A Causal Type System for Ambient Movements

Torben Amtoft
Church Project

Boston University/Heriot-Watt University
The Ambient Calculus

Proposed by Cardelli & Gordon to model notions of

- **Location**: ambients are nested, forming a dynamic tree structure
The Ambient Calculus

Proposed by Cardelli & Gordon to model notions of

- **Location**: ambients are nested, forming a dynamic tree structure
- **Mobility**: ambients can enter and exit other ambients
The Ambient Calculus

Proposed by Cardelli & Gordon to model notions of

- **Location**: ambients are nested, forming a dynamic tree structure
- **Mobility**: ambients can enter and exit other ambients
- **Communication**: inside an ambient, values can be exchanged between processes
The Ambient Calculus

Proposed by Cardelli & Gordon to model notions of

- **Location**: ambients are nested, forming a dynamic tree structure
- **Mobility**: ambients can enter and exit other ambients
- **Communication**: inside an ambient, values can be exchanged between processes

Our proposal is to verify certain safety and security properties using
The Ambient Calculus

Proposed by Cardelli & Gordon to model notions of

- **Location**: ambients are nested, forming a dynamic tree structure
- **Mobility**: ambients can enter and exit other ambients
- **Communication**: inside an ambient, values can be exchanged between processes

Our proposal is to verify certain safety and security properties using

- a type system
The Ambient Calculus

Proposed by Cardelli & Gordon to model notions of

- **Location**: ambients are nested, forming a dynamic tree structure

- **Mobility**: ambients can enter and exit other ambients

- **Communication**: inside an ambient, values can be exchanged between processes

Our proposal is to verify certain safety and security properties using

- a type system

- with causality information
Outline of Talk

- introduce the ambient calculus
Outline of Talk

- introduce the ambient calculus
- explain the security issue to be addressed
Outline of Talk

- introduce the ambient calculus
- explain the security issue to be addressed
- motivate our approach
Outline of Talk

- introduce the ambient calculus
- explain the security issue to be addressed
- motivate our approach
- sketch the formal development
Firewall and Agent Example

\( w \) secret name

\( k' \) password
Firewall and Agent Example

\[
\begin{align*}
&w \text{ secret name} \\
&\begin{array}{c}
\text{out } w.\text{in } k'.\text{in } w \\
\text{open } k'.\text{open } k''. P
\end{array} \\
&\begin{array}{c}
\text{open } k.k''[Q]
\end{array}
\end{align*}
\]
Firewall and Agent Example

**secret name**

\[
\begin{array}{c}
\text{k} \\
\text{out w.in k'.in w} \\
\text{open k'.open k''.P}
\end{array}
\]

**password**

\[
\begin{array}{c}
\text{k'} \\
\text{open k.k''[Q]}
\end{array}
\]

\[
\begin{array}{c}
\text{in k'.in w} \\
\text{open k'.open k''.P}
\end{array}
\]

\[
\begin{array}{c}
\text{k} \\
\text{open k.k''[Q]}
\end{array}
\]
Firewall and Agent Example

\[ w \text{ secret name} \]
\[ k \text{ out } w.\text{in } k'.\text{in } w \]
\[ \text{open } k'.\text{open } k''.P \]
\[ k' \text{ password} \]
\[ \text{open } k.k''.[Q] \]

\[ w \]
\[ \text{open } k'.\text{open } k''.P \]
\[ k \]
\[ \text{in } k'.\text{in } w \]
\[ k' \]
\[ \text{open } k.k''.[Q] \]

\[ w \]
\[ \text{open } k'.\text{open } k''.P \]
\[ k \]
\[ \text{in } w \]
\[ k' \]
\[ \text{open } k.k''.[Q] \]
**Firewall and Agent Example**

\[
\begin{align*}
    & w \quad \text{secret name} \\
    & k \quad \text{out } w.\text{in } k'.\text{in } w \\
    & \text{open } k'.\text{open } k''.P \quad k' \quad \text{password} \\
    & \text{open } k.\text{open } k''.[Q] \\
    & w \quad \text{in } k'.\text{in } w \\
    & \text{in } w \quad \text{open } k.\text{open } k''.[Q] \\
    & w \quad \text{in } w \\
    & \text{in } w \\
    & \text{open } k'.\text{open } k''.P \\
    & k' \quad \text{Q} \\
    & \text{in } w \\
    & \text{in } w \\
    & \text{open } k'.\text{open } k''.P \\
    & k' \quad \text{Q} \\
\end{align*}
\]
Useful Logics, Types, Rewriting, and their Automation

Firewall and Agent Example

\( w \) secret name

\( k \)

\[ \text{out } w.\text{in } k'.\text{in } w \]

\( \text{open } k'.\text{open } k''.P \)

\( k' \) password

\[ \text{open } k.\text{?}[Q] \]

\( w \)

\[ \text{open } k'.\text{open } k''.P \]

\( k \)

\[ \text{in } k'.\text{in } w \]

\( w \)

\[ \text{open } k'.\text{open } k''.P \]

\( k' \)

\[ \text{open } k.\text{?}[Q] \]

\( w \)

\[ \text{open } k'.\text{open } k''.P \]

\( k' \)

\[ \text{in } w \]

\[ k'' \]

\( Q \)

\( w \)

\[ \text{open } k'.\text{open } k''.P \]

\[ Q \]

\[ Q \]
Trojan Horse Example

\[ a \]
\[ \text{open } b.\text{in } c \]

\[ b \]
\[ \text{in } a.\text{in } d \]

\[ c \]
\[ P \]
\[ d \]
\[ Q \]
Trojan Horse Example

\[
a \text{open } b \text{.in } c \quad b \text{in } a \text{.in } d \quad c \text{[P } \mid d \text{[Q]]}
\]
Trojan Horse Example

\[ a[\text{open } b.\text{in } c] \mid b[\text{in } a.\text{in } d] \mid c[\text{P } \mid \text{d}[Q]] \]
Trojan Horse Example

