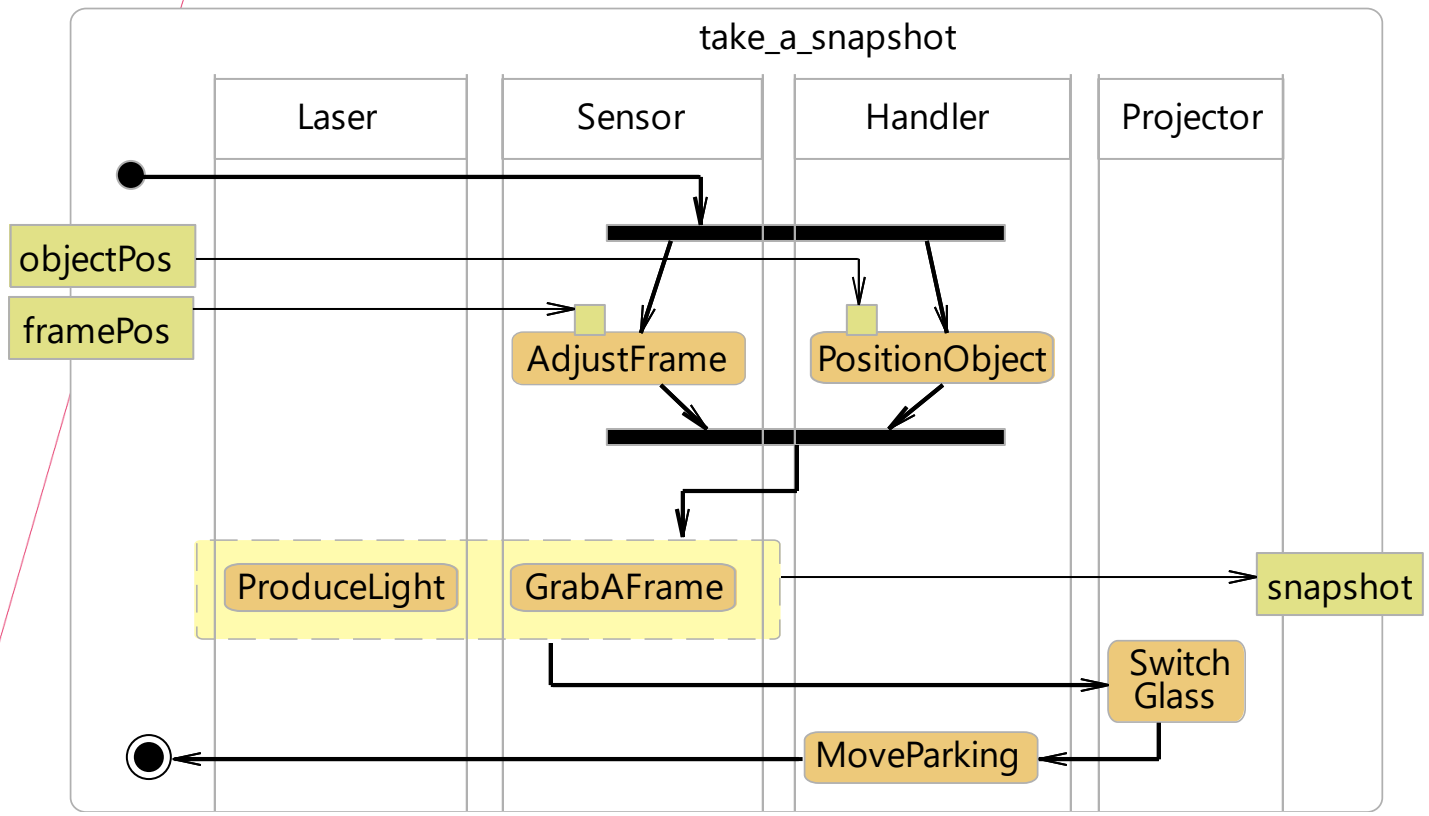
Technische Universiteit
Eindhoven
University of Technology

Reusable specification templates for defining dynamic semantics of DSLs

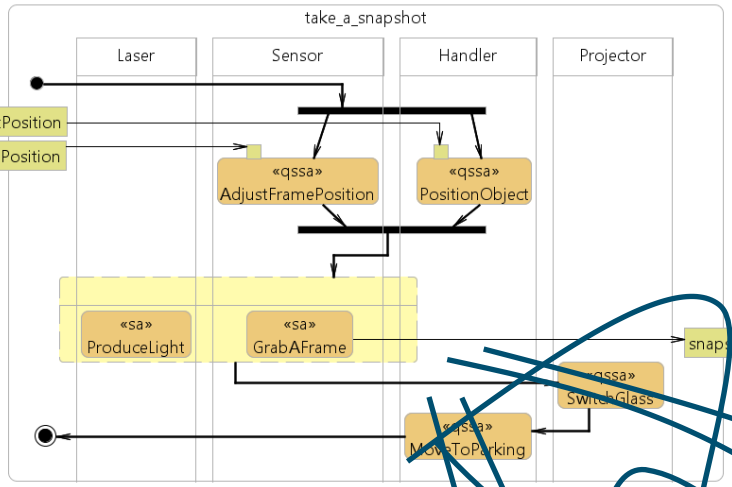
Ulyana Tikhonova
ulyana.tikhonova@constelle.net

Where innovation starts

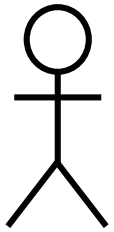
Tim Willemse Mark van den Brand



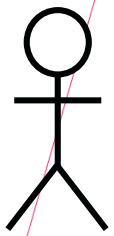
Domain Specific Languages (DSLs)



Code generation



DSL developer

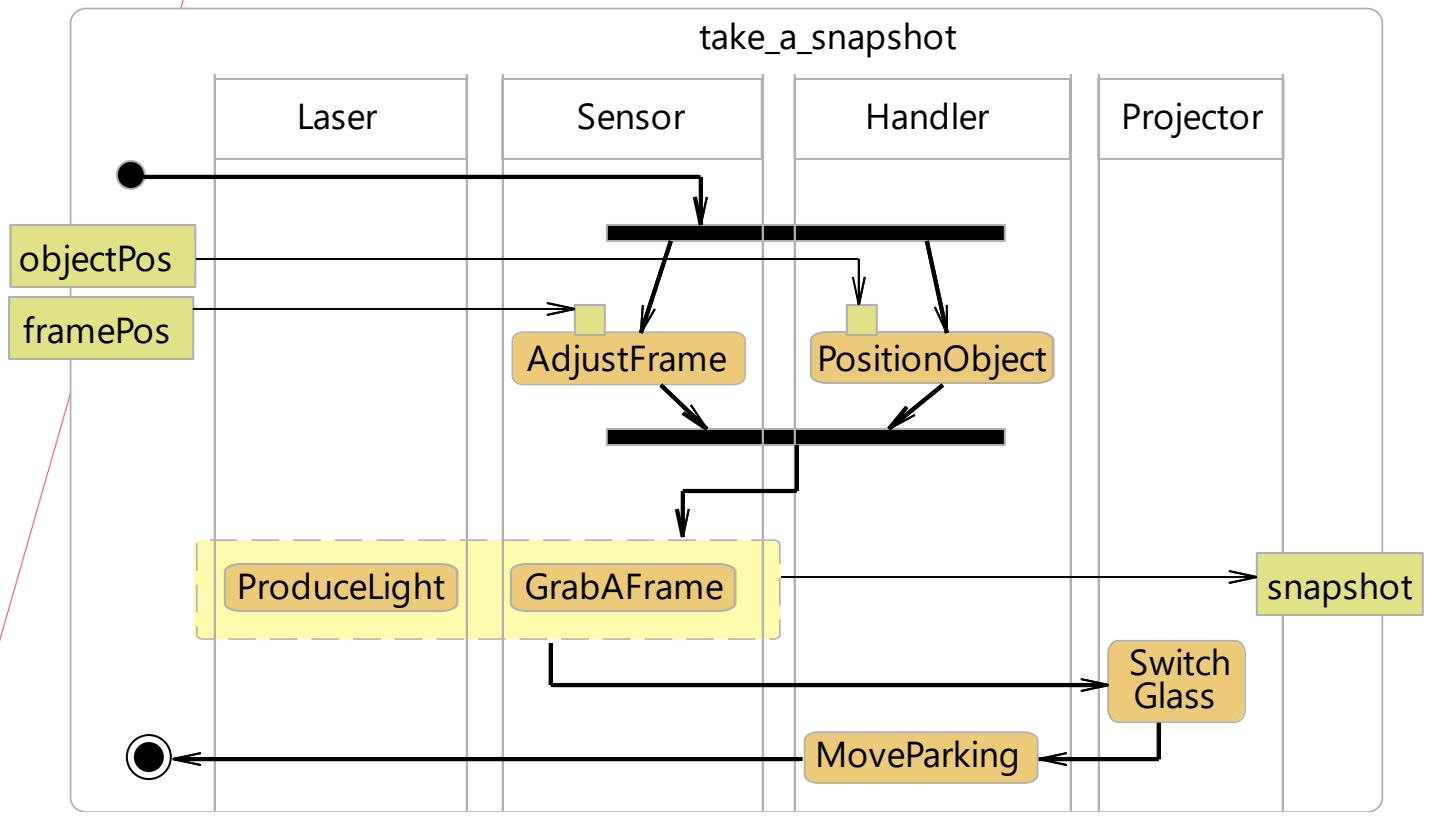


DSL user

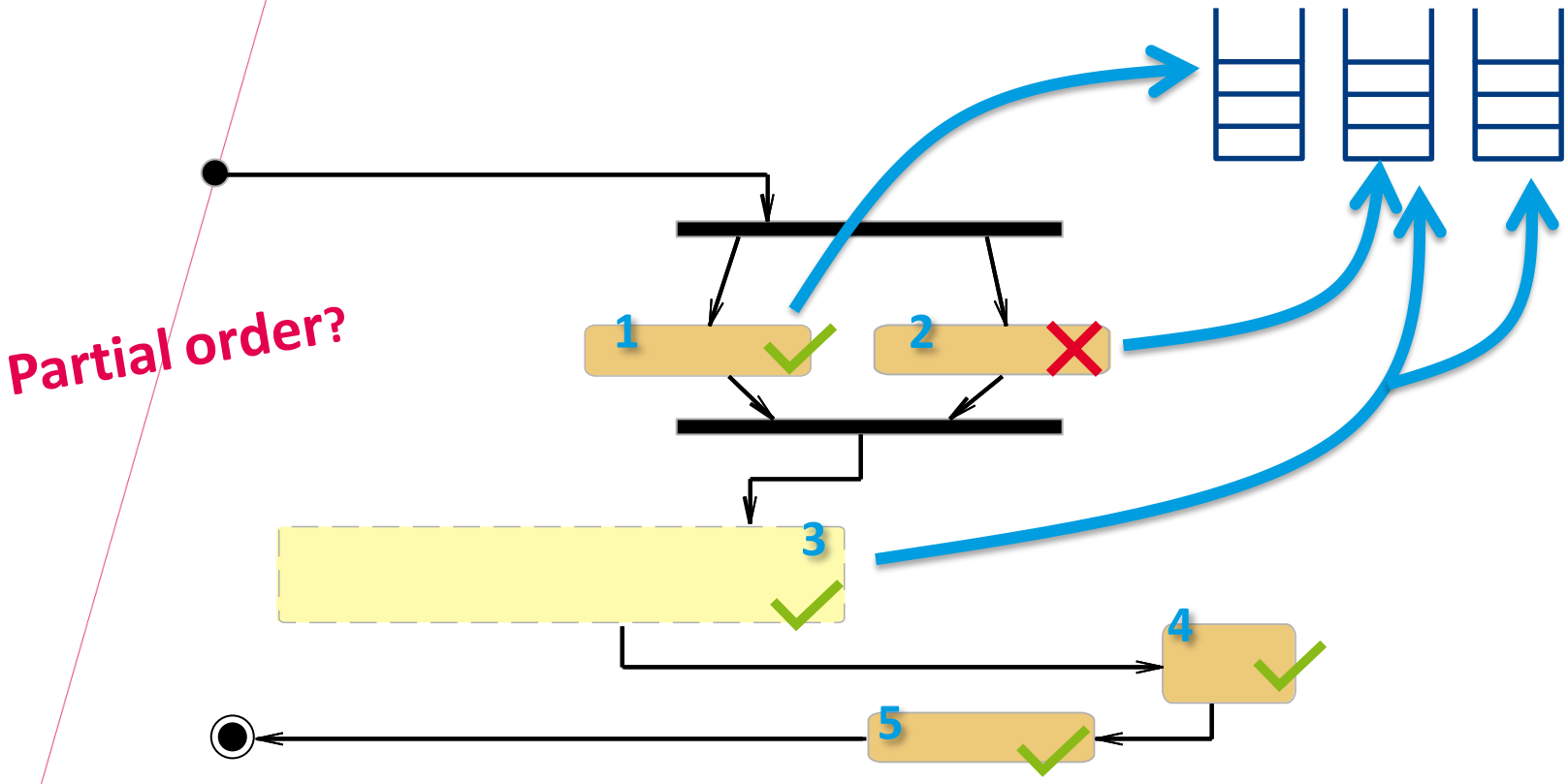
- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics
- To understand and debug DSL programs

Precise and executable definition

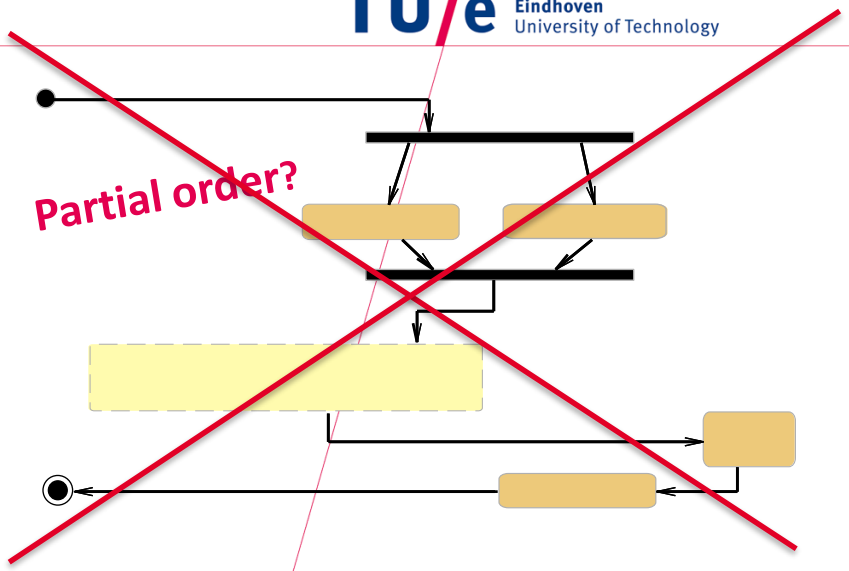
Why to define dynamic semantics?

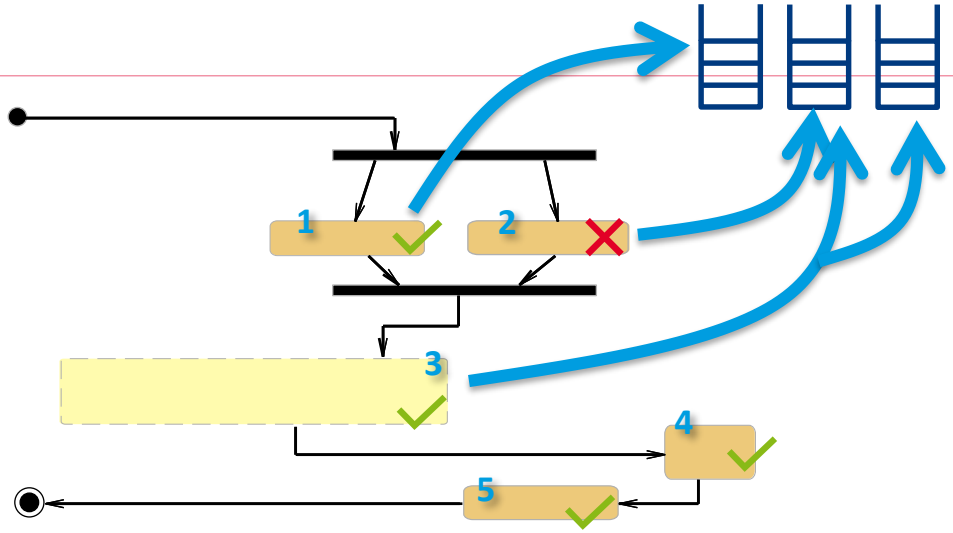


What is dynamic semantics?



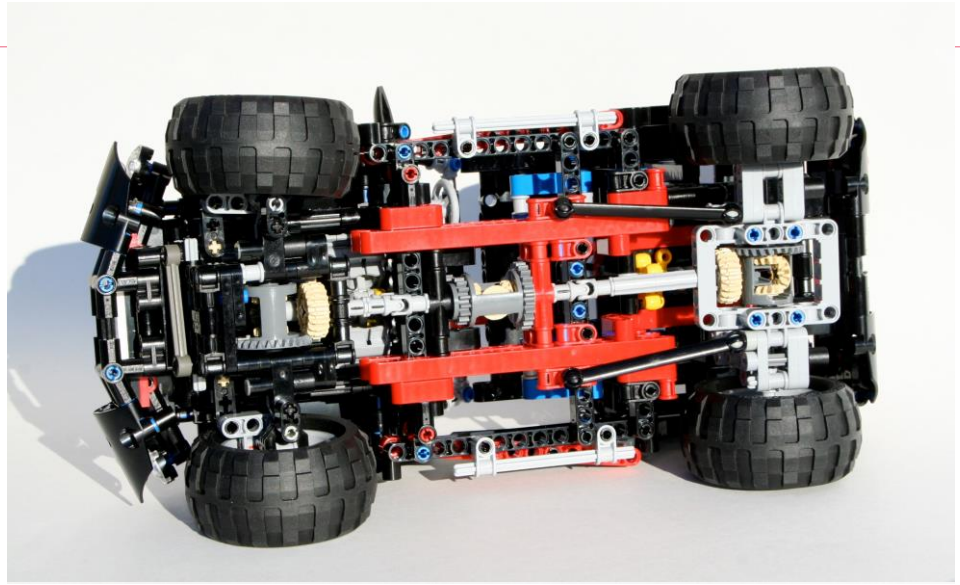
What is dynamic semantics?





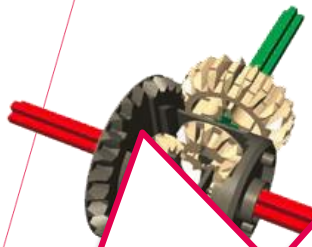
- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics
- To understand and debug DSL programs

What is dynamic semantics?

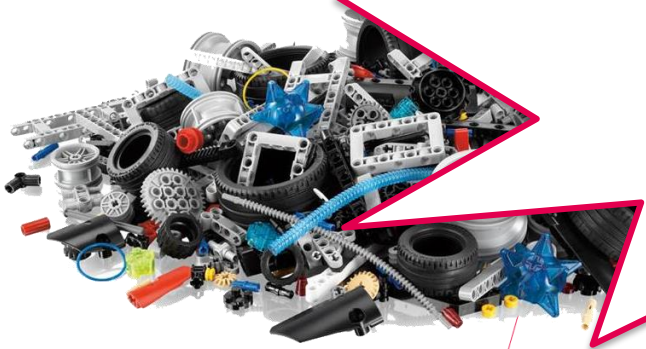


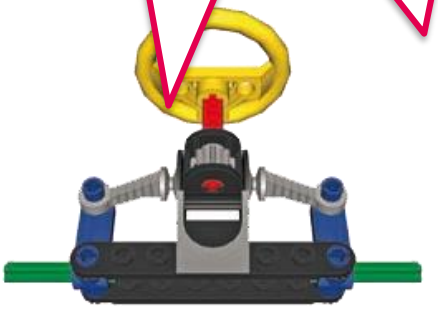
- To understand the dynamic semantics
- To analyze the dynamic semantics
- To validate the dynamic semantics
- To understand and debug DSL programs

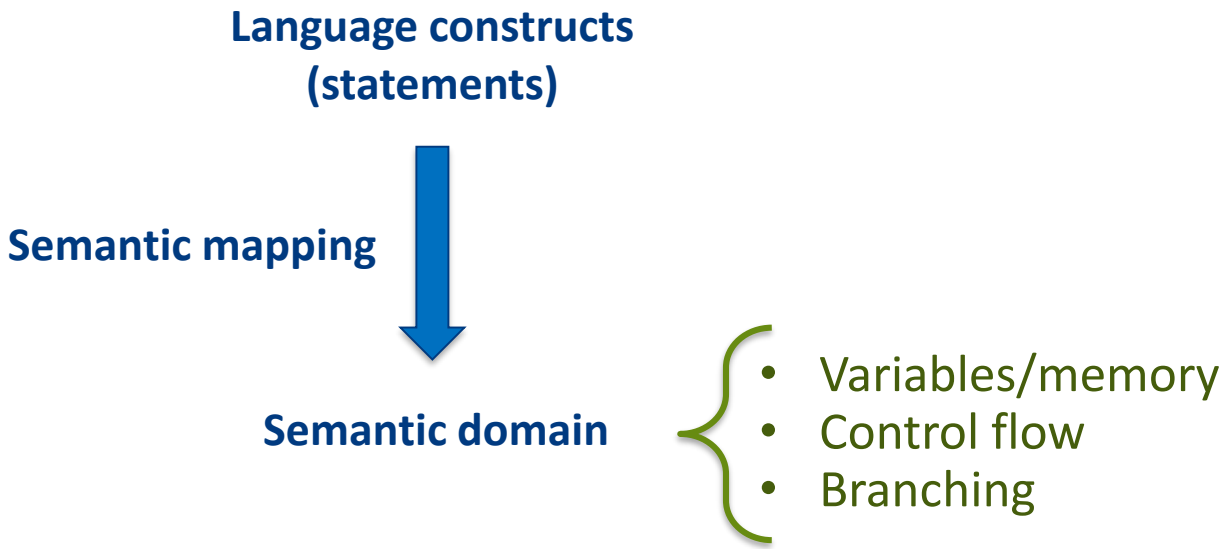
Dynamic semantics as requirements vs. dynamic semantics as an actual implementation



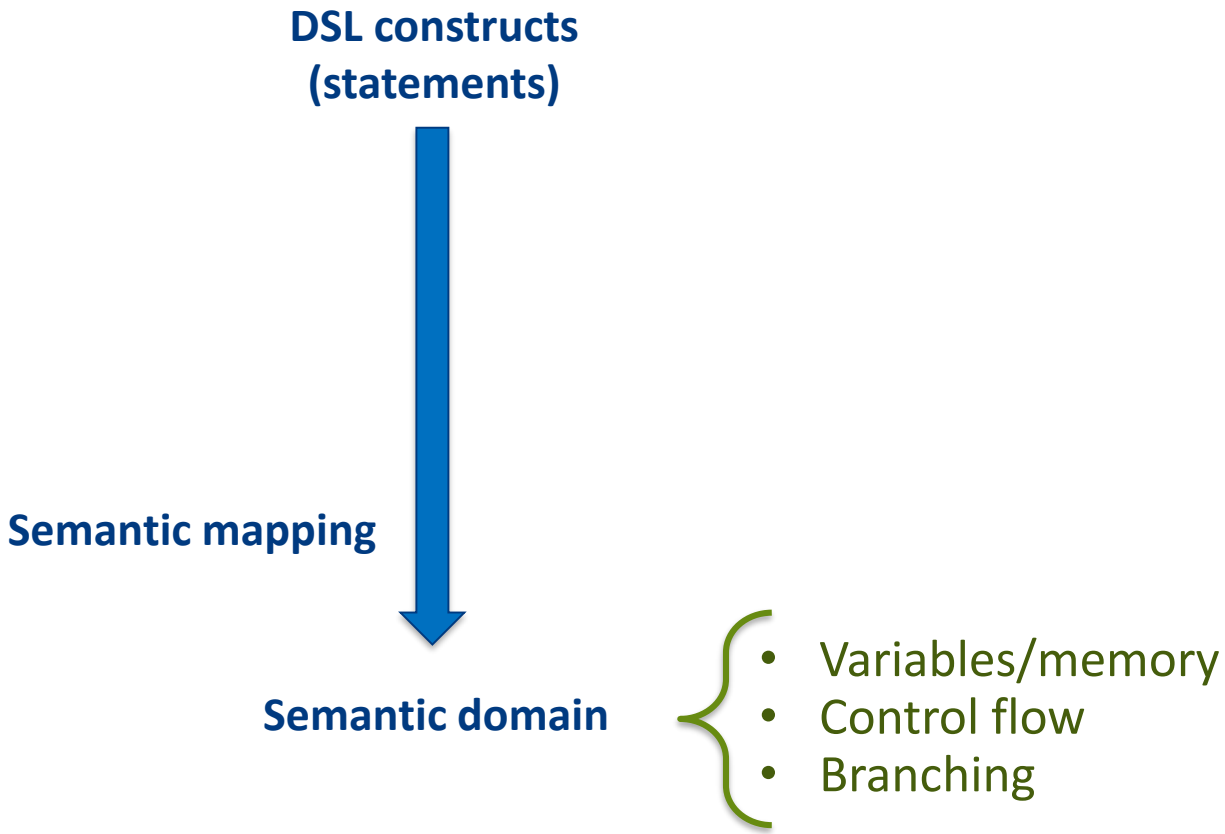
Specification templates



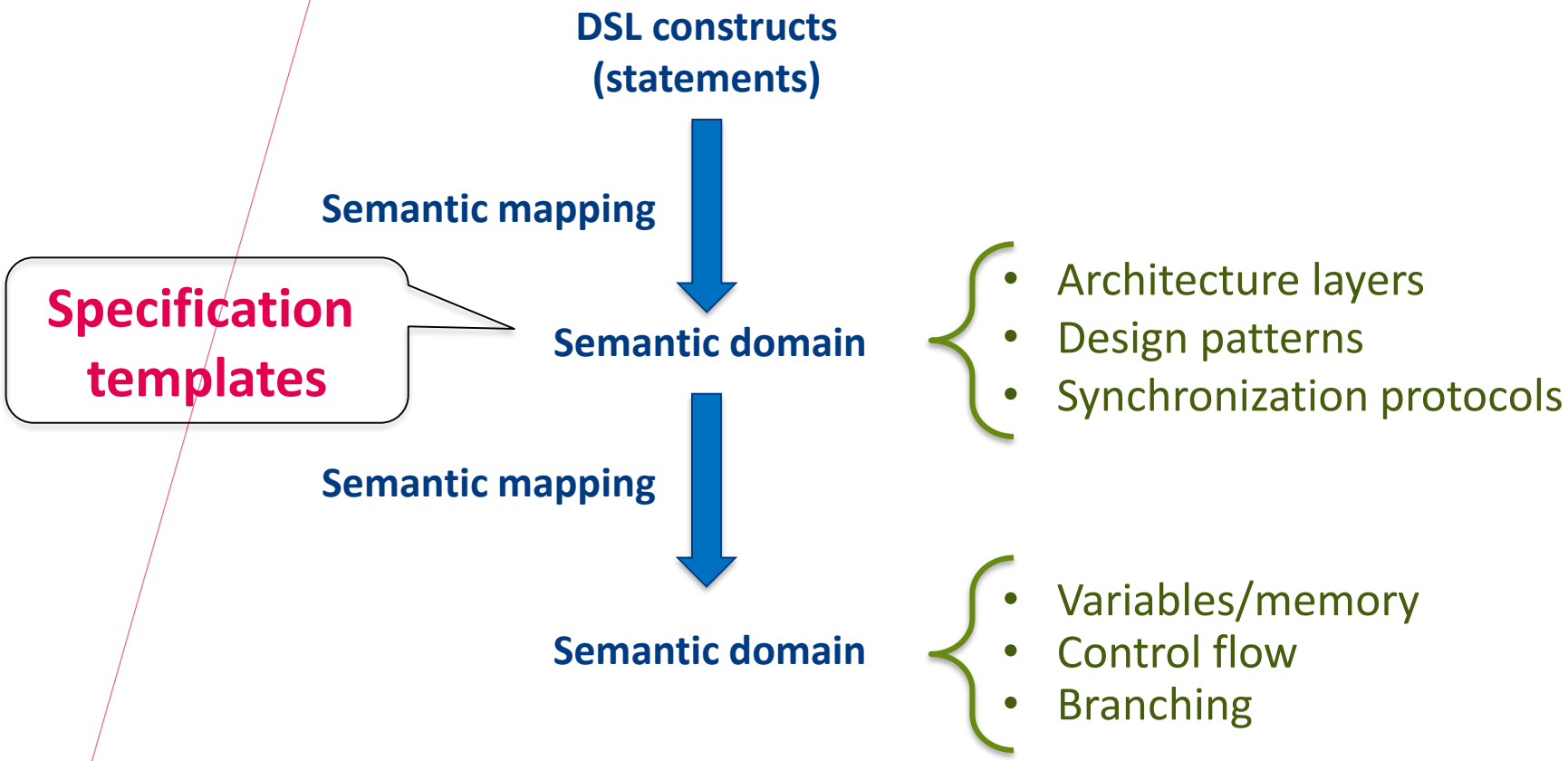




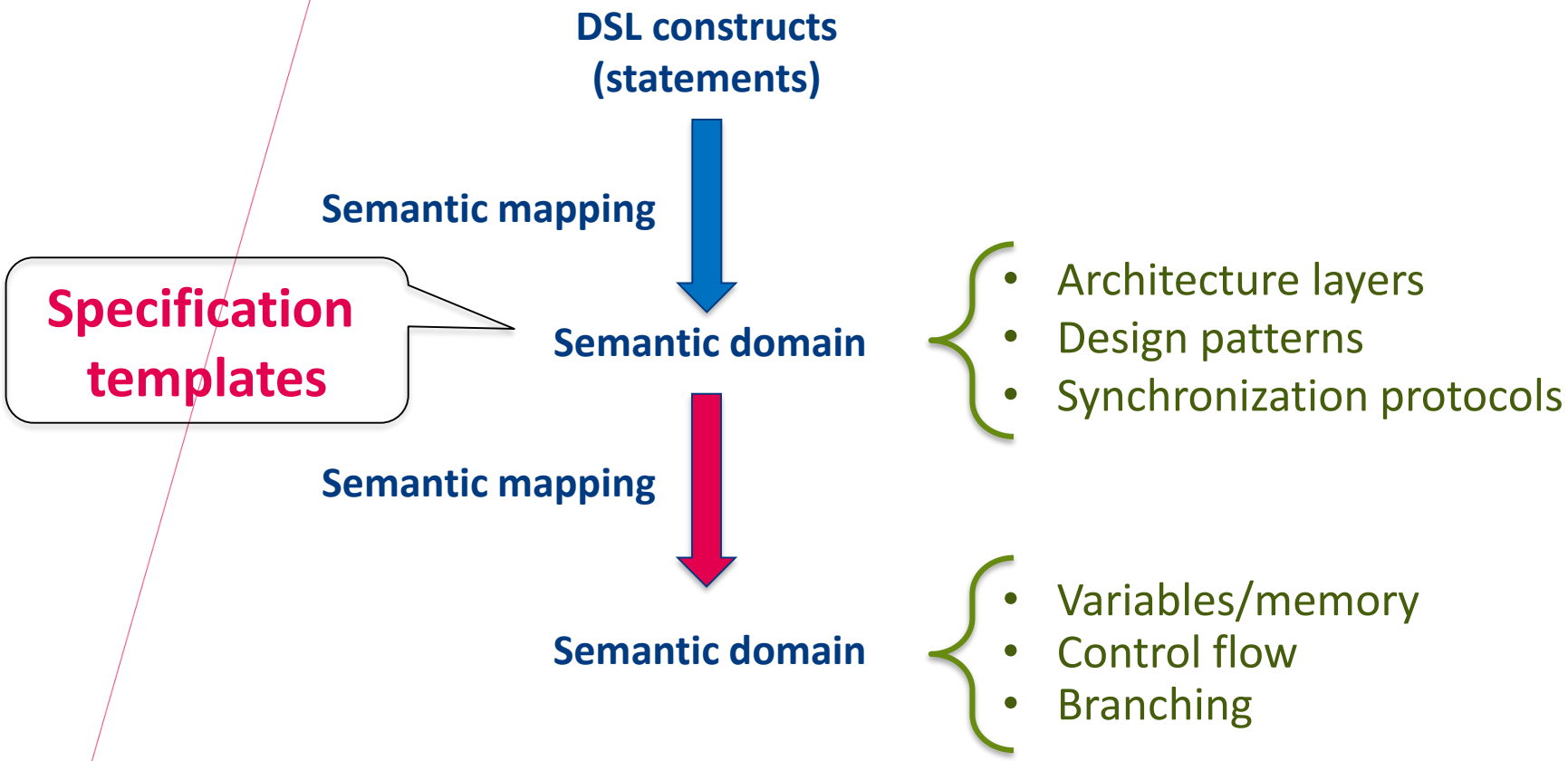
Defining dynamic semantics of programming languages



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 ∈ 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)

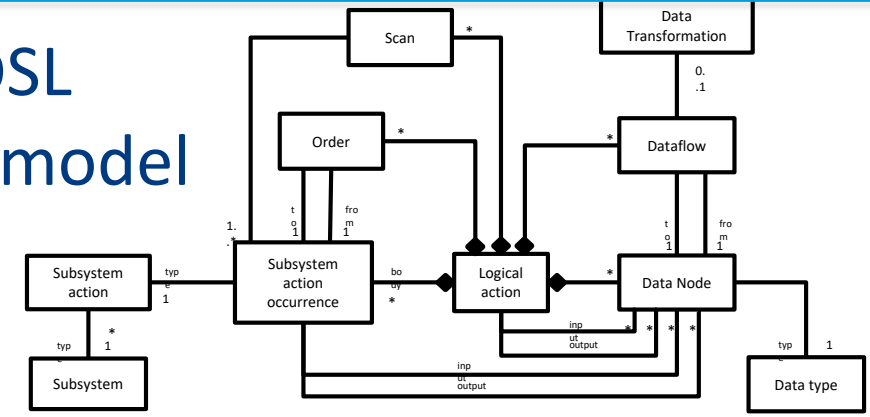
THEN

act2: queue ≔ queue ∪ {index ↦ element}

END

END

DSL metamodel



Constelle

Structural interface

Substitution

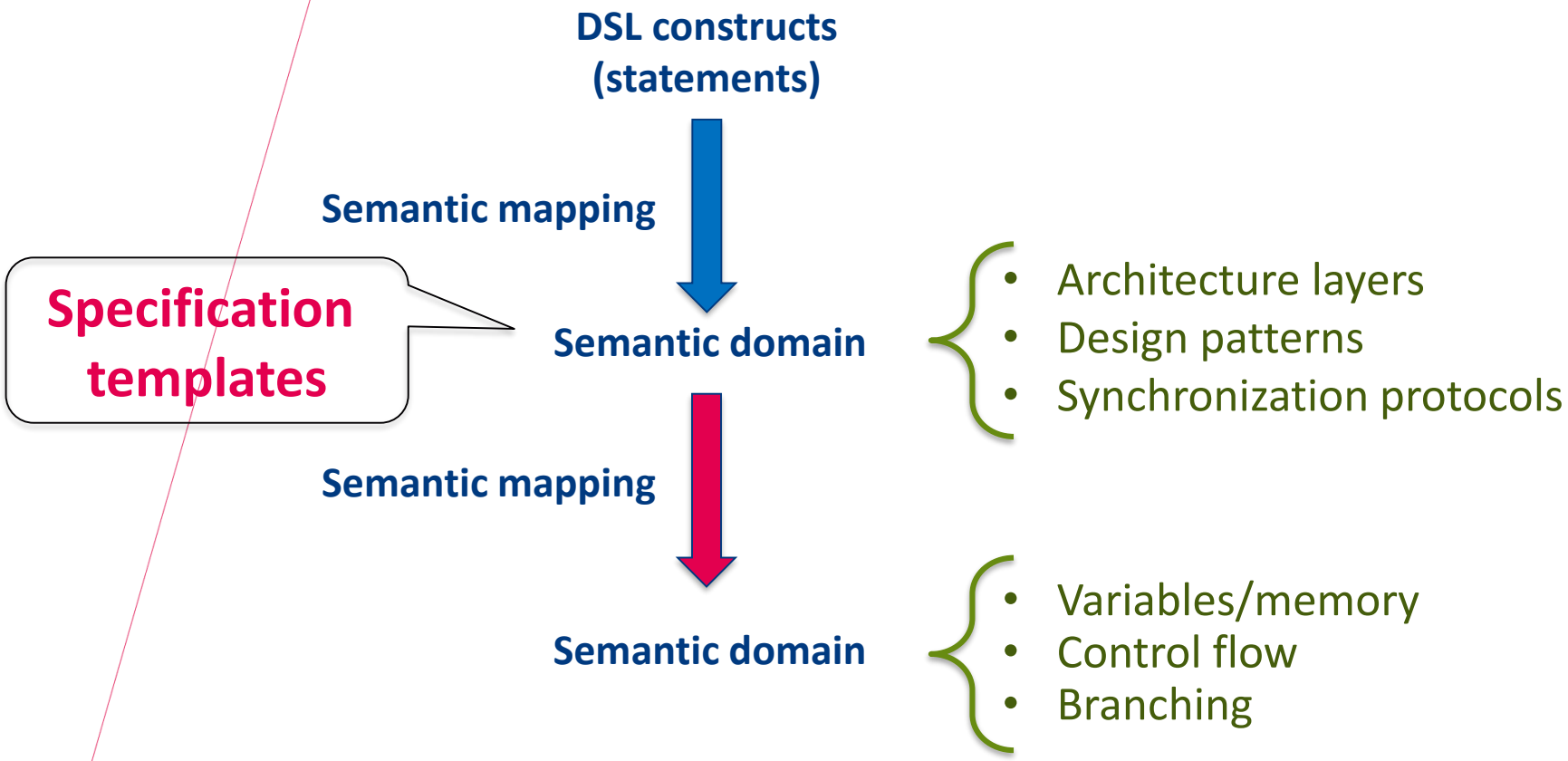
Structural interface

Event-B specification

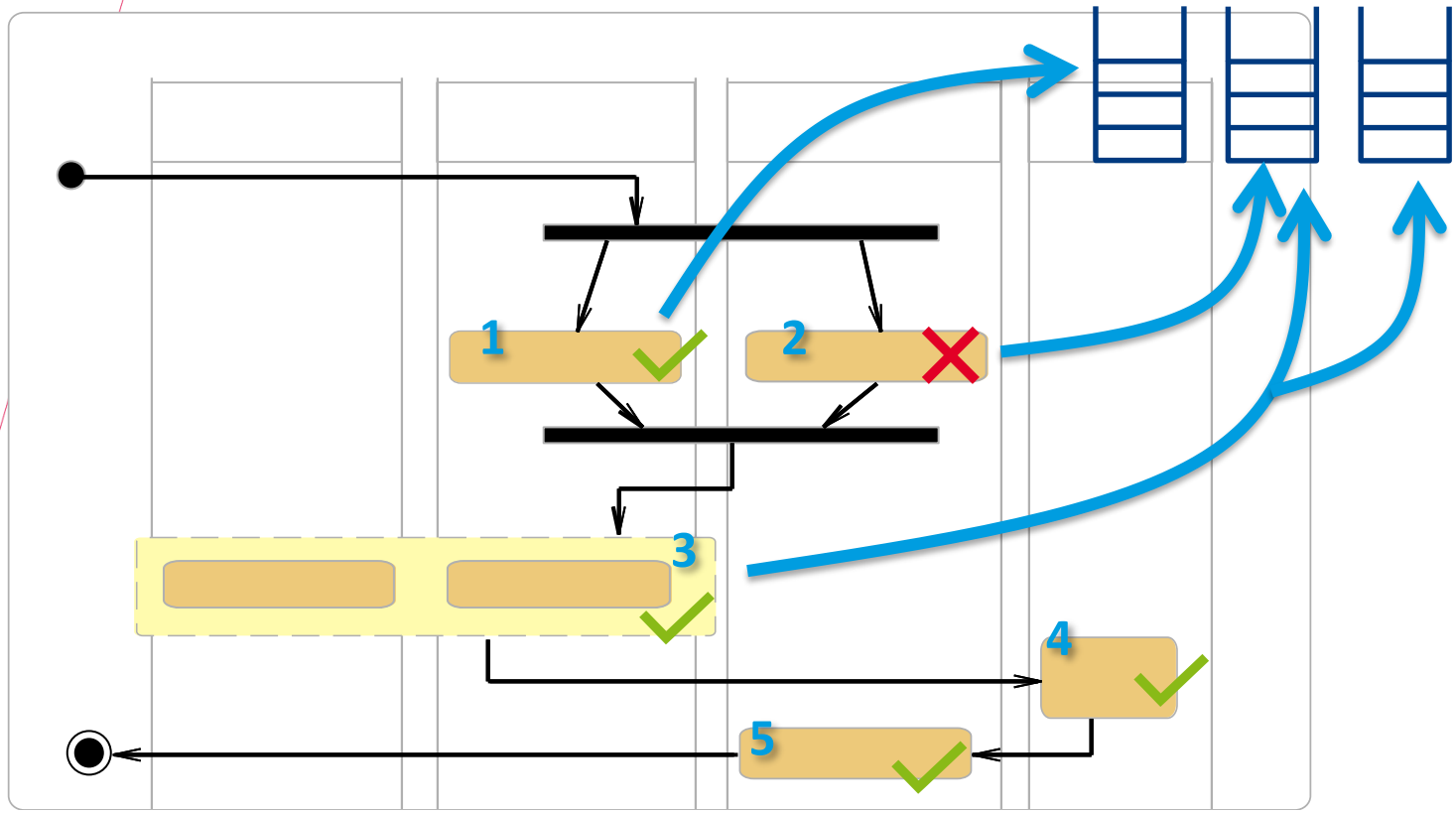
MACHINE queue_machine
SEES queue_context
VARIABLES queue
INVARIANTS
inv1: queue ∈ ℕ ⇸ MyType
EVENTS
INITIALISATION ≙
act1: queue ≔ ∅
END

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 ∈ 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

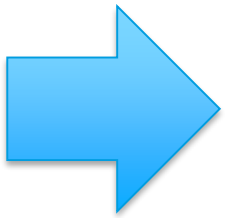
ssa_output ≔ ssa_output ∪ { n ↦ ssaction }

END

operation1
- subscribe
- init_partial_order

operation2
- notify
- enqueue
- is_max_element

operation3
- dequeue
- remove_element



Queue
enqueue
dequeue

Listener
subscribe
notify

Partial Order
init_partial_order
is_max_element
remove_element

Aspect Oriented Programming: cross cutting concerns

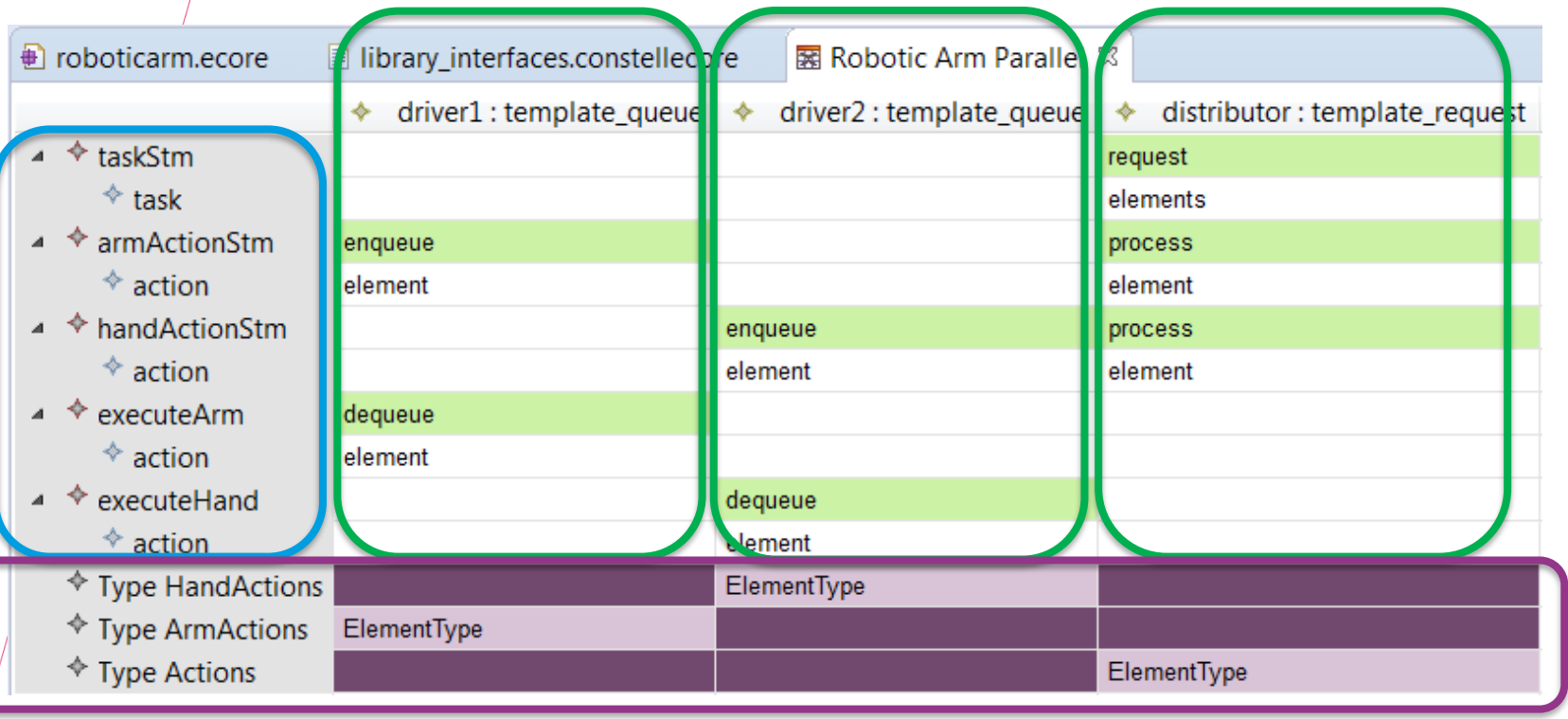
	Listener	Queue	Partial Order
operation1	subscribe		init_partial_order
operation2	notify	enqueue	is_max_element
operation3		dequeue	remove_element

Specializations of specification templates from the library

Aspect Oriented Programming: composing cross cutting concerns

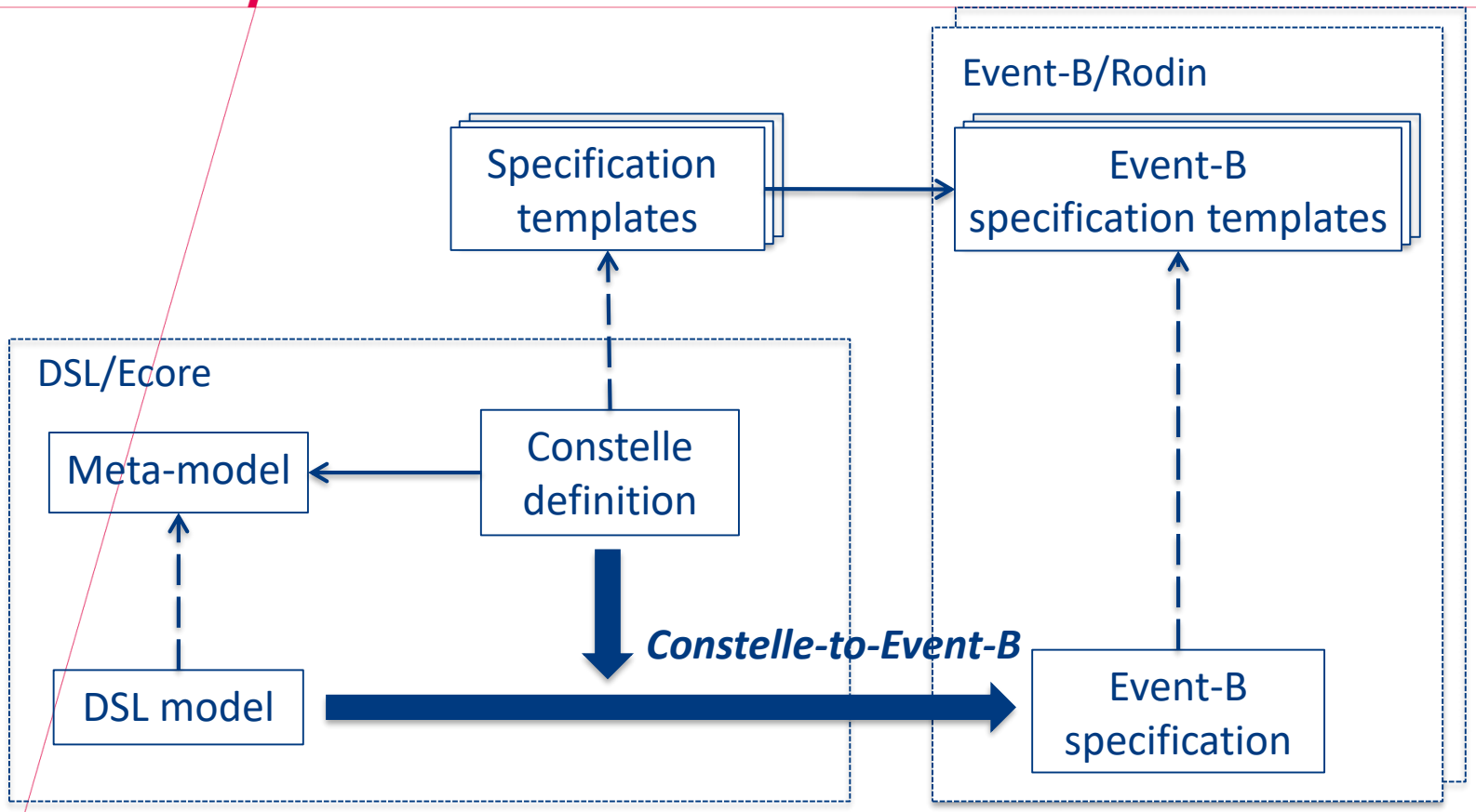
Aspects = specification templates

Operations of the DSL dynamic semantics



Substitution of template (static) parameters with DSL types

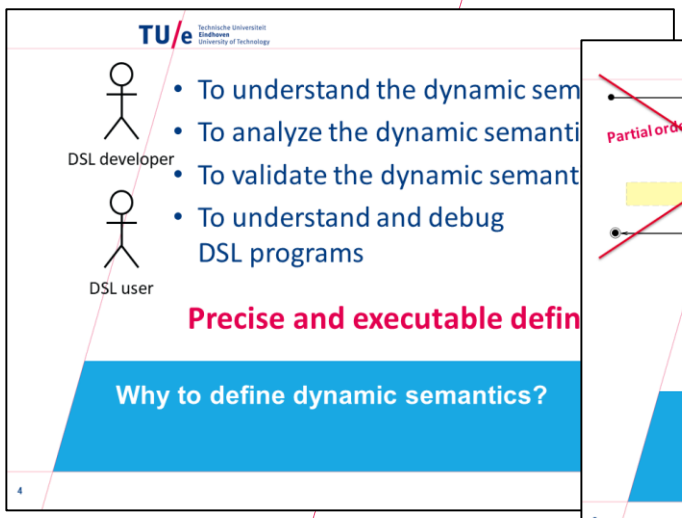
Constelle

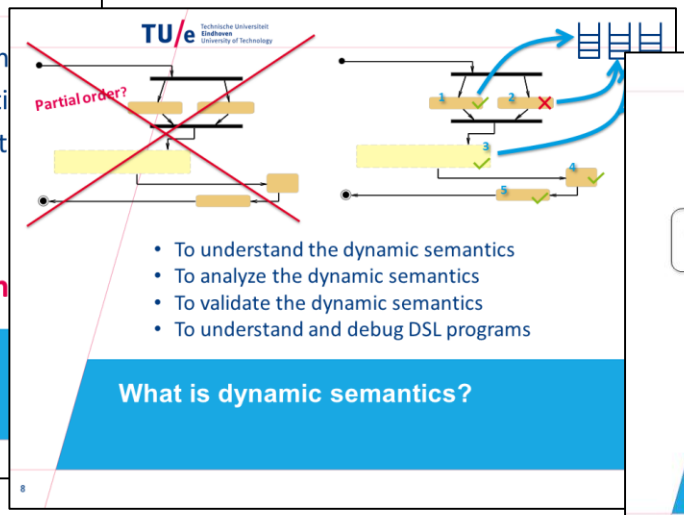

Event-B/Rodin
Specification templates
Event-B specification templates
DSL/Ecore
Meta-model
Constelle definition
Constelle-to-Event-B
DSL model
Event-B specification

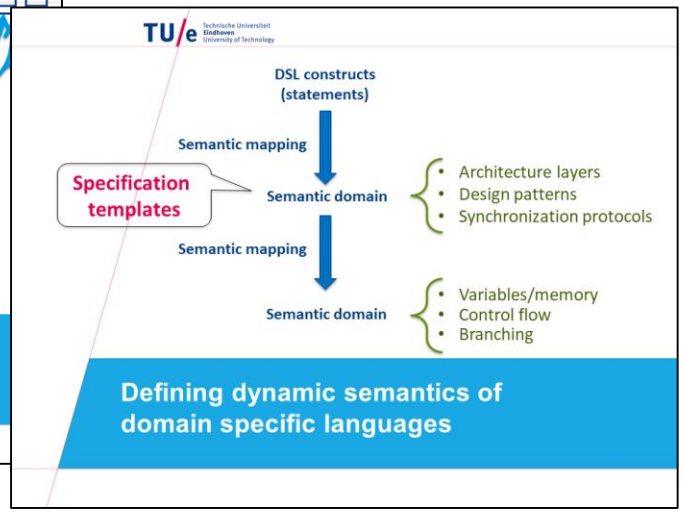
Constelle

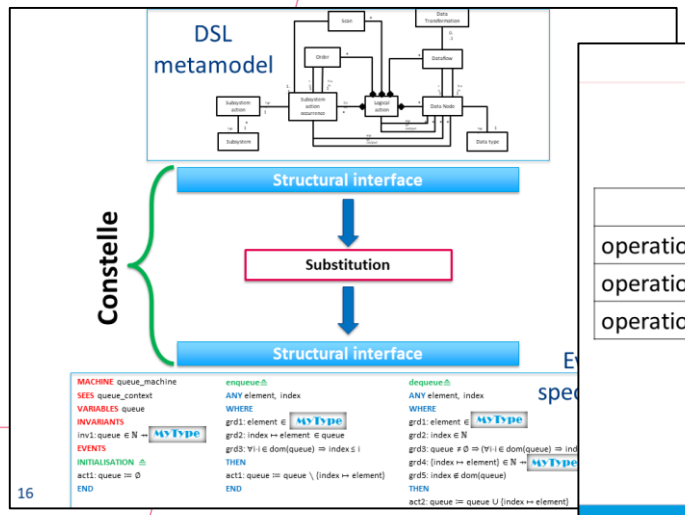
- Scalability and applicability of the proposed method
  - Validated on three different DSLs
- A library of specification templates
  - State machine, communication channels, data typing, design patterns (Listener), data structures (Queue)
- Specification templates for bridging technological diversity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

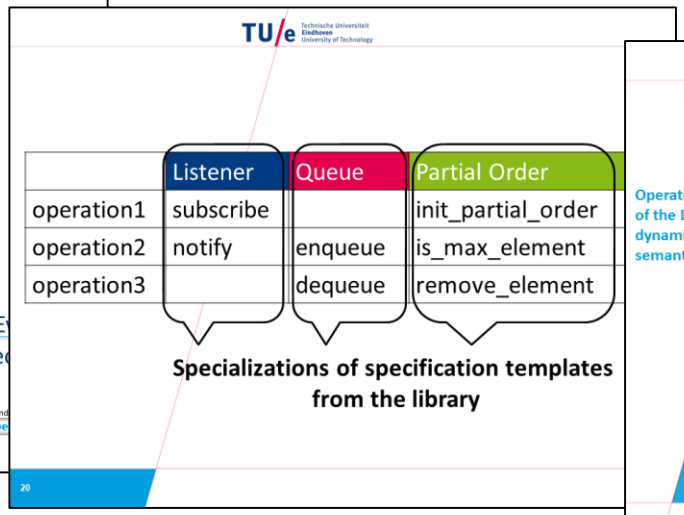
Future work

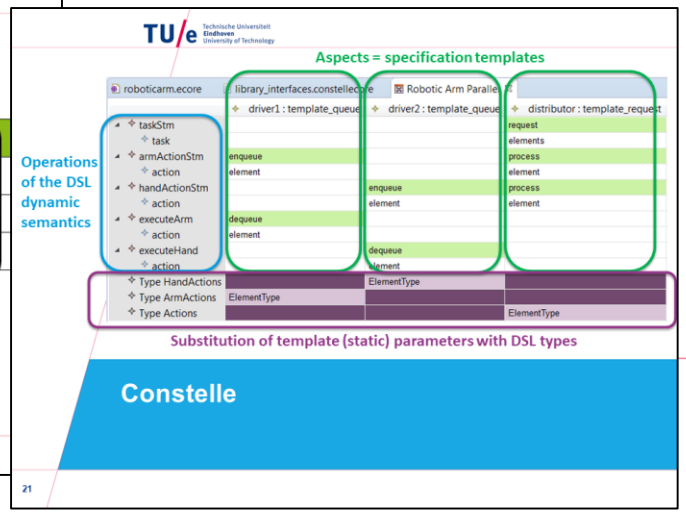












More details: SoSyM paper & PhD thesis