



Introduction to Process Calculi and to the Poly* System

Jan Jakubův

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

ULTRA Group, MACS
Heriot-Watt University

September 11, 2008



Outline

Why Process Calculi?

Motivation of Process Calculi

Existing Process Calculi

The π -calculus and Mobile Ambients

The π -calculus

Mobile Ambients

Type Systems for Process Calculi

The Poly* System

Overview of the Poly* System

Properties and Usage of Poly*



Outline

Why Process Calculi?

Motivation of Process Calculi

Existing Process Calculi

The π -calculus and Mobile Ambients

The π -calculus

Mobile Ambients

Type Systems for Process Calculi

The Poly* System

Overview of the Poly* System

Properties and Usage of Poly*



Outline

Why Process Calculi?

Motivation of Process Calculi

Existing Process Calculi

The π -calculus and Mobile Ambients

The π -calculus

Mobile Ambients

Type Systems for Process Calculi

The Poly \star System

Overview of the Poly \star System

Properties and Usage of Poly \star



Outline

Why Process Calculi?

Motivation of Process Calculi

Existing Process Calculi

The π -calculus and Mobile Ambients

The π -calculus

Mobile Ambients

Type Systems for Process Calculi

The Poly* System

Overview of the Poly* System

Properties and Usage of Poly*



What are concurrent systems?

- *Concurrent systems are systems where several interacting units engage in activity at the same time.*
- there may be several possible next states to which a system can evolve from the current state
- a system can interact with its environment
- there is not necessarily any final state
- Key concepts: interaction, parallelism, non-determinism.



What are concurrent systems?

- *Concurrent systems are systems where several interacting units engage in activity at the same time.*
- there may be several possible next states to which a system can evolve from the current state
- a system can interact with its environment
- there is not necessarily any final state
- Key concepts: interaction, parallelism, non-determinism.



What are concurrent systems?

- *Concurrent systems are systems where several interacting units engage in activity at the same time.*
- there may be several possible next states to which a system can evolve from the current state
- a system can interact with its environment
- there is not necessarily any final state
- Key concepts: interaction, parallelism, non-determinism.



What are concurrent systems?

- *Concurrent systems are systems where several interacting units engage in activity at the same time.*
- there may be several possible next states to which a system can evolve from the current state
- a system can interact with its environment
- there is not necessarily any final state
- Key concepts: interaction, parallelism, non-determinism.



What are concurrent systems?

- *Concurrent systems are systems where several interacting units engage in activity at the same time.*
- there may be several possible next states to which a system can evolve from the current state
- a system can interact with its environment
- there is not necessarily any final state
- Key concepts: interaction, parallelism, non-determinism.



What are concurrent systems?

- *Concurrent systems are systems where several **interacting units** engage in activity at the same time.*
- there may be several possible next states to which a system can evolve from the current state
- a system can interact with its environment
- there is not necessarily any final state
- Key concepts: **interaction**, parallelism, non-determinism.



What are concurrent systems?

- *Concurrent systems are systems where several interacting units engage in activity **at the same time**.*
- there may be several possible next states to which a system can evolve from the current state
- a system can interact with its environment
- there is not necessarily any final state
- Key concepts: interaction, **parallelism**, non-determinism.



What are concurrent systems?

- *Concurrent systems are systems where several interacting units engage in activity at the same time.*
- there may be **several possible next states** to which a system can evolve from the current state
- a system can **interact with its environment**
- there is not necessarily any final state
- Key concepts: interaction, parallelism, **non-determinism**.



What are concurrent systems?

- *Concurrent systems are systems where several **interacting units** engage in activity **at the same time**.*
- there may be **several possible next states** to which a system can evolve from the current state
- a system can **interact with its environment**
- there is not necessarily any final state
- Key concepts: interaction, parallelism, non-determinism.



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



What are process calculi?

- *Process calculi are intended to model concurrent systems.*
- interacting units are called **processes** (P, Q, R)
- processes are build from atoms called **names** (a, b, \dots)
- processes are **composed** using process constructors to form more complex processes:
 - sequential composition (prefixing with an action A): $A.P$
 - parallel composition: $P \mid Q$
- **reduction semantics** is given by a rewrite system:
 - a binary reduction relation \rightarrow on processes is defined



Usage of existing process calculi

- *Process calculi are used to emulation of and to formal reasoning about various concurrent systems.*
 - usage inside CS: processes in operating systems, services in networks
 - usage outside CS: molecular and other systems in biology and chemistry, work-flow in management.
- Variants of process calculi addressed particular aspects of concurrent systems like: data treatment, time treatment, probability, and mobility.
- From calculi concerning mobility the most famous are the π -calculus [1] and Mobile Ambients [2].

Usage of existing process calculi

- *Process calculi are used to emulation of and to formal reasoning about various concurrent systems.*
 - usage inside CS: processes in operating systems, services in networks
 - usage outside CS: molecular and other systems in biology and chemistry, work-flow in management.
- Variants of process calculi addressed particular aspects of concurrent systems like: data treatment, time treatment, probability, and mobility.
- From calculi concerning mobility the most famous are the π -calculus [1] and Mobile Ambients [2].



Usage of existing process calculi

- *Process calculi are used to emulation of and to formal reasoning about various concurrent systems.*
 - usage inside CS: processes in operating systems, services in networks
 - usage outside CS: molecular and other systems in biology and chemistry, work-flow in management.
- Variants of process calculi addressed particular aspects of concurrent systems like: data treatment, time treatment, probability, and mobility.
- From calculi concerning mobility the most famous are the π -calculus [1] and Mobile Ambients [2].



Usage of existing process calculi

- *Process calculi are used to emulation of and to formal reasoning about various concurrent systems.*
 - usage inside CS: processes in operating systems, services in networks
 - usage outside CS: molecular and other systems in biology and chemistry, work-flow in management.
- Variants of process calculi addressed particular aspects of concurrent systems like: data treatment, time treatment, probability, and mobility.
- From calculi concerning mobility the most famous are the π -calculus [1] and Mobile Ambients [2].



Usage of existing process calculi

- *Process calculi are used to emulation of and to formal reasoning about various concurrent systems.*
 - usage inside CS: processes in operating systems, services in networks
 - usage outside CS: molecular and other systems in biology and chemistry, work-flow in management.
- Variants of process calculi addressed particular aspects of concurrent systems like: data treatment, time treatment, probability, and **mobility**.
- From calculi concerning mobility the most famous are the π -calculus [1] and Mobile Ambients [2].



Outline

Why Process Calculi?

Motivation of Process Calculi

Existing Process Calculi

The π -calculus and Mobile Ambients

The π -calculus

Mobile Ambients

Type Systems for Process Calculi

The Poly* System

Overview of the Poly* System

Properties and Usage of Poly*



Overview of the π -calculus

- **interaction** is abstracted as communication over named channels (links)
- objects of communication and channel identifiers are **both** modeled by (atomic) names
- **mobility** is abstracted as exchange of channel names
= **link passing**



Overview of the π -calculus

- **interaction** is abstracted as communication over named channels (links)
- objects of communication and channel identifiers are **both** modeled by (atomic) names
- **mobility** is abstracted as exchange of channel names
= **link passing**

Overview of the π -calculus

- **interaction** is abstracted as communication over named channels (links)
- objects of communication and channel identifiers are **both** modeled by (atomic) names
- **mobility** is abstracted as exchange of channel names
= **link passing**



Overview of the π -calculus

- **interaction** is abstracted as communication over named channels (links)
- objects of communication and channel identifiers are **both** modeled by (atomic) names
- **mobility** is abstracted as exchange of channel names
= **link passing**



Syntax and Semantics of the π -calculus

- 0 - inactive (null) process
- $c \langle a \rangle . P$ - sending process
send the name a over the channel c and continue as P
- $c(x) . P$ - receiving process
receive the name on the channel c and continue as P with the received name substituted for x
- $!P$ - replicated process
act as infinitely many P 's in parallel



Syntax and Semantics of the π -calculus

- 0 - inactive (null) process
- $c\langle a \rangle.P$ - sending process
send the name a over the channel c and continue as P
- $c(x).P$ - receiving process
receive the name on the channel c and continue as P with the received name substituted for x
- $!P$ - replicated process
act as infinitely many P 's in parallel



Syntax and Semantics of the π -calculus

- 0 - inactive (null) process
- $c\langle a \rangle.P$ - sending process
send the name a over the channel c and continue as P
- $c(x).P$ - receiving process
receive the name on the channel c and continue as P with the received name substituted for x
- $!P$ - replicated process
act as infinitely many P 's in parallel



Syntax and Semantics of the π -calculus

- 0 - inactive (null) process
- $c\langle a \rangle.P$ - sending process
send the name a over the channel c and continue as P
- $c(x).P$ - receiving process
receive the name on the channel c and continue as P with the received name substituted for x
- $!P$ - replicated process
act as infinitely many P 's in parallel



Reduction Semantics

- communication rule:

$$c\langle a \rangle.P \mid c(x).Q \mid R \rightarrow P \mid (Q\{x \mapsto a\}) \mid R$$

- replication: $!P \mid R \rightarrow P \mid !P \mid R$
- \mid is commutative and associative with unit 0



Reduction Semantics

- communication rule:

$$c\langle a \rangle.P \mid c(x).Q \mid R \rightarrow P \mid (Q\{x \mapsto a\}) \mid R$$

- replication: $!P \mid R \rightarrow P \mid !P \mid R$
- \mid is commutative and associative with unit 0



Reduction Semantics

- communication rule:

$$c\langle a \rangle.P \mid c(x).Q \mid R \rightarrow P \mid (Q\{x \mapsto a\}) \mid R$$

- replication: $!P \mid R \rightarrow P \mid !P \mid R$
- \mid is commutative and associative with unit 0



Reduction Semantics

- communication rule:

$$c\langle a \rangle.P \mid c(x).Q \mid R \rightarrow P \mid (Q\{x \mapsto a\}) \mid R$$

- replication: $!P \mid R \rightarrow P \mid !P \mid R$

- \mid is commutative and associative with unit 0



Reduction Semantics

- communication rule:

$$c\langle a \rangle.P \mid c(x).Q \mid R \rightarrow P \mid (Q\{x \mapsto a\}) \mid R$$

- replication: $!P \mid R \rightarrow P \mid !P \mid R$

- \mid is commutative and associative with unit 0



Reduction Semantics

- communication rule:

$$c\langle a \rangle.P \mid c(x).Q \mid R \rightarrow P \mid (Q\{x \mapsto a\}) \mid R$$

- replication: $!P \mid R \rightarrow P \mid !P \mid R$

- \mid is commutative and associative with unit 0

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid 0 \mid c\langle d \rangle.0 \mid !p(z).0$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a π -calculus process (*server* | *user* | *printer*):

$$!s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid s\langle p \rangle.0 \mid s(x).x\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid p\langle d \rangle.0 \mid p(z).c\langle z \rangle.0 \mid !p(z).c\langle z \rangle.0$$

$$\rightarrow !s\langle p \rangle.0 \mid c\langle d \rangle.0 \mid !p(z).0$$



Overview of Mobile Ambients

- processes are placed in a tree hierarchy of bounded locations (**ambients**)
- ambients are referenced by names
- processes are allowed to change the locations hierarchy by executing **capabilities**
- processes can send/receive messages to/from **all** processes at the same location (anonymous messages)
- message can be either a single name or a sequence of capabilities
- **mobility** is abstracted as a change of a locations hierarchy



Overview of Mobile Ambients

- processes are placed in a tree hierarchy of bounded locations (**ambients**)
- ambients are referenced by names
- processes are allowed to change the locations hierarchy by executing **capabilities**
- processes can send/receive messages to/from **all** processes at the same location (anonymous messages)
- message can be either a single name or a sequence of capabilities
- **mobility** is abstracted as a change of a locations hierarchy



Overview of Mobile Ambients

- processes are placed in a tree hierarchy of bounded locations (**ambients**)
- ambients are referenced by names
- processes are allowed to change the locations hierarchy by executing **capabilities**
- processes can send/receive messages to/from **all** processes at the same location (anonymous messages)
- message can be either a single name or a sequence of capabilities
- **mobility** is abstracted as a change of a locations hierarchy



Overview of Mobile Ambients

- processes are placed in a tree hierarchy of bounded locations (**ambients**)
- ambients are referenced by names
- processes are allowed to change the locations hierarchy by executing **capabilities**
- processes can send/receive messages to/from **all** processes at the same location (anonymous messages)
- message can be either a single name or a sequence of capabilities
- **mobility** is abstracted as a change of a locations hierarchy



Overview of Mobile Ambients

- processes are placed in a tree hierarchy of bounded locations (**ambients**)
- ambients are referenced by names
- processes are allowed to change the locations hierarchy by executing **capabilities**
- processes can send/receive messages to/from **all** processes at the same location (anonymous messages)
- message can be either a single name or a sequence of capabilities
- **mobility** is abstracted as a change of a locations hierarchy



Overview of Mobile Ambients

- processes are placed in a tree hierarchy of bounded locations (**ambients**)
- ambients are referenced by names
- processes are allowed to change the locations hierarchy by executing **capabilities**
- processes can send/receive messages to/from **all** processes at the same location (anonymous messages)
- message can be either a single name or a sequence of capabilities
- **mobility** is abstracted as a change of a locations hierarchy



Syntax and Semantics of Mobile Ambients

- $a[P]$ - the process containing the process P running inside the ambient a
- $A.P$ - the process that executes the action A (a communication action or a capability) and continues as P

Executing a capability can instruct an ambient a to

1. to move inside a child ambient b (**in b**)
ex.: $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
2. to move out of the parent ambient b (**out b**)
3. dissolve the boundary of a child ambient b (**open b**)
ex.: $\text{open } b.P \mid b[Q] \rightarrow P \mid Q$

Syntax and Semantics of Mobile Ambients

- $a[P]$ - the process containing the process P running inside the ambient a
- $A.P$ - the process that executes the action A (a communication action or a capability) and continues as P

Executing a capability can instruct an ambient a to

1. to move inside a child ambient b (**in b**)
ex.: $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
2. to move out of the parent ambient b (**out b**)
3. dissolve the boundary of a child ambient b (**open b**)
ex.: $\text{open } b.P \mid b[Q] \rightarrow P \mid Q$

Syntax and Semantics of Mobile Ambients

- $a[P]$ - the process containing the process P running inside the ambient a
- $A.P$ - the process that executes the action A (a communication action or a capability) and continues as P

Executing a capability can instruct an ambient a to

1. to move inside a child ambient b (**in b**)
 ex.: $a[\mathbf{in\ } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
2. to move out of the parent ambient b (**out b**)
3. dissolve the boundary of a child ambient b (**open b**)
 ex.: $\mathbf{open\ } b.P \mid b[Q] \rightarrow P \mid Q$

Syntax and Semantics of Mobile Ambients

- $a[P]$ - the process containing the process P running inside the ambient a
- $A.P$ - the process that executes the action A (a communication action or a capability) and continues as P

Executing a capability can instruct an ambient a to

1. to move inside a child ambient b (**in b**)
 ex.: $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
2. to move out of the parent ambient b (**out b**)
3. dissolve the boundary of a child ambient b (**open b**)
 ex.: $\text{open } b.P \mid b[Q] \rightarrow P \mid Q$

Syntax and Semantics of Mobile Ambients

- $a[P]$ - the process containing the process P running inside the ambient a
- $A.P$ - the process that executes the action A (a communication action or a capability) and continues as P

Executing a capability can instruct an ambient a to

1. to move inside a child ambient b (**in b**)
 ex.: $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
2. to move out of the parent ambient b (**out b**)
3. dissolve the boundary of a child ambient b (**open b**)
 ex.: $\text{open } b.P \mid b[Q] \rightarrow P \mid Q$

Syntax and Semantics of Mobile Ambients

- $a[P]$ - the process containing the process P running inside the ambient a
- $A.P$ - the process that executes the action A (a communication action or a capability) and continues as P

Executing a capability can instruct an ambient a to

1. to move inside a child ambient b (**in b**)
 ex.: $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
2. to move out of the parent ambient b (**out b**)
3. dissolve the boundary of a child ambient b (**open b**)
 ex.: $\text{open } b.P \mid b[Q] \rightarrow P \mid Q$

Syntax and Semantics of Mobile Ambients

- $a[P]$ - the process containing the process P running inside the ambient a
- $A.P$ - the process that executes the action A (a communication action or a capability) and continues as P

Executing a capability can instruct an ambient a to

1. to move inside a child ambient b (**in b**)
 ex.: $a[\text{in } b.P] \mid b[Q] \rightarrow b[a[P] \mid Q]$
2. to move out of the parent ambient b (**out b**)
3. dissolve the boundary of a child ambient b (**open b**)
 ex.: $\text{open } b.P \mid b[Q] \rightarrow P \mid Q$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$!\langle \text{in } p \rangle . 0 \mid (x) . e[x.\langle d \rangle . 0] \mid p[!\text{open } e.(z) . c[\langle z \rangle]]$$

$$\Rightarrow !\langle \text{in } p \rangle . 0 \mid e[\text{in } p.\langle d \rangle . 0] \mid p[!\text{open } e.(z) . c[\langle z \rangle]]$$

$$\rightarrow !\langle \text{in } p \rangle . 0 \mid p[e[\langle d \rangle . 0] \mid !\text{open } e.(z) . c[\langle z \rangle]]$$

$$\Rightarrow !\langle \text{in } p \rangle . 0 \mid p[\langle d \rangle . 0 \mid (z) . c[\langle z \rangle] \mid !\text{open } e.(z) . c[\langle z \rangle]]$$

$$\rightarrow !\langle \text{in } p \rangle . 0 \mid p[c[\langle d \rangle] \mid !\text{open } e.(z) . c[\langle z \rangle]]$$



Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle]] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle]] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle]] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle]] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Example situation:

The *server* (s) provides connection to the *printer* (p). A *user* would like to print the document (d). The user has to connect to the server to obtain printer's address.

As a Mobile Ambients process (*server* | *user* | *printer*):

$$! \langle \text{in } p \rangle . 0 \mid (x) . e [x . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid e [\text{in } p . \langle d \rangle . 0] \mid p [! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [e [\langle d \rangle . 0] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\Rightarrow ! \langle \text{in } p \rangle . 0 \mid p [\langle d \rangle . 0 \mid (z) . c [\langle z \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

$$\rightarrow ! \langle \text{in } p \rangle . 0 \mid p [c [\langle d \rangle] \mid ! \text{open } e . (z) . c [\langle z \rangle]]$$

Type Systems for Process Calculi

- *Type Systems are used to describe various properties of processes.*
- provides **types** (T) representing the properties in question
- provides **type judgment relation** ($\vdash P : T$), which says that the process P has the type (property) T
- grants that types are preserved under reductions (**subject reduction**)
- usual described properties: “process is well-formed”, “communication in a process is well-formed”, “ambient a will never be opened”.

Type Systems for Process Calculi

- *Type Systems are used to describe various properties of processes.*
- provides **types** (T) representing the properties in question
- provides **type judgment relation** ($\vdash P : T$), which says that the process P has the type (property) T
- grants that types are preserved under reductions (**subject reduction**)
- usual described properties: “process is well-formed”, “communication in a process is well-formed”, “ambient a will never be opened”.



Type Systems for Process Calculi

- *Type Systems are used to describe various properties of processes.*
- provides **types** (T) representing the properties in question
- provides **type judgment relation** ($\vdash P : T$), which says that the process P has the type (property) T
- grants that types are preserved under reductions (**subject reduction**)
- usual described properties: “process is well-formed”, “communication in a process is well-formed”, “ambient a will never be opened”.



Type Systems for Process Calculi

- *Type Systems are used to describe various properties of processes.*
- provides **types** (T) representing the properties in question
- provides **type judgment relation** ($\vdash P : T$), which says that the process P has the type (property) T
- grants that types are preserved under reductions (**subject reduction**)
- usual described properties: “process is well-formed”, “communication in a process is well-formed”, “ambient a will never be opened”.



Type Systems for Process Calculi

- *Type Systems are used to describe various properties of processes.*
- provides **types** (T) representing the properties in question
- provides **type judgment relation** ($\vdash P : T$), which says that the process P has the type (property) T
- grants that types are preserved under reductions (**subject reduction**)
- usual described properties: “process is well-formed”, “communication in a process is well-formed”, “ambient a will never be opened”.



Outline

Why Process Calculi?

Motivation of Process Calculi

Existing Process Calculi

The π -calculus and Mobile Ambients

The π -calculus

Mobile Ambients

Type Systems for Process Calculi

The Poly★ System

Overview of the Poly★ System

Properties and Usage of Poly★



Overview of the Poly \star System

- The system [3] consists of two parts: Meta \star and Poly \star
- Meta \star
 - is general syntax of processes
 - provides the way to describe a particular semantics.
- Poly \star is a type system for every process calculus that can be described in Meta \star .

Overview of the Poly★ System

- The system [3] consists of two parts: Meta★ and Poly★
- **Meta★**
 - is general syntax of processes, and
 - provides the way to describe a particular semantics.
- **Poly★** is a type system for every process calculus that can be described in Meta★.



Overview of the Poly \star System

- The system [3] consists of two parts: Meta \star and Poly \star
- **Meta \star**
 - is general syntax of processes, and
 - provides the way to describe a particular semantics.
- **Poly \star** is a type system for every process calculus that can be described in Meta \star .



Overview of the Poly \star System

- The system [3] consists of two parts: Meta \star and Poly \star
- **Meta \star**
 - is general syntax of processes, and
 - provides the way to describe a particular semantics.
- **Poly \star** is a type system for every process calculus that can be described in Meta \star .

Meta★ provides

- common operators found in process calculi ‘|’, ‘.’, ‘!’
- general shape of actions that includes $c\langle a \rangle$, (a) , in a
- syntax to describe reduction rules, as the following:
 - $\text{reduce}(\tilde{c}\langle \tilde{a} \rangle . \tilde{P} \mid \tilde{c}(\tilde{x}) . \tilde{Q} \hookrightarrow \tilde{P} \mid \{\tilde{x} := \tilde{a}\} \tilde{Q})$
 - $\text{reduce}(\text{open } \tilde{a} . \tilde{P} \mid \tilde{a}[\tilde{R}] \hookrightarrow \tilde{P} \mid \tilde{Q})$
- reduction relation on Meta★ processes $\hookrightarrow_{\mathcal{R}}$ derived from the descriptions of reduction rules \mathcal{R}

Meta★ provides

- common operators found in process calculi ‘|’, ‘.’, ‘!’
- general shape of actions that includes $c\langle a \rangle$, (a) , in a
- syntax to describe reduction rules, as the following:
 - $\text{reduce}(\tilde{c}\langle \tilde{a} \rangle . \tilde{P} \mid \tilde{c}(\tilde{x}) . \tilde{Q} \hookrightarrow \tilde{P} \mid \{\tilde{x} := \tilde{a}\} \tilde{Q})$
 - $\text{reduce}(\text{open } \tilde{a} . \tilde{P} \mid \tilde{a}[\tilde{R}] \hookrightarrow \tilde{P} \mid \tilde{Q})$
- reduction relation on Meta★ processes $\hookrightarrow_{\mathcal{R}}$ derived from the descriptions of reduction rules \mathcal{R}



Meta* provides

- common operators found in process calculi ‘|’, ‘.’, ‘!’
- general shape of actions that includes $c\langle a \rangle$, (a) , in a
- syntax to describe reduction rules, as the following:
 - **reduce** $\{ \check{c}\langle \check{a} \rangle . \check{P} \mid \check{c}(\check{x}) . \check{Q} \hookrightarrow \check{P} \mid \{ \check{x} := \check{a} \} \check{Q} \}$
 - **reduce** $\{ \text{open } \check{a} . \check{P} \mid \check{a}[\check{R}] \hookrightarrow \check{P} \mid \check{Q} \}$
- reduction relation on Meta* processes $\hookrightarrow_{\mathcal{R}}$ derived from the descriptions of reduction rules \mathcal{R}



Meta* provides

- common operators found in process calculi ‘|’, ‘.’, ‘!’
- general shape of actions that includes $c\langle a \rangle$, (a) , in a
- syntax to describe reduction rules, as the following:
 - **reduce** $\{ \check{c}\langle \check{a} \rangle . \check{P} \mid \check{c}(\check{x}) . \check{Q} \hookrightarrow \check{P} \mid \{ \check{x} := \check{a} \} \check{Q} \}$
 - **reduce** $\{ \text{open } \check{a} . \check{P} \mid \check{a}[\check{R}] \hookrightarrow \check{P} \mid \check{Q} \}$
- reduction relation on Meta* processes $\hookrightarrow_{\mathcal{R}}$ derived from the descriptions of reduction rules \mathcal{R}



Meta* provides

- common operators found in process calculi ‘|’, ‘.’, ‘!’
- general shape of actions that includes $c\langle a \rangle$, (a) , in a
- syntax to describe reduction rules, as the following:
 - **reduce** $\{ \check{c}\langle \check{a} \rangle . \check{P} \mid \check{c}(\check{x}) . \check{Q} \hookrightarrow \check{P} \mid \{ \check{x} := \check{a} \} \check{Q} \}$
 - **reduce** $\{ \text{open } \check{a} . \check{P} \mid \check{a}[\check{R}] \hookrightarrow \check{P} \mid \check{Q} \}$
- reduction relation on Meta* processes $\hookrightarrow_{\mathcal{R}}$ derived from the descriptions of reduction rules \mathcal{R}

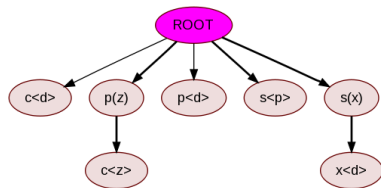


Meta \star provides

- common operators found in process calculi ‘|’, ‘.’, ‘!’
- general shape of actions that includes $c\langle a \rangle$, (a) , in a
- syntax to describe reduction rules, as the following:
 - **reduce** $\{ c\langle a \rangle.P \mid c(\tilde{x}).Q \hookrightarrow P \mid \{\tilde{x} := a\}Q \}$
 - **reduce** $\{ \text{open } a.P \mid a[R] \hookrightarrow P \mid Q \}$
- reduction relation on Meta \star processes $\hookrightarrow_{\mathcal{R}}$ derived from the descriptions of reduction rules \mathcal{R}

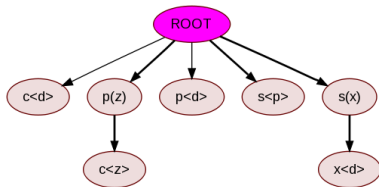
Poly* types

- looks like processes
- contains all possible future states smashed together in a single place
- sometimes describe states that can not be reached (**over approximation**)



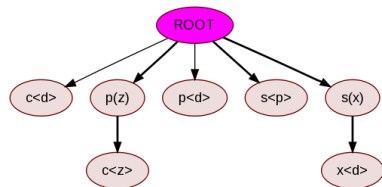
Poly* types

- looks like processes
- contains all possible future states smashed together in a single place
- sometimes describe states that can not be reached (over approximation)



Poly* types

- looks like processes
- contains all possible future states smashed together in a single place
- sometimes describe states that can not be reached (**over approximation**)





Poly*

- can be used to grant simple properties of the modeled system (like well-formed communication)
- can be used to grant that some states can never be reached
- can be used for **static analysis** of the modeled system
- has an **implemented** type inference algorithm



Poly*

- can be used to grant simple properties of the modeled system (like well-formed communication)
- can be used to grant that some states can never be reached
- can be used for **static analysis** of the modeled system
- has an **implemented** type inference algorithm



Poly*

- can be used to grant simple properties of the modeled system (like well-formed communication)
- can be used to grant that some states can never be reached
- can be used for **static analysis** of the modeled system
- has an **implemented** type inference algorithm

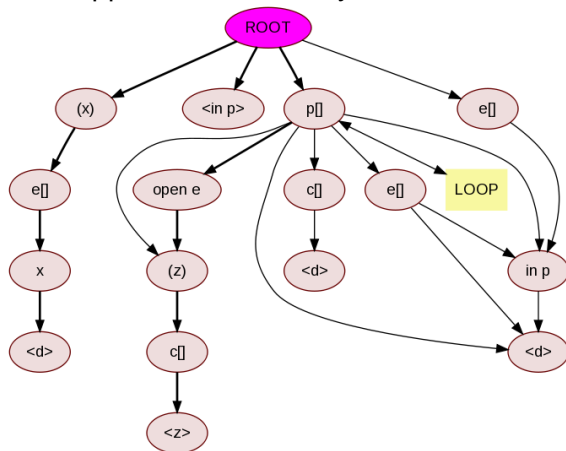


Poly*

- can be used to grant simple properties of the modeled system (like well-formed communication)
- can be used to grant that some states can never be reached
- can be used for **static analysis** of the modeled system
- has an **implemented** type inference algorithm



Over approximation in Poly*



Current work on Poly★

- comparison with existing type systems for Mobile Ambient:
In Poly★, we can construct the type $\pi_{E,T}^P$ such that:

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

- comparison with existing system for static analysis of BioAmbients
- extending Poly★ adding expressiveness by providing better support for: the name restriction operator, recursion, the choice operator

Current work on Poly★

- comparison with existing type systems for Mobile Ambient:
In Poly★, we can construct the type $\pi_{E,T}^P$ such that:

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

- comparison with existing system for static analysis of BioAmbients
- extending Poly★ adding expressiveness by providing better support for: the name restriction operator, recursion, the choice operator

Current work on Poly★

- comparison with existing type systems for Mobile Ambient:
In Poly★, we can construct the type $\pi_{E,T}^P$ such that:

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

- comparison with existing system for static analysis of BioAmbients
- extending Poly★ adding expressiveness by providing better support for: the name restriction operator, recursion, the choice operator



Summary

- Many **process calculi** were developed to study various aspects of concurrent systems.
- Poly* provides **a uniform type system** for a large family of process calculi.
- Poly* can also be used for **static analysis** of modeled systems.
- Outlook
 - name restriction operator, recursion, choice operator
 - formal comparison(s) of Poly* with related systems



Summary

- Many **process calculi** were developed to study various aspects of concurrent systems.
- Poly* provides **a uniform type system** for a large family of process calculi.
- Poly* can also be used for **static analysis** of modeled systems.
- Outlook
 - name restriction operator, recursion, choice operator
 - formal comparison(s) of Poly* with related systems

Summary

- Many **process calculi** were developed to study various aspects of concurrent systems.
- Poly* provides **a uniform type system** for a large family of process calculi.
- Poly* can also be used for **static analysis** of modeled systems.
- Outlook
 - name restriction operator, recursion, choice operator
 - formal comparison(s) of Poly* with related systems



Summary

- Many **process calculi** were developed to study various aspects of concurrent systems.
- Poly* provides **a uniform type system** for a large family of process calculi.
- Poly* can also be used for **static analysis** of modeled systems.

- Outlook
 - name restriction operator, recursion, choice operator
 - formal comparison(s) of Poly* with related systems

For Further Reading



J. Parrow

An introduction to the π -calculus

in *Handbook of Process Algebra*, p. 479–543, NH, 2001



L. Cardelli and A. D. Gordon

Mobile Ambients

in *Proc. FoSSaCS'98*, p. 140–155, SV, 1998



H. Makholm and J. B. Wells

Instant polymorphic type systems for mobile process calculi

in *Proc. ESOP'05*, p. 389–407, SV, 2005