Verifying Concurrency Runtimes using Graph Transformation Systems

Claudio Corrodi[1], Chris Poskitt[2], Alexander Heußner[3]

[1]Software Composition Group, University of Bern, Switzerland

[2]Singapore University of Technology and Design, Singapore

[3]Software Technologies Research Group, University of Bamberg, Germany

ETH zürich

Chair of
Software Engineering

ETH zürich

Chair of
Software Engineering

Software Technologies Research Group

Concurrency Made Easy

O-O Concurrency models

Verification

Testing

Robotics

Concurrency Made Easy

O-O Concurrency models

Verification

Testing

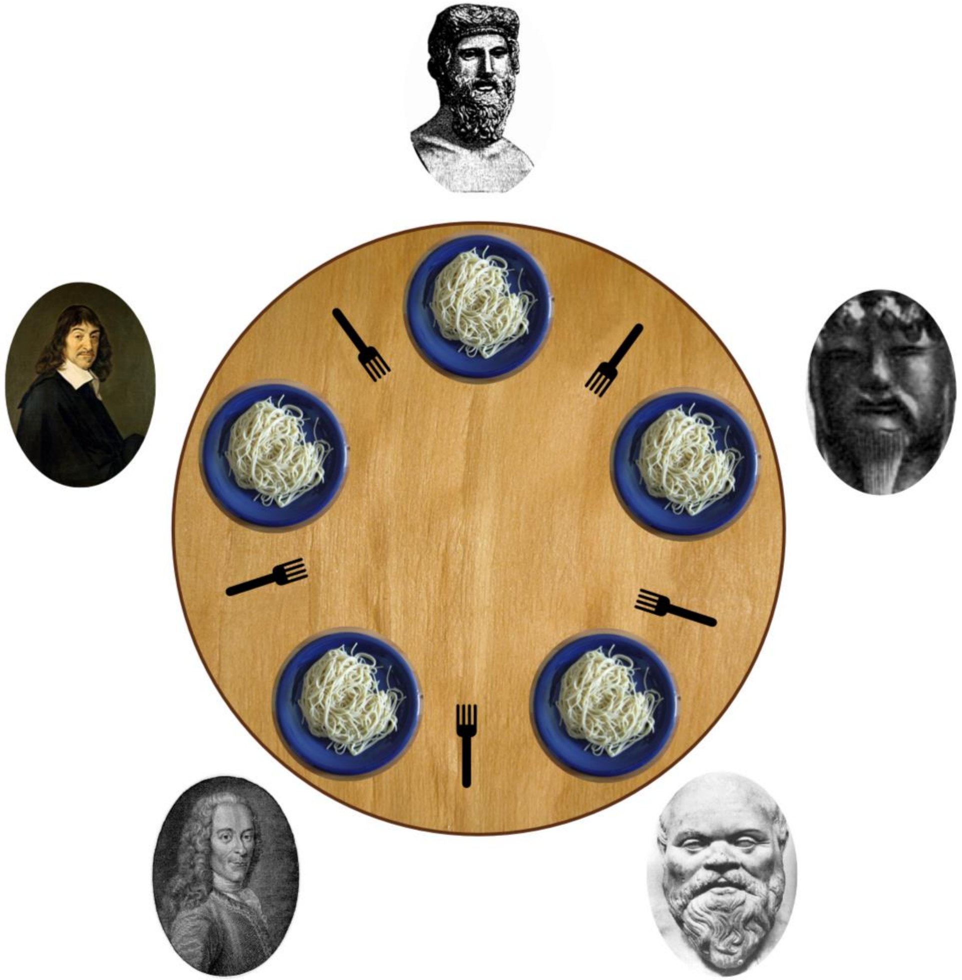
Robotics

SCOOP

Simple Concurrent Object-Oriented Programming

Goal: Raise concurrency abstractions from error-prone (lock based) models to O-O programming

SCOOP

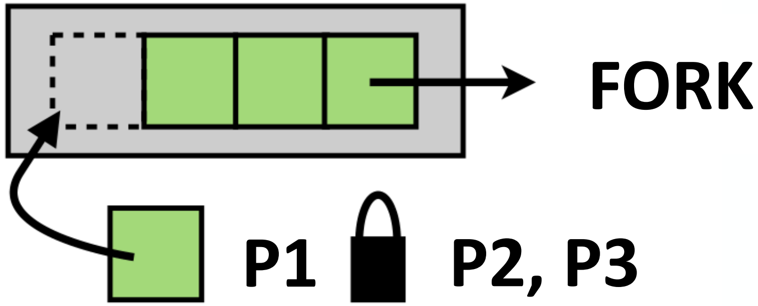


SCOOP

```
eat (left, right: separate FORK)
     do
          left.pick_up
          right.pick_up
          print (“I am eating!”)
          left.put_down
          right.put_down
     end
```

Separate block: No intervening calls between “pick_up” and “put_down”

