

Reusable specification templates

for defining dynamic semantics of DSLs

Ulyana Tikhonova

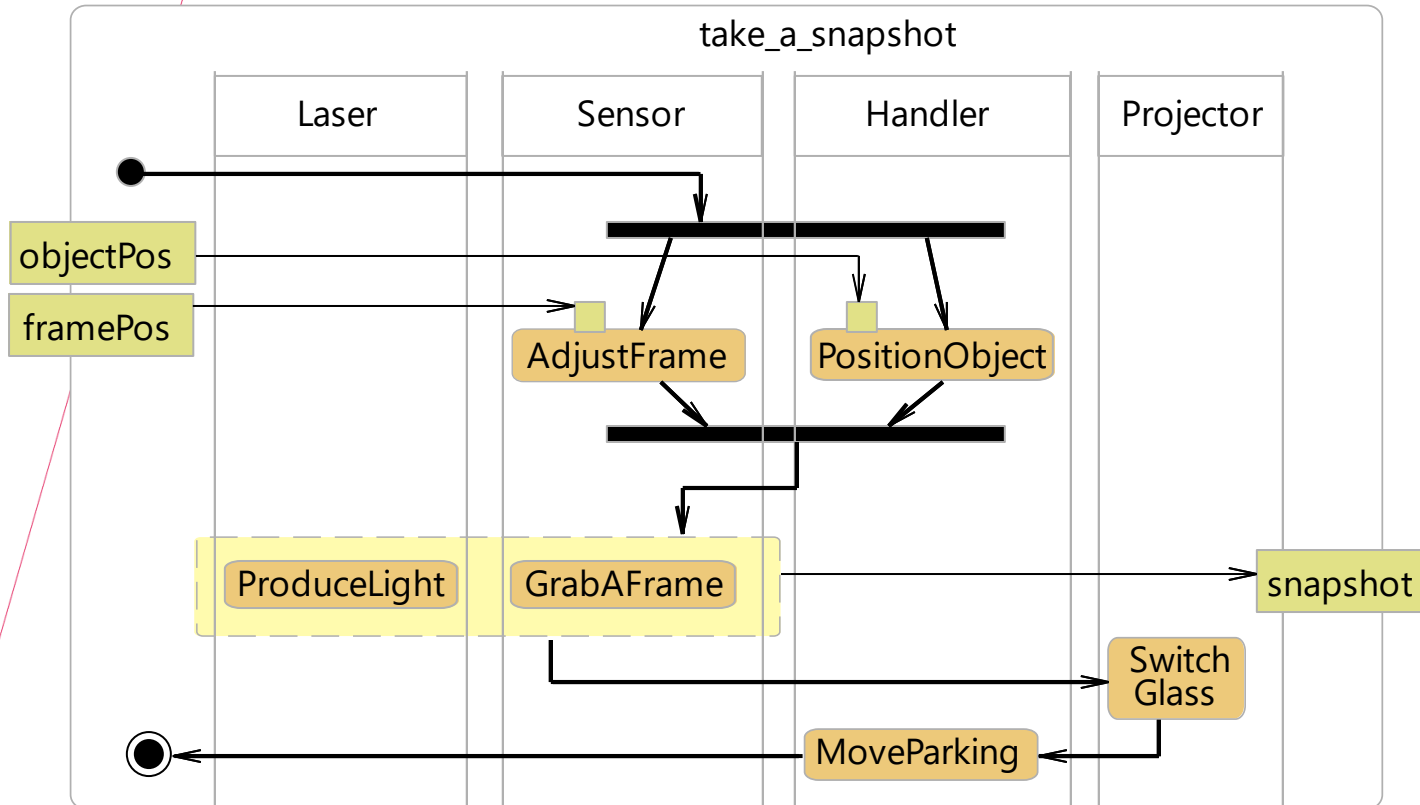
ulyana.tikhonova@constelle.net



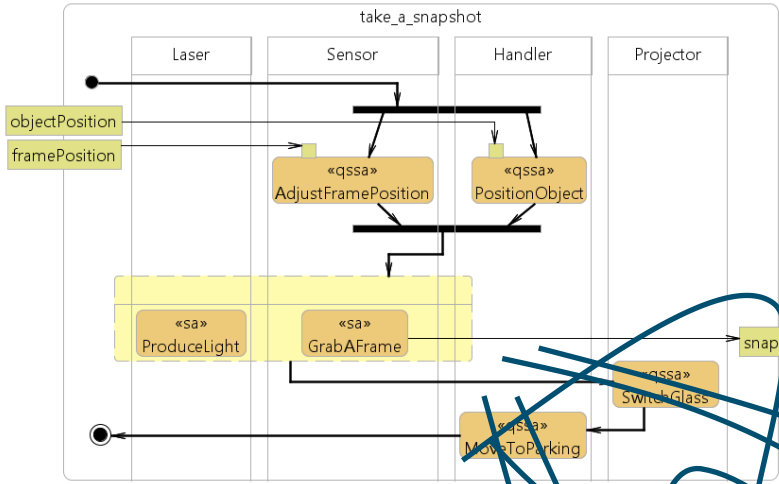
Tim Willemse



Mark van den Brand

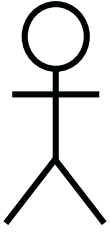


Domain Specific Languages (DSLs)

