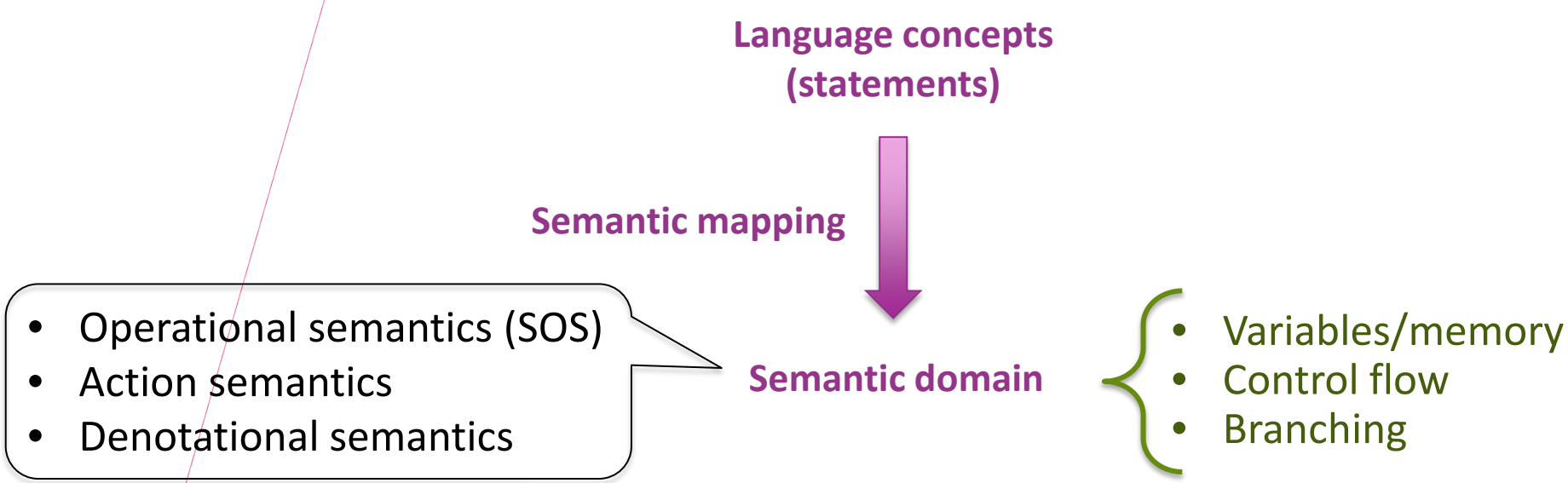
Technische Universiteit
Eindhoven
University of Technology

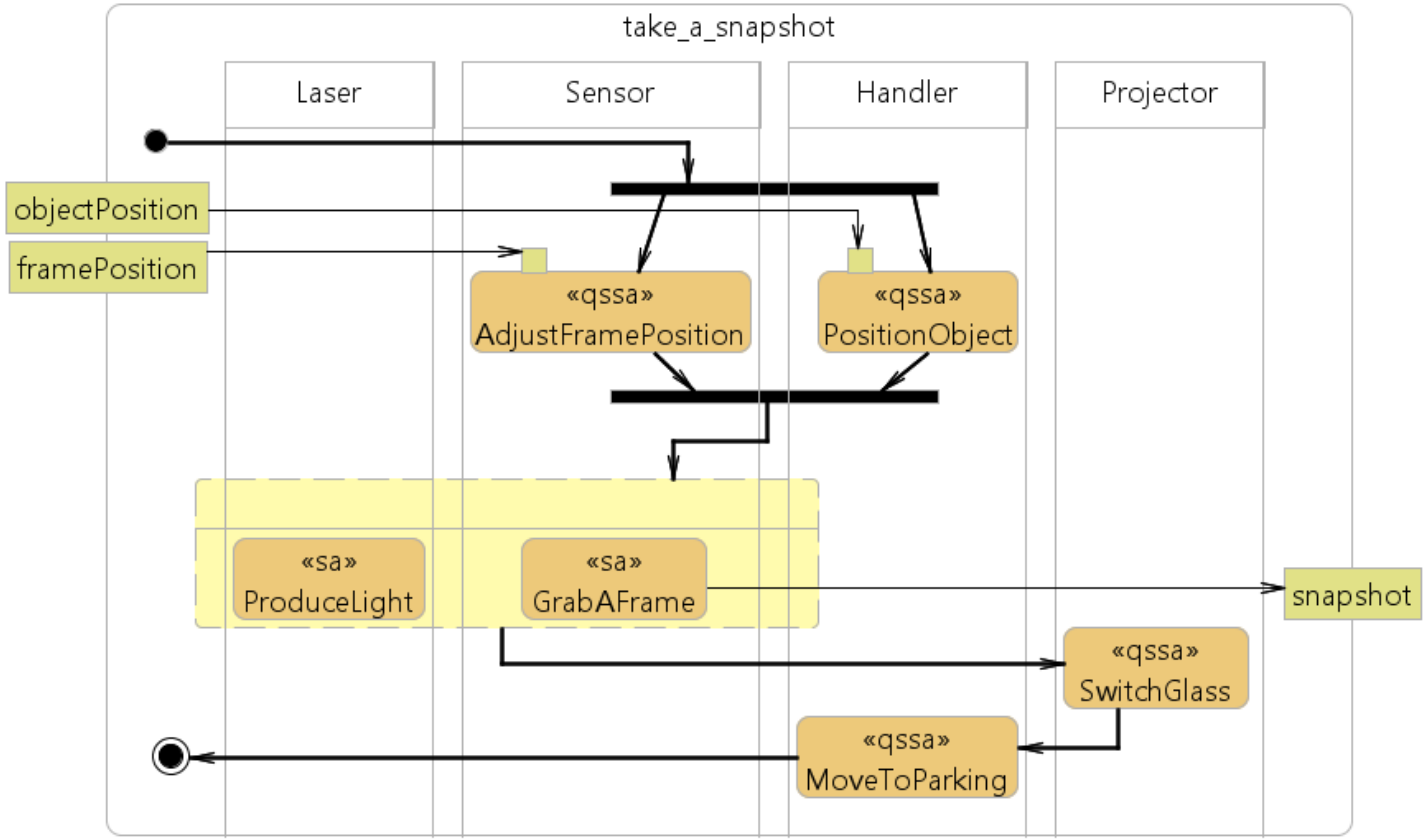
Event-B specification templates for defining dynamic semantics of DSLs

Ulyana Tikhonova
u.tikhonova@tue.nl

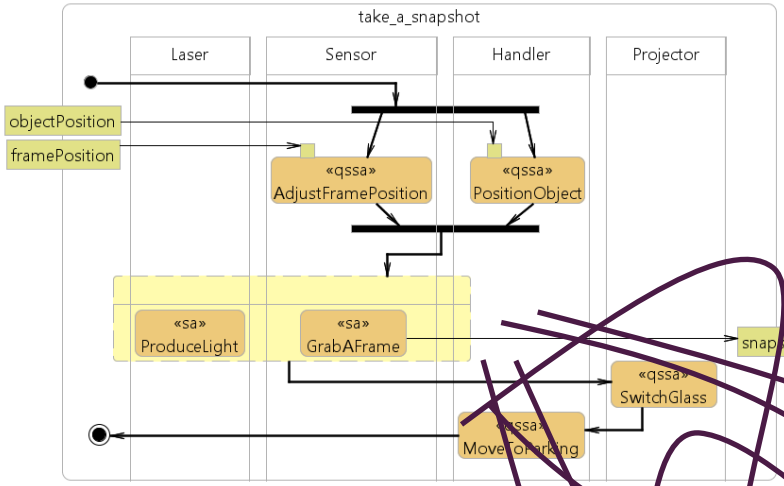
Mark van den Brand, Tim Willemse,
Tom Verhoeff, Maarten Manders



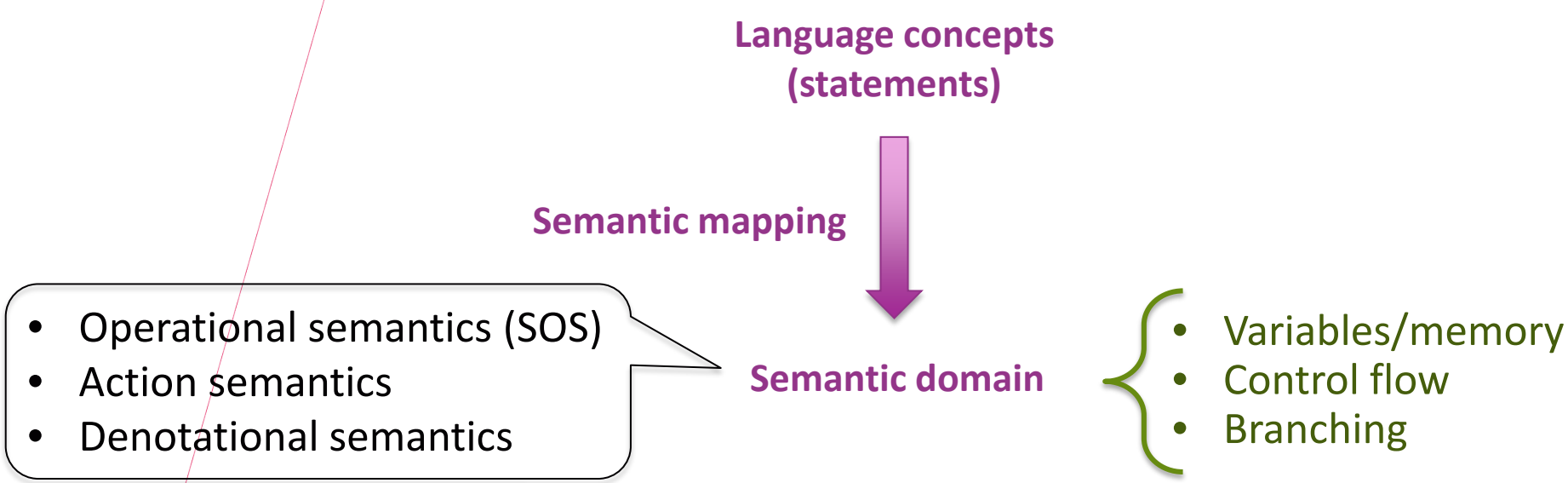
Defining dynamic semantics of programming languages



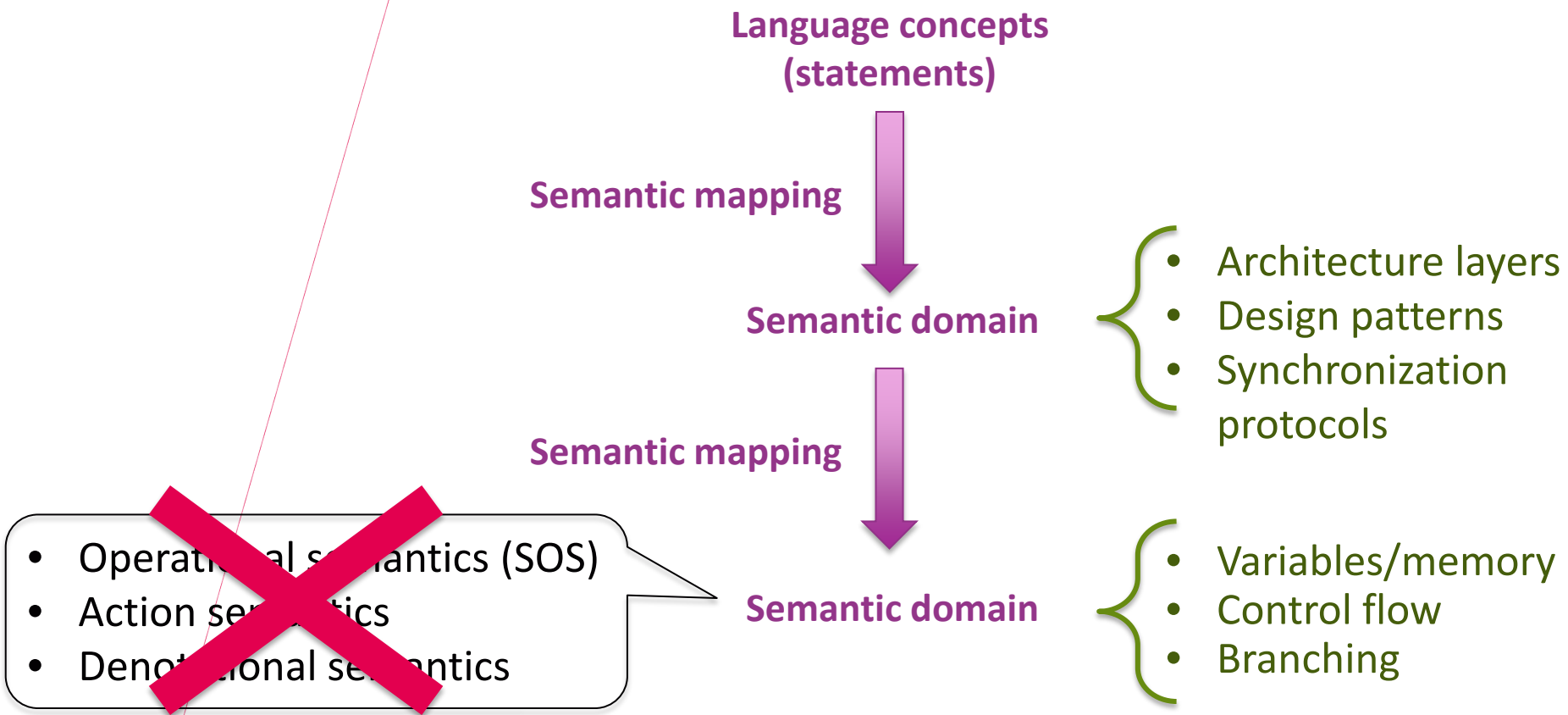
Domain-Specific Languages (DSLs)



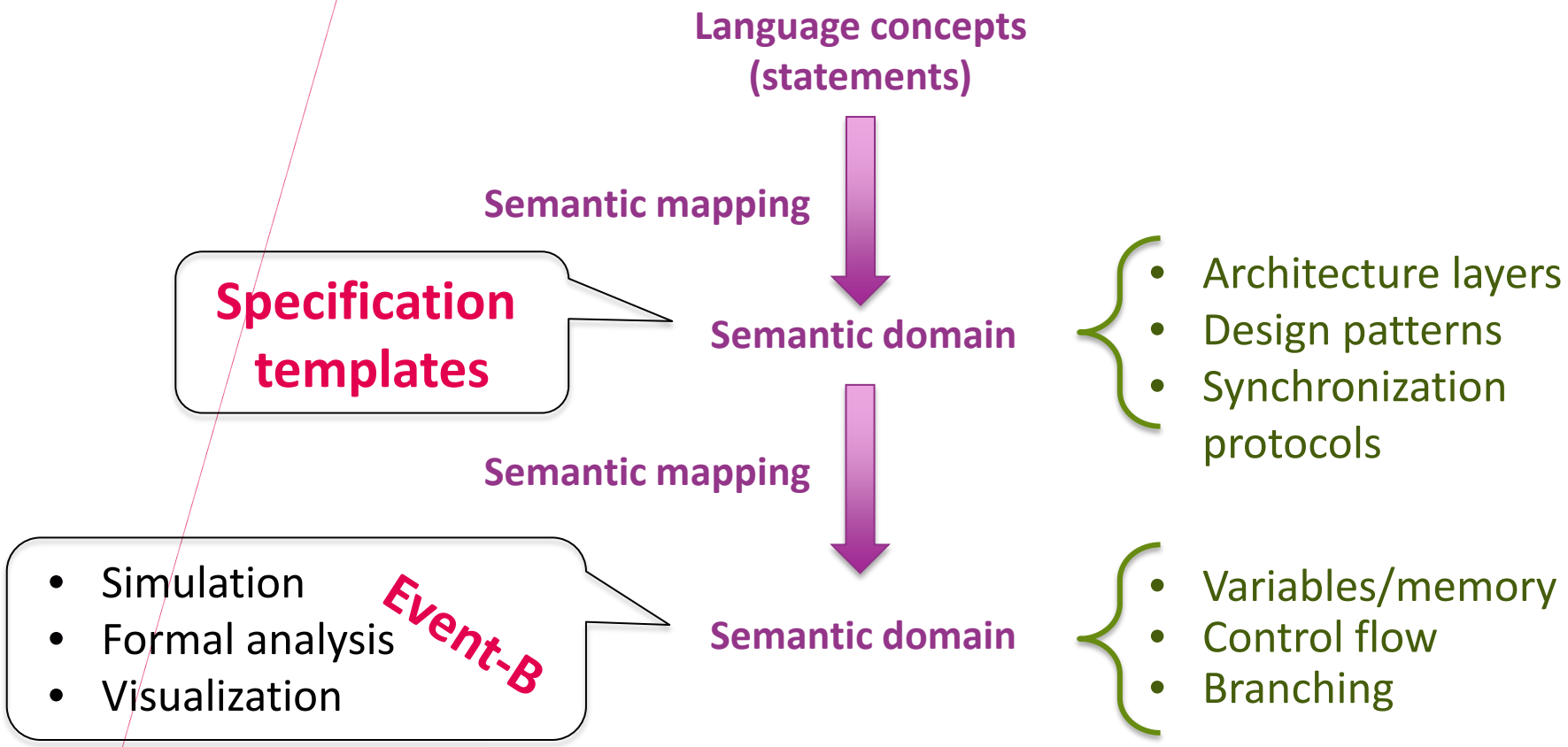
Code generation



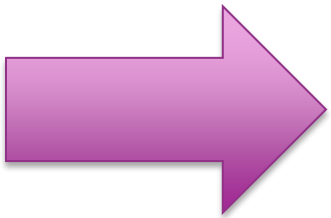
Defining dynamic semantics of domain specific languages

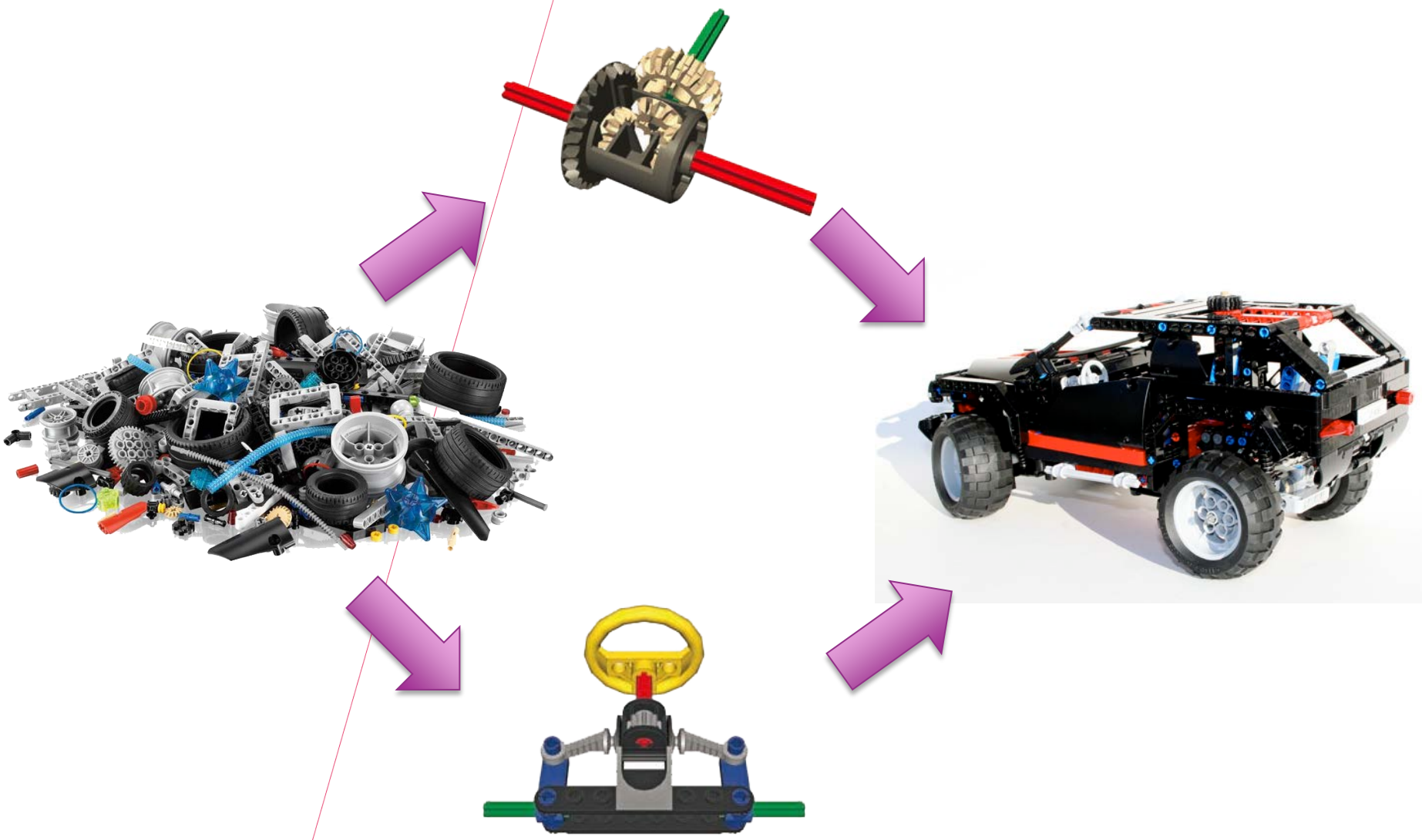


Defining dynamic semantics of domain specific languages



Defining dynamic semantics of domain specific languages





VARIABLES curr_job, curr_la, la_input, ssa_output

INVARIANTS

- la_input ∈ ℕ ⇸ LogicalActions
- ssa_output ∈ ℕ ⇸ SSActions
- curr_job ∈ ℙ(SSAOccurences)
- curr_la ∈ LogicalActions

EVENTS

Initialisation

- curr_la :∈ LogicalActions
- curr_job ≔ ∅
- la_input ≔ ∅
- ssa_output ≔ ∅

request_la (la, n)

- **where**
- la ∈ LogicalActions
- curr_job = ∅
- n ∈ ℕ
- la_input ≠ ∅ ⇒ ∀ i · i ∈ dom(la_input) ⇒ n > i
- **then**
- curr_job ≔ dom(LALabelDef(la))
- curr_la ≔ la
- la_input ≔ la_input ∪ { n ↦ la }

request_ssa (ssaction, occurence)

- **where**
- occurence ∈ curr_job
- occurence ↦ ssaction ∈ LALabelDef (curr_la)
- **then**
- curr_job ≔ curr_job \ {occurence}

execute_ssa (ssaction, n)

- **where**
- ssaction ∈ SSActions
- n ∈ ℕ
- ssa_output ≠ ∅ ⇒ ∀ i · i ∈ dom(ssa_output) ⇒ n > i
- **then**
- ssa_output ≔ ssa_output ∪ { n ↦ ssaction }

END

```
static void MakeAtLeast<T>(T[] list, T lowest)
where T : IComparable<T>
{
      for (int i = 0; i < list.Length; i++)
            if (list[i].CompareTo(lowest) < 0)
                  list[i] = lowest;
}
static void Main()
{
      int[] array = { 0, 1, 2, 3 };
      MakeAtLeast<int>(array, 2);
}
```

Generic programming: reuse of code

MACHINE queue_machine

SEES queue_context

VARIABLES queue

INVARIANTS

inv1: queue ∈ ℕ ⇸ MyType

EVENTS

INITIALISATION ≙

act1: queue ≔ ∅

END

enqueue≙

ANY element, index

WHERE

grd1: element ∈ MyType

grd2: index ↦ element ∈ queue

grd3: ∀i·i ∈ dom(queue) ⇒ index ≤ i

THEN

act1: queue ≔ queue \ {index ↦ element}

END

Generic instantiation

dequeue≙

ANY element, index

WHERE

grd1: element ∈ MyType

grd2: index ∈ ℕ

grd3: queue ≠ ∅ ⇒

(∀i·i ∈ dom(queue) ⇒ index > i)

grd4: {index ↦ element} ∈ ℕ ⇸ MyType

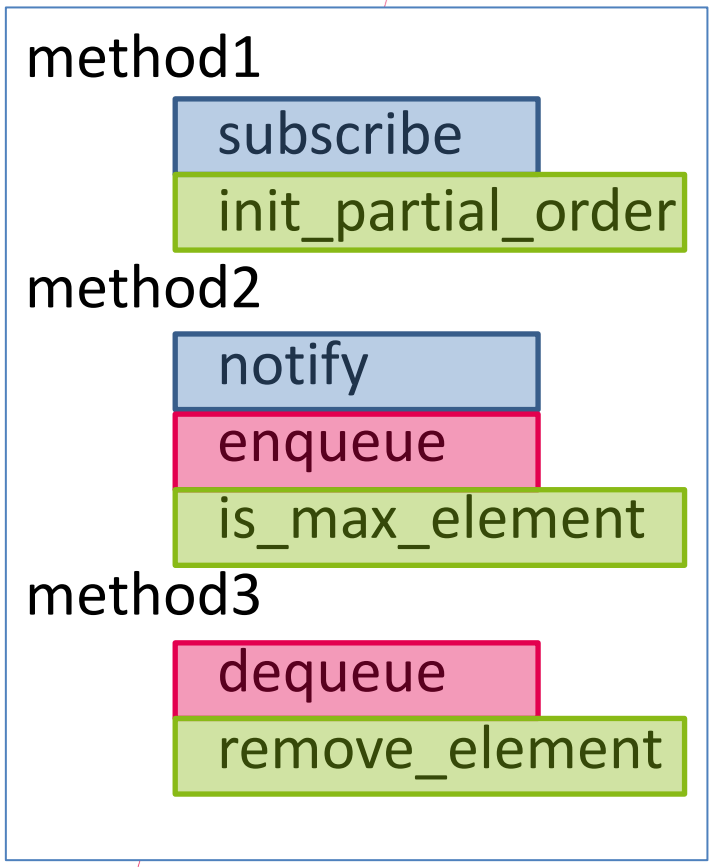
grd5: index ∉ dom(queue)

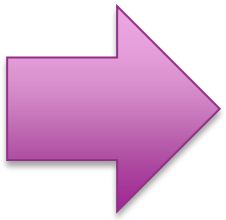
THEN

act2: queue ≔ queue ∪ {index ↦ element}

END

END





Queue
enqueue
dequeue

Listener
subscribe
notify

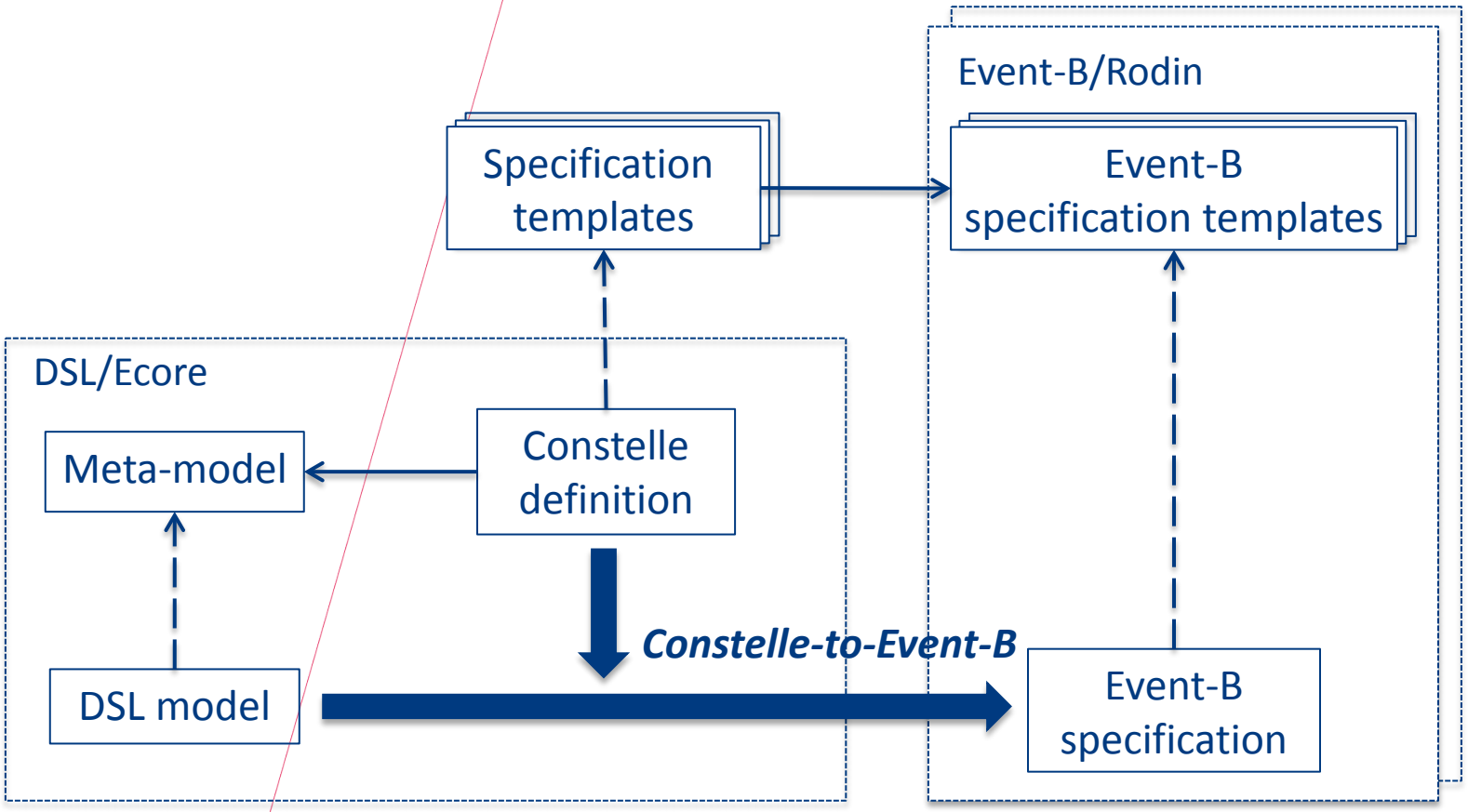
Partial Order
init_partial_order
is_max_element
remove_element

Aspect Oriented Programming: cross cutting concerns

Shared-event composition

	Listener	Queue	Partial Order
method1	subscribe		init_partial_order
method2	notify	enqueue	is_max_element
method3		dequeue	remove_element

Specializations of specification templates from the generic library


Event-B/Rodin
Specification templates
Event-B specification templates
DSL/Ecore
Meta-model
Constelle definition
Constelle-to-Event-B
DSL model
Event-B specification

Conclusions

- Constelle as a front-end
  - Reuse of Event-B code via generic programming
  - Clear design via composition/mapping of aspects
  - Intermediate layer for bridging different technological platforms
- Event-B as a back-end
  - Generic instantiation
  - (Shared event) composition
  - **Library of reusable specification templates**