by introducing the *shared variables*.
3. Decompose the model in the previous step according to processes.
4. Develop each sub-model from the previous step independently.

Key aspects

- Step 2: Derive the specification of future processes
  from the intended final result of the program.
- Step 4: Develop a process with the abstraction of other processes.
- Step 4: Refinement allows us to have different implementations.

The Context

The Context

	1	2	3	⋯	M
$ARRAY$	F	F	T	⋯	T

constants: $M, ARRAY$

axioms:
axm0_1 : $M \in \mathbb{N}_1$
axm0_2 : $ARRAY \in 1\,..\,M \rightarrow \mathrm{BOOL}$

Step 1. The One-shot Specification

The state and events

	1	2	3	⋯	M
$ARRAY$	F	F	T	⋯	T

variables: $result$

invariants:
inv0_1 : $result \in \mathbb{Z}$

init
begin
$result :\in \mathbb{Z}$
end

final
any k **where**
$k \in 1\,..\,M+1$
$\forall j \cdot j \in 1\,..\,k-1 \Rightarrow ARRAY(j) = F$
$k \neq M+1 \Rightarrow ARRAY(k) = T$
then
$result := k$
end

Step 2. Introducing the Shared Variables (1/5)

The published values of two processes

variables: $\ldots, finish1, finish2, publish1, publish2$

init
begin
$\ldots$
$finish1 := F$
$finish2 := F$
$publish1 := M+1$
$publish2 := M+1$
end

Step 2. Introducing the Shared Variables (2/5)

Refinement of the final event

(abs_)final
any k **where**
$k \in 1\,..\,M+1$
$\forall j \cdot j \in 1\,..\,k-1 \Rightarrow ARRAY(j) = F$
$k \neq M+1 \Rightarrow ARRAY(k) = T$
then
$result := k$
end

(conc_)final
when
$finish1 = T$
$finish2 = T$
with
$k = min(\{publish1, publish2\}$
then
$result := min(\{publish1, publish2\})$
end

Step 2. Introducing the Shared Variables (3/5)

The invariants

invariants:

$publish1 \neq M+1 \Rightarrow finish1 = T$

$publish1 \neq M+1 \Rightarrow publish1 \in PART1$

$publish1 \neq M+1 \Rightarrow ARRAY(publish1) = T$

$publish1 \neq M+1 \Rightarrow$
$(\forall i \cdot i \in PART1 \wedge i < publish1 \Rightarrow ARRAY(i) = F)$

$finish1 = T \wedge publish1 = M+1 \Rightarrow$
$(\forall i \cdot i \in PART1 \wedge i < publish2 \Rightarrow ARRAY(i) = F)$

...

Step 2. Introducing the Shared Variables (4/5)

found_1 event

invariants:

$publish1 \neq M+1 \Rightarrow publish1 \in PART1$

$publish1 \neq M+1 \Rightarrow ARRAY(publish1) = T$

$publish1 \neq M+1 \Rightarrow$
$(\forall i \cdot i \in PART1 \wedge i < publish1 \Rightarrow ARRAY(i) = F)$

found_1
any k **where**
$finish1 = F$
$k \in PART1$
$ARRAY(k) = T$
$\forall i \cdot i \in PART1 \wedge i < k \Rightarrow ARRAY(i) = F$
$\mathtt{publish1} = \mathtt{M} + 1$
then
$finish1, publish1 := T, k$
end

Step 2. Introducing the Shared Variables (5/5)

not_found_1 event

invariants:

$finish1 = T \wedge publish1 = M+1 \Rightarrow$
$(\forall i \cdot i \in PART1 \wedge i < publish2 \Rightarrow ARRAY(i) = F)$

not_found_1
when
$finish1 = F$
$\forall i \cdot i \in PART1 \wedge i < publish2 \Rightarrow ARRAY(i) = F$
then
$finish1 := T$
end

Step 3. Decomposition

Event partition

main: final

process1: not_found_1 and found_1.

process2: not_found_2 and found_2.

Step 4. Further Refinements (1/2)

Constraints during refinement

- Shared variables cannot be removed.
- External events cannot be changed.
- External events must preserve the newly introduced invariants.