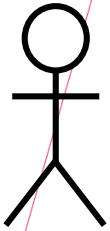


Code generation





DSL developer

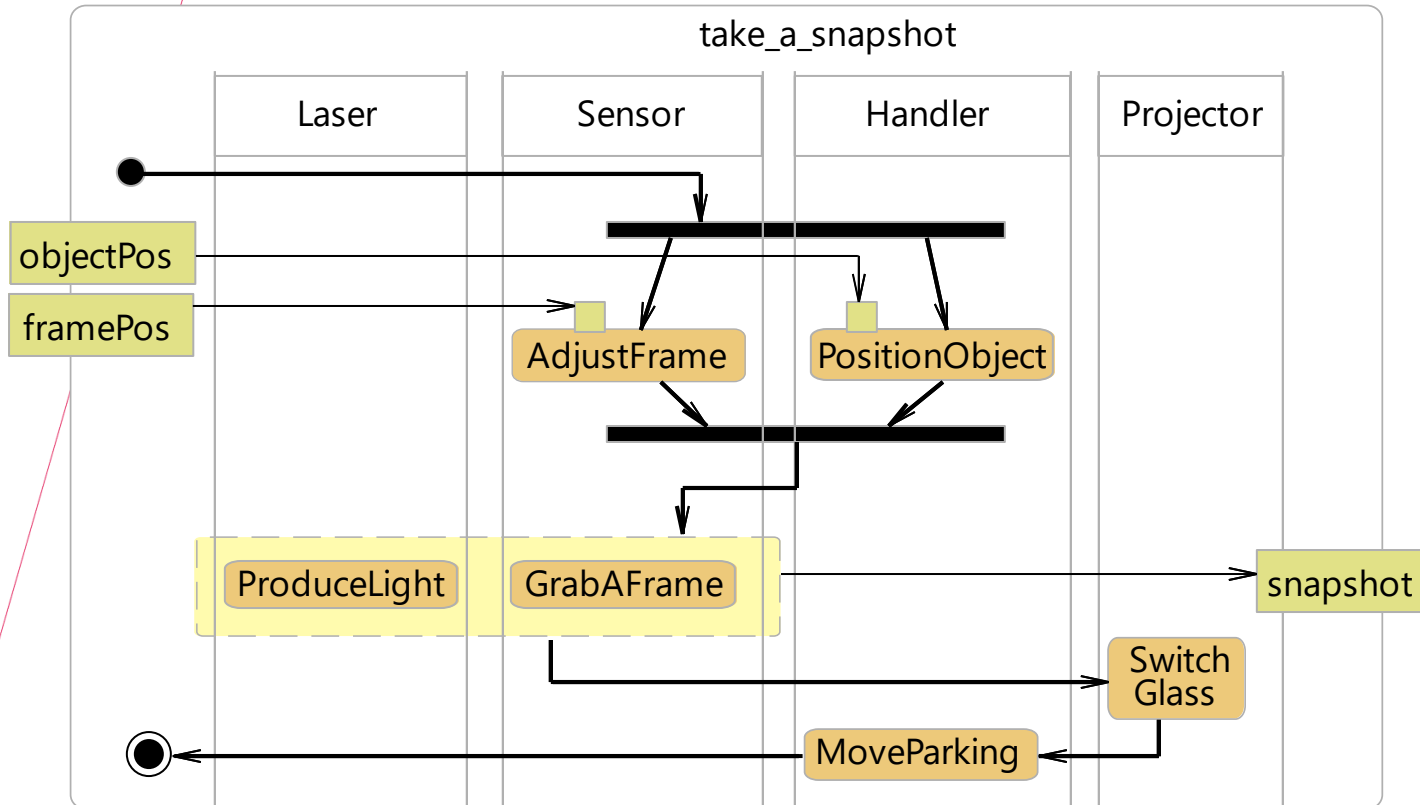


DSL user

- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics
- To understand and debug DSL programs

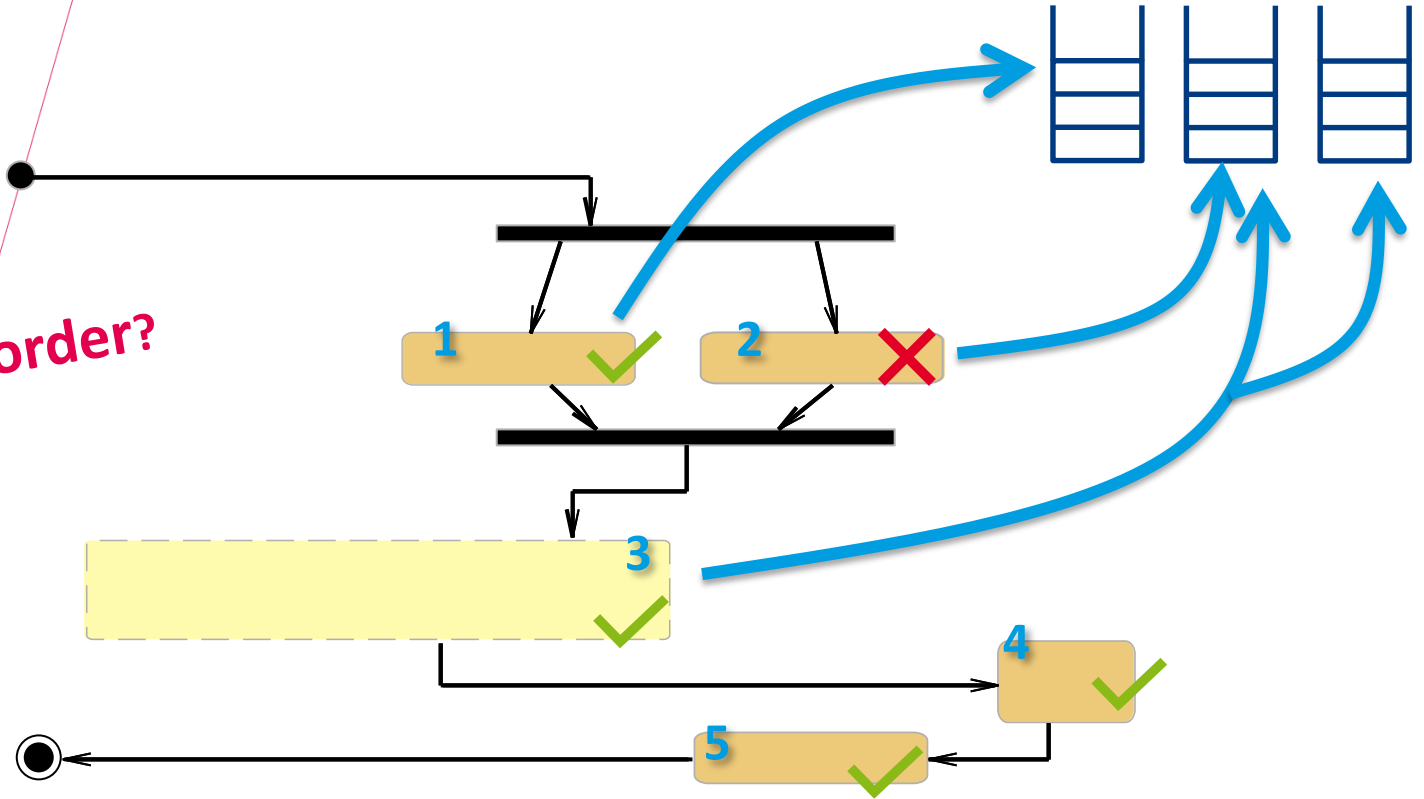
Precise and executable definition

Why to define dynamic semantics?