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} \\
\text{open } \text{b.in c} & \quad \text{in a.in d} & \quad \text{in d} \\
\text{a[open b.in c] | b[in a.in d] | c[P | d[Q]]} \end{align*}
\]
Trojan Horse Example

\[
a [\text{open } b.\text{in } c] \mid b[\text{in } a.\text{in } d] \mid c[P \mid d[Q]]
\]
Trojan Horse Example

Security breach: $a$ enters $d$
Security Constraints

Static: \( I(a, b) \)
\( a \) is allowed to be a child of \( b \)
Security Constraints

**Static:** $\mathcal{I}(a, b)$

$a$ is allowed to be a child of $b$

**Dynamic:** $\mathcal{O}(a, b)$

$b$ is allowed to open $a$ (must imply $\mathcal{I}(a, b)$)
Security Constraints

Static: $\mathcal{I}(a, b)$
$a$ is allowed to be a child of $b$

Dynamic: $\mathcal{O}(a, b)$
$b$ is allowed to open $a$ (must imply $\mathcal{I}(a, b)$)

_prescriptive view:_ The user imposes the constraints
Security Constraints

Static: $\mathcal{I}(a, b)$

$a$ is allowed to be a child of $b$

Dynamic: $\mathcal{O}(a, b)$

$b$ is allowed to open $a$ (must imply $\mathcal{I}(a, b)$)

- **Prescriptive view**: The user imposes the constraints
- **Descriptive view**: An algorithm infers the (minimal) constraints needed for typability
Security Constraints

**Static:** $\mathcal{I}(a, b)$
$a$ is allowed to be a child of $b$

**Dynamic:** $\mathcal{O}(a, b)$
$b$ is allowed to open $a$ (must imply $\mathcal{I}(a, b)$)

- **prescriptive view:** The user imposes the constraints
- **descriptive view:** An algorithm infers the (minimal) constraints needed for typability

If $a$ is a child of $b$, and $c$ opens $b$, then $a$ becomes a child of $c$. 
Security Constraints

**Static**: $\mathcal{I}(a, b)$

- $a$ is allowed to be a child of $b$

**Dynamic**: $\mathcal{O}(a, b)$

- $b$ is allowed to open $a$ (must imply $\mathcal{I}(a, b)$)

  - **prescriptive view**: The user imposes the constraints
  - **descriptive view**: An algorithm infers the (minimal) constraints needed for typability

If $a$ is a child of $b$, and $c$ opens $b$, then $a$ becomes a child of $c$.

This motivates that we say that a set $H$ is **upward closed** if for all $a \in H$ and all $b$ with $\mathcal{O}(a, b)$ it holds that $b \in H$. 
Security Constraints

Static: $\mathcal{I}(a, b)$
$a$ is allowed to be a child of $b$

Dynamic: $\mathcal{O}(a, b)$
$b$ is allowed to open $a$ (must imply $\mathcal{I}(a, b)$)

- **prescriptive view:** The user imposes the constraints
- **descriptive view:** An algorithm infers the (minimal) constraints needed for typability

If $a$ is a child of $b$, and $c$ opens $b$, then $a$ becomes a child of $c$.

This motivates that we say that a set $H$ is **upward closed** if for all $a \in H$ and all $b$ with $\mathcal{O}(a, b)$ it holds that $b \in H$.

$a'^\uparrow$ denotes the least upwards closed set containing $a$. 

Useful Logics, Types, Rewriting, and their Automation
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can $p$ enter $r$ in the processes below?

1. $p[\text{open } q] \mid q[\text{in } p.\text{in } r.\text{out } r] \mid r[0]
2. p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0]
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can $p$ enter $r$ in the processes below?

1. $p[\text{open } q] | q[\text{in } p.\text{in } r.\text{out } r] | r[0]$  
   Yes!

2. $p[\text{open } q] | q[\text{in } r.\text{out } r.\text{in } p] | r[0]$
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can \( p \) enter \( r \) in the processes below?

- \( p[\text{open } q] \mid q[\text{in } p.\text{in } r.\text{out } r] \mid r[0] \)  
  \( \text{Yes!} \)

- \( p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \)  
  \( \text{No!} \)

Lessons learned: Causality matters! We must keep track of when \( q \) is opened.

The popular dialect Safe ambients employ "co-capabilities":

\( p[\text{open } q] \mid q[\text{in } p.\text{co-open } q: \text{in } r.\text{out } r] \mid r[0] \)  
But we shall stick with the pure calculus.
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can $p$ enter $r$ in the processes below?

- $p[\text{open } q] | q[\text{in } p.\text{in } r.\text{out } r] | r[0]$  
  Yes!

- $p[\text{open } q] | q[\text{in } r.\text{out } r.\text{in } p] | r[0]$  
  No!

Lessons learned:

- Causality matters!
- We must keep track of when $q$ is opened.
- The popular dialect Safe ambients employ "co-capabilities":
  - $q[j][\text{in } p] \text{co-open } q : \text{in } r \text{ out } r \text{ in } p \text{ co-open } q[j][0]$
  - But we shall stick with the pure calculus.
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can $p$ enter $r$ in the processes below?

- $p[\text{open } q] | q[\text{in } p.\text{in } r.\text{out } r] | r[0]$  
  Yes!

- $p[\text{open } q] | q[\text{in } r.\text{out } r.\text{in } p] | r[0]$  
  No!

Lessons learned:

- Causality matters!
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can $p$ enter $r$ in the processes below?

- $p[\text{open } q] \parallel q[\text{in } p.\text{in } r.\text{out } r] \parallel r[0]$  
  Yes!

- $p[\text{open } q] \parallel q[\text{in } r.\text{out } r.\text{in } p] \parallel r[0]$  
  No!

Lessons learned:

- Causality matters!
- we must keep track of when $q$ is opened.
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can \( p \) enter \( r \) in the processes below?

