

# 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

- 1 Introduction to Process Calculi
  - Motivation, history and usage
  - Mobile Ambients
- 2 The Poly\* System
  - Meta\*: syntax and description of reduction rules
  - Poly\*: shape predicates and types for Meta\*
- 3 Usage of the Poly\* System
  - Poly\* as a type system for Mobile Ambients
  - Poly\* and a flow analysis

## Outline

- 1 Introduction to Process Calculi
  - Motivation, history and usage
  - Mobile Ambients
- 2 The Poly\* System
  - Meta\*: syntax and description of reduction rules
  - Poly\*: shape predicates and types for Meta\*
- 3 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.

# 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.

# 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.

# History of modeling of concurrent systems

- History can be traced back to the 70th of the 20th century.
- The first formalism that deals with concurrency:
  - Petri Nets of C. A. Petri (1962)
- The first process calculi
  - Calculus of Communicating Systems (CCS)  
of R. Milner (1980)
  - Communicating Sequential Processes (CSP)  
of C. A. R. Hoare (1978)
- Modern process calculi (addressing mobility)
  - $\pi$ -calculus [Par01]  
of R. Milner, J. Parrow, and D. Walker (1992)
  - Mobile Ambients [CG98]  
of L. Cardelli and A. D. Gordon (1998)

# History of modeling of concurrent systems

- History can be traced back to the 70th of the 20th century.
- The first formalism that deals with concurrency:
  - Petri Nets of C. A. Petri (1962)
- The first process calculi
  - Calculus of Communicating Systems (CCS)  
of R. Milner (1980)
  - Communicating Sequential Processes (CSP)  
of C. A. R. Hoare (1978)
- Modern process calculi (addressing mobility)
  - $\pi$ -calculus [Par01]  
of R. Milner, J. Parrow, and D. Walker (1992)
  - Mobile Ambients [CG98]  
of L. Cardelli and A. D. Gordon (1998)

# History of modeling of concurrent systems

- History can be traced back to the 70th of the 20th century.
- The first formalism that deals with concurrency:
  - Petri Nets of C. A. Petri (1962)
- The first process calculi
  - Calculus of Communicating Systems (CCS)  
of R. Milner (1980)
  - Communicating Sequential Processes (CSP)  
of C. A. R. Hoare (1978)
- Modern process calculi (addressing mobility)
  - $\pi$ -calculus [Par01]  
of R. Milner, J. Parrow, and D. Walker (1992)
  - Mobile Ambients [CG98]  
of L. Cardelli and A. D. Gordon (1998)

# History of modeling of concurrent systems

- History can be traced back to the 70th of the 20th century.
- The first formalism that deals with concurrency:
  - Petri Nets of C. A. Petri (1962)
- The first process calculi
  - Calculus of Communicating Systems (CCS)  
of R. Milner (1980)
  - Communicating Sequential Processes (CSP)  
of C. A. R. Hoare (1978)
- Modern process calculi (addressing mobility)
  - $\pi$ -calculus [Par01]  
of R. Milner, J. Parrow, and D. Walker (1992)
  - Mobile Ambients [CG98]  
of L. Cardelli and A. D. Gordon (1998)

# Usage of process calculi

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

## Outline

- 1 Introduction to Process Calculi
  - Motivation, history and usage
  - **Mobile Ambients**
- 2 The Poly\* System
  - Meta\*: syntax and description of reduction rules
  - Poly\*: shape predicates and types for Meta\*
- 3 Usage of the Poly\* System
  - Poly\* as a type system for Mobile Ambients
  - Poly\* and a flow analysis

# Mobile Ambients I

- processes ( $P$ ) can run in *parallel*:  $P_1 \mid P_2$
- process can run inside an *ambient* boundary:  $a[P]$
- ambients are named by single *names* (denoted  $a$ )
- an ambient can contain other ambients:  
 $a[a_1[\dots] \mid a_2[\dots] \mid P]$
- ambients form a *hierarchy tree*
- syntactically,  $a[P]$  is also a process

# Mobile Ambients I

- processes ( $P$ ) can run in *parallel*:  $P_1 \mid P_2$
- process can run inside an *ambient* boundary:  $a[P]$
- ambients are named by single *names* (denoted  $a$ )
- an ambient can contain other ambients:  
 $a[a_1[\dots] \mid a_2[\dots] \mid P]$
- ambients form a *hierarchy tree*
- syntactically,  $a[P]$  is also a process

# Mobile Ambients I

- processes ( $P$ ) can run in *parallel*:  $P_1 \mid P_2$
- process can run inside an *ambient* boundary:  $a[P]$
- ambients are named by single *names* (denoted  $a$ )
- an ambient can contain other ambients:  
 $a[a_1[\dots] \mid a_2[\dots] \mid P]$
- ambients form a *hierarchy tree*
- syntactically,  $a[P]$  is also a process

# Mobile Ambients I

- processes ( $P$ ) can run in *parallel*:  $P_1 \mid P_2$
- process can run inside an *ambient* boundary:  $a[P]$
- ambients are named by single *names* (denoted  $a$ )
- an ambient can contain other ambients:  
 $a[a_1[\dots] \mid a_2[\dots] \mid P]$
- ambients form a *hierarchy tree*
- syntactically,  $a[P]$  is also a process

# Mobile Ambients I

- processes ( $P$ ) can run in *parallel*:  $P_1 \mid P_2$
- process can run inside an *ambient* boundary:  $a[P]$
- ambients are named by single *names* (denoted  $a$ )
- an ambient can contain other ambients:  
$$a[a_1[\dots] \mid a_2[\dots] \mid P]$$
- ambients form a *hierarchy tree*
- syntactically,  $a[P]$  is also a process

# Mobile Ambients I

- processes ( $P$ ) can run in *parallel*:  $P_1 \mid P_2$
- process can run inside an *ambient* boundary:  $a[P]$
- ambients are named by single *names* (denoted  $a$ )
- an ambient can contain other ambients:  
$$a[a_1[\dots] \mid a_2[\dots] \mid P]$$
- ambients form a *hierarchy tree*
- syntactically,  $a[P]$  is also a process

# Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle . P$  - send  $a$  and continue as  $P$
  - $(x) . P$  - receive a value and continue like  $P$  with received value substituted for  $x$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle . P$  - send  $a$  and continue as  $P$
  - $(x) . P$  - receive a value and continue like  $P$  with received value substituted for  $x$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$   
 $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle.P$  - send  $a$  and continue as  $P$
  - $(x).P$  - receive a value and continue like  $P$  with received value substituted for  $x$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$   
 $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle.P$  - send  $a$  and continue as  $P$
  - $(x).P$  - receive a value and continue like  $P$  with received value substituted for  $x$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle . P$  - send  $a$  and continue as  $P$
  - $(x) . P$  - receive a value and continue like  $P$  with received value substituted for  $x$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle . P$  - send  $a$  and continue as  $P$
  - $(x) . P$  - receive a value and continue like  $P$  with received value substituted for  $x$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle . P$  - send  $a$  and continue as  $P$
  - $(x) . P$  - receive a value and continue like  $P$  with received value substituted for  $x$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle . P$  - send  $a$  and continue as  $P$
  - $(x) . P$  - receive a value and continue like  $P$  with received value substituted for  $x$
  - $\langle a \rangle . P \mid (x) . Q \rightarrow P \mid Q\{x \mapsto a\}$

## Mobile Ambients II

- processes can change the hierarchy tree by executing *capabilities*:
  - in  $a$  - move the surrounding ambient into  $a$
  - out  $a$  - move the surrounding ambient out of  $a$
  - open  $a$  - open a sibling ambient named  $a$
- processes in the same ambient can *communicate*:
  - a name or sequence of capabilities can be sent
  - $\langle a \rangle . P$  - send  $a$  and continue as  $P$
  - $(x) . P$  - receive a value and continue like  $P$  with received value substituted for  $x$
  - $\langle a \rangle . P \mid (x) . Q \rightarrow P \mid Q\{x \mapsto a\}$

## Outline

- 1 Introduction to Process Calculi
  - Motivation, history and usage
  - Mobile Ambients
- 2 The Poly\* System
  - Meta\*: syntax and description of reduction rules
  - Poly\*: shape predicates and types for Meta\*
- 3 Usage of the Poly\* System
  - Poly\* as a type system for Mobile Ambients
  - Poly\* and a flow analysis

# What is Meta\*?

- Meta\* [MW04] 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
    - ... by providing descriptions of reduction rules
  - So, Meta\* is
- 
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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
    - ... by providing descriptions of reduction rules
  - So, Meta\* is
- 
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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
  - ... by providing descriptions of reduction rules
- So, Meta\* is
  
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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
  - ... by providing descriptions of reduction rules
- So, Meta\* is
  
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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  
...by providing descriptions of reduction rules
- So, Meta\* is
  
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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  
...by providing descriptions of reduction rules
- So, Meta\* is ...
  - A generic syntax of process terms
  - a generic way to specify a particular semantics
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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  
... by providing descriptions of reduction rules
- So, Meta\* is
  - 1 A generic syntax of process terms, that provides
  - 2 a generic way to specify a particular semantics
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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  
... by providing descriptions of reduction rules
- So, Meta\* is
  - 1 A generic syntax of process terms, that provides
  - 2 a generic way to specify a particular semantics
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

# What is Meta\*?

- Meta\* [MW04] 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  
... by providing descriptions of reduction rules
- So, Meta\* is
  - 1 A generic syntax of process terms, that provides
  - 2 a generic way to specify a particular semantics
- Thus, Meta\* can be **instantiated** to a specific process calculus (the  $\pi$ -calculus, Mobile Ambients, and many others)

## Example of a Meta\* instantiation

... to Mobile Ambients

$$\mathcal{A} = \left\{ \begin{array}{l} \mathbf{active}\{ P \mathbf{in} a[P] \}, \\ \mathbf{reduce}\{ a[\mathbf{in} b.P \mid Q] \mid b[R] \hookrightarrow b[a[P \mid Q] \mid R] \}, \\ \mathbf{reduce}\{ a[b[\mathbf{out} a.P \mid Q] \mid R] \hookrightarrow a[R] \mid b[P \mid Q] \}, \\ \mathbf{reduce}\{ \mathbf{open} a.P \mid a[R] \hookrightarrow P \mid R \} \\ \mathbf{reduce}\{ \langle M \rangle.P \mid (x).Q \hookrightarrow P \mid \{x := M\}Q \} \end{array} \right\}$$

- From this descriptions of reduction rules, a reduction relation  $\hookrightarrow^{\mathcal{A}}$  is automatically inferred.

## Outline

- 1 Introduction to Process Calculi
  - Motivation, history and usage
  - Mobile Ambients
- 2 The Poly\* System
  - Meta\*: syntax and description of reduction rules
  - Poly\*: shape predicates and types for Meta\*
- 3 Usage of the Poly\* System
  - Poly\* as a type system for Mobile Ambients
  - Poly\* and a flow analysis

# What is Poly\*?

- Poly\* [MW04] is a **type system** for every process calculus that can be described in Meta\*
- Type systems (not only) in process calculi
  - are used to describe various properties of (process) terms
  - provide a **set of types** that represent those properties
  - provide a **type judgment relation** ( $\vdash P : T$ )
  - grant that a type is preserved under reductions (**subject reduction**)
- Usual properties in process calculi:
  - a term is *well-formed*
  - communication in a term is *well-formed*
  - the ambient 'a' will never move

# What is Poly\*?

- Poly\* [MW04] is a **type system** for every process calculus that can be described in Meta\*
- Type systems (not only) in process calculi
  - are used to describe various properties of (process) terms
  - provide a **set of types** that represent those properties
  - provide a **type judgment relation** ( $\vdash P : T$ ), which says that the process  $P$  has the type (property)  $T$
  - grant that a type is preserved under reductions (**subject reduction**)
- Usual properties in process calculi:
  - a term is *well-formed*
  - communication in a term is *well-formed*
  - the ambient 'a' will never move

# What is Poly\*?

- Poly\* [MW04] is a **type system** for every process calculus that can be described in Meta\*
- Type systems (not only) in process calculi
  - are used to describe various properties of (process) terms
  - provide a **set of types** that represent those properties
  - provide a **type judgment relation** ( $\vdash P : T$ ), which says that the process  $P$  has the type (property)  $T$
  - grant that a type is preserved under reductions (**subject reduction**)
- Usual properties in process calculi:
  - a term is *well-formed*
  - communication in a term is *well-formed*
  - the ambient 'a' will never move

# What is Poly\*?

- Poly\* [MW04] is a **type system** for every process calculus that can be described in Meta\*
- Type systems (not only) in process calculi
  - are used to describe various properties of (process) terms
  - provide a **set of types** that represent those properties
  - provide a **type judgment relation** ( $\vdash P : T$ ), which says that the process  $P$  has the type (property)  $T$
  - grant that a type is preserved under reductions (**subject reduction**)
- Usual properties in process calculi:
  - a term is *well-formed*
  - communication in a term is *well-formed*
  - the ambient 'a' will never move

# What is Poly\*?

- Poly\* [MW04] is a **type system** for every process calculus that can be described in Meta\*
- Type systems (not only) in process calculi
  - are used to describe various properties of (process) terms
  - provide a **set of types** that represent those properties
  - provide a **type judgment relation** ( $\vdash P : T$ ), which says that the process  $P$  has the type (property)  $T$
  - grant that a type is preserved under reductions (**subject reduction**)
- Usual properties in process calculi:
  - a term is *well-formed*
  - communication in a term is *well-formed*
  - the ambient 'a' will never move

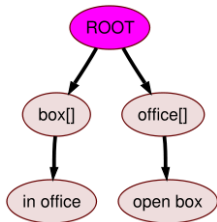
# What is Poly\*?

- Poly\* [MW04] is a **type system** for every process calculus that can be described in Meta\*
- Type systems (not only) in process calculi
  - are used to describe various properties of (process) terms
  - provide a **set of types** that represent those properties
  - provide a **type judgment relation** ( $\vdash P : T$ ), which says that the process  $P$  has the type (property)  $T$
  - grant that a type is preserved under reductions (**subject reduction**)
- Usual properties in process calculi:
  - a term is *well-formed*
  - communication in a term is *well-formed*
  - the ambient 'a' will never move

# What is Poly\*?

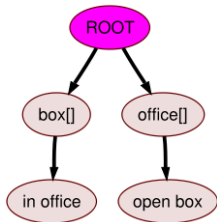
- Poly\* [MW04] is a **type system** for every process calculus that can be described in Meta\*
- Type systems (not only) in process calculi
  - are used to describe various properties of (process) terms
  - provide a **set of types** that represent those properties
  - provide a **type judgment relation** ( $\vdash P : T$ ), which says that the process  $P$  has the type (property)  $T$
  - grant that a type is preserved under reductions (**subject reduction**)
- Usual properties in process calculi:
  - a term is *well-formed*
  - communication in a term is *well-formed*
  - the ambient 'a' will never move

# Shape predicates and types



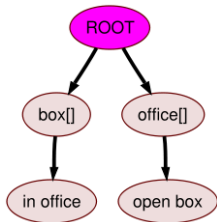
- A **shape predicate** is a graph that looks like a term syntax tree.
- A term **matches** a shape predicate if its syntax tree can be bent into the shape of a shape predicate.
- A shape predicate that is closed under the given reduction rules is called **type**.
- A type represents all possible computation futures of the source term smashed together in one place.

# Shape predicates and types



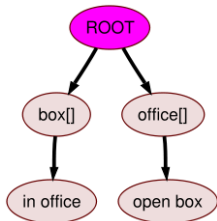
- A **shape predicate** is a graph that looks like a term syntax tree.
- A term **matches** a shape predicate if its syntax tree can be bent into the shape of a shape predicate.
- A shape predicate that is closed under the given reduction rules is called **type**.
- A type represents all possible computation futures of the source term smashed together in one place.

# Shape predicates and types



- A **shape predicate** 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 a 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.

# Shape predicates and types



- A **shape predicate** 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 a 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.

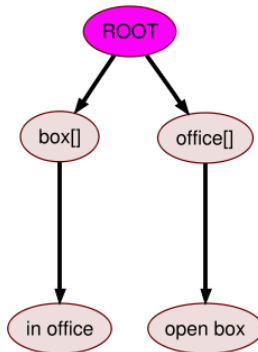
# Poly\* types

- types have to be closed under reduction rules:

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

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

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



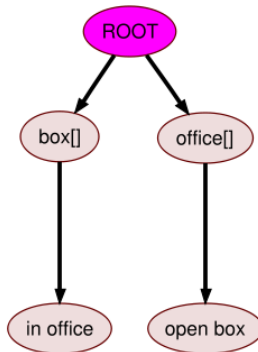
# Poly\* types

- types have to be closed under reduction rules:

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

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

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



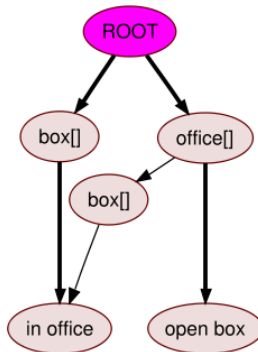
## Poly\* types

- types have to be closed under reduction rules:

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

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

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



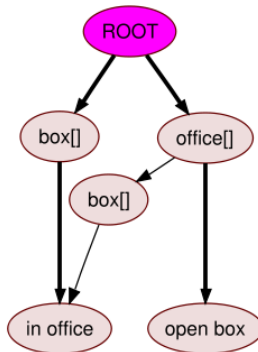
# Poly\* types

- types have to be closed under reduction rules:

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

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

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



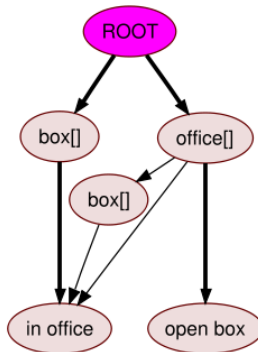
## Poly\* types

- types have to be closed under reduction rules:

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

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

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



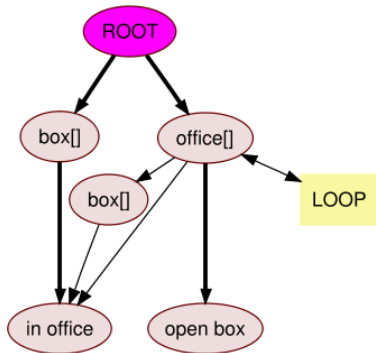
# Poly\* types

- types have to be **closed** under reduction rules:

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

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

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



## Outline

- 1 Introduction to Process Calculi
  - Motivation, history and usage
  - Mobile Ambients
- 2 The Poly\* System
  - Meta\*: syntax and description of reduction rules
  - Poly\*: shape predicates and types for Meta\*
- 3 Usage of the Poly\* System
  - Poly\* as a type system for Mobile Ambients
  - Poly\* and a flow analysis

# Poly\* and the seminal Mobile Ambients type system

- There are many type systems for Mobile Ambients
- The seminal one [CG99] is by Cardeli and Gordon. It's intended to allow only well-formed
  - terms; not like 'in (out a).0', '(in a) [P]'
  - communication; not like '(x).in x.0 | <out a>.0'
- It has type judgments of the form  $E \vdash P : T$ .
- In Poly\*, we can construct the shape predicate  $\Pi_{E,T}^P$  such that:

$$E \vdash P : T \quad \Leftrightarrow \quad \vdash \ulcorner P \urcorner : \Pi_{E,T}^P \quad (\ulcorner P \urcorner \text{ is } P \text{ in Meta*})$$

# Poly\* and the seminal Mobile Ambients type system

- There are many type systems for Mobile Ambients
- The seminal one [CG99] is by Cardeli and Gordon. It's intended to allow only well-formed
  - terms; not like 'in (out a).0', '(in a) [P]'.
  - communication; not like '(x).in x.0 | <out a>.0'.
- It has type judgments of the form  $E \vdash P : T$ .
- In Poly\*, we can construct the shape predicate  $\Pi_{E,T}^P$  such that:

$$E \vdash P : T \quad \Leftrightarrow \quad \vdash \ulcorner P \urcorner : \Pi_{E,T}^P \quad (\ulcorner P \urcorner \text{ is } P \text{ in Meta*})$$

# Poly\* and the seminal Mobile Ambients type system

- There are many type systems for Mobile Ambients
- The seminal one [CG99] is by Cardeli and Gordon. It's intended to allow only well-formed
  - terms; not like 'in (out a).0', '(in a) [P]'.
  - communication; not like '(x) .in x.0 | <out a>.0'.
- It has type judgments of the form  $E \vdash P : T$ .
- In Poly\*, we can construct the shape predicate  $\Pi_{E,T}^P$  such that:

$$E \vdash P : T \quad \Leftrightarrow \quad \vdash \ulcorner P \urcorner : \Pi_{E,T}^P \quad (\ulcorner P \urcorner \text{ is } P \text{ in Meta*})$$

# Poly\* and the seminal Mobile Ambients type system

- There are many type systems for Mobile Ambients
- The seminal one [CG99] is by Cardeli and Gordon. It's intended to allow only well-formed
  - terms; not like 'in (out a).0', '(in a) [P]'.
  - communication; not like '(x).in x.0 | <out a>.0'.
- It has type judgments of the form  $E \vdash P : T$ .
- In Poly\*, we can construct the shape predicate  $\Pi_{E,T}^P$  such that:

$$E \vdash P : T \quad \Leftrightarrow \quad \vdash \ulcorner P \urcorner : \Pi_{E,T}^P \quad (\ulcorner P \urcorner \text{ is } P \text{ in Meta*})$$

# Poly\* as a type system for Mobile Ambients

- Poly\* can also grant well-formed communication and well-formed ambient terms by its own means.
- Poly\* **can** grant properties like:
  - an ambient  $a$  will never be opened/moved
  - an ambient  $a$  will never contain an ambient  $b$  as a child
  - an ambient  $a$  does not perform communication
  - an ambient  $a$  can contain a child ambient  $b$  only when  $a$  is a child of an ambient  $c$  (**spatial polymorphisms**)
- Poly\* **can not** easily grant properties like:
  - an ambient  $a$  has exactly two child ambients  $b$
  - an ambient  $a$  has two or more child ambients  $b$
  - an ambient  $a$  has always a child ambient  $b$

# Poly\* as a type system for Mobile Ambients

- Poly\* can also grant well-formed communication and well-formed ambient terms by its own means.
- Poly\* **can** grant properties like:
  - an ambient  $a$  will never be opened/moved
  - an ambient  $a$  will never contain an ambient  $b$  as a child
  - an ambient  $a$  does not perform communication
  - an ambient  $a$  can contain a child ambient  $b$  only when  $a$  is a child of an ambient  $c$  (**spatial polymorphisms**)
- Poly\* **can not** easily grant properties like:
  - an ambient  $a$  has exactly two child ambients  $b$
  - an ambient  $a$  has two or more child ambients  $b$
  - an ambient  $a$  has always a child ambient  $b$

# Poly\* as a type system for Mobile Ambients

- Poly\* can also grant well-formed communication and well-formed ambient terms by its own means.
- Poly\* **can** grant properties like:
  - an ambient  $a$  will never be opened/moved
  - an ambient  $a$  will never contain an ambient  $b$  as a child
  - an ambient  $a$  does not perform communication
  - an ambient  $a$  can contain a child ambient  $b$  only when  $a$  is a child of an ambient  $c$  (**spatial polymorphisms**)
- Poly\* **can not** easily grant properties like:
  - an ambient  $a$  has exactly two child ambients  $b$
  - an ambient  $a$  has two or more child ambients  $b$
  - an ambient  $a$  has always a child ambient  $b$

## Outline

- 1 Introduction to Process Calculi
  - Motivation, history and usage
  - Mobile Ambients
- 2 The Poly\* System
  - Meta\*: syntax and description of reduction rules
  - Poly\*: shape predicates and types for Meta\*
- 3 Usage of the Poly\* System
  - Poly\* as a type system for Mobile Ambients
  - Poly\* and a flow analysis

# 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** [RPS<sup>+</sup>04] of Nielson et. al. [NNPR07]
- BioAmbients [RPS<sup>+</sup>04] 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** [RPS<sup>+</sup>04] of Nielson et. al. [NNPR07]
- BioAmbients [RPS<sup>+</sup>04] is a variant of Mobile Ambients intended to model biological systems.

# Poly\* and a flow analysis

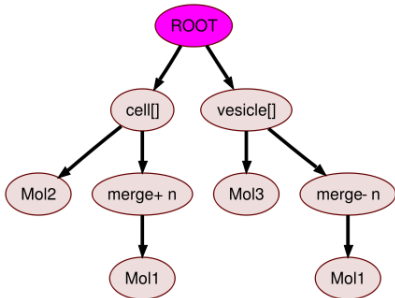
- Poly\* can be used for a **flow analysis** of systems described in Meta\*.
- A flow analysis provides a way to determine **over approximation** of all possible states of the modeled system.
- Poly\* provides, for example, comparable results with a flow analysis for **BioAmbients** [RPS<sup>+</sup>04] of Nielson et. al. [NNPR07]
- BioAmbients [RPS<sup>+</sup>04] 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** [RPS<sup>+</sup>04] of Nielson et. al. [NNPR07]
- BioAmbients [RPS<sup>+</sup>04] is a variant of Mobile Ambients intended to model biological systems.

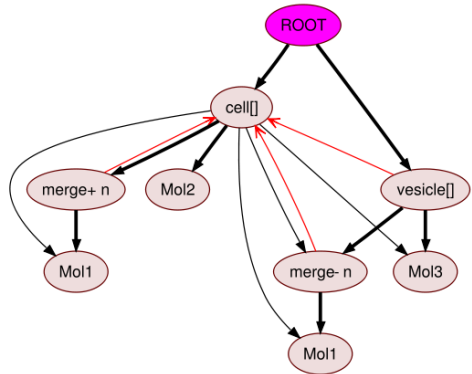
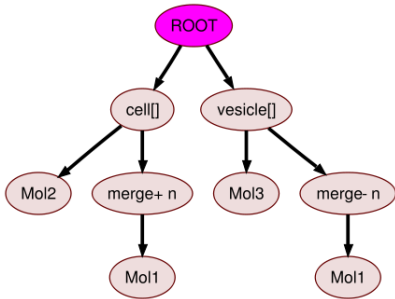
# Example of a flow analysis in Poly\*

## 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.



L. Cardelli and A. D. Gordon.

Mobile ambients.

In *Proc. FoSSaCS '98*, vol. 1378 of *LNCS*, pp. 140–155.  
Springer-Verlag, 1998.



L. Cardelli and A. D. Gordon.

Types for mobile ambients.

In *Conf. Rec. POPL '99: 26th ACM Symp. Princ. of Prog. Langs.*, pp. 79–92, 1999.



H. Makholm and J. B. Wells.

Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close.

Technical Report HW-MACS-TR-0022, Heriot-Watt Univ.,  
School of Math. & Comput. Sci., Nov. 2004.

A shorter successor is [MW05].



H. Makholm and J. B. Wells.

Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close.

In *Programming Languages & Systems, 14th European Symp. Programming*, vol. 3444 of *LNCS*, pp. 389–407. Springer-Verlag, 2005.

A more detailed predecessor is [MW04].



F. Nielson, H. R. Nielson, C. Priami, and D. Rosa.

Control flow analysis for bioambients.

*ENTCS*, 180(3):65–79, 2007.

A preliminary version appeared at Bio-CONCUR 2003.



J. Parrow.

An introduction to the  $\pi$ -calculus.

In *Handbook of Process Algebra*, pp. 479–543.

North-Holland, Amsterdam, 2001.



A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and  
E. Shapiro.

Bioambients: An abstraction for biological compartments.  
*Theoret. Comput. Sci.*, 325(1):141–167, Sept. 2004.