# Event-B Decomposition for Parallel Programs

Thai Son Hoang

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

Dagstuhl Seminar, 13th-18th September, 2009

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

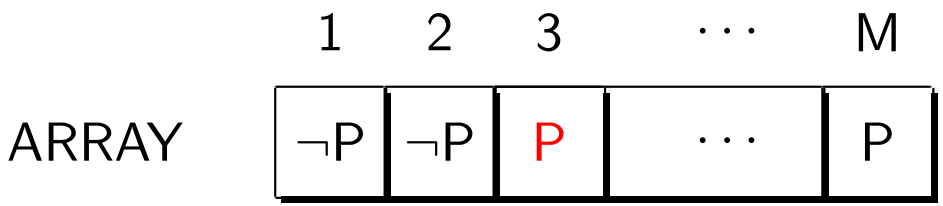
# Outline



# Motivation

- Parallel programs.
- Event-B for discrete transition systems.
- Work on "interference-free" (by S. Owicki and D. Gries).
- Work on Rely/Guarantee (by C. Jones)
- Example: FindP program.

# Overview




## FindP Program

Finding the first index $k$ of an array $ARRAY$, if there is one, such that $ARRAY(k)$ satisfied some property $P$. 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.

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

# 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** $ARRAY(index1) = \mathrm{TRUE}$ **then**
    $publish1 := index1$
  **else**
    $index1 :=$ `the-next-index-in-PART1`
  **end**
**end**

# FindP with Parallel Processes

## Main programs

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

## Process: **process1**

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

# Unfolding process1

## Process: **process1**

$1 : (read)$ $\quad read1 := publish2;$
$2 :$ $\quad$ **if** $index1 < min(\{publish1, read1\})$ **then**
$\quad$ **if** $ARRAY(index1) = \mathrm{TRUE}$ **then**
$(found)$ $\quad publish1 := index1 \ \| \ goto\ 3(end);$
$\quad$ **else**
$(inc)$ $\quad index1 := \texttt{next-in-PART1} \ \| \ goto\ 1(read);$
$\quad$ **end**
$\quad$ **else**
$(not\_found)$ $\quad goto\ 3(end)$
$\quad$ **end**
$3 : (end)$

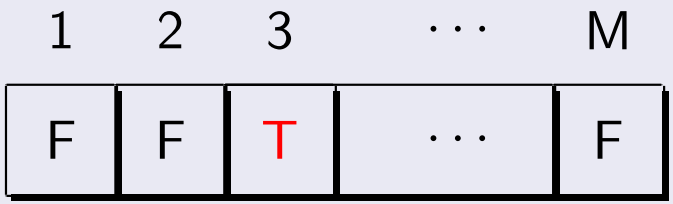
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Ideas for Decomposition

- Specify the program globally.
- Decomposing the program into different processes: main, process1, process2.

# The Context

## The Context




**constants:** $M, ARRAY$

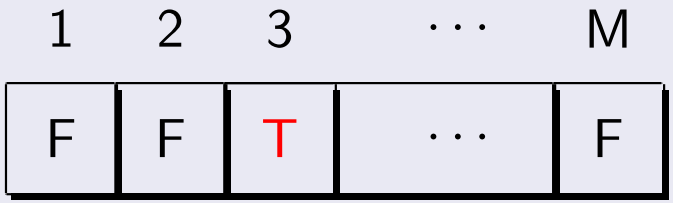
**axioms:**

**axm0_1:** $M \in \mathbb{N}_1$

**axm0_2:** $ARRAY \in 1 \mathinner{..} M \rightarrow \mathrm{BOOL}$

# The Specification

## The state and events




**variables:** *result*

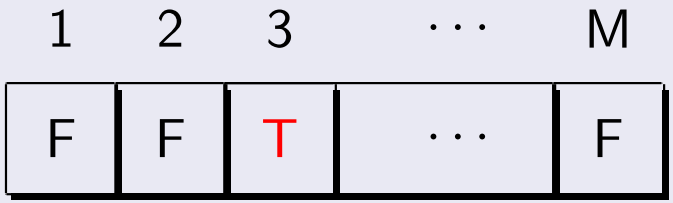
**invariants:**
**inv0_1:** $result \in \mathbb{Z}$