Superposition refinements strategy

1. 1st Ref.: Introducing the local index of the array.
2. 2nd Ref.: Introducing the read value.
3. 3rd Ref.: Introducing the address counter for sequencing the events.

Step 4. Further Refinements (2/2)

Final events of process1

read1
when
$address1 = 1$
then
$address1, read1 := 2, publish2$
end

not_found_1
when
$address1 = 2$
$\neg(index1 < min(\{publish1, read1\}))$
then
$address1, finish1 := 3, T$
end

found_1
when
$address1 = 2$
$index1 < min(\{publish1, read1\})$
$ARRAY(index1) = T$
then
$address1 := 3$
$finish1 := T$
$publish1 := index1$
end

inc_1
any i **where**
$address1 = 2$
$index1 < min(\{publish1, read1\})$
$ARRAY(index1) = F$
$i \neq M + 1 \Rightarrow i \in PART1$
$index1 < i$
$\forall j \cdot j \in PART1 \land index1 < j \Rightarrow i \leq j$
then
$address1, index1 := 1, i$
end

Proof Statistics

Proof Statistics

Developing using the RODIN Platform with decomposition plug-in.

Model	Total	Auto.(%)	Manual (%)
Initial context	0	0 (N/A)	0 (N/A)
Initial model	3	3 (100%)	0 (0%)
First extended context	0	0 (N/A)	0 (N/A)
First refinement	46	44 (96%)	2 (4%)
First sub-refinement	14	10 (71%)	4 (29%)
Second sub-refinement	6	5 (83%)	1 (17%)
Third sub-refinement	22	16 (73%)	6 (27%)
Total	91	78 (86%)	13 (14%)

Conclusions and Future Work

- Decomposition allows us to reduce the complexity in developing parallel programs.
- The interactions between processes are introduced early in the development.
- Apply the method to other standard parallel programs.

For Further Reading I

J-R. Abrial.
Event model decomposition,.
ETH Zurich Tech. Rep., 2009.

C. Jones.
Splitting atoms safely,.
Theor. Comput. Sci., 2007.

S. Owicki and D.Gries.
An Axiomatic Proof Technique for Parallel Programs I.
Acta Inf. 6, 1976.

Interference-free

- Notion "Interference-free" from Owicki-Gries.

  Consider a proof of $\{P\}S\{Q\}$ and a statement T with precondition $pre(T)$, T does not interfere with $\{P\}S\{Q\}$ if

  Inf1 $\{Q \wedge pre(T)\}T\{Q\}$.

  Inf2 Let S' be any statement within S, then $\{pre(S') \wedge pre(T)\}T\{pre(S')\}$
- Compare to our work:
  - S is an internal event of **process1**.
  - T is an external event of **process1**.
  - The condition **Inf1** is proved at the level before decomposing.
  - S' is introduced during the refinement of S.
  - $pre(S')$ are the invariants introduced during refinement.
  - The condition **Inf2** is proved during refinement:
    external event preserves invariants.
  - Advantage of our approach: T is at the abstract level.

Rely/Guarantee (1/2)

- Rely/Guarantee method from Jones.
  - Extending the Hoare's triple to include the Rely/Guarantee conditions R and G, i.e. $\{P, R\}S\{G, Q\}$.
  - An example rule for parallel composition

$$\textbf{PAR-I}\quad \frac{\begin{array}{l} R \vee G_1 \Rightarrow R_2 \\ R \vee G_2 \Rightarrow R_1 \\ G_1 \vee G_2 \Rightarrow G \\ \{P, R_1\}S_1\{G_1, Q_1\} \\ \{P, R_2\}S_2\{G_2, Q_2\} \end{array}}{\{P, R\}S_1 || S_2\{G, Q_1 \wedge Q_2\}}$$

Rely/Guarantee (2/2)

- The rely/guarantee condition are relations over the two states.
- A pair of external/internal events
  - External event: Rely condition.
  - Internal event: Guarantee condition.
- $\Rightarrow$ relation of rely/guarantee conditions becomes event refinement.
- The generated pair of external/internal events
  satisfies the rules for parallel composition.
- However, this generated external events might be too "concrete".
- In the FindP example, the external events just need to guarantee
  to decrease the published value monotonically.
- User-defined external events?