- \( p[\text{open } q] \mid q[\text{in } p.\text{in } r.\text{out } r] \mid r[0] \) \hspace{1cm} \text{Yes!}
- \( p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \) \hspace{1cm} \text{No!}

Lessons learned:

- Causality matters!
- we must keep track of when \( q \) is opened.

The popular dialect \textit{Safe ambients} employ “co-capabilities”:

- \( q[\text{in } p.\text{co-open } q.\text{in } r.\text{out } r] \)
- \( q[\text{in } r.\text{out } r.\text{in } p.\text{co-open } q] \)
Motivating Example

Question, essentially posed by (Cardelli & Ghelli & Gordon, ICALP’99): can $p$ enter $r$ in the processes below?

- $p[\text{open } q] \mid q[\text{in } p.\text{in } r.\text{out } r] \mid r[0]$  
  Yes!
- $p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0]$  
  No!

Lessons learned:

- Causality matters!
- we must keep track of when $q$ is opened.

The popular dialect Safe ambients employ “co-capabilities”:

- $q[\text{in } p.\text{co-open } q.\text{in } r.\text{out } r]$
- $q[\text{in } r.\text{out } r.\text{in } p.\text{co-open } q]$

But we shall stick with the pure calculus.
Basic Idea

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r\text{.out } r\text{.in } p] \mid r[0] \]
Basic Idea

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \# denoting the “global” ambient

\[#\]
Basic Idea

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \( \# \) denoting the “global” ambient

\[ \# \xrightarrow{\text{in } r} r \]
Basic Idea

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \# denoting the “global” ambient

\[
\begin{array}{c}
\# \quad \text{in } r \quad \text{out } r
\end{array}
\]
Basic Idea

Consider again the process

\[ \text{p}[\text{open } q] \mid \text{q}[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \# denoting the “global” ambient

\[ \# \xrightarrow{\text{in } r} r \xrightarrow{\text{out } r} \# \]

For the above calculation, we need to know about the whereabouts of \( r \): that it is always on top-level, so when \( q \) exits \( r \) it becomes child of \#. 
Basic Idea

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \( # \) denoting the “global” ambient

\[ \# \xrightarrow{\text{in } r} r \xrightarrow{\text{out } r} \# \xrightarrow{\text{in } p} p \]

For the above calculation, we need to know about the whereabouts of \( r \): that it is always on top-level, so when \( q \) exits \( r \) it becomes child of \( # \).
**Basic Idea**

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \( \# \) denoting the “global” ambient

\[
\begin{align*}
\# & \xrightarrow{\text{in } r} r & \xrightarrow{\text{out } r} \# & \xrightarrow{\text{in } p} p
\end{align*}
\]

When \( q \) is opened in \( p \), all actions have been “consumed”, so no actions will be “unleashed”.

For the above calculation, we need to know about the whereabouts of \( r \): that it is always on top-level, so when \( q \) exits \( r \) it becomes child of \( \# \).
Basic Idea

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \( \# \) denoting the “global” ambient

\[ \# \xrightarrow{\text{in } r} r \xrightarrow{\text{out } r} \# \xrightarrow{\text{in } p} p \]

When \( q \) is opened in \( p \), all actions have been “consumed”, so no actions will be “unleashed”.

For the above calculation, we need to know about the whereabouts of \( r \): that it is always on top-level, so when \( q \) exits \( r \) it becomes child of \( \# \).

Chicken and egg problem? No, just find a consistent solution!
Basic Idea

Consider again the process

\[ p[\text{open } q] \mid q[\text{in } r.\text{out } r.\text{in } p] \mid r[0] \]

The path of \( q \) is, with \# denoting the “global” ambient

\[ \# \xrightarrow{\text{in } r} r \xrightarrow{\text{out } r} \# \xrightarrow{\text{in } p} p \]

When \( q \) is opened in \( p \), all actions have been “consumed”, so no actions will be “unleashed”.

For the above calculation, we need to know about the whereabouts of \( r \): that it is always on top-level, so when \( q \) exits \( r \) it becomes child of \#.

Chicken and egg problem? No, just find a consistent solution!

Information for \( q \): \( \text{amb} \#_{rp}[p : \varepsilon] \)
Trojan Horse, revisited

\[ a[\text{open } b.\text{in } c] \mid b[\text{in } a.\text{in } d] \mid c[P \mid d[Q]] \]
Trojan Horse, revisited

\[ a[\text{open } b.\text{in } c] \mid b[\text{in } a.\text{in } d] \mid c[P \mid d[Q]] \]

Typing:

\[ d : \text{amb}_c \]
Trojan Horse, revisited

\[ a[\text{open } b.\text{in } c] \mid b[\text{in } a.\text{in } d] \mid c[P \mid d[Q]] \]

Typing:

\[ d : \text{amb}_c \]

\[ c : \text{amb}_{\#} \]
Trojan Horse, revisited

\[ a[\text{open } b.\text{in } c] \mid b[\text{in } a.\text{in } d] \mid c[P] \mid d[Q] \]

Typing:

- \( d : \text{amb}_c \)
- \( c : \text{amb}_\# \)
- \( b : \text{amb}_\#a[a : c \text{ enter}(d)] \)
Trojan Horse, revisited

\[ \text{Typing:} \]

- \( d : \text{amb}_c \)
- \( c : \text{amb}_# \)
- \( b : \text{amb}_#a[a : c \text{ enter}(d)] \)
- \( a : \text{amb}_#cd \)
Trojan Horse, revisited

\[ a[\text{open } b.\text{in } c] \mid b[\text{in } a.\text{in } d] \mid c[P \mid d[Q]] \]

Typing:

\[ d : \text{amb}_c \]
\[ c : \text{amb}_\# \]
\[ b : \text{amb}_\#a[a :^c \text{enter}(d)] \]
\[ a : \text{amb}_\#cd \]

Security breach detected!
Firewall, revisited

\[(\nu w).w[k[\text{out } w.\text{in } k'.\text{in } w] \mid \text{open } k'.\text{open } k''.P] \mid k'[\text{open } k.k''[Q]]]\]
Firewall, revisited

\[(\nu w).w[k[\text{out } w.\text{in } k'.\text{in } w] \mid \text{open } k'.\text{open } k''.\text{P}] \mid k'[\text{open } k.k''[Q]]\]

\[w : \text{amb#}\]
Firewall, revisited

\[(\nu w).w[k[\text{out } w.\text{in } k'.\text{in } w] \mid \text{open } k'.\text{open } k''.P] \]

\[\mid k'[\text{open } k.k''[Q]]\]

\[w : \text{amb}_{\#}\]

\[k : \text{amb}_{w#k'}[k' : \# \text{ enter}(w)]\]
Firewall, revisited

\[(\nu w).w[k]\text{out } w\text{.in } k'.\text{in } w]\mid \text{open } k'.\text{open } k''.P]\mid k'[\text{open } k.k''[Q]]

- \(w : \text{amb}_{\#}\)
- \(k : \text{amb}_{w\#k'}[k' : \# \text{ enter}(w)]\)
- \(k' : \text{amb}_{w}[w : \varepsilon]\)
Firewall, revisited

\[(vw).w[k[\text{out } w.\text{in } k'.\text{in } w] \mid \text{open } k'.\text{open } k''.P] \mid k'[\text{open } k.k''[Q]]\]

- \( w : \text{amb}_\# \)
- \( k : \text{amb}_{w#k'}[k' : # \text{ enter}(w)] \)
- \( k' : \text{amb}_{#w}[w : \varepsilon] \)
- \( k'' : \text{amb}_{k'w}[w : ?] \)
Firewall, revisited

$$(\nu w).w[k[\text{out } w\text{.in } k'.\text{in } w] \mid \text{open } k'.\text{open } k''.P]$$

$$\mid k'[\text{open } k.k''[Q]]$$

- $w : \text{amb}_\#$
- $k : \text{amb}_{w\#k'}[k' : \# \text{ enter}(w)]$
- $k' : \text{amb}_{w}[w : \varepsilon]$
- $k'' : \text{amb}_{k'w}[w : ?]$}

The secret name $w$ should not be known in the type of $k'$ and $k''$. 

Useful Logics, Types, Rewriting, and their Automation
Firewall, revisited

\[(\nu w).w[k[\text{out} w.\text{in} k'.\text{in} w] | \text{open} k'.\text{open} k''.P] \]

\[| \quad k'[\text{open} k.k''[Q]] \]

- \( w : \text{amb} \# \)
- \( k : \text{amb}_w \# k'[k' : \# \text{enter}(w)] \)
- \( k' : \text{amb}_w \# [w : \varepsilon] \)
- \( k'' : \text{amb}_{k'w} [w : ?] \)

The secret name \( w \) should not be known in the type of \( k' \) and \( k'' \). Therefore we follow (Cardelli & Ghelli & Gordon) and introduce groups.
The secret name $w$ should not be known in the type of $k'$ and $k''$. Therefore we follow (Cardelli & Ghelli & Gordon) and introduce groups. With $W$ the group of $w$, etc, we have

$$ k' : \text{amb}_{\#W}^K[W : \varepsilon] $$
**Contribution**

First type system giving a precise estimate of the location of an ambient.
Contribution

First **type system** giving a precise estimate of the location of an ambient.

- semantically sound
Contribution

First type system giving a precise estimate of the location of an ambient.

- semantically sound
- type checking algorithm
Contribution

First *type system* giving a precise estimate of the location of an ambient.

- semantically sound
- type checking algorithm
- limited type inference
Contribution

First **type system** giving a precise estimate of the location of an ambient.

- semantically sound
- type checking algorithm
- limited type inference

Other approaches

- **Control Flow Analysis** (Nielson et. al, CONCUR’99).
  No causality, so very imprecise
**Contribution**

First **type system** giving a precise estimate of the location of an ambient.

- semantically sound
- type checking algorithm
- limited type inference

Other approaches

- **Control Flow Analysis** (Nielson et. al, CONCUR’99).
  No causality, so very imprecise

- **Abstract Interpretation** (Levi & Maffeis, SAS’01)
  Quite precise, and yet polynomial
Contribution

First type system giving a precise estimate of the location of an ambient.

- semantically sound
- type checking algorithm
- limited type inference

Other approaches

- Control Flow Analysis (Nielson et. al, CONCUR’99). No causality, so very imprecise
- Abstract Interpretation (Levi & Maffeis, SAS’01) Quite precise, and yet polynomial
- Shape grammars (Nielson & Nielson, POPL’00) Very precise, but potentially very expensive
Extra Contribution

We allow each ambient to hold *multiple topics of conversation*.
Extra Contribution

We allow each ambient to hold multiple topics of conversation

Communication omitted in (Levi & Maffeis, SAS’01) and (Nielson & Nielson, POPL’00)
Extra Contribution

We allow each ambient to hold **multiple topics of conversation**

- Communication omitted in (Levi & Maffeis, SAS’01) and (Nielson & Nielson, POPL’00)
- Only single topic of conversation allowed in (Cardelli & Gordon, POPL’99)
Extra Contribution

We allow each ambient to hold multiple topics of conversation

- Communication omitted in (Levi & Maffeis, SAS’01) and (Nielson & Nielson, POPL’00)
- Only single topic of conversation allowed in (Cardelli & Gordon, POPL’99)

Below, the topic of conversation in $r$ is `1st int` and then `bool`

$$r[\langle 7 \rangle | (z : \text{int}).\text{open } q.(z = 42) | q[(y : \text{bool}).P]]$$
Extra Contribution

We allow each ambient to hold multiple topics of conversation

Communication omitted in (Levi & Maffeis, SAS’01) and (Nielson & Nielson, POPL’00)

Only single topic of conversation allowed in (Cardelli & Gordon, POPL’99)

Below, the topic of conversation in $r$ is $\text{rst int}$ and then $\text{bool}$

$$r[\langle 7 \rangle \mid (z : \text{int}).\text{open } q. \langle z = 42 \rangle \mid q[(y : \text{bool}).P]]$$

$$\longrightarrow r[\text{open } q. \langle 7 = 42 \rangle \mid q[(y : \text{bool}).P]]$$
Extra Contribution

We allow each ambient to hold multiple topics of conversation

- Communication omitted in (Levi & Maffeis, SAS’01) and (Nielson & Nielson, POPL’00)
- Only single topic of conversation allowed in (Cardelli & Gordon, POPL’99)

Below, the topic of conversation in \( r \) is first int and then bool

\[
\begin{align*}
    r[\langle 7 \rangle \mid (z : \text{int}).\text{open } q.\langle z = 42 \rangle \mid q[(y : \text{bool}).P]] \\
    \quad \rightarrow r[\text{open } q.\langle 7 = 42 \rangle \mid q[(y : \text{bool}).P]] \\
    \quad \rightarrow r[\langle 7 = 42 \rangle \mid (y : \text{bool}).P] \rightarrow r[P[y := \text{false}]]
\end{align*}
\]
Extra Contribution

We allow each ambient to hold multiple topics of conversation

- Communication omitted in (Levi & Maffeis, SAS’01) and (Nielson & Nielson, POPL’00)
- Only single topic of conversation allowed in (Cardelli & Gordon, POPL’99)

Below, the topic of conversation in $r$ is $\texttt{first int}$ and then $\texttt{bool}$

$$
\begin{align*}
   r[\langle 7 \rangle \mid (z : \texttt{int}).\text{open } q.\langle z = 42 \rangle \mid q[(y : \texttt{bool}).P]] \\
   \quad \rightarrow \quad r[\text{open } q.\langle 7 = 42 \rangle \mid q[(y : \texttt{bool}).P]] \\
   \quad \rightarrow \quad r[\langle 7 = 42 \rangle \mid (y : \texttt{bool}).P] \rightarrow r[P[y := \texttt{false}]]
\end{align*}
$$

This is typable, with $q$ having type $\texttt{amb}[r : \text{get(bool)}]$
Language

Expressions:

\[ M \in \text{Exp} ::= \text{in } M \mid \text{out } M \mid \text{open } M \mid n \mid \epsilon \mid M_1.M_2 \]
Language

Expressions:

\[ M \in \text{Exp} ::= \text{in } M \mid \text{out } M \mid \text{open } M \]
\[ \quad \mid n \mid \epsilon \mid M_1.M_2 \]

Processes:

\[ P \in \text{Proc} ::= 0 \mid P_1 \mid P_2 \mid !P \mid (\nu n : \tau).P \]
\[ \quad \mid M.P \mid M[P]^{\xi} \]
\[ \quad \mid (n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \]
Reduction Semantics

\[ m[\text{in } n. P | Q]^\xi | n[R]^\chi \xrightarrow{\xi: \text{enter } \chi} n[m[P | Q]^\xi | R]^\chi \]
Reduction Semantics

\[ m[\text{in } n.P \mid Q]^\xi \mid n[R]^\chi \xrightarrow{\xi: \text{enter } \chi} n[m[P \mid Q]^\xi \mid R]^\chi \]

\[ n[m[\text{out } n.P \mid Q]^\xi \mid R]^\chi \xrightarrow{\xi: \text{exit } \chi} m[P \mid Q]^\xi \mid n[R]^\chi \]
Reduction Semantics

\[ m[\text{in } n. P \mid Q]^{\xi} \mid n[R]^\chi \xrightarrow{\xi: \text{enter } \chi} n[m[P \mid Q]^{\xi} \mid R]^\chi \]

\[ n[m[\text{out } n. P \mid Q]^{\xi} \mid R]^\chi \xrightarrow{\xi: \text{exit } \chi} m[P \mid Q]^{\xi} \mid n[R]^\chi \]

\[ m[\text{open } n. P \mid n[Q]^{\chi} \mid R]^{\xi} \xrightarrow{\xi: \text{open } \chi} m[P \mid Q \mid R]^{\xi} \]
Reduction Semantics

\[ m[\text{in } n.P | Q]^{\xi} | n[R]^{\chi} \xrightarrow{\xi:\text{enter } \chi} n[m[P | Q]^{\xi} | R]^{\chi} \]

\[ n[m[\text{out } n.P | Q]^{\xi} | R]^{\chi} \xrightarrow{\xi:\text{exit } \chi} m[P | Q]^{\xi} | n[R]^{\chi} \]

\[ m[\text{open } n.P | n[Q]^{\chi} | R]^{\xi} \xrightarrow{\xi:\text{open } \chi} m[P | Q | R]^{\xi} \]

\[ m[(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P | \langle M_1 \ldots M_k \rangle | Q]^{\xi} \xrightarrow{\xi:\text{comm } \times (\tau_1, \ldots, \tau_k)} m[P[n_i := M_i] | Q]^{\xi} \]
Reduction Semantics

\[\begin{align*}
m[\text{in } n.P \mid Q]^{\xi} \mid n[R]^\chi & \xrightarrow{\xi:\text{enter } \chi} n[m[P \mid Q]^{\xi} \mid R]^\chi \\
n[m[\text{out } n.P \mid Q]^{\xi} \mid R]^\chi & \xrightarrow{\xi:\text{exit } \chi} m[P \mid Q]^{\xi} \mid n[R]^\chi \\
m[\text{open } n.P \mid n[Q]^\chi \mid R]^{\xi} & \xrightarrow{\xi:\text{open } \chi} m[P \mid Q \mid R]^{\xi} \\
m[(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \mid Q]^{\xi} & \xrightarrow{\xi:\text{comm } \times (\tau_1, \ldots, \tau_k)} m[P[n_i := M_i] \mid Q]^{\xi} \\
!P & \xrightarrow{\epsilon} P' \mid !P \quad \text{if } P' \text{ equals } P \text{ except for tags}
\end{align*}\]
Reduction Semantics

\[
m[\text{in } n.P \mid Q]^{\xi} \mid n[R]^\chi \xrightarrow{\xi: \text{enter } \chi} n[m[P \mid Q]^{\xi} \mid R]^\chi
\]

\[
n[\text{out } n.P \mid Q]^{\xi} \mid R]^\chi \xrightarrow{\xi: \text{exit } \chi} m[P \mid Q]^{\xi} \mid n[R]^\chi
\]

\[
m[\text{open } n.P \mid n[Q]^{\chi} \mid R]^{\xi} \xrightarrow{\xi: \text{open } \chi} m[P \mid Q]^{\xi} \mid R]^{\xi}
\]

\[
m[(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \mid Q]^{\xi}
\]

\[
\xrightarrow{\xi: \text{comm } \times (\tau_1, \ldots, \tau_k)} m[P[n_i := M_i] \mid Q]^{\xi}
\]

\[
!P \xrightarrow{\epsilon} P' \mid !P \quad \text{if } P' \text{ equals } P \text{ except for tags}
\]

\[
\text{If } P' \equiv P, P \xrightarrow{\ell} Q, Q \equiv Q' \text{ then } P' \xrightarrow{\ell} Q'
\]
Reduction Semantics

\[
m[\text{in } n.P \mid Q]^{\xi} \mid n[R]^{\chi} \xrightarrow{\xi:\text{enter } \chi} n[m[P \mid Q]^{\xi} \mid R]^{\chi}
\]

\[
n[m[\text{out } n.P \mid Q]^{\xi} \mid R]^{\chi} \xrightarrow{\xi:\text{exit } \chi} m[P \mid Q]^{\xi} \mid n[R]^{\chi}
\]

\[
m[\text{open } n.P \mid n[Q]^{\chi} \mid R]^{\xi} \xrightarrow{\xi:\text{open } \chi} m[P \mid Q \mid R]^{\xi}
\]

\[
m[(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \mid Q]^{\xi} \xrightarrow{\xi:\text{comm } \times (\tau_1, \ldots, \tau_k)} m[P[n_i := M_i] \mid Q]^{\xi}
\]

\[
!P \xrightarrow{c} P' \mid !P \quad \text{if } P' \text{ equals } P \text{ except for tags}
\]

If \( P' \equiv P, P \xrightarrow{\ell} Q, Q \equiv Q' \) then \( P' \xrightarrow{\ell} Q' \)

If \( P \xrightarrow{\ell} Q \) then \( \mathcal{P}C[P] \xrightarrow{\ell} \mathcal{P}C[Q] \)
Reduction Semantics

\[ m[\text{in } n. P \mid Q]^\xi \mid n[R]^\chi \xrightarrow{\xi:\text{enter } \chi} n[m[P \mid Q]^\xi \mid R]^\chi \]

\[ n[m[\text{out } n. P \mid Q]^\xi \mid R]^\chi \xrightarrow{\xi:\text{exit } \chi} m[P \mid Q]^\xi \mid n[R]^\chi \]

\[ m[\text{open } n. P \mid n[Q]^\chi \mid R]^\xi \xrightarrow{\xi:\text{open } \chi} m[P \mid Q \mid R]^\xi \]

\[ m[(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \mid Q]^\xi \xrightarrow{\xi:\text{comm } \times (\tau_1, \ldots, \tau_k)} m[P[n_i := M_i] \mid Q]^\xi \]

\[ !P \xrightarrow{\epsilon} P' \mid !P \quad \text{if } P' \text{ equals } P \text{ except for tags} \]

\[ \text{If } P' \equiv P, P \xrightarrow{\ell} Q, Q \equiv Q' \text{ then } P' \xrightarrow{\ell} Q' \]

\[ \text{If } P \xrightarrow{\ell} Q \text{ then } \mathcal{P}C[P] \xrightarrow{\ell} \mathcal{P}C[Q] \]

where \[ \mathcal{P}C ::= \Box \mid \mathcal{P}C \mid P \mid (\nu n : \tau).\mathcal{P}C \mid n[\mathcal{P}C]^\xi \]
Type Judgements

Our type system assigns
Type Judgements

Our type system assigns

\[ \text{types } \tau \text{ to expressions: } E \vdash M : \tau \]
Type Judgements

Our type system assigns

- types \( \tau \) to expressions: \( E \vdash M : \tau \)
- behaviors \( b \) to processes: \( \Delta, E \vdash_g P : b \)
Type Judgements

Our type system assigns

- types $\tau$ to expressions: $E \vdash M : \tau$

- behaviors $b$ to processes: $\Delta, E \vdash_g P : b$

where $g$ is the location of $P$
Type Judgements

Our type system assigns

- **types** $\tau$ to expressions: $E \vdash M : \tau$
- **behaviors** $b$ to processes: $\Delta, E \vdash_g P : b$

where $g$ is the location of $P$, and $\Delta$ maps tags into behaviors (needed to express semantic soundness)
Type Judgements

Our type system assigns

- **types** $\tau$ to expressions: $E \vdash M : \tau$

- **behaviors** $b$ to processes: $\Delta, E \vdash_g P : b$
  where $g$ is the location of $P$, and $\Delta$ maps tags into behaviors
  (needed to express semantic soundness)

The system employs subtyping (to be defined also for behaviors).
Behaviors

Action \( a \) ::= \( H \) enter\((g)\) \hspace{1cm} \text{steers an ambient from } H \text{ to } g
| \hspace{1cm} \( g \) exit\((H)\) \hspace{1cm} \text{steers an ambient from } g \text{ to } H
| \hspace{1cm} G \text{ open}(g) \hspace{1cm} \text{if executed in } G, \text{ opens } g
| \hspace{1cm} \text{put}(\sigma) \hspace{1cm} \text{output tuple of type } \sigma
| \hspace{1cm} \text{get}(\sigma) \hspace{1cm} \text{input tuple of type } \sigma
Behaviors

Action \( a := \) [\( \uparrow \) enter(\( g \)] \) steers an ambient from \( H \) to \( g \)
| [\( \downarrow \) exit(\( H \)] \) steers an ambient from \( g \) to \( H \)
| \( G \) open(\( g \)) if executed in \( G \), opens \( g \)
| put(\( \sigma \)) output tuple of type \( \sigma \)
| get(\( \sigma \)) input tuple of type \( \sigma \)

A Trace \( tr \) is a finite sequence of actions.
Behaviors

Action \( a \) ::= \( H \) enter\((g)\)  
\( g \) exit\((H)\)  
\( G \) open\((g)\)  
put\((\sigma)\)  
get\((\sigma)\)

steers an ambient from \( H \) to \( g \)
steers an ambient from \( g \) to \( H \)
if executed in \( G \), opens \( g \)
output tuple of type \( \sigma \)
input tuple of type \( \sigma \)

A Trace \( tr \) is a finite sequence of actions.
A Behavior \( b \) is a non-empty regular set of traces.
Behaviors

**Action** $a ::= \begin{array}{l} \text{enter}(g) \quad \text{steers an ambient from } H \text{ to } g \\ \text{exit}(H) \quad \text{steers an ambient from } g \text{ to } H \\ \text{open}(g) \quad \text{if executed in } G, \text{ opens } g \\ \text{put}(\sigma) \quad \text{output tuple of type } \sigma \\ \text{get}(\sigma) \quad \text{input tuple of type } \sigma \end{array}$

A **Trace** $tr$ is a finite sequence of actions.

A **Behavior** $b$ is a non-empty regular set of traces.

$\varepsilon$ denotes $\{\bullet\}$, and we define the operators

$$a \cdot b = \{a \diamond tr \mid tr \in b\}$$
$$b_1 \parallel b_2 = \bigcup_{tr_1 \in b_1, tr_2 \in b_2} tr_1 \parallel tr_2$$
Behaviors

Action $a ::= {}^H \text{enter}(g)$ ⊕ steers an ambient from $H$ to $g$
| $g \text{exit}(H)$ ⊕ steers an ambient from $g$ to $H$
| $G \text{open}(g)$ ⊕ if executed in $G$, opens $g$
| $\text{put}(\sigma)$ ⊕ output tuple of type $\sigma$
| $\text{get}(\sigma)$ ⊕ input tuple of type $\sigma$

A Trace $tr$ is a finite sequence of actions. Pointwise ordering

A Behavior $b$ is a non-empty regular set of traces.

$b_1 \leq b_2$ iff for all $tr_1 \in b_1$ there exists $tr_2 \in b_2$ such that $tr_1 \leq tr_2$.

$\varepsilon$ denotes $\{\bullet\}$, and we define the operators

$$a \cdot b = \{a \diamond tr \mid tr \in b\}$$
$$b_1 \parallel b_2 = \bigcup_{tr_1 \in b_1, tr_2 \in b_2} tr_1 \parallel tr_2$$
Estimating Destination

\[ \text{Dest}(H, H_0 \text{ enter}(g)) = \begin{cases} \text{if } H \cap H_0 \neq \emptyset & \text{then } g^\uparrow \text{ else } \emptyset \\ \text{Dest}(H, g \text{ exit}(H_0)) = \begin{cases} \text{if } g \in H & \text{then } H_0 \text{ else } \emptyset \\ \text{Dest}(H, a) = H \text{ otherwise} \\ \text{Dest}(H, a \diamond tr) = \text{Dest}(\text{Dest}(H, a), tr) \end{cases} \end{cases} \]
Estimating Destination

\[ \text{Dest}(H, H_0 \text{ enter}(g)) = \begin{cases} g^\uparrow & \text{if } H \cap H_0 \neq \emptyset \\ \emptyset & \text{else} \end{cases} \]

\[ \text{Dest}(H, g \text{ exit}(H_0)) = \begin{cases} H_0 & \text{if } g \in H \\ \emptyset & \text{else} \end{cases} \]

\[ \text{Dest}(H, a) = \begin{cases} H & \text{otherwise} \end{cases} \]

\[ \text{Dest}(H, a \diamond tr) = \text{Dest}(\text{Dest}(H, a), tr) \]

Trojan Horse:

\[ \text{Dest}(\#, \#^{CD} \text{ enter}(A) ^C \text{ enter}(D)) = \text{Dest}(A, ^C \text{ enter}(D)) = \emptyset \]
Estimating Destination

\[
\begin{align*}
\text{Dest}(H, \overset{H_0}{\text{enter}}(g)) &= \text{if } H \cap H_0 \neq \emptyset \text{ then } g^\uparrow \text{ else } \emptyset \\
\text{Dest}(H, \overset{g}{\text{exit}}(H_0)) &= \text{if } g \in H \text{ then } H_0 \text{ else } \emptyset \\
\text{Dest}(H, a) &= H \text{ otherwise} \\
\text{Dest}(H, a \diamond \text{tr}) &= \text{Dest}(\text{Dest}(H, a), \text{tr})
\end{align*}
\]

Trojan Horse:
\[
\text{Dest}(\#, \overset{CD}{\text{enter}}(A) \overset{C}{\text{enter}}(D)) = \text{Dest}(A, \overset{C}{\text{enter}}(D)) = \emptyset
\]

For \( tr \) is stuck until its ambient is dissolved while in \( A \)
Estimating Destination

\[ \text{Dest}(H, H_0 \text{ enter}(g)) = \begin{cases} g \uparrow & \text{if } H \cap H_0 \neq \emptyset \\ \emptyset & \text{else} \end{cases} \]

\[ \text{Dest}(H, g \text{ exit}(H_0)) = \begin{cases} H_0 & \text{if } g \in H \\ \emptyset & \text{else} \end{cases} \]

\[ \text{Dest}(H, a) = H \text{ otherwise} \]

\[ \text{Dest}(H, a \Diamond tr) = \text{Dest}(\text{Dest}(H, a), tr) \]

Trojan Horse:
\[ \text{Dest}(\#, \#^{CD} \text{ enter}(A) \uparrow^C \text{ enter}(D)) = \text{Dest}(A, ^C \text{ enter}(D)) = \emptyset \]

For \( tr \) is stuck until its ambient is dissolved while in \( A \)
\( tr \) is feasible from \( H \) if it can “execute on its own”:

\[ \text{Dest}(H, tr) \neq \emptyset \]
Estimating Destination

\[
\begin{align*}
\text{Dest}(H, H_0 \text{ enter}(g)) & = \text{if } H \cap H_0 \neq \emptyset \text{ then } g^\uparrow \text{ else } \emptyset \\
\text{Dest}(H, g \text{ exit}(H_0)) & = \text{if } g \in H \text{ then } H_0 \text{ else } \emptyset \\
\text{Dest}(H, a) & = H \text{ otherwise} \\
\text{Dest}(H, a \diamond tr) & = \text{Dest}(\text{Dest}(H, a), tr)
\end{align*}
\]

Trojan Horse:

\[
\text{Dest}(\#, \#^{CD} \text{ enter}(A) \, C \text{ enter}(D)) = \text{Dest}(A, C \text{ enter}(D)) = \emptyset
\]

For \(tr\) is stuck until its ambient is dissolved while in \(A\)

\(tr\) is feasible from \(H\) if it can “execute on its own”:

\[
\text{Dest}(H, tr) \neq \emptyset
\]

\(\text{put}(_)\) always preceding \(\text{get}(_)\), and vice versa.
For ambient name:

$$\text{amb}^g_H[\{g_i : b_i\}_{i \in I}]$$
Types

For ambient name:

\[ \text{amb}_H^g [\{g_i : b_i\}_{i \in I}] \]

Demands: \( H \neq \emptyset \) and \( \mathcal{I}(g, H) \) and \( \forall i \in I : \mathcal{O}(g, g_i) \)
Types

For ambient name:

\[ \text{amb}^g_H \{ g_i : b_i \}_{i \in I} \]

Demands: \( H \neq \emptyset \) and \( \mathcal{I}(g, H) \) and \( \forall i \in I : \mathcal{O}(g, g_i) \)

Special cases: \( \text{amb}^g_H \) (if \( I = \emptyset \)); \( \text{amb}^g_H[g_i : b_i] \) (if \( I = \{ i \} \))
Types

- For ambient name:
  \[ \text{amb}_H^g [\{g_i : b_i\}_{i \in I}] \]

  - Demands: \( H \neq \emptyset \) and \( \mathcal{I}(g, H) \) and \( \forall i \in I : \mathcal{O}(g, g_i) \)
  - Special cases: \( \text{amb}_H^g \) (if \( I = \emptyset \)); \( \text{amb}_H^g [g_i : b_i] \) (if \( I = \{ i \} \))

- For capability (such as in \( p \)):
  \[ \text{cap}[B] \]

  with \( B \) a behavior context:
  \[ \square \mid a.B \mid (b \mid B) \]
Types

For ambient name:

\[ \text{amb}^g_H[\{g_i : b_i\}_{i \in I}] \quad \text{amb}^=[=](\text{could split}) \]

Demands: \( H \neq \emptyset \) and \( \mathcal{T}(g, H) \) and \( \forall i \in I : \mathcal{O}(g, g_i) \)

Special cases: \( \text{amb}^g_H \) (if \( I = \emptyset \)); \( \text{amb}^g_H[g_i : b_i] \) (if \( I = \{i\} \))

For capability (such as in \( p \)):

\[ \text{cap}[B] \quad \text{cap}[\oplus] \]

with \( B \) a behavior context:

\[ \square | a.B | (b \mid B) \]

\[ B_1 \leq B_2 \text{ iff } \forall b: B_1[b] \leq B_2[b] \]
Selected Clauses, I

\[ \Delta, E \vdash_g 0 : \varepsilon \]
Selected Clauses, I

\[ \Delta, E \vdash_p 0 : \varepsilon \]

\[ \Delta, E \vdash_p P_1 : b_1 \quad \Delta, E \vdash_p P_2 : b_2 \]

\[ \Delta, E \vdash_p P_1 \parallel P_2 : b_1 \parallel b_2 \]
Selected Clauses, I

\[
\Delta, E \vdash_g 0 : \varepsilon
\]

\[
\Delta, E \vdash_g P_1 : b_1 \quad \Delta, E \vdash_g P_2 : b_2
\]

\[
\Delta, E \vdash_g P_1 \mid P_2 : b_1 \mid b_2
\]

\[
\forall i \in \{1 \ldots k\} : E \vdash M_i : \tau_i
\]

\[
\Delta, E \vdash_g \langle M_1 \ldots M_k \rangle : \text{put}(\times(\tau_1, \ldots, \tau_k))
\]
 Selected Clauses, I

\[
\begin{align*}
\Delta, E \vdash_g 0 : \varepsilon \\
\Delta, E \vdash_g P_1 : b_1 & \quad \Delta, E \vdash_g P_2 : b_2 \\
\Delta, E \vdash_g P_1 \parallel P_2 : b_1 \parallel b_2 \\
\forall i \in \{1 \ldots k\} : E \vdash M_i : \tau_i \\
\Delta, E \vdash_g \langle M_1 \ldots M_k \rangle : \text{put}(\times(\tau_1, \ldots, \tau_k)) \\
\Delta, E, n_1 : \tau_1, \