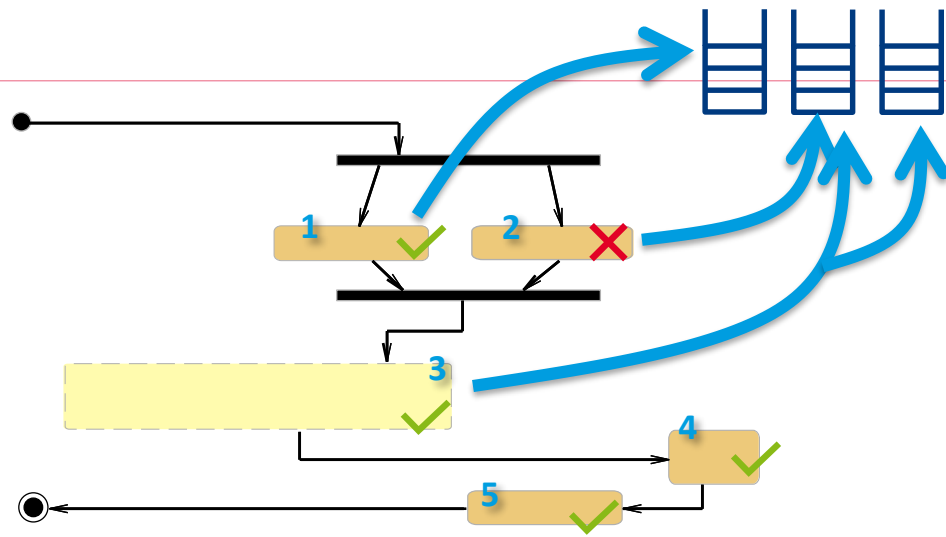
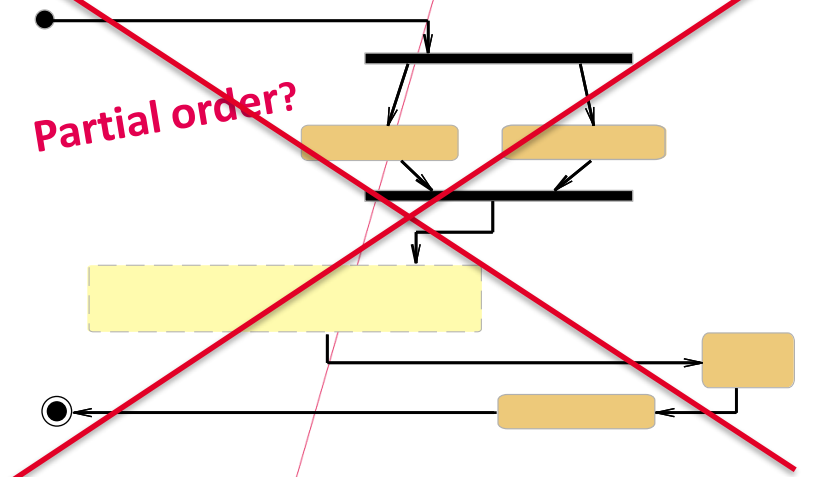
What is dynamic semantics?

Partial order?



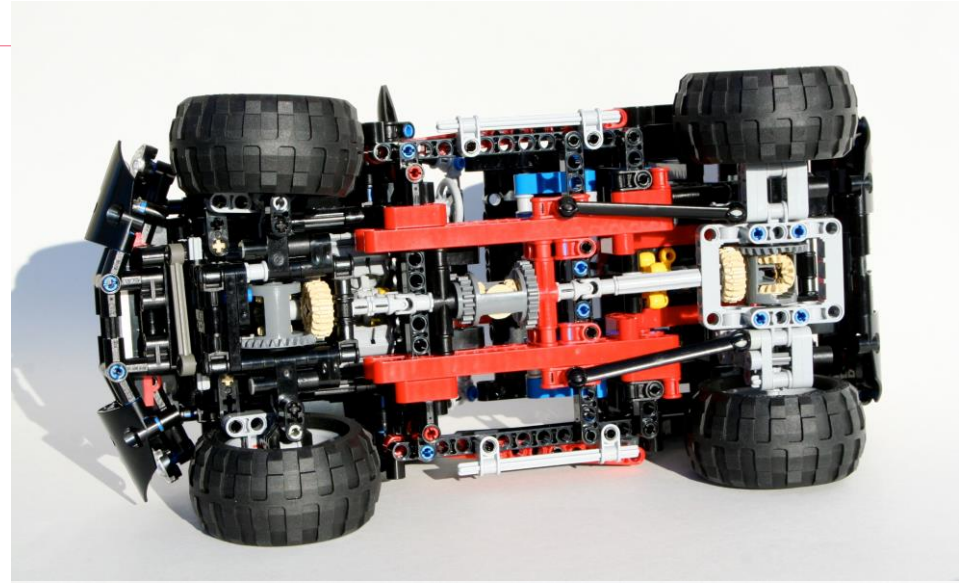
What is dynamic semantics?

Partial order?



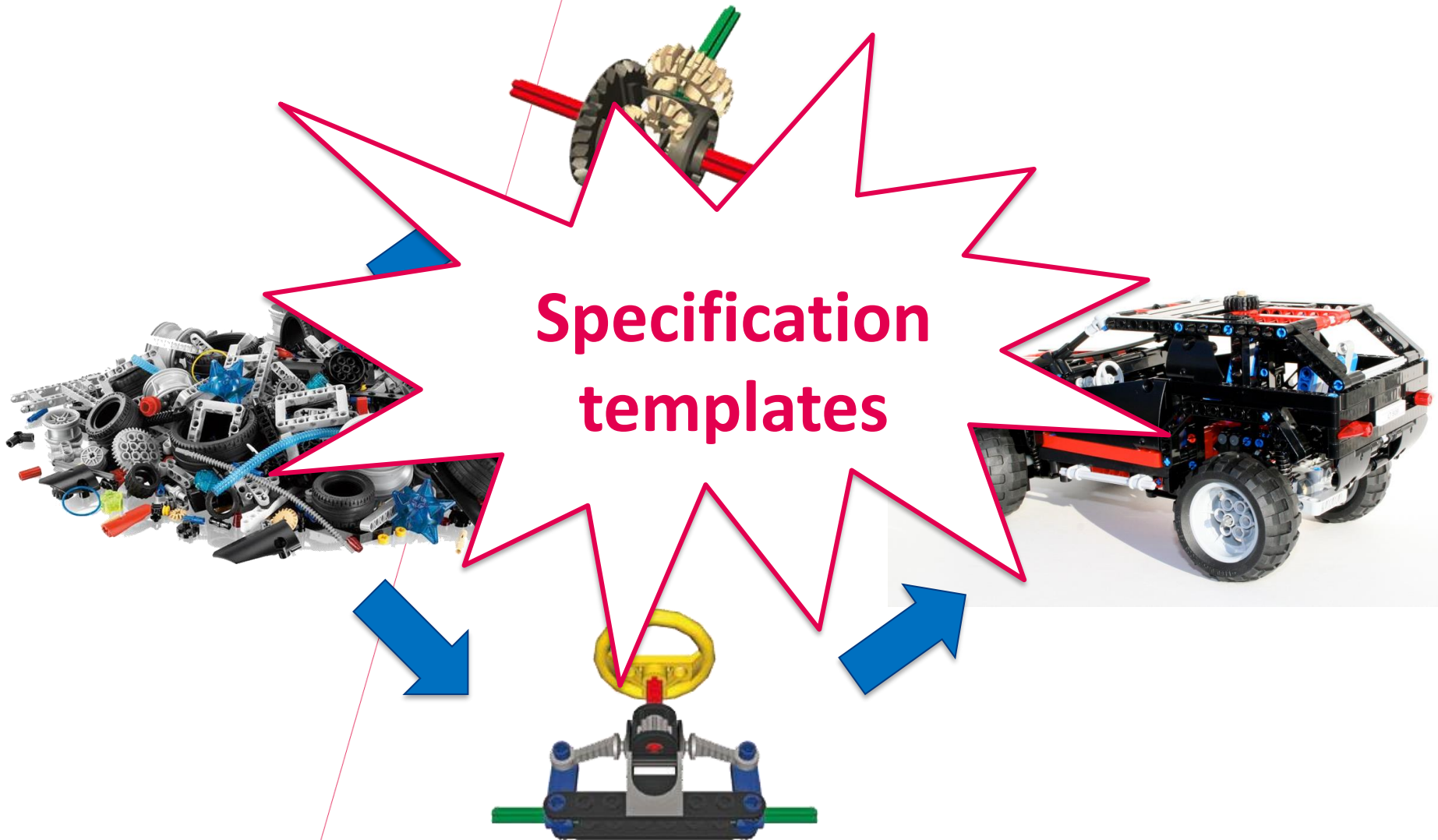
- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics
- To understand and debug DSL programs