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

final
**any** $k$ **where**
$k \in 1 \,..\, M+1$
$\forall j \cdot j \in 1 \,..\, k-1 \Rightarrow ARRAY(j) = \text{FALSE}$
$k \neq M+1 \Rightarrow ARRAY(k) = \text{TRUE}$
**then**
$result := k$
**end**

# The Specification

## The state and events




**variables:** *result*

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

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

final
**any** $k$ **where**
$k \in 1 \,..\, M+1$
$\forall j \cdot j \in 1 \,..\, k-1 \Rightarrow ARRAY(j) = \text{FALSE}$
$k \neq M+1 \Rightarrow ARRAY(k) = \text{TRUE}$
**then**
$result := k$
**end**

# The Refinement

## The published values of two processes

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

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

# The Final Event

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

(conc_)final
**refines** (abs_)final
**when**
$finish1 = \text{TRUE}$
$finish2 = \text{TRUE}$
**with**
$k = min(\{publish1, publish2\})$
**then**
$result := min(\{publish1, publish2\})$
**end**

# The Invariants

## The invariants

**invariants:**

$publish1 \neq M + 1 \Rightarrow finish1 = \text{TRUE}$
$publish1 \neq M + 1 \Rightarrow publish1 \in PART1$
$publish1 \neq M + 1 \Rightarrow ARRAY(publish1) = \text{TRUE}$
$publish1 \neq M + 1 \Rightarrow (\forall i \cdot i \in PART1 \wedge i < publish1 \Rightarrow ARRAY(i) = \text{FALSE})$
$finish1 = \text{TRUE} \wedge publish1 = M + 1 \Rightarrow$
$(\forall i \cdot i \in PART1 \wedge i < publish2 \Rightarrow ARRAY(i) = \text{FALSE})$
$\ldots$

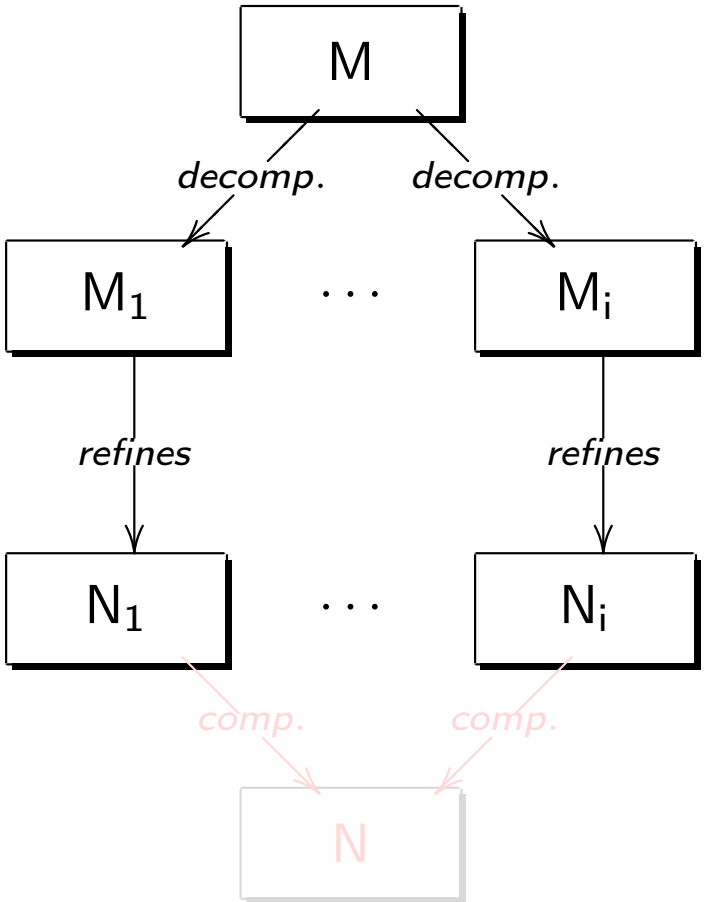
ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

