Instant Polymorphic Type Systems for Mobile Process Calculi: Just Add Reduction Rules and Close

Henning Makholm and Joe Wells

Heriot-Watt University

ESOP'05 – April 8, 2005

Work supported by EU/IST/FET grant 2001-33477 (DART)

Useful
Logics,
Types,
Rewriting, and their
Automation

Mobile process calculi

- *Mobility and process calculi* are used to model and reason about systems with *mobile devices*, *mobile code*, *dynamically changing networks*, ... and to model *biological systems* and *business processes*.
- *Many such calculi* exist:
  - The π-calculus – and variants
  - Mobile Ambients – and variants
  - Safe Ambients, Boxed Ambients, Seal – and variants
  - Dπ, Higher-order π-calculus – and variants
  - Join calculus – and variants
- There is no obvious *best* calculus. For different purposes one may need different calculi, and needs are likely to change.

Types for process calculi

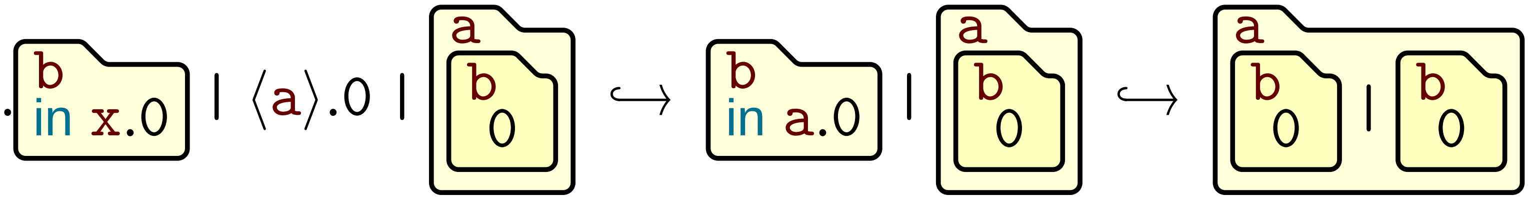
- Any process and mobility calculus can benefit from having a *type system*.
  - For pinpointing *programming errors*
  - To prove that programs or systems are *safe*
  - To provide *flow information* for automatic analyses
- Traditionally *each new calculus* has a type system designed specifically for it.
- We present the *re-targetable* type system Poly✶ which *automatically adapts* to new calculi or variants.
  - Allows easy *experimentation* with calculus variants
Just write down your reduction rules. Poly✶ does the rest.
  - Experimenting with *type system features*:
Which features do I need to handle this kind of code?

Plan

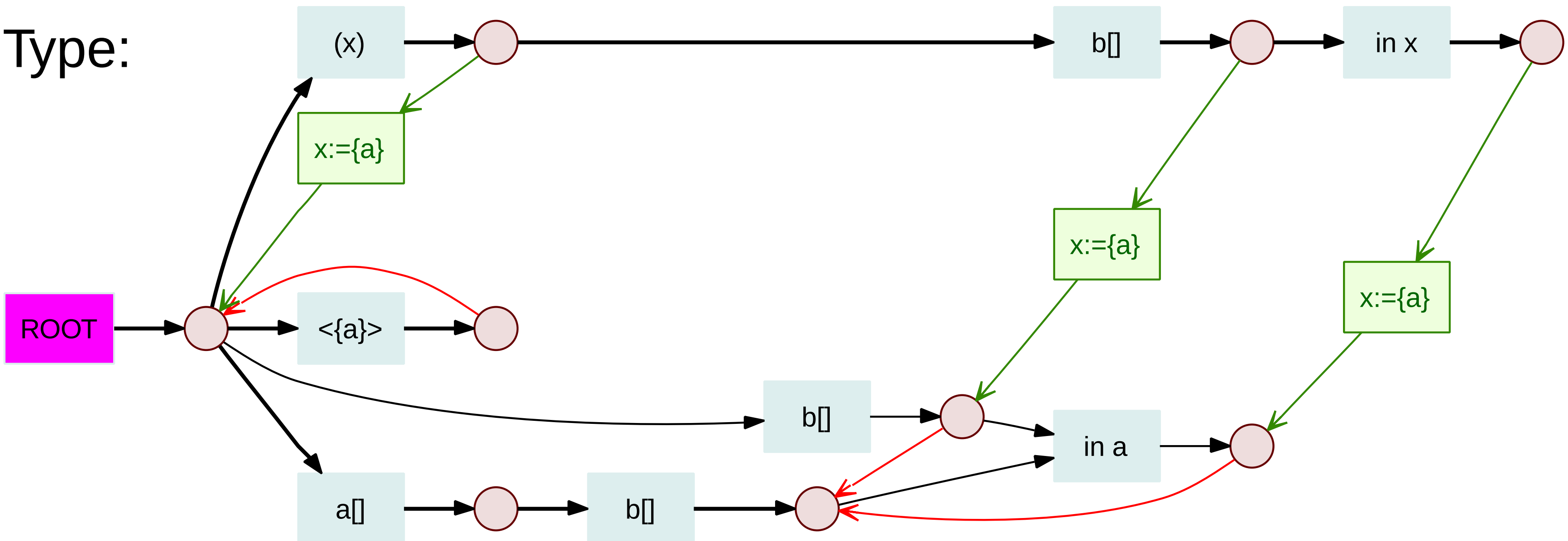
- Poly✶ example
- Case study: Evolution of calculi
- Spatial polymorphism
- Theoretical properties
- Conclusion

An example Poly★ type for an ambient term

Term: (x).



Type:



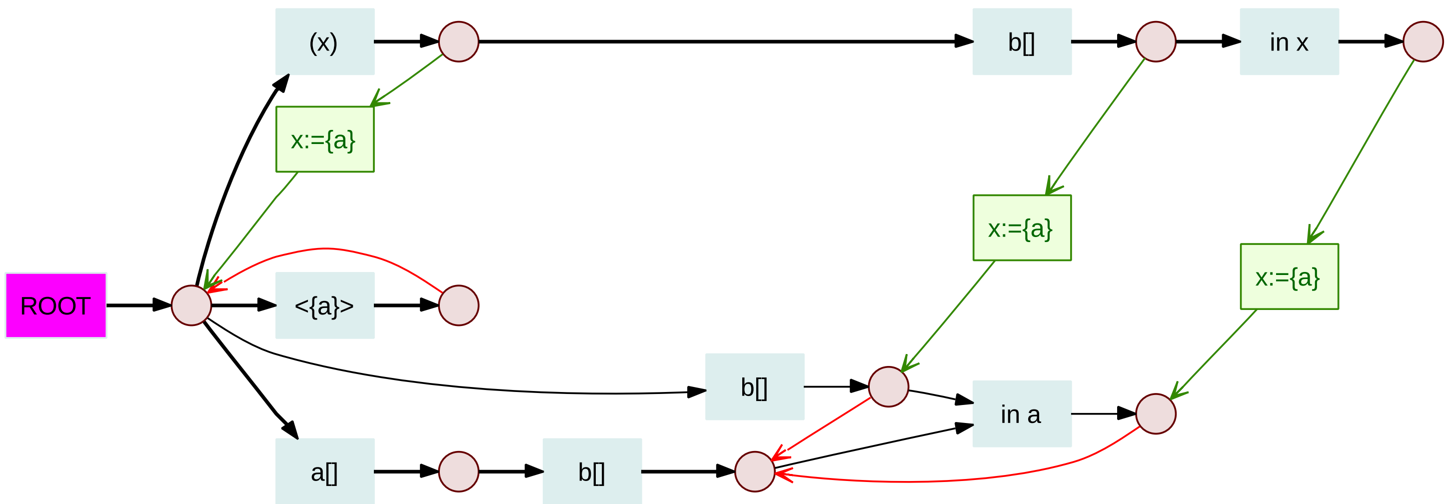
Black edges with labels define the possible term structure.

Red edges encode *flow*, which is the same as *subtyping*.

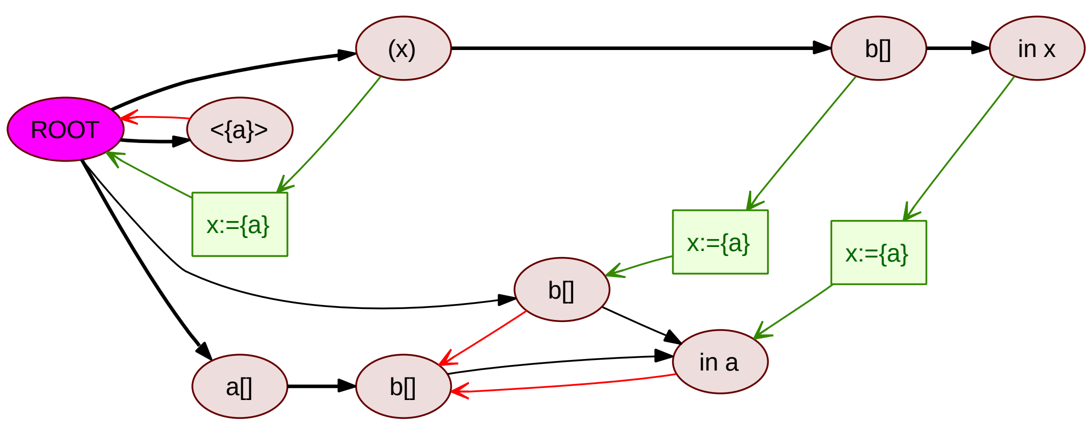
Green edges encode flow/subtyping with substitutions.

Useful
Logics,
Types,
Rewriting, and their
Automation

Drawing the type graph more compactly



When all black edges leading to a node have the same label, we write the label inside the target node:



Useful
Logics,
Types,
Rewriting, and their
Automation

Plan

- Poly✶ example
- Case study: Evolution of calculi
- Spatial polymorphism
- Theoretical properties
- Conclusion

The siege of Troy

Term:

```
horse        Ulysses                  Troy
in Troy      in horse.out horse         0
```

Input to Poly✶ type inference tool:

```
active{ P : a[P] }
reduce{ a["in" b.P | Q] | b[S]  --> b[a[P|Q] | S] }
reduce{ a[b["out" a.P | Q] | S] --> a[S] | b[P|Q] }
reduce{ "open" a.P | a[R]       --> P | R         }

term{ horse[in Troy]
    | Ulysses[in horse.out horse]
    | Troy[0]
    }
```

The siege of Troy

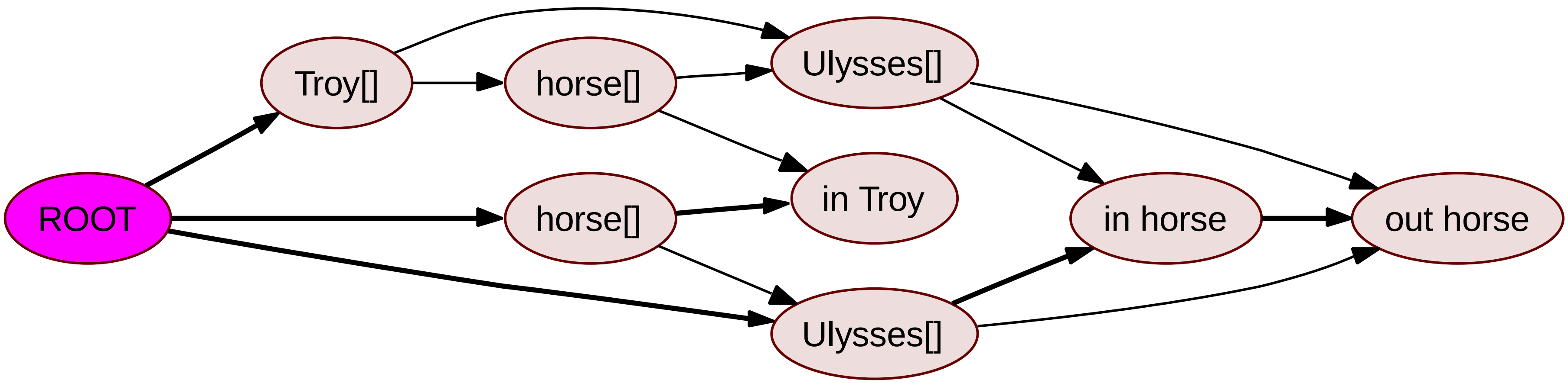
Term:

horse
in Troy

Ulysses
in horse.out horse

Troy
0

Inferred type:



Safe Ambients, first try

What if one needed *permission* to enter and exit ambients?

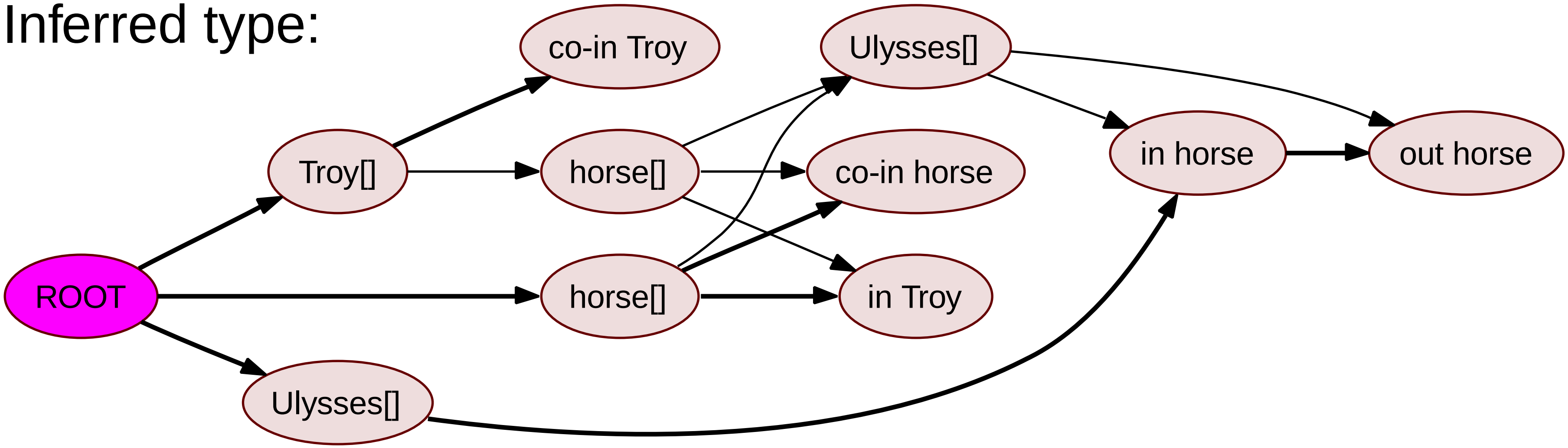
```
active{ P : a[P] }
reduce{ a["in" b.P | Q] | b[S | "co-in" b.R ]
                             --> b[a[P|Q] | S | R] }
reduce{ a[b["out" a.P | Q] | S] | "co-out" a.R
                             --> a[S] | b[P|Q] | R }
reduce{ "open" a.P | a[R | "co-open" a.S]
                             --> P | R | S         }

term{ horse[in Troy | co-in horse]
    | Ulysses[in horse.out horse]
    | Troy[co-in Troy.0]
    }
```

Useful
Logics,
Types,
Rewriting, and their
Automation

Safe Ambients, first try

What if one needed *permission* to enter and exit ambients?

Inferred type:



Hmm. This seems to work. Or does it?

The first try did not work

Unfortunately, Ulysses is rather clever.

If the horse can use the “co-in Troy”, then so can he.

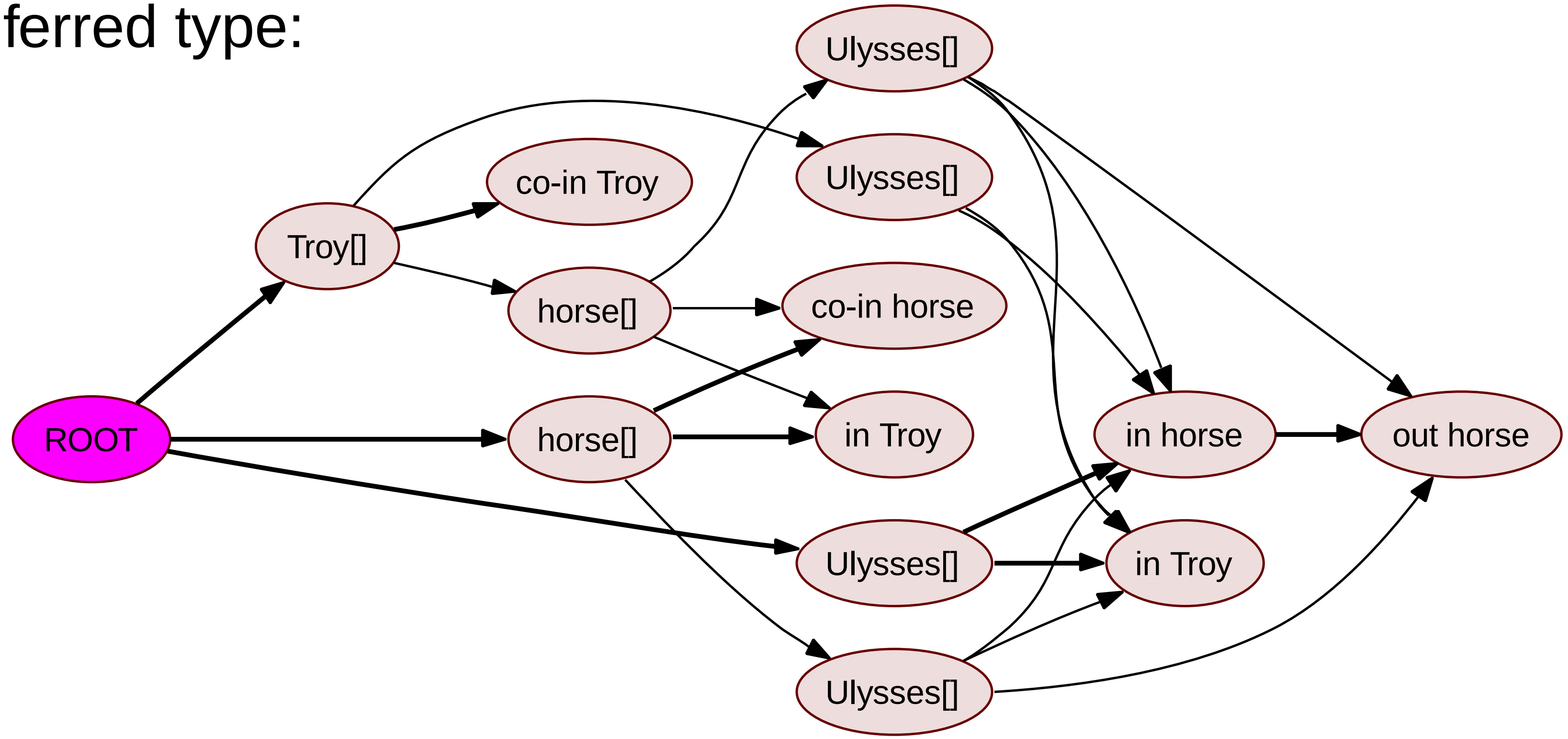
```
active{ P : a[P] }
reduce{ a["in" b.P | Q] | b[S | "co-in" b.R ]
                         --> b[a[P|Q] | S | R] }
reduce{ a[b["out" a.P | Q] | S] | "co-out" a.R
                         --> a[S] | b[P|Q] | R }
reduce{ "open" a.P | a[R | "co-open" a.S]
                         --> P | R | S          }

term{ horse[in Troy | co-in horse]
    | Ulysses[in horse.out horse | in Troy ]
    | Troy[co-in Troy.0]
    }
```

The first try did not work

Unfortunately, Ulysses is rather clever.

If the horse can use the "co-in Troy", then so can he.

Inferred type:



Modern Safe Ambients

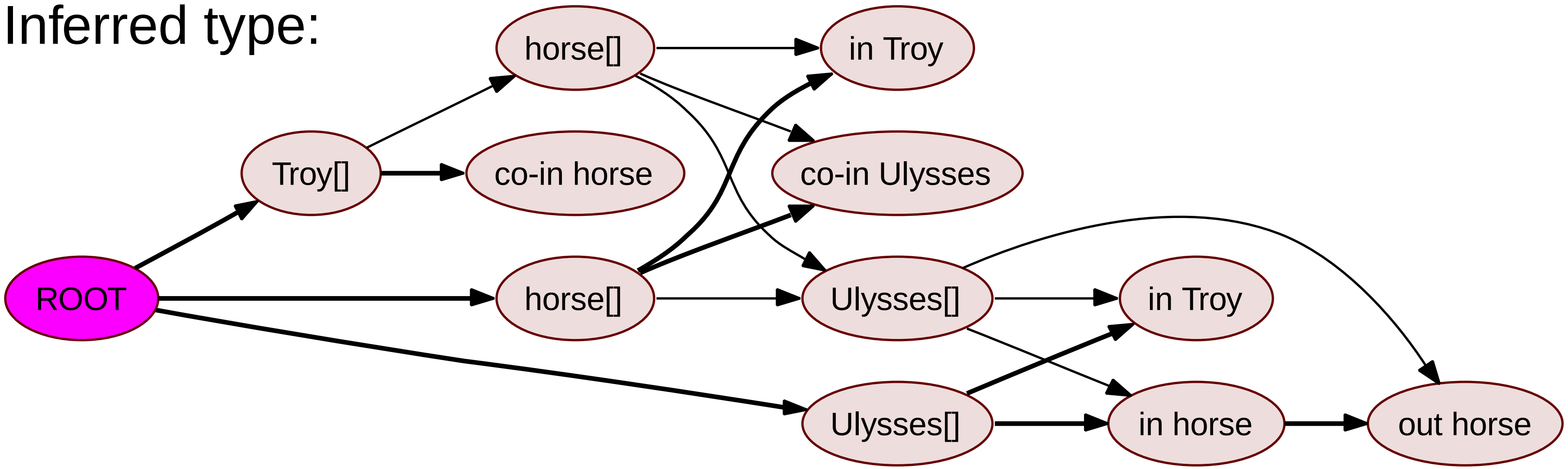
It would be better if the permissions say *who* can enter instead of *where* the permission itself is located.

```
active{ P : a[P] }
reduce{ a["in" b.P | Q] | b[S | "co-in" a.R ]
                          --> b[a[P|Q] | S | R] }
reduce{ a[b["out" a.P | Q] | S] | "co-out" b.R
                          --> a[S] | b[P|Q] | R }
reduce{ "open" a.P | a[R | "co-open" a.S]
                          --> P | R | S         }

term{ horse[in Troy | co-in Ulysses]
    | Ulysses[in horse.out horse | in Troy ]
    | Troy[co-in horse.0]
    }
```

Modern Safe Ambients

It would be better if the permissions say *who* can enter instead of *where* the permission itself is located.

Inferred type:



This works!

Plan

- Poly✶ example
- Case study: Evolution of calculi
- Spatial polymorphism
- Theoretical properties
- Conclusion

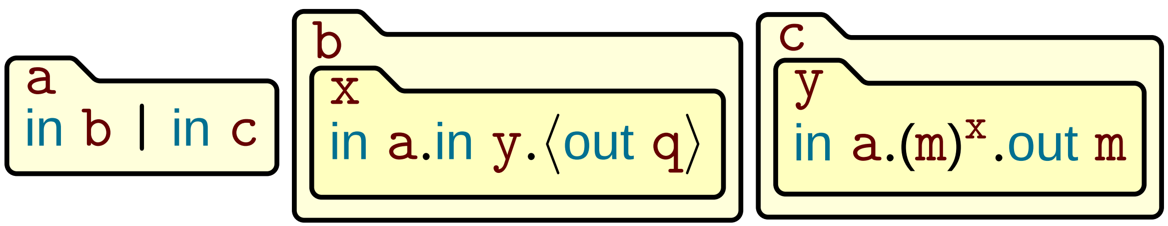
Spatial polymorphism

- The core of Poly✶ descends from earlier work on PolyA for Mobile Ambients [Amtoft, Makholm, Wells].

  It inherits the notion of *spatial polymorphism*:

  - A *single* process can have *multiple* future *type descriptions*, depending on *where it moves*.

- Example. Consider the Boxed Ambients term

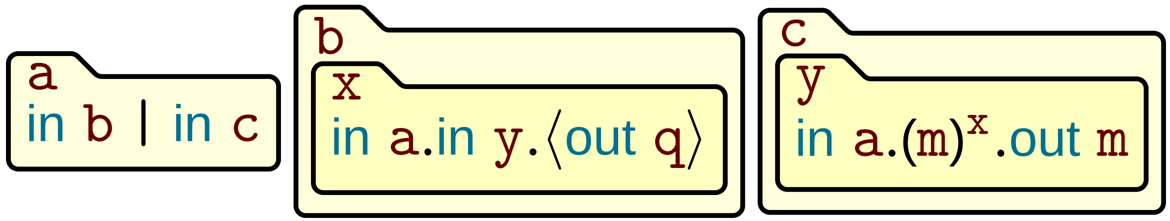


  - If x and y were to enter a simultaneously, ⟨out q⟩ and (m)^x.out m would communicate, causing a run-time error. This term's Poly✶ type verifies this does not happen.

  - *Spatial polymorphism* allows a type to express that a can contain x when found inside one b, or y when found inside the other, but never both.

Spatial polymorphism example

Term:



Input to type inference tool:

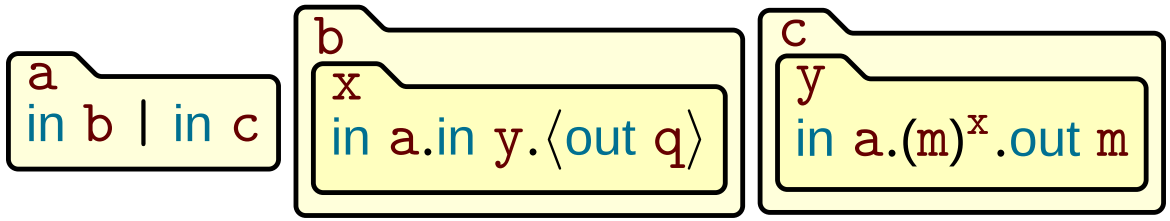
```
active{ P : a[P] }
reduce{ a["in" b.P | Q] | b[S]    --> b[a[P|Q] | S]        }
reduce{ a[b["out" a.P | Q] | S]   --> a[S] | b[P|Q]        }
reduce{ <M>  .P | n[(a)^^.Q | R] --> P | n[{a:=M}Q | R] }
```

(4 other communication rules go here)

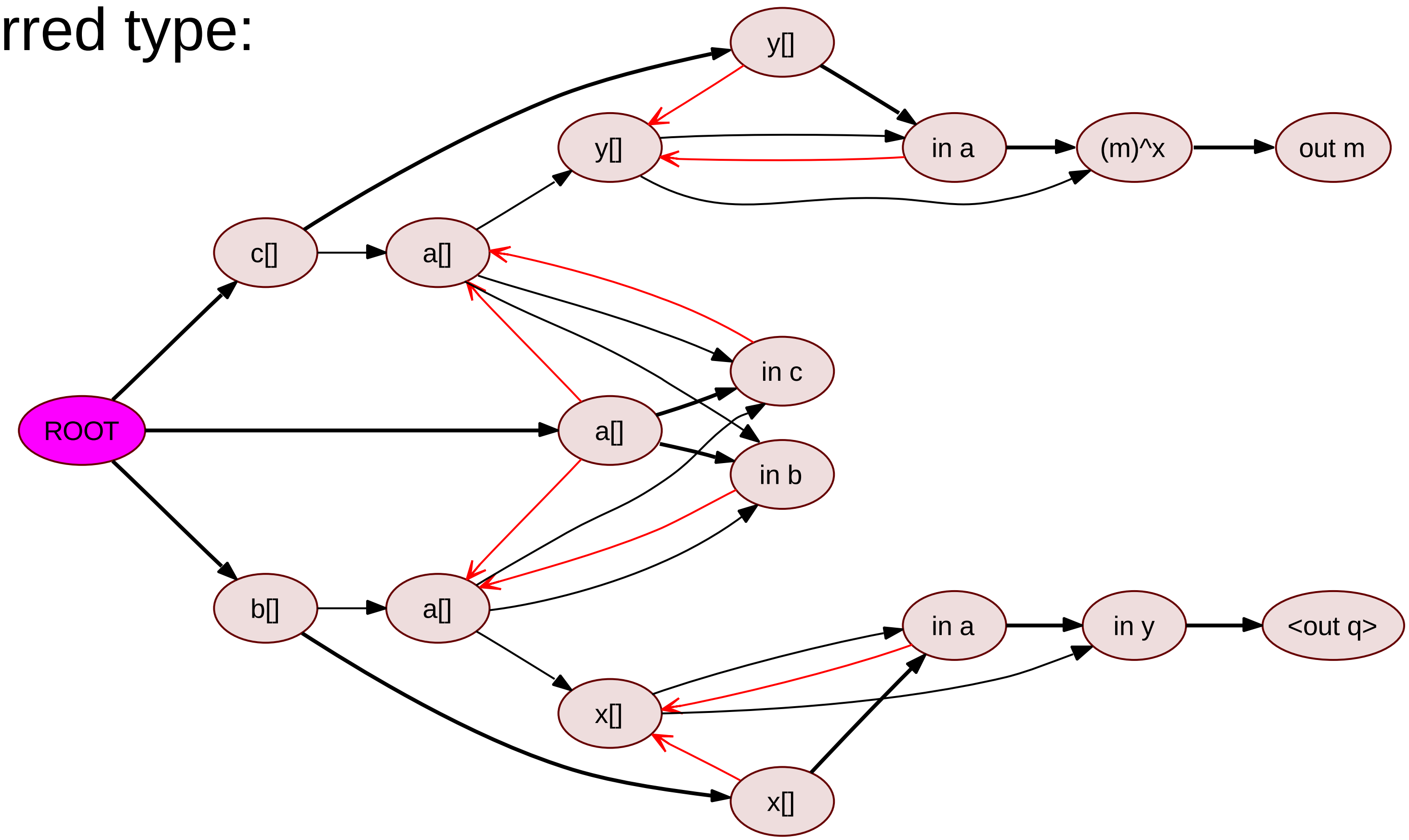
```
term{ a[in b | in c]
    | b[x[in a.in y.<out q>]]
    | c[y[in a.(m)^x.out m]] }
```

Spatial polymorphism example

Term:



Inferred type:



Turning off spatial polymorphism

Input to type inference tool:

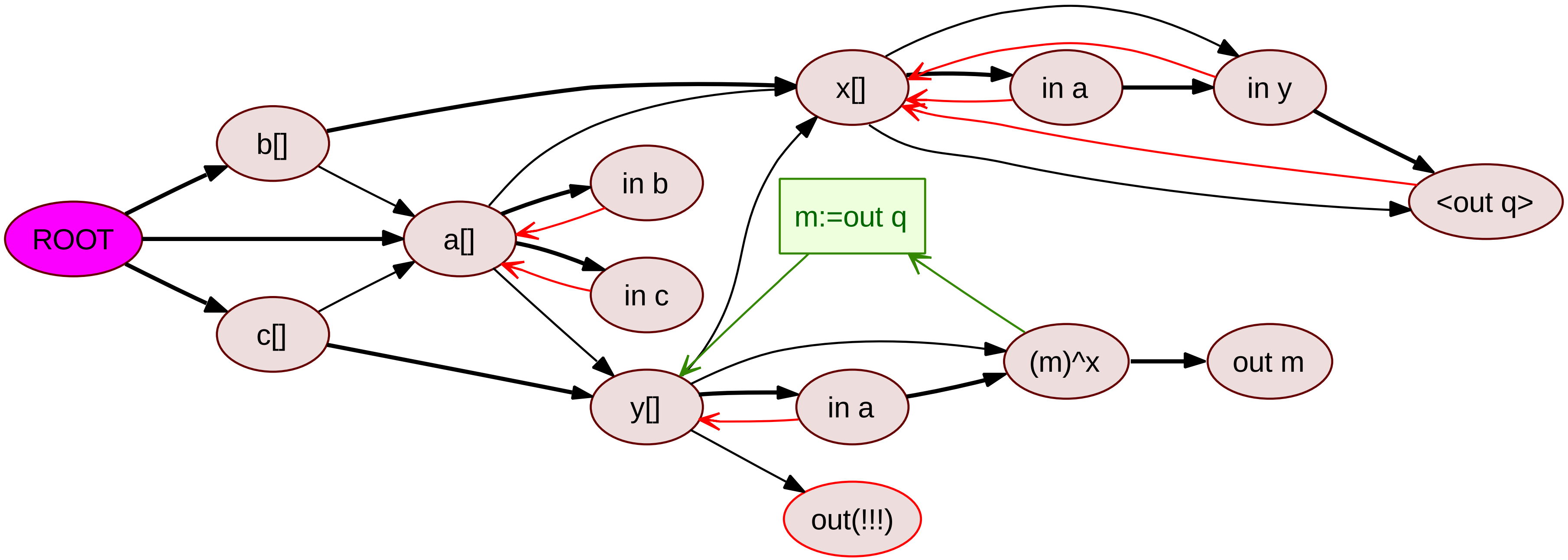
```
active{ P : a[P] }
reduce{ a["in" b.P | Q] | b[S]    --> b[a[P|Q] | S]      }
reduce{ a[b["out" a.P | Q] | S]   --> a[S] | b[P|Q]      }
reduce{ <M>  .P | n[(a)^^.Q | R] --> P | n[{a:=M}Q | R] }
```

(4 other communication rules go here)

```
term{ a[in b | in c]
    | b[x[in a.in y.<out q>]]
    | c[y[in a.(m)^x.out m]] }

option{ smash + n[] }
```

Turning off spatial polymorphism

Inferred type:



The form "out (!!!)" in the red circle (which would be "out •" in the paper's notation) indicates that Poly✶ has detected a possible run-time error, namely an ill-formed substitution result.

History polymorphism

History polymorphism allows having multiple type descriptions for possible processes at a location, depending on *where they came from*.

History polymorphism is built on top of spatial polymorphism using *origin marks*.

Unfortunately we don't have time to describe it now. Feel free to ask us after the session for a demonstration!

Plan

- Poly✶ example
- Case study: Evolution of calculi
- Spatial polymorphism
- Theoretical properties
- Conclusion

The metacalculus Meta$\star$

- A single syntax that allows one to write process terms from many concrete calculi.

$$
\begin{array}{lrl}
\text{Processes:} & P, Q ::= & (P \mid Q) \mid 0 \mid\ !P \\
& \mid & \nu(x).P \\
& \mid & F.P \\
\text{Forms:} & F ::= & E_1\ E_2 \ldots E_k \\
\text{Elements:} & E ::= & x \mid (x_1, \ldots, x_k) \mid \texttt{<}M_1, \ldots, M_k\texttt{>} \\
\text{Messages:} & M ::= & 0 \mid F_1. \cdots .F_k
\end{array}
$$

- Key concept: the *form* F. Examples:
“in Troy”, “open x”, “$\langle$out a.in b, k$\rangle^{\uparrow}$”, “$\overline{\mathtt{c}}$(z)”, “a[]”, “q”.
- Where are keywords? E.g., in or out? They are *names*.
- Punctuation? “$\langle$out a.in b, k$\rangle^{\uparrow}$” $\Rightarrow$ “`<out a.in b, k> ^ ^`”.
- Ambients? Sugar E[P] $\Rightarrow$ E[].P

The usual nice properties of types

- Straightforward *subject reduction* result holds for a large class of *closed* type graphs.
- In a narrower class, defined by *width* and *depth* restrictions, *principal typings* exist: Each process term has a *best* type that is a stronger predicate on terms than any of its other types. Our type inference algorithm *infers* principal typings.
- Typing derivations are *easily checkable* by purely *local* rules.
  - It may be difficult to *compute* a type, but it is easy to check whether a purported type is good for a term.
  - In contrast, for non-type-based program analyses, validating analysis results typically costs as much as computing them from scratch.
- All properties also hold for interesting *restrictions* that give *smaller* types or *faster* inference.

Answers to common questions

- This seems more like program analysis than types.

  Answer: There is no clean division between type systems and other forms of program analysis. Types must become more detailed to obtain principal typings.

- The types seem large compared to the terms they describe.

  Answer: Our examples show the most precise version of our system. Our system can be fine-tuned to trade space for expressive strength. There are versions of our system with smaller types that are as crude as previous type systems.

More questions and answers are at
⟨`http://www.macs.hw.ac.uk/DART/software/PolyStar/FAQ`⟩

(or Google for “PolyStar type inference FAQ”).

Plan

- Poly✶ example
- Case study: Evolution of calculi
- Spatial polymorphism
- Theoretical properties
- Conclusion

Future work

Lift restrictions on calculi

- Eliminate current *invariant:* Names bound by forms never need to be α-renamed.
  - Reduction rules that risk breaking this (by moving binders into each other) are rejected by the system.
- Allow more structured *messages* than just names and "flat" forms. (This would allow *spi-calculus*).

Make type-system core stronger

- Add some form of *single-threadedness* tracking.
- Incorporate the form of polymorphism commonly used for the π-calculus. ($\overline{c}(x).\nu(k).\cdots$)

Conclusion

- The metacalculus Meta✶ can be instantiated to many proposed process calculi
- The type system Poly✶ applies to each instantiation . . .
  - . . . and provides *spatial polymorphism* (or not)
  - . . . and *history polymorphism* (or not)
- The *strength* of Poly✶ is adjustable in many orthogonal *dimensions*.
- A very flexible *implementation of type inference* is available:

  ⟨`http://www.macs.hw.ac.uk/DART/software/PolyStar/`⟩

  (or ⟨`http://henning.makholm.net/`⟩ $\longrightarrow$ software $\longrightarrow$ Poly✶)

Thank you