

# Process Calculi and the Poly★ System

## A First Year Report

Jan Jakubův

Supervisors: Dr. J. B. Wells and Prof. F. Kamareddine

School of Mathematical and Computer Sciences  
Heriot-Watt University

June 4, 2008

# Outline

## Introduction to Process Calculi

### The Poly★ System

- Meta★: syntax and description of reduction rules

- Poly★: shape predicates and types for Meta★

### Usage of the Poly★ System

- Poly★ as a type system for Mobile Ambients

- Poly★ and a flow analysis

# What are process calculi?

- ▶ Process Calculi are intended to model concurrent systems, that is, systems where several **interacting** units (called *processes*) engage in activity at the same time.
- ▶ There may be several ways of evaluation which leads to different states
- ▶ The current state of the modeled system may depend on interaction with its environment.
- ▶ Process Calculi use rewrite systems to model concurrent systems.

# What are process calculi?

- ▶ Process Calculi are intended to model concurrent systems, that is, systems where several **interacting** units (called *processes*) engage in activity at the same time.
- ▶ There may be several ways of evaluation which leads to different states
- ▶ The current state of the modeled system may depend on interaction with its environment.
- ▶ Process Calculi use rewrite systems to model concurrent systems.

# What are process calculi?

- ▶ Process Calculi are intended to model concurrent systems, that is, systems where several **interacting** units (called *processes*) engage in activity at the same time.
- ▶ There may be several ways of evaluation which leads to different states
- ▶ The current state of the modeled system may depend on interaction with its environment.
- ▶ Process Calculi use rewrite systems to model concurrent systems.

# What are process calculi?

- ▶ Process Calculi are intended to model concurrent systems, that is, systems where several **interacting** units (called *processes*) engage in activity at the same time.
- ▶ There may be several ways of evaluation which leads to different states
- ▶ The current state of the modeled system may depend on interaction with its environment.
- ▶ Process Calculi use rewrite systems to model concurrent systems.

# Usage of process calculi

- ▶ In the Computer Science
  - ▶ computer systems with programs running in parallel
  - ▶ computer networks
  - ▶ security and cryptography
- ▶ Outside of CS
  - ▶ biological/molecular systems
  - ▶ work flow in business management

# Usage of process calculi

- ▶ In the Computer Science
  - ▶ computer systems with programs running in parallel
  - ▶ computer networks
  - ▶ security and cryptography
- ▶ Outside of CS
  - ▶ biological/molecular systems
  - ▶ work flow in business management

# Main properties of process calculi

- ▶ **concurrency** - processes can run in parallel
- ▶ **communication** - processes can communicate among themselves by sending simple values
- ▶ **mobility** - processes are connected or placed in a hierarchy, and connections or the hierarchy can change during computation

# Main properties of process calculi

- ▶ **concurrency** - processes can run in parallel
- ▶ **communication** - processes can communicate among themselves by sending simple values
- ▶ **mobility** - processes are connected or placed in a hierarchy, and connections or the hierarchy can change during computation

# Main properties of process calculi

- ▶ **concurrency** - processes can run in parallel
- ▶ **communication** - processes can communicate among themselves by sending simple values
- ▶ **mobility** - processes are connected or placed in a hierarchy, and connections or the hierarchy can change during computation

# What is Meta★?

- ▶ **Meta★ provides a generic syntax of process terms**
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is

▶ A generic syntax of process terms

▶ A generic way to specify a particular semantics

# What is Meta★?

- ▶ Meta★ provides a generic syntax of process terms
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is

▶ a generic syntax of process terms

▶ a generic way to specify a particular semantics

# What is Meta★?

- ▶ Meta★ provides a generic syntax of process terms
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is

# What is Meta★?

- ▶ Meta★ provides a generic syntax of process terms
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is

# What is Meta★?

- ▶ Meta★ provides a generic syntax of process terms
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is

# What is Meta★?

- ▶ Meta★ provides a generic syntax of process terms
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is ...

1. A generic syntax of process terms, that provides
2. a generic way to describe a particular semantics

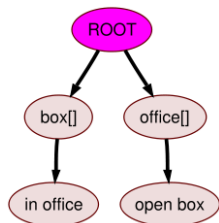
# What is Meta★?

- ▶ Meta★ provides a generic syntax of process terms
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is
  1. A generic syntax of process terms, that provides
  2. a generic way to **describe** a particular semantics

# What is Meta★?

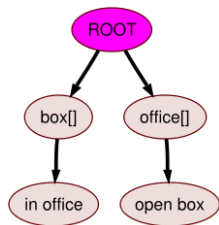
- ▶ Meta★ provides a generic syntax of process terms
- ▶ it supports basic constructions found in process calculi
- ▶ it does not provide a fixed semantics
- ▶ it provides a generic way to specify a particular semantics
- ▶ thus, Meta★ can be **instantiated** to a specific calculus
- ▶ So, Meta★ is
  1. A generic syntax of process terms, that provides
  2. a generic way to **describe** a particular semantics

# What is Poly★?



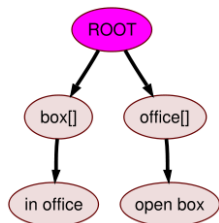
- ▶ Poly★ is a **type system** for every calculus described in Meta★.
- ▶ Central concept of Poly★ is a **shape predicate**: it is a graph that looks like a term syntax tree.
- ▶ A term **matches** a shape predicate if its syntax tree can be bent into the shape of the shape predicate.
- ▶ A shape predicate that is closed under the given reduction rules is called **type**.
- ▶ A type represents all possible computation futures of the source term smashed together in one place.

# What is Poly★?



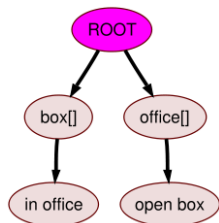
- ▶ Poly★ is a **type system** for every calculus described in Meta★.
- ▶ Central concept of Poly★ is a **shape predicate**: it is a graph that looks like a term syntax tree.
- ▶ A term **matches** a shape predicate if its syntax tree can be bent into the shape of the shape predicate.
- ▶ A shape predicate that is closed under the given reduction rules is called **type**.
- ▶ A type represents all possible computation futures of the source term smashed together in one place.

# What is Poly★?



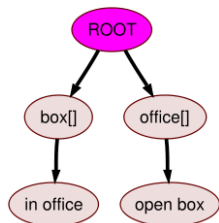
- ▶ Poly★ is a **type system** for every calculus described in Meta★.
- ▶ Central concept of Poly★ is a **shape predicate**: it is a graph that looks like a term syntax tree.
- ▶ A term **matches** a shape predicate if its syntax tree can be bent into the shape of the shape predicate.
- ▶ A shape predicate that is closed under the given reduction rules is called **type**.
- ▶ A type represents all possible computation futures of the source term smashed together in one place.

# What is Poly★?



- ▶ Poly★ is a **type system** for every calculus described in Meta★.
- ▶ Central concept of Poly★ is a **shape predicate**: it is a graph that looks like a term syntax tree.
- ▶ A term **matches** a shape predicate if its syntax tree can be bent into the shape of the shape predicate.
- ▶ A shape predicate that is closed under the given reduction rules is called **type**.
- ▶ A type represents all possible computation futures of the source term smashed together in one place.

# What is Poly★?



- ▶ Poly★ is a **type system** for every calculus described in Meta★.
- ▶ Central concept of Poly★ is a **shape predicate**: it is a graph that looks like a term syntax tree.
- ▶ A term **matches** a shape predicate if its syntax tree can be bent into the shape of the shape predicate.
- ▶ A shape predicate that is closed under the given reduction rules is called **type**.
- ▶ A type represents all possible computation futures of the source term smashed together in one place.

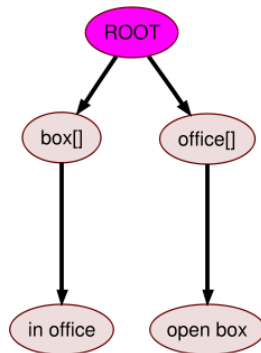
# Poly★ types

- ▶ types have to be closed under reduction rules:

**box [in office.0] | office [open box.0] →**

office [box [0] | open box.0] →

office [0 | 0]



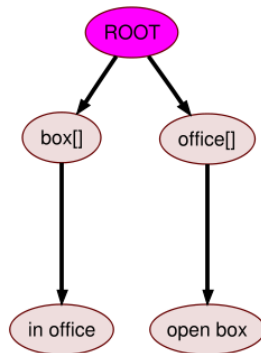
# Poly★ types

- types have to be closed under reduction rules:

`box [in office.0] | office [open box.0] →`

`office [box [0] | open box.0] →`

`office [0 | 0]`



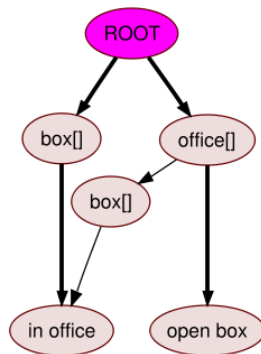
# Poly★ types

- types have to be closed under reduction rules:

$\text{box}[\text{in office}.0] \mid \text{office}[\text{open box}.0] \rightarrow$

$\text{office}[\text{box}[0] \mid \text{open box}.0] \rightarrow$

$\text{office}[0 \mid 0]$



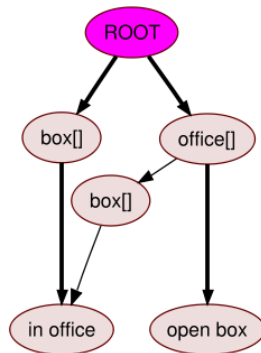
# Poly★ types

- types have to be closed under reduction rules:

$\text{box}[\text{in office}.0] \mid \text{office}[\text{open box}.0] \rightarrow$

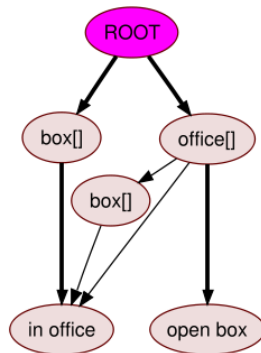
$\text{office}[\text{box}[0] \mid \text{open box}.0] \rightarrow$

$\text{office}[0 \mid 0]$



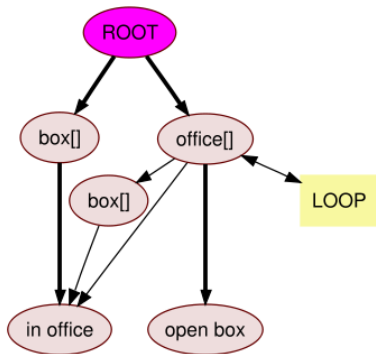
# Poly★ types

- types have to be closed under reduction rules:  
 $\text{box}[\text{in office}.0] \mid \text{office}[\text{open box}.0] \rightarrow$   
 $\text{office}[\text{box}[0] \mid \text{open box}.0] \rightarrow$   
 $\text{office}[0 \mid 0]$



# Poly★ types

- types have to be **closed** under reduction rules:  
 $\text{box}[\text{in office}.0] \mid \text{office}[\text{open box}.0] \rightarrow$   
 $\text{office}[\text{box}[0] \mid \text{open box}.0] \rightarrow$   
 $\text{office}[0 \mid 0]$



# Poly★ and the seminal Mobile Ambients type system

- ▶ There are many type systems for Mobile Ambients.
- ▶ The seminal one is by Cardeli and Gordon, here called TMA.
- ▶ TMA is intended to allow only **well-formed** terms and communication.
- ▶ TMA's type judgment relation can be exactly emulated by Poly★'s type judgment relation.
- ▶ In the simplified version of TMA, also the meaning of TMA's types can be exactly captured by Poly★'s types.

# Poly★ and the seminal Mobile Ambients type system

- ▶ There are many type systems for Mobile Ambients.
- ▶ The seminal one is by Cardeli and Gordon, here called TMA.
- ▶ TMA is intended to allow only **well-formed** terms and communication.
- ▶ TMA's type judgment relation can be exactly emulated by Poly★'s type judgment relation.
- ▶ In the simplified version of TMA, also the meaning of TMA's types can be exactly captured by Poly★'s types.

# Poly★ and the seminal Mobile Ambients type system

- ▶ There are many type systems for Mobile Ambients.
- ▶ The seminal one is by Cardeli and Gordon, here called TMA.
- ▶ TMA is intended to allow only **well-formed** terms and communication.
- ▶ TMA's type judgment relation can be exactly emulated by Poly★'s type judgment relation.
- ▶ In the simplified version of TMA, also the meaning of TMA's types can be exactly captured by Poly★'s types.

# Poly★ and the seminal Mobile Ambients type system

- ▶ There are many type systems for Mobile Ambients.
- ▶ The seminal one is by Cardeli and Gordon, here called TMA.
- ▶ TMA is intended to allow only **well-formed** terms and communication.
- ▶ TMA's type judgment relation can be exactly emulated by Poly★'s type judgment relation.
- ▶ In the simplified version of TMA, also the meaning of TMA's types can be exactly captured by Poly★'s types.

# Poly★ and the seminal Mobile Ambients type system

- ▶ There are many type systems for Mobile Ambients.
- ▶ The seminal one is by Cardeli and Gordon, here called TMA.
- ▶ TMA is intended to allow only **well-formed** terms and communication.
- ▶ TMA's type judgment relation can be exactly emulated by Poly★'s type judgment relation.
- ▶ In the simplified version of TMA, also the meaning of TMA's types can be exactly captured by Poly★'s types.

# Poly★ and a flow analysis

- ▶ Poly★ can be used for a **flow analysis** of systems described in Meta★.
- ▶ A flow analysis provides a way to determine **over approximation** of all possible states of the modeled system.
- ▶ Poly★ provides, for example, comparable results with a flow analysis for **BioAmbients** of Nielson et. al.
- ▶ BioAmbients is a variant of Mobile Ambients intended to model biological systems.

# Poly★ and a flow analysis

- ▶ Poly★ can be used for a **flow analysis** of systems described in Meta★.
- ▶ A flow analysis provides a way to determine **over approximation** of all possible states of the modeled system.
- ▶ Poly★ provides, for example, comparable results with a flow analysis for **BioAmbients** of Nielson et. al.
- ▶ BioAmbients is a variant of Mobile Ambients intended to model biological systems.

# Poly★ and a flow analysis

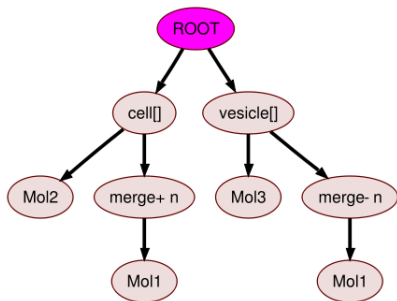
- ▶ Poly★ can be used for a **flow analysis** of systems described in Meta★.
- ▶ A flow analysis provides a way to determine **over approximation** of all possible states of the modeled system.
- ▶ Poly★ provides, for example, comparable results with a flow analysis for **BioAmbients** of Nielson et. al.
- ▶ BioAmbients is a variant of Mobile Ambients intended to model biological systems.

# Poly★ and a flow analysis

- ▶ Poly★ can be used for a **flow analysis** of systems described in Meta★.
- ▶ A flow analysis provides a way to determine **over approximation** of all possible states of the modeled system.
- ▶ Poly★ provides, for example, comparable results with a flow analysis for **BioAmbients** of Nielson et. al.
- ▶ BioAmbients is a variant of Mobile Ambients intended to model biological systems.

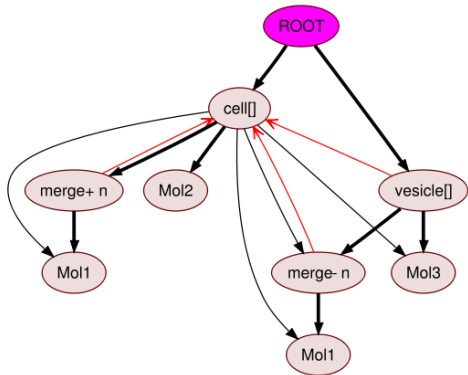
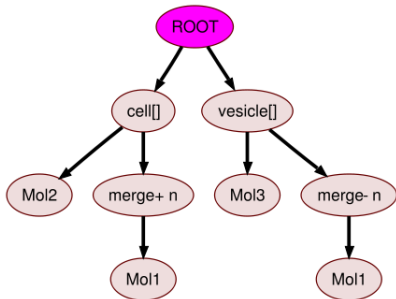
# Example of a flow analysis in Poly★

Membrane fusion as ambient merger in BioAmbients



# Example of a flow analysis in Poly★

Membrane fusion as ambient merger in BioAmbients



## Summary

- ▶ Poly★ is a **general** type system for a **large** family of process calculi.
- ▶ Poly★ type system enjoys some properties that are not in common for other process calculi type systems (**spatial polymorphisms**).
- ▶ Poly★ has an **implemented** type inference algorithm.

## Summary of the work done in the first year

1. Comparison of Poly★ with the seminal Mobile Ambients type system.
2. Comparison of Poly★ with the flow analysis of BioAmbients.
3. Extension of the Poly★ type inference implementation to support the 'rec' operator.
4. Comparison and study of approaches to recursion and nondeterminism in process calculi.