# The Abstract Events for **process1**.

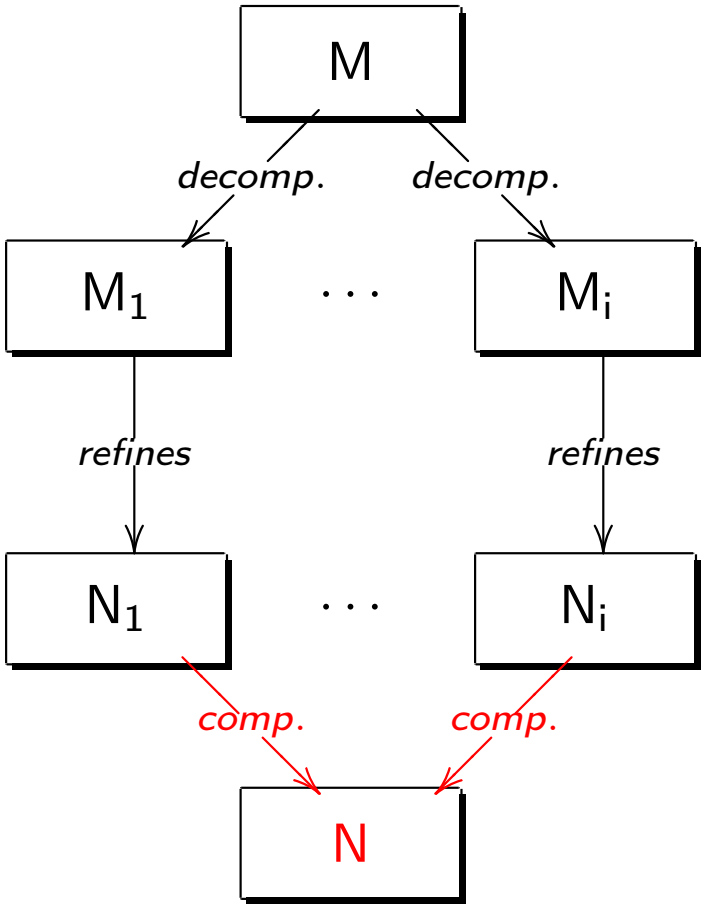
found_1
**any** $k$ **where**
$finish1 = \text{FALSE}$
$k \in PART1$
$ARRAY(k) = \text{TRUE}$
$\forall i \cdot i \in PART1 \wedge i < k \Rightarrow ARRAY(i) = \text{FALSE}$
$publish1 = M + 1$
**then**
$finish1 := \text{TRUE}$
$publish1 := k$
**end**

not_found_1
**when**
$finish1 = \text{FALSE}$
$\forall i \cdot i \in PART1 \wedge i < publish2 \Rightarrow ARRAY(i) = \text{FALSE}$
**then**
$finish1 := \text{TRUE}$
**end**

# Overview



# Overview



# Shared Variables Decomposition in Event-B

(Also called A-Style decomposition)

- Sub-models shared 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)

- Assume model M has the following events:
  $e_1(a)$, $e_2(a, c)$, $e_3(b, c)$, $e_4(b)$.
- Events partition:
  - $M_1$: $e_1$, $e_2$.
  - $M_2$: $e_3$, $e_4$.
- Variables distribution (calculated 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 $(ext\_)e_2(c)$.