Execution Models

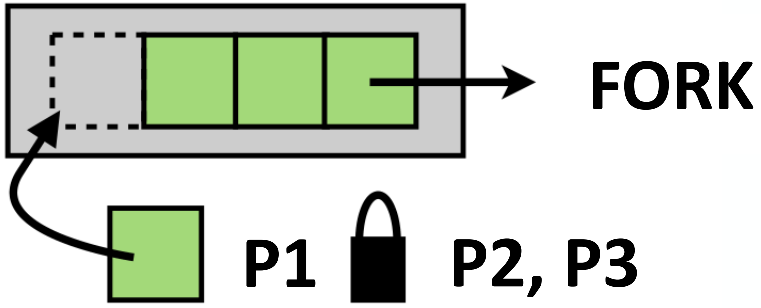


"Request Queues"

Separate block guarantees?

Performant?

Execution Models

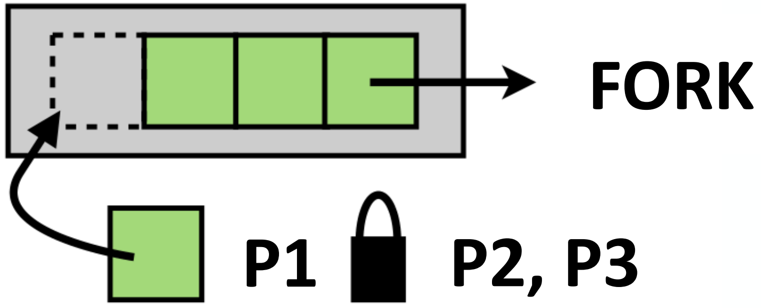


"Request Queues"

Separate block guarantees? ✓

Performant? ✗

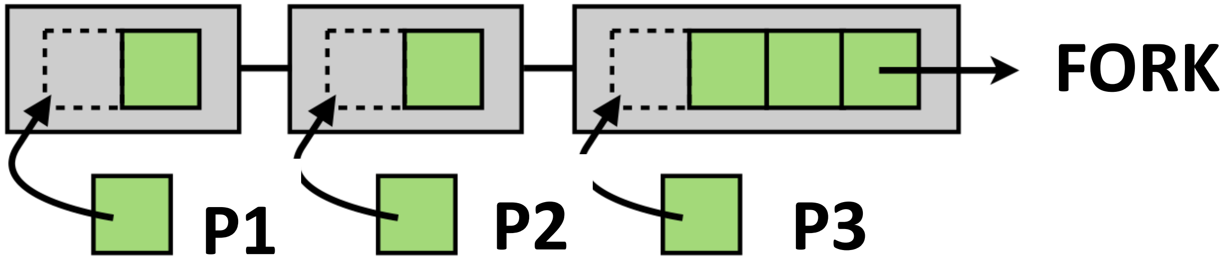
Execution Models



"Request Queues"

Separate block guarantees? ✓

Performant? ✗

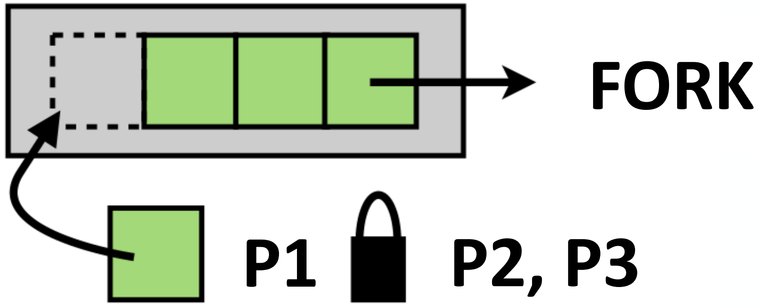


"Queues of Queues"

Separate block guarantees?

Performant?

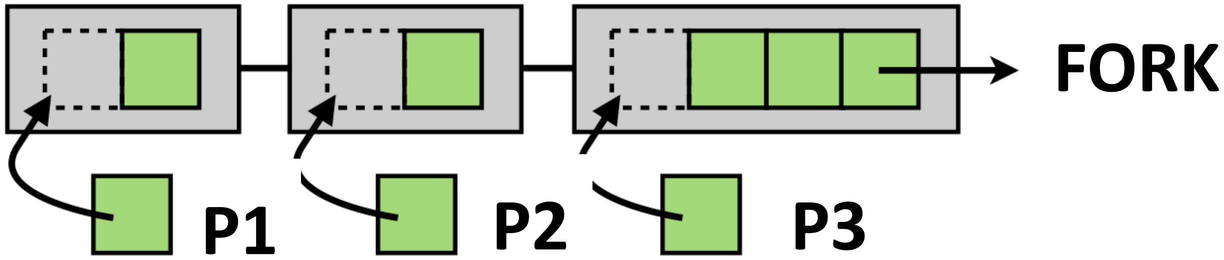
Execution Models



“Request Queues”

Separate block guarantees? ✓

Performant? ✗

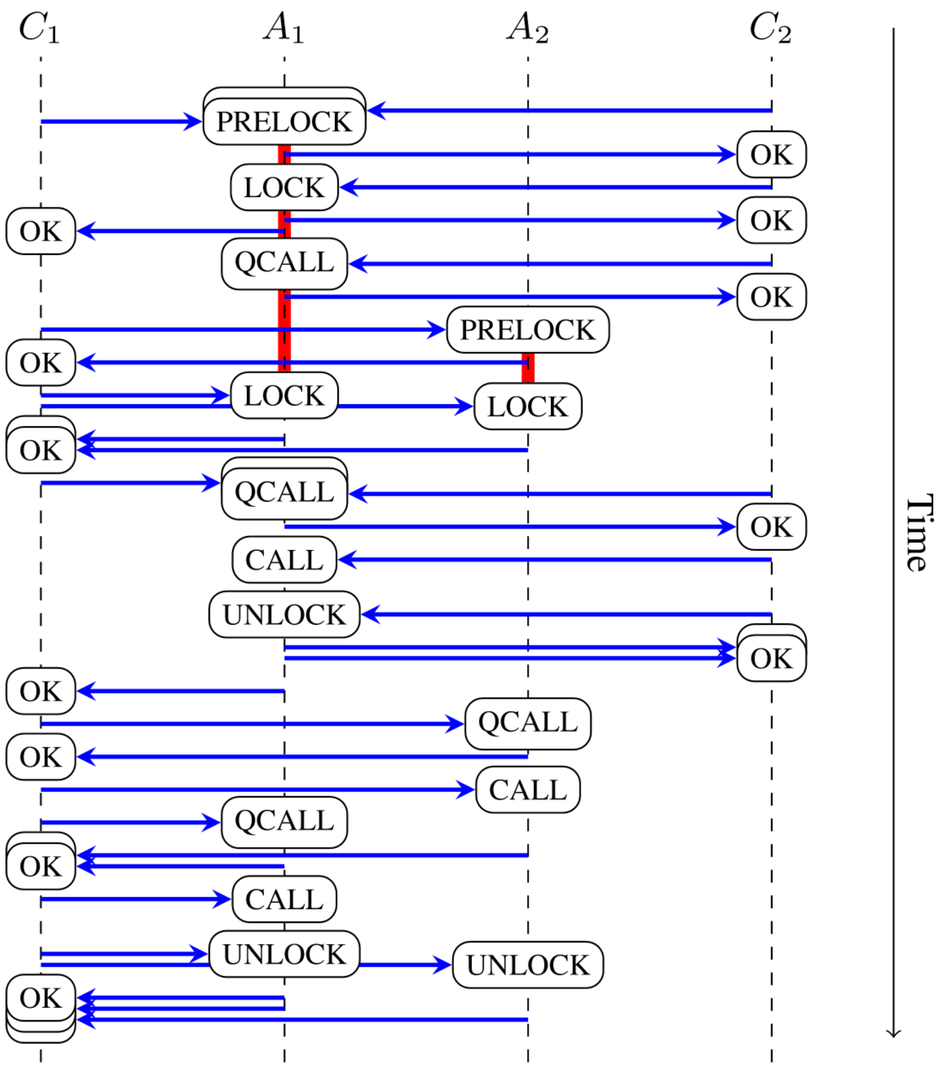


“Queues of Queues”

Separate block guarantees? ✓

Performant? ✓

Execution Models



“Distributed SCOOP”

Extension of “Queues of Queues” model

Correctness

No race conditions?

Absence of deadlocks?

Correctness

No race conditions?

Absence of deadlocks?

Is this still a solution?

```
eat (left, right: separate FORK)
        do
                print (“I am eating!”)
        end
```

Our work

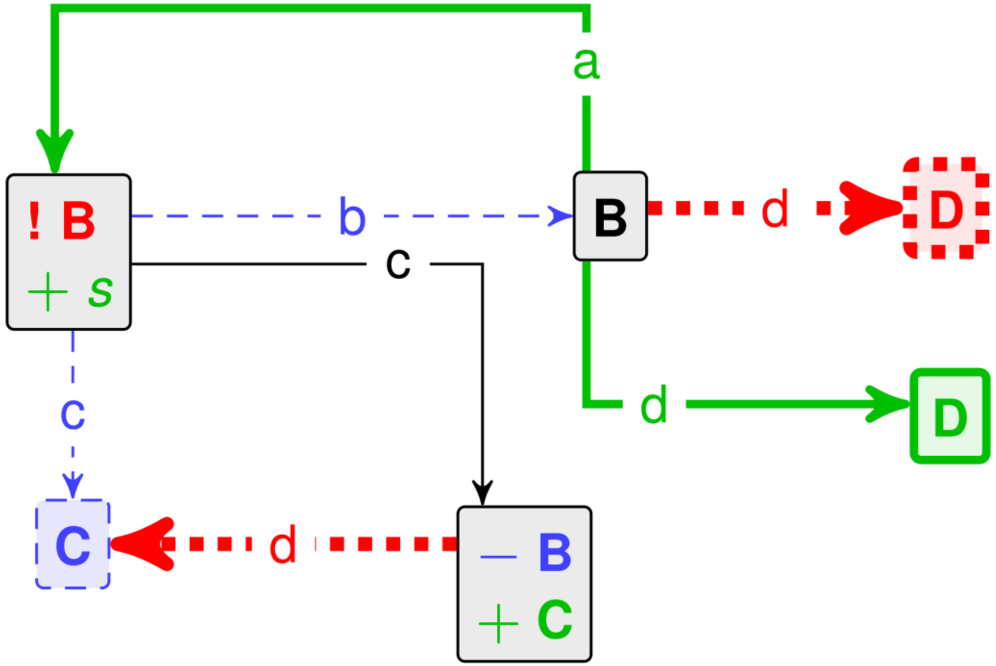
Can we model and simulate—modularly—
competing semantics for a language like SCOOP,
and analyse them for semantic discrepancies?

Approach

- Model runtimes as **graph transformation systems**

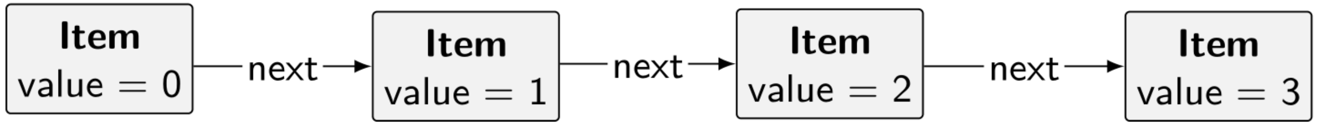
- Modular / paramterisable semantics

- Analyse parameterised GTS against **representative programs** in GROOVE



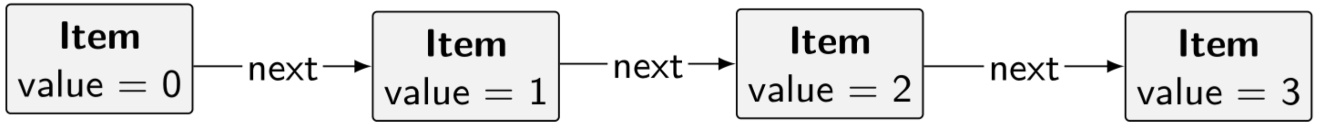
Graph Transformation Systems

Configuration / state graph

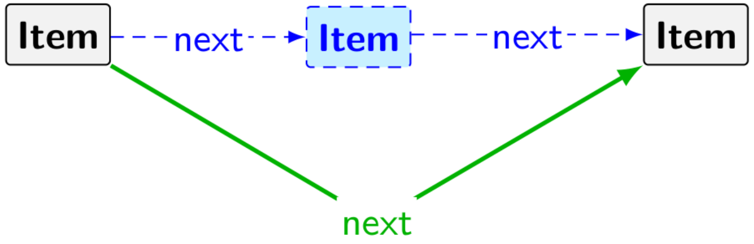


Graph Transformation Systems

Configuration / state graph



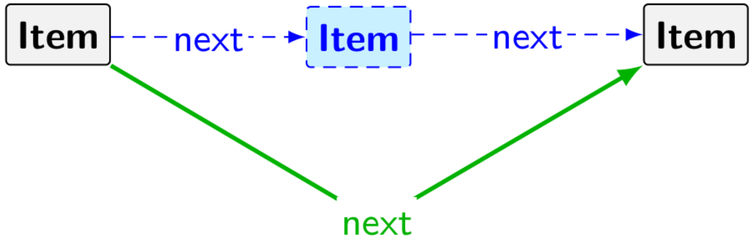
Transformation rule



Graph Transformation Systems

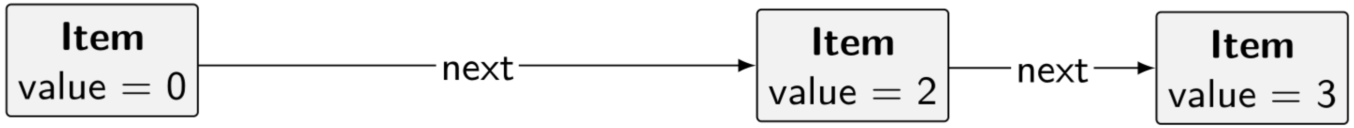
Configuration / state graph

Transformation rule

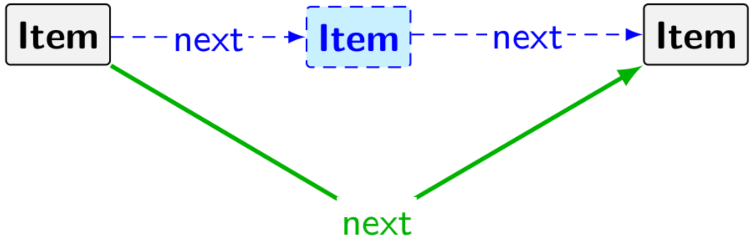


Graph Transformation Systems

Configuration / state graph



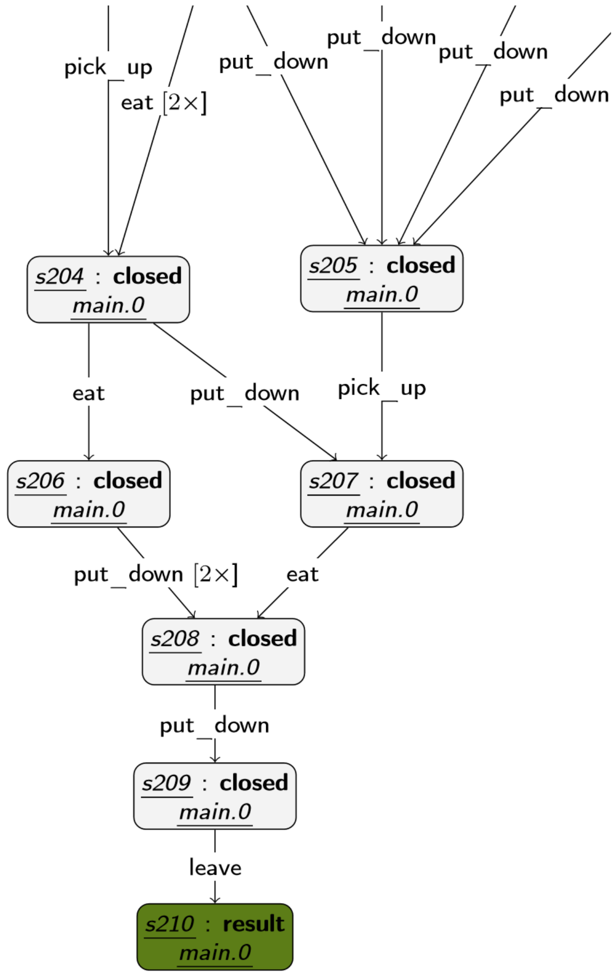
Transformation rule



Graph Transformation Systems

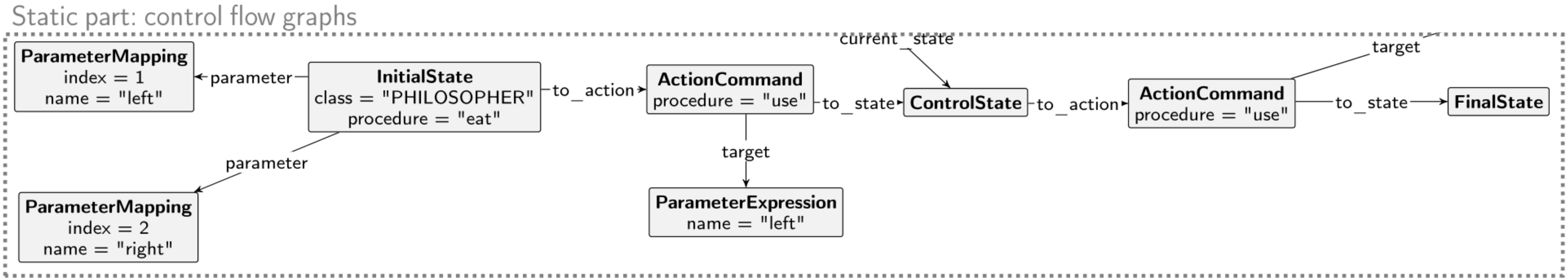
State-space exploration

Nondeterministic application of any matching rule

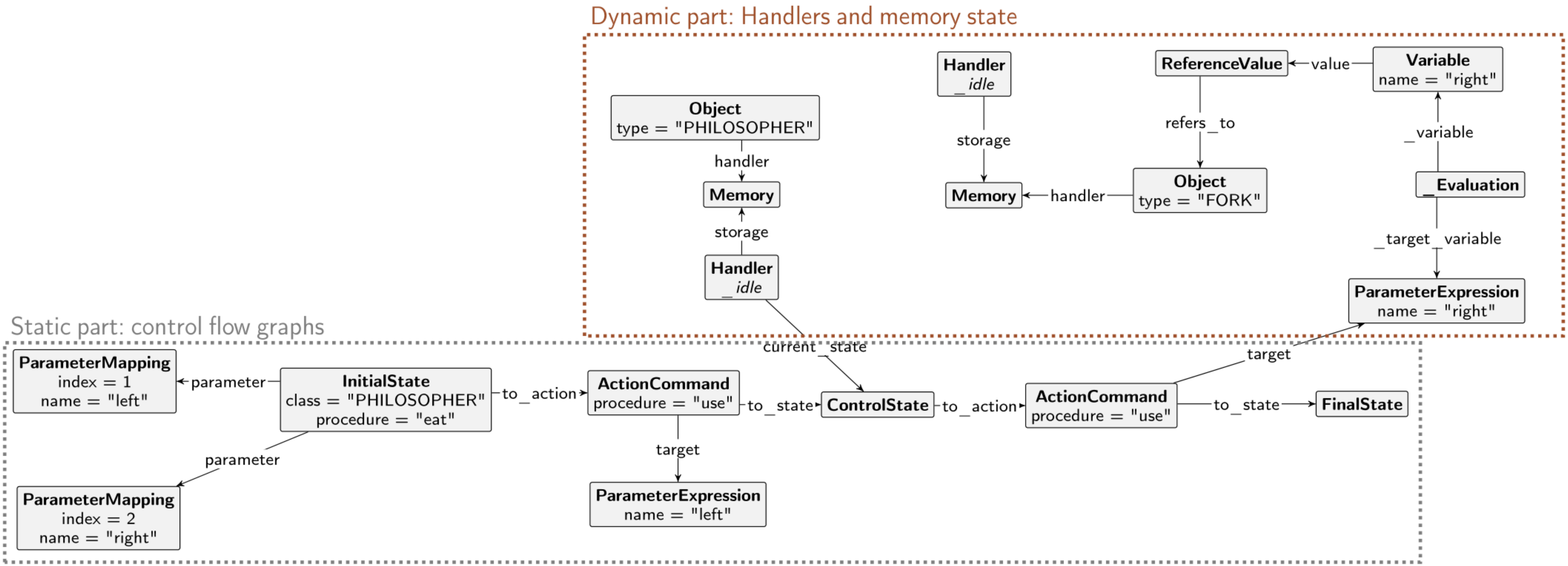


(labeled transition system)

SCOOP GTS

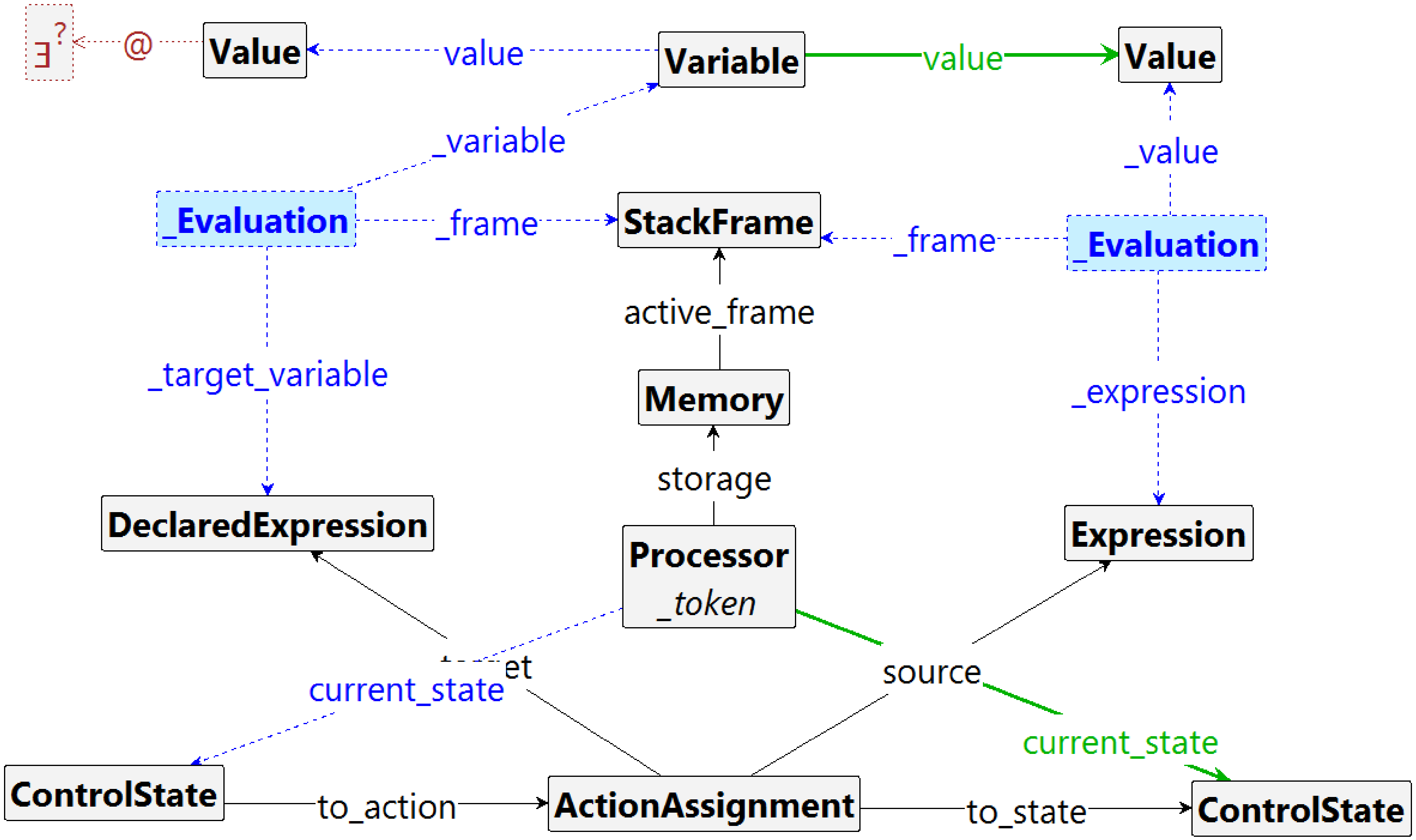

Static part: control flow graphs
ParameterMapping
index = 1
name = "left"
parameter
InitialState
class = "PHILOSOPHER"
procedure = "eat"
to_action
ActionCommand
procedure = "use"
to_state
current_state
ControlState
to_action
target
ActionCommand
procedure = "use"
to_state
FinalState
parameter
target
ParameterMapping
index = 2
name = "right"
ParameterExpression
name = "left"

SCOOP GTS

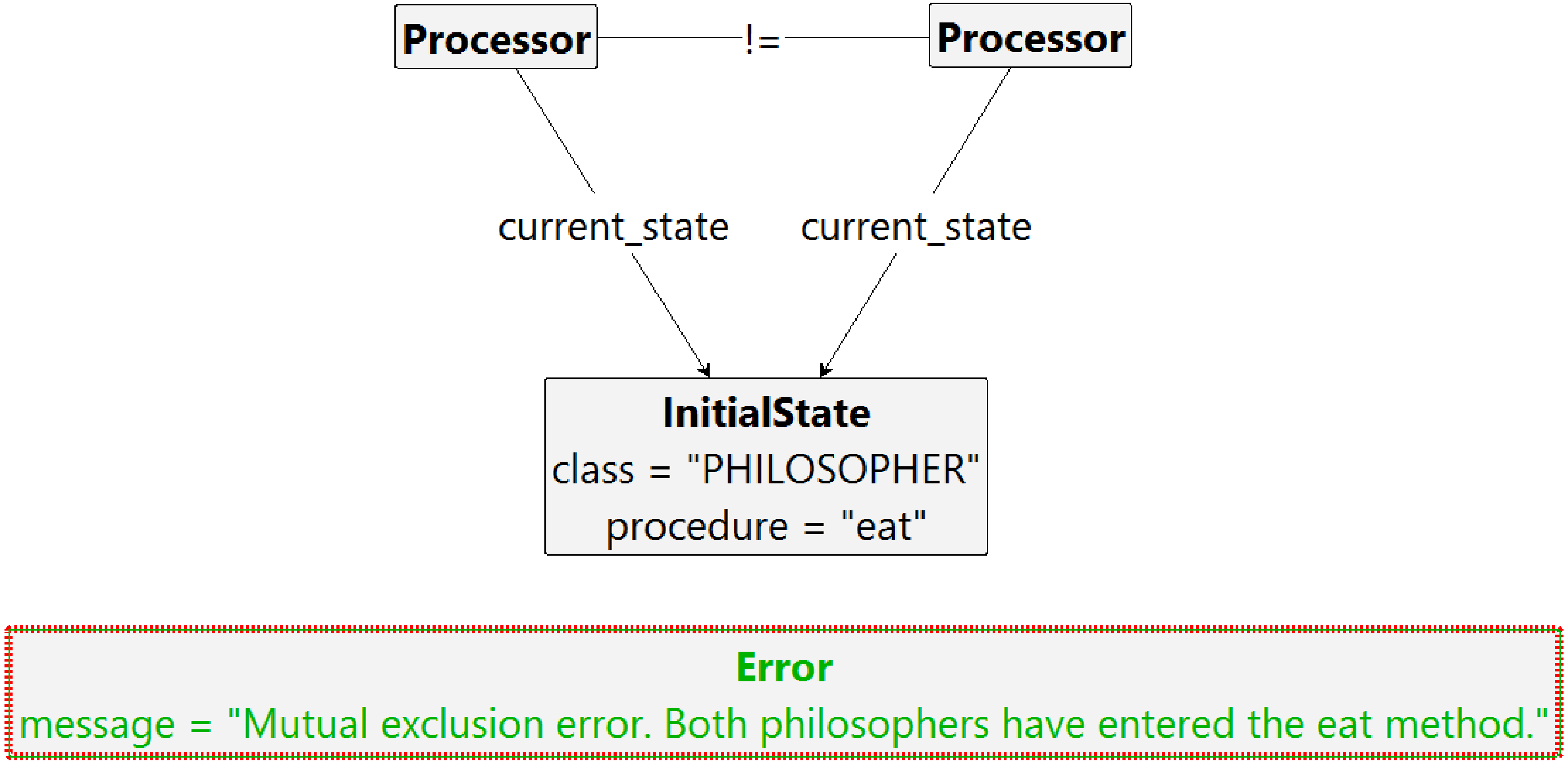


Dynamic part: Handlers and memory state
Static part: control flow graphs
Object
type = "PHILOSOPHER"
handler
Memory
storage
Handler
_idle
current_state
Handler
_idle
storage
Memory
handler
Object
type = "FORK"
ReferenceValue
refers_to
value
Variable
name = "right"
_variable
_Evaluation
_target_variable
ParameterExpression
name = "right"
ParameterMapping
index = 1
name = "left"
parameter
InitialState
class = "PHILOSOPHER"
procedure = "eat"
to_action
ActionCommand
procedure = "use"
to_state
ControlState
to_action
ActionCommand
procedure = "use"
to_state
FinalState
target
parameter
ParameterMapping
index = 2
name = "right"
target
ParameterExpression
name = "left"

SCOOP GTS


∃
?
@
Value
value
Variable
value
Value
_variable
_value
_Evaluation
_frame
StackFrame
_frame
_Evaluation
active_frame
_target_variable
Memory
_expression
storage
DeclaredExpression
Processor
_token
Expression
et
current_state
source
current_state
ControlState
to_action
ActionAssignment
to_state
ControlState

Detecting errors



Graph	Runtime	Configurations	Time [s]
DP 2 eager	QoQ	5,863	25.5
	RQ	4,219	18.2
	DSCOOP	13,046	52.9
DP 2 lazy	QoQ	9,609	40.8
	RQ	5,679	23.5
	DSCOOP	18,874	73.0
DP 3 eager	QoQ	227,797	1,480.6
	RQ	99,198	436.3
	DSCOOP	523,513	2,726.0
DP 3 lazy	QoQ	444,689	2,424.9
	RQ	170,249	1,090.1
	DSCOOP	1,288,663	5,999.5
PC 20	QoQ	50,286	575.0
	RQ	12,890	141.6
	DSCOOP	90,434	997.7

Graphs as Models
2015

Publications

Towards Practical Graph-Based Verification for an Object-Oriented Concurrency Model

Alexander Heußner
University of Bamberg, Germany

Christopher M. Poskitt Claudio Corrodi
Benjamin Morandi
Department of Computer Science
ETH Zürich, Switzerland

To harness the power of multi-core and distributed platforms, and to make the development of concurrent software more accessible to software engineers, different object-oriented concurrency models such as SCOOP have been proposed. Despite the practical importance of analysing SCOOP programs, there are currently no general verification approaches that operate directly on program code without additional annotations. One reason for this is the multitude of partially conflicting semantic formalisations for SCOOP (either in theory or by-implementation). Here, we propose a simple graph transformation system (GTS) based run-time semantics for SCOOP that grasps the most common features of all known semantics of the language. This run-time model is implemented in the state-of-the-art GTS tool GROOVE, which allows us to simulate, analyse, and verify a subset of SCOOP

FASE 2016

Publications

A Graph-Based Semantics Workbench for Concurrent Asynchronous Programs

Claudio Corrodi[1,2]⋆, Alexander Heußner[3], and Christopher M. Poskitt[1,4]⋆

[1] Department of Computer Science, ETH Zürich, Switzerland
[2] Software Composition Group, University of Bern, Switzerland
[3] Software Technologies Research Group, University of Bamberg, Germany
[4] Singapore University of Technology and Design, Singapore

Abstract. A number of novel programming languages and libraries have been proposed that offer simpler-to-use models of concurrency than threads. It is challenging, however, to devise execution models that successfully realise their abstractions without forfeiting performance or introducing unintended behaviours. This is exemplified by SCOOP—a concurrent object-oriented message-passing language—which has seen multiple semantics proposed and implemented over its evolution. We propose

FAC 2017
(under review)

Publications

A Semantics Comparison Workbench for Concurrent, Asynchronous, Distributed Programs

Claudio Corrodi[1], Alexander Heußner[2], and Christopher M. Poskitt[3]

[1]Software Composition Group, University of Bern, Switzerland
[2]Software Technologies Research Group, University of Bamberg, Germany
[3]Singapore University of Technology and Design, Singapore

Abstract. A number of high-level languages and libraries have been proposed that offer novel and simple to use abstractions for concurrent, asynchronous, and distributed programming. The execution models that realise them, however, often change over time—whether to improve performance, or to extend them to new language features—potentially affecting behavioural and safety properties of existing programs. This is exemplified by SCOOP, a message-passing approach to concurrent object-oriented programming that has

Acknowledgments

- Many slides are adapted from related similar presentations given by Chris Poskitt and Alexander Heußner

- Dining philosophers image taken from Wikipedia

- D-SCOOP figure taken from “An Interference-Free Programming Model for Network Objects” (Schill, Poskitt, Meyer; 2016)