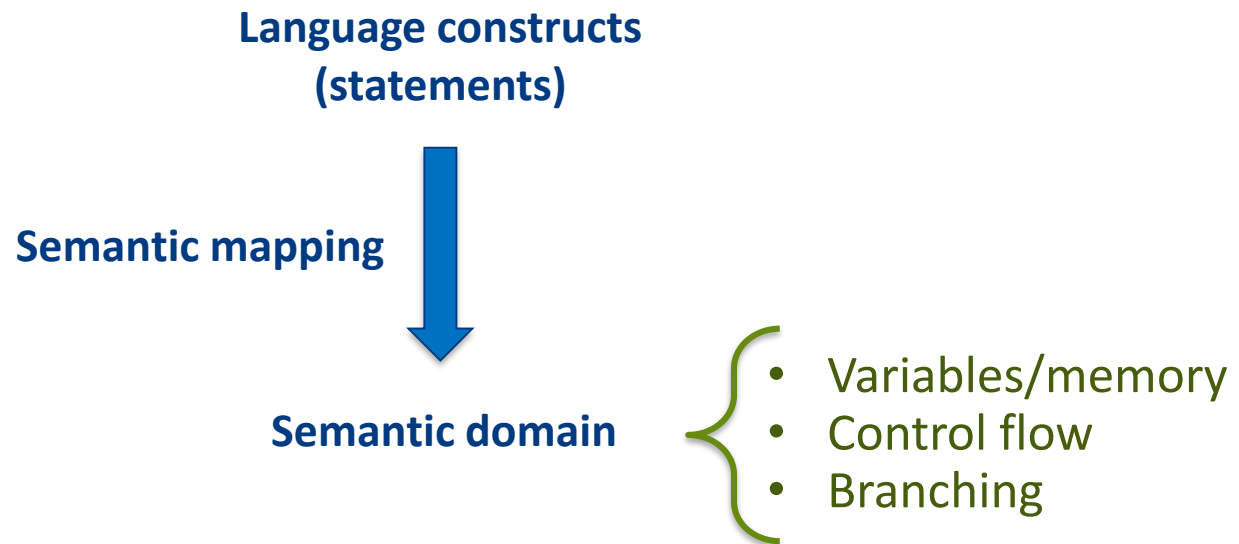
What is dynamic semantics?



- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics
- To understand and debug DSL programs

**Dynamic semantics as requirements vs.
dynamic semantics as an actual implementation**





Defining dynamic semantics of programming languages

**DSL constructs
(statements)**

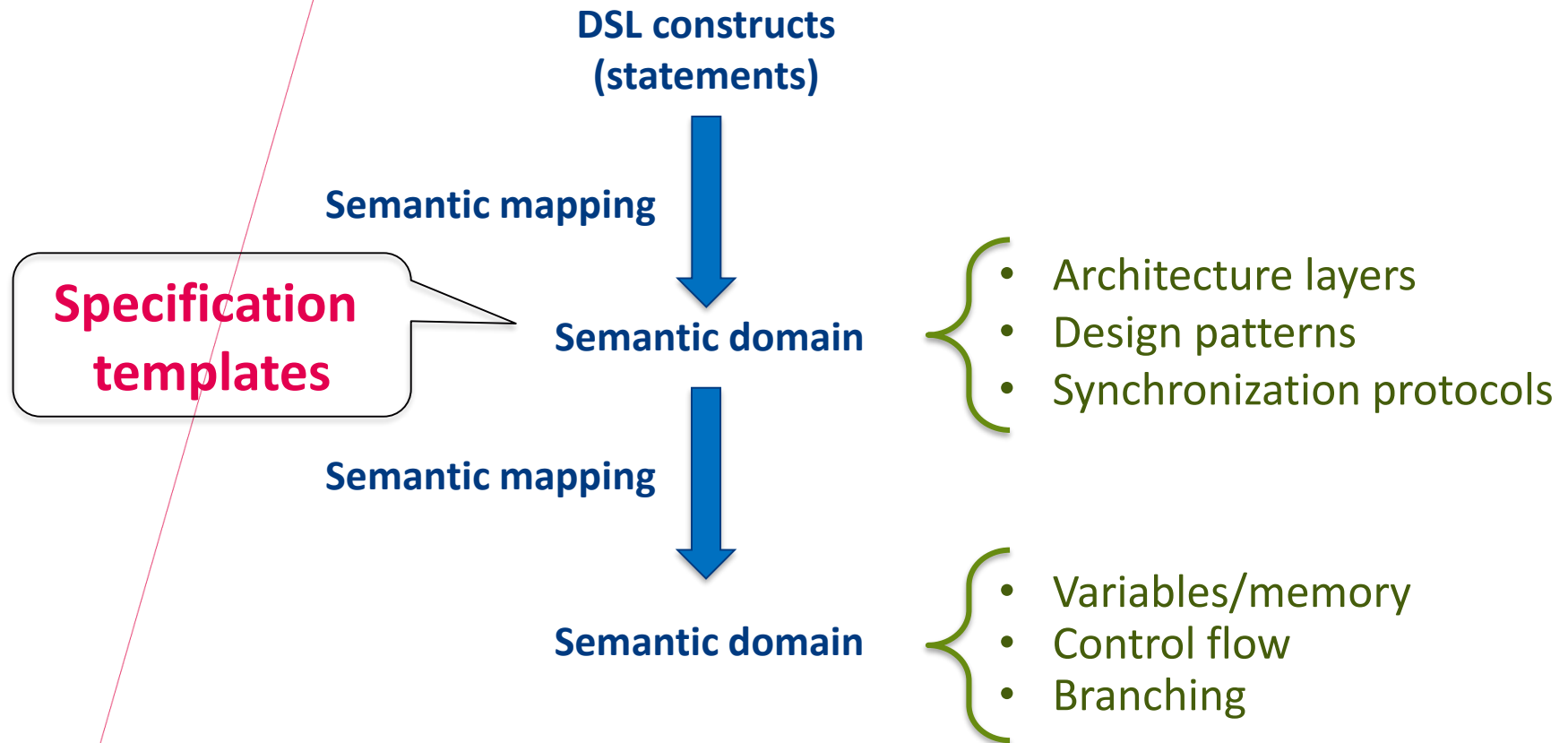


Semantic mapping

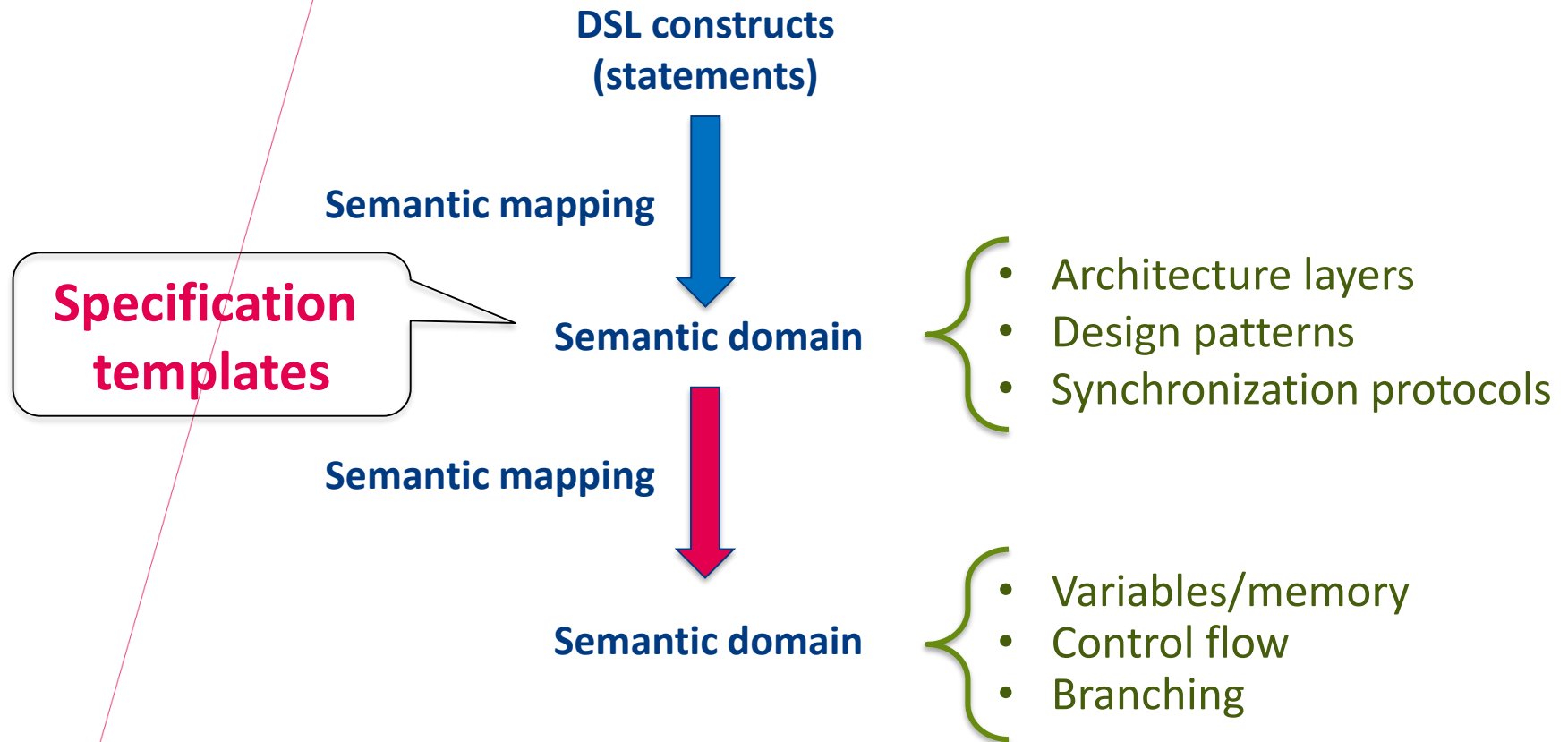
Semantic domain

- Variables/memory
- Control flow
- Branching

**Defining dynamic semantics of
domain specific languages**



Defining dynamic semantics of domain specific languages



Defining dynamic semantics of domain specific languages

MACHINE queue_machine

SEES queue_context

VARIABLES queue

INVARIANTS

inv1: queue $\in \mathbb{N} \mapsto$ **MyType**

EVENTS

INITIALISATION \triangleq

act1: queue := \emptyset

END

enqueue \triangleq

ANY element, index

WHERE

grd1: element \in **MyType**

grd2: index \mapsto element \in queue

grd3: $\forall i \cdot i \in \text{dom}(\text{queue}) \Rightarrow \text{index} \leq i$

THEN

act1: queue := queue \setminus {index \mapsto element}

END

dequeue \triangleq

ANY element, index

WHERE

grd1: element \in **MyType**

grd2: index $\in \mathbb{N}$

grd3: queue $\neq \emptyset \Rightarrow$

$(\forall i \cdot i \in \text{dom}(\text{queue}) \Rightarrow \text{index} > i)$

grd4: {index \mapsto element} $\in \mathbb{N} \mapsto$ **MyType**

grd5: index $\notin \text{dom}(\text{queue})$

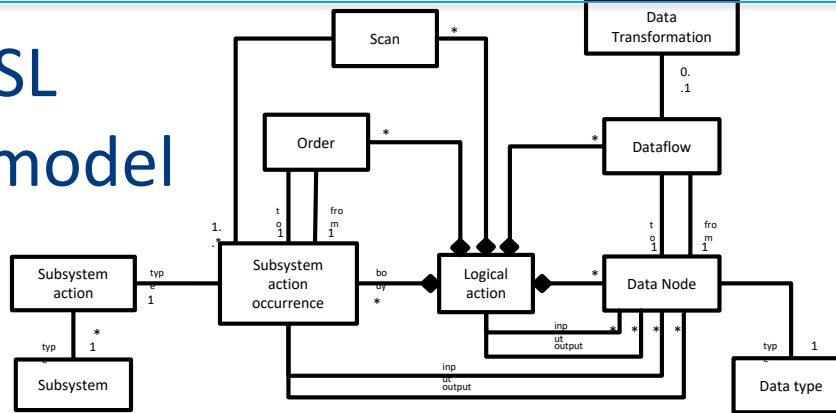
THEN

act2: queue := queue \cup {index \mapsto element}

END

END

DSL metamodel



Constelle

Structural interface

Substitution

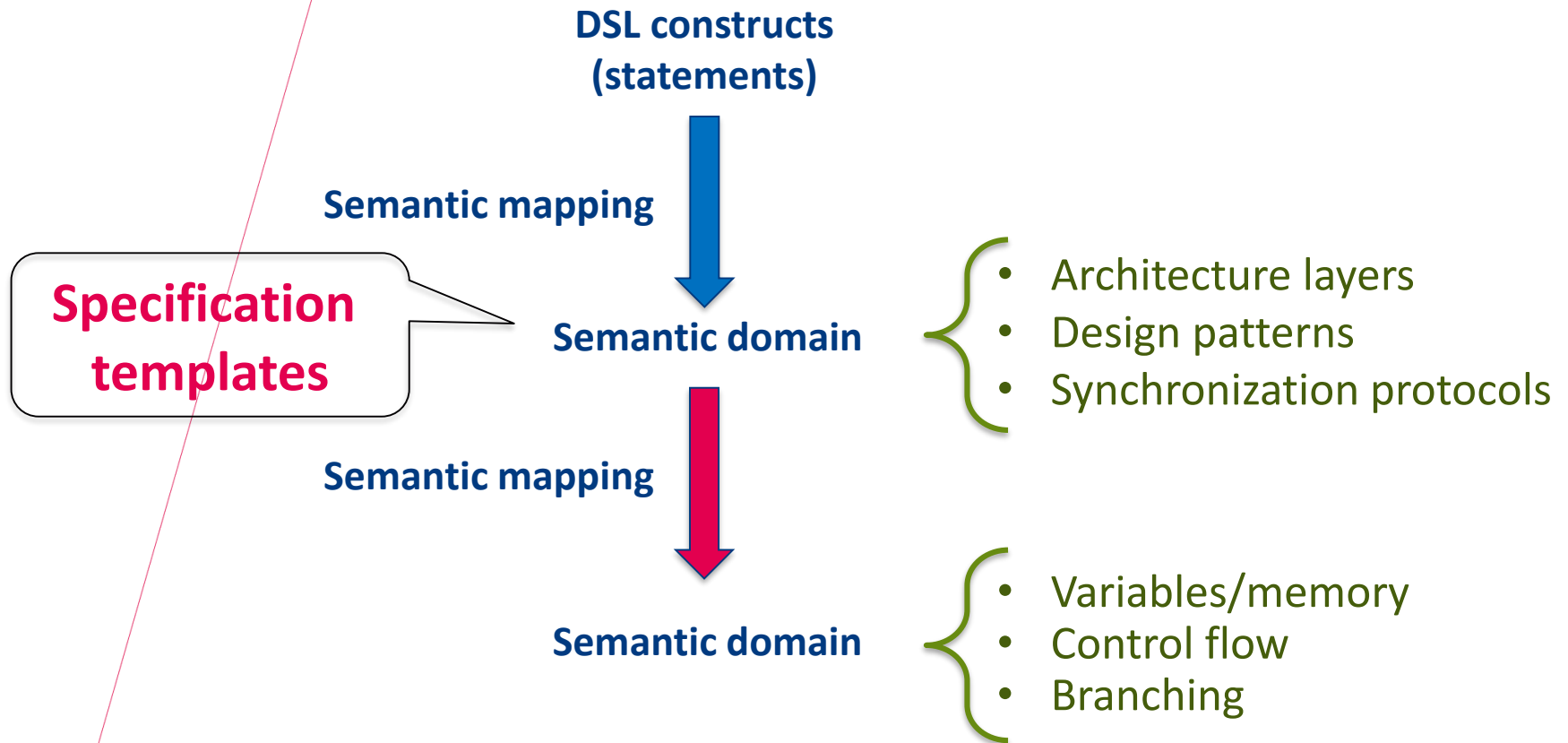
Structural interface

Event-B specification

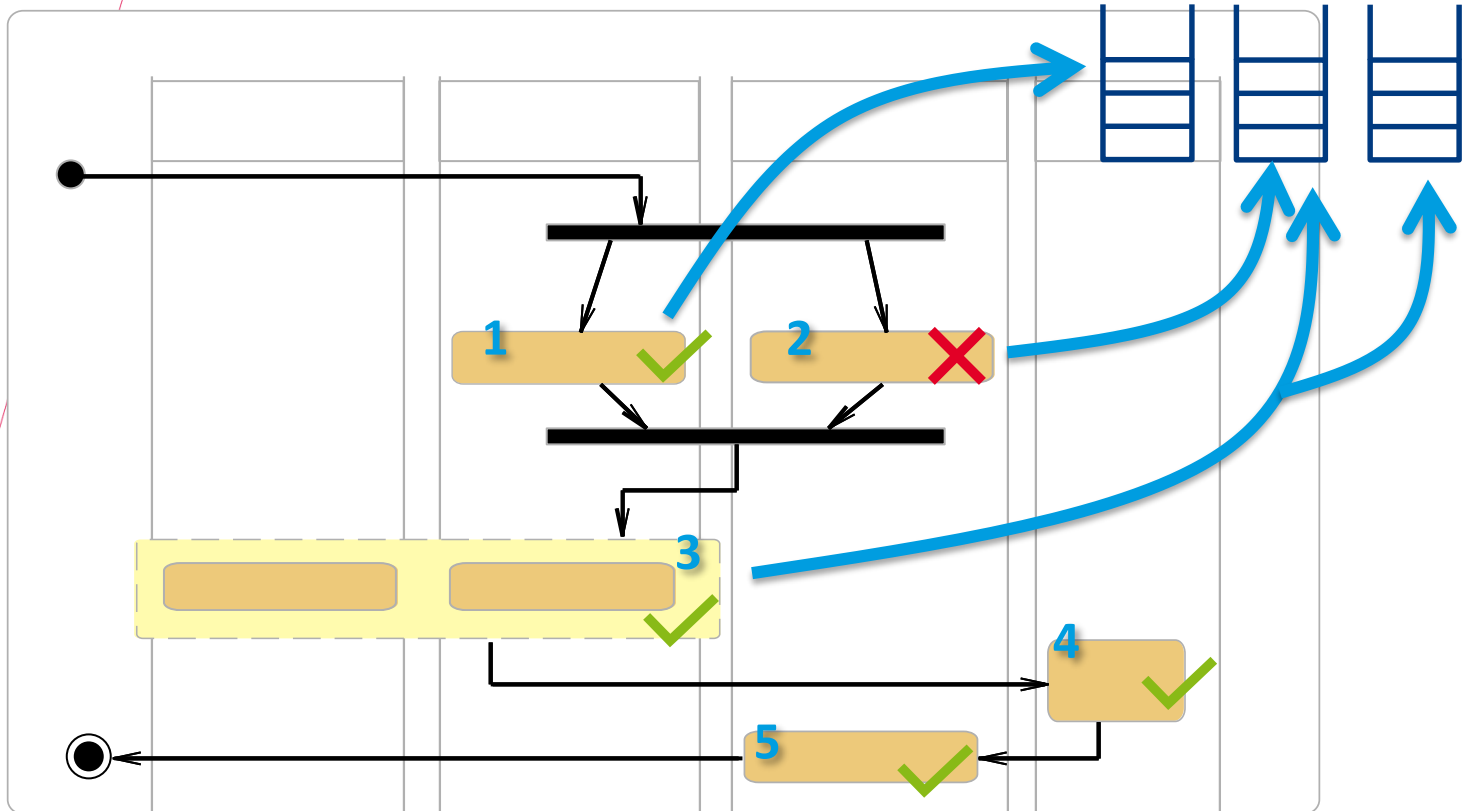
MACHINE queue_machine
SEES queue_context
VARIABLES queue
INVARIANTS
 inv1: queue ∈ ℕ → **MyType**
EVENTS
INITIALISATION ≙
 act1: queue := ∅
END

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

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}



Defining dynamic semantics of domain specific languages



Mapping DSL constructs to specification templates

VARIABLES curr_job, curr_la, la_input, ssa_output

INVARIANTS

la_input $\in \mathbb{N} \mapsto \text{LogicalActions}$

ssa_output $\in \mathbb{N} \mapsto \text{SSActions}$

curr_job $\in \mathbb{P}(\text{SSAOccurrences})$

curr_la $\in \text{LogicalActions}$

EVENTS

Initialisation

curr_la $:\in \text{LogicalActions}$

curr_job $:= \emptyset$

la_input $:= \emptyset$

ssa_output $:= \emptyset$

request la (la, n)

where

la $\in \text{LogicalActions}$

curr_job $= \emptyset$

n $\in \mathbb{N}$

la_input $\neq \emptyset \Rightarrow \forall i \cdot i \in \text{dom}(\text{la_input}) \Rightarrow n > i$

then

curr_job $:= \text{dom}(\text{LALabelDef}(\text{la}))$

curr_la $:= \text{la}$

la_input $:= \text{la_input} \cup \{ n \mapsto \text{la} \}$

request_ssa (ssaction, occurrence)

where

occurrence $\in \text{curr_job}$

occurrence $\mapsto \text{ssaction} \in \text{LALabelDef}(\text{curr_la})$

then

curr_job $:= \text{curr_job} \setminus \{\text{occurrence}\}$

execute_ssa (ssaction, n)

where

ssaction $\in \text{SSActions}$

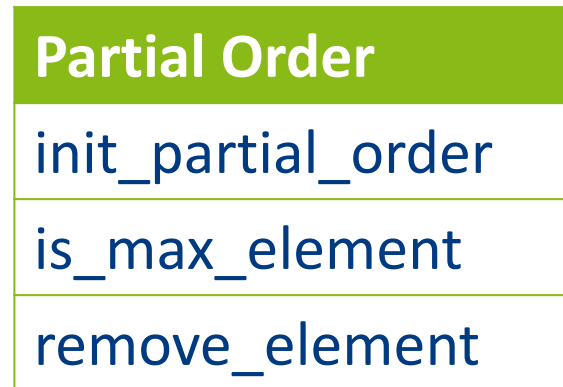
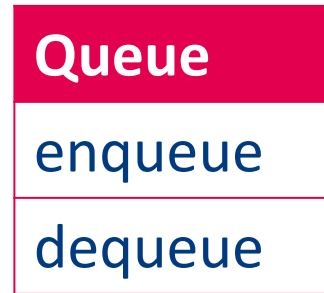
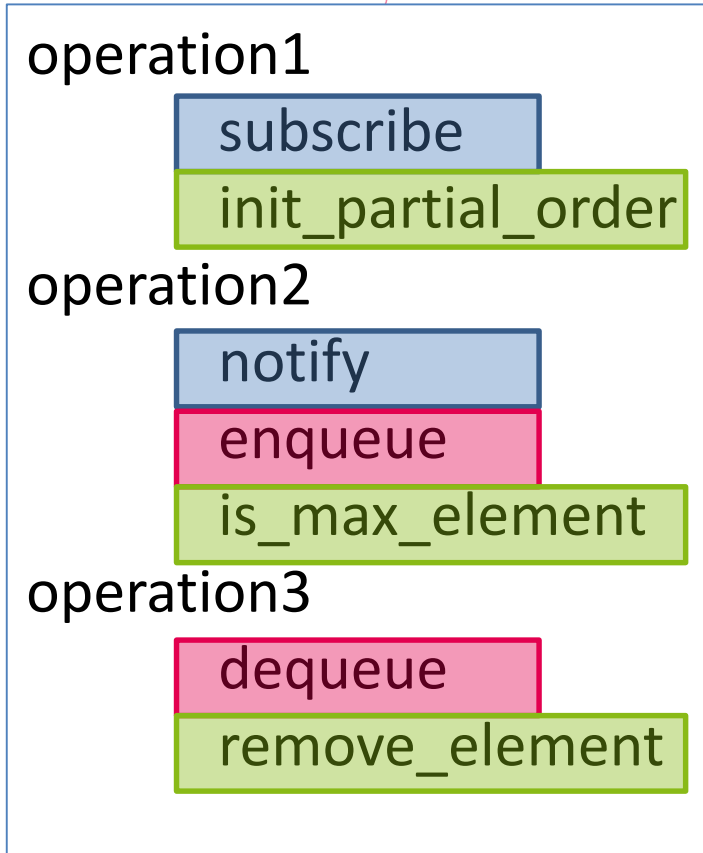
n $\in \mathbb{N}$

ssa_output $\neq \emptyset \Rightarrow \forall i \cdot i \in \text{dom}(\text{ssa_output}) \Rightarrow n > i$

then

ssa_output $:= \text{ssa_output} \cup \{ n \mapsto \text{ssaction} \}$

END



Aspect Oriented Programming: cross cutting concerns

	Listener	Queue	Partial Order
operation1	subscribe		init_partial_order
operation2	notify	enqueue	is_max_element
operation3		dequeue	remove_element

**Specializations of specification templates
from the library**

**Aspect Oriented Programming:
composing cross cutting concerns**

Aspects = specification templates

Operations of the DSL dynamic semantics

The screenshot shows an IDE with three files open: `roboticarm.ecore`, `library_interfaces.constellecore`, and `Robotic Arm Parallel`. The `library_interfaces.constellecore` file contains the following code:

```

driver1 : template_queue
driver2 : template_queue
distributor : template_request

request
elements
process
element

enqueue
element

dequeue
element

enqueue
element

dequeue
element

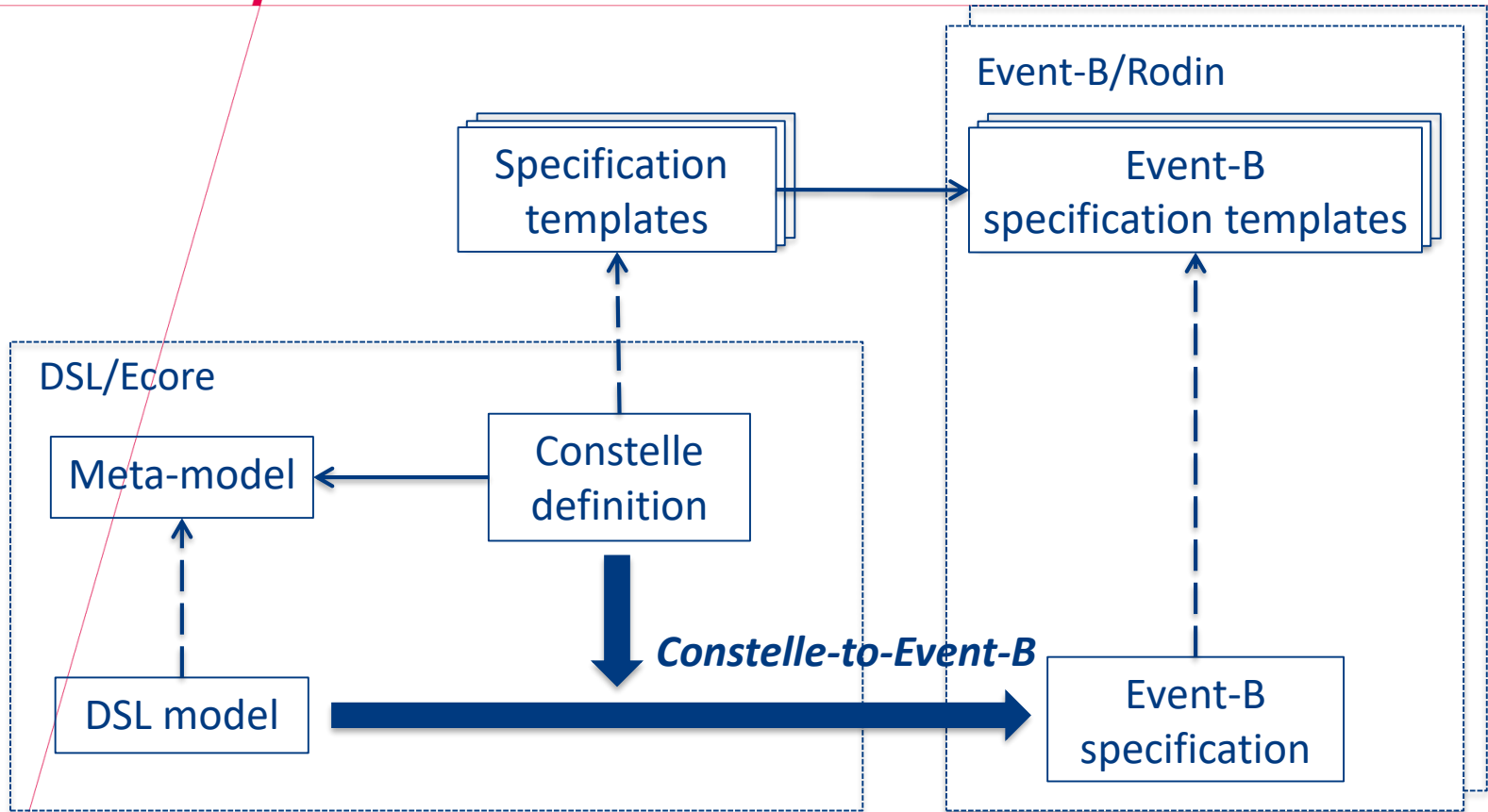
```

Below the code is a table showing the substitution of static parameters with DSL types:

Type HandActions		ElementType	
Type ArmActions	ElementType		
Type Actions			ElementType

Substitution of template (static) parameters with DSL types

Constelle



Constelle

- Scalability and applicability of the proposed method
 - Validated on three different DSLs
- A library of specification templates
 - State machine, communication channels, data typing, design patterns (Listener), data structures (Queue)
- Specification templates for bridging technological diversity
 - Source code, visualization templates, formal notations

Future work

TU/e Technische Universiteit Eindhoven University of Technology

DSL developer

- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics

DSL user

- To understand and debug DSL programs

Precise and executable definitions

Why to define dynamic semantics?

TU/e Technische Universiteit Eindhoven University of Technology

Partial order?

- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics
- To understand and debug DSL programs

What is dynamic semantics?

TU/e Technische Universiteit Eindhoven University of Technology

DSL constructs (statements)

Semantic mapping

Specification templates

Semantic domain

- Architecture layers
- Design patterns
- Synchronization protocols

Semantic mapping

Semantic domain

- Variables/memory
- Control flow
- Branching

Defining dynamic semantics of domain specific languages

DSL metamodel

Constelle

Structural interface

Substitution

Structural interface

```

MACHINE queue_machine
SEES queue_context
VARIABLES queue
INVARIANTS
  inv1: queue ∈ N
EVENTS
INITIALISATION
  act1: queue := 0
END
  
```

enqueue: Δ
ANY element, index
WHERE
grd1: element ∈ queue
grd2: index ⇒ element ∈ queue
grd3: Yi ∈ dom(queue) ⇒ index ≤ i
THEN
act1: queue := queue \ (index ⇒ element)
END

TU/e Technische Universiteit Eindhoven University of Technology

	Listener	Queue	Partial Order
operation1	subscribe		init_partial_order
operation2	notify	enqueue	is_max_element
operation3		dequeue	remove_element

Specializations of specification templates from the library

TU/e Technische Universiteit Eindhoven University of Technology

Aspects = specification templates

roboticarm.ecore

```

library_interfaces.constelle
  driver1: template_queue
  driver2: template_queue
  distributor: template_request
  request
  elements
  process
  element
  process
  element
  dequeue
  element
  enqueue
  element
  dequeue
  element
  
```

Operations of the DSL dynamic semantics

Substitution of template (static) parameters with DSL types

Constelle