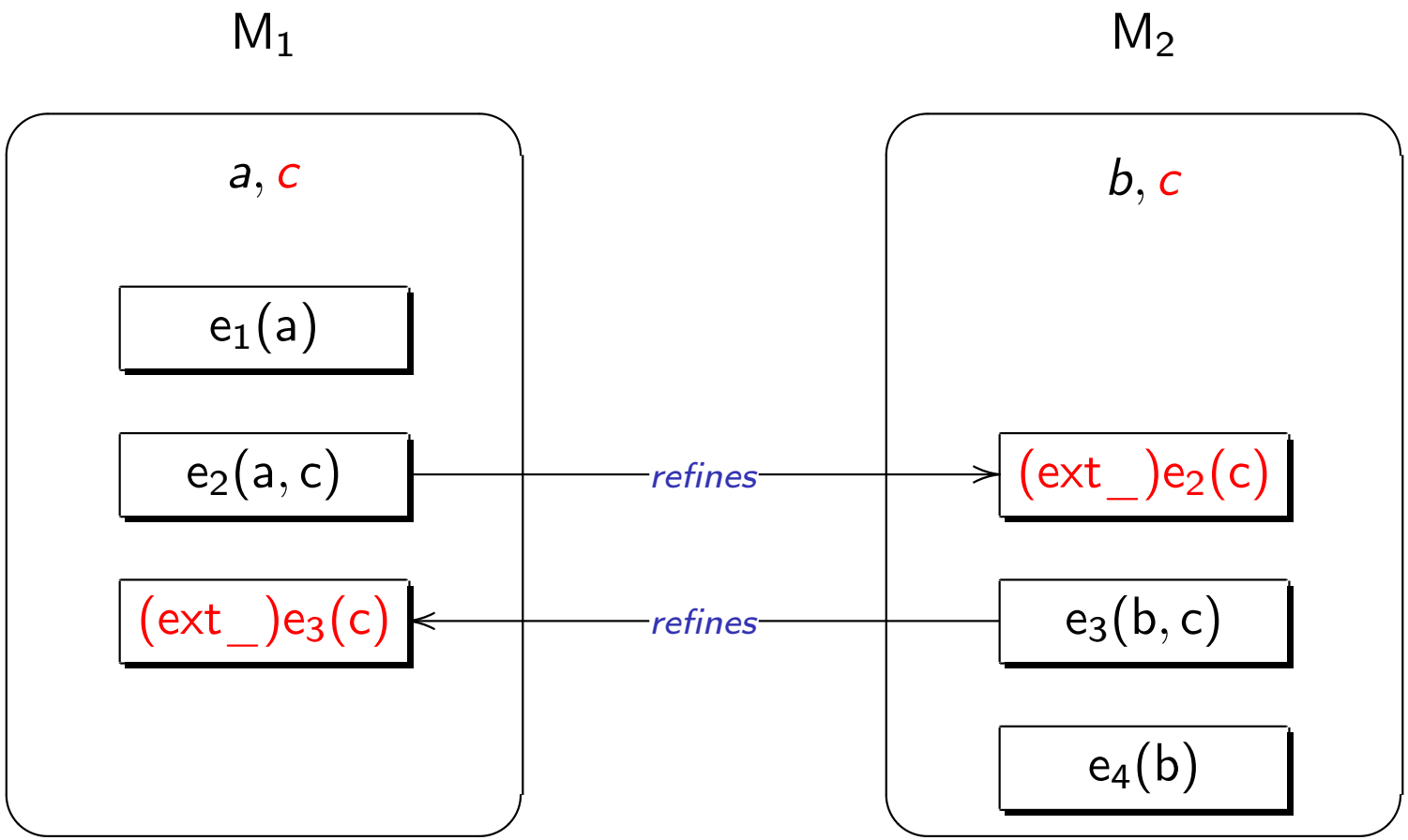
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# An Example (1)

- Assume model M has the following events:
  $e_1(a)$, $e_2(a, c)$, $e_3(b, c)$, $e_4(b)$.
- Events partition:
  - $M_1$: $e_1$, $e_2$.
  - $M_2$: $e_3$, $e_4$.
- Variables distribution (calculated 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 $(ext\_)e_2(c)$.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# An Example (2)



# Constructing External Events

## Informally ...

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

## More precisely ...

$M_1(a, c)$

$e_2$
**any** $t$ **where**
$G(t, a, c)$
**then**
$a, c :| Q(t, a, c, a', c')$
**end**

$M_2(b, c)$

(ext_)$e_2$
**any** $t, a$ **where**
$G(t, a, c)$
**then**
$c :| \exists a' \cdot Q(t, a, c, a', c')$
**end**

# Back to FindP

## Decomposition Ideas

**main:** final

**process1:** not_found_1 and found_1.

**process2:** not_found_2 and found_2.

# Refinement Strategy for **process1**

## Constraints during refinement

- Shared variables and external events cannot be refined.
- External events must preserve the newly introduced invariants.

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.

# Related Work (1)

- 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 $\{pre(S') \wedge pre(T)\}T\{pre(S')\}$

- Comparing the 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.

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

# Related Work (1)

- 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 $\{pre(S') \wedge pre(T)\}T\{pre(S')\}$

- Comparing the 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.

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

# Related Work (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

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

# Related Work (3)

- 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.

# For Further Reading I


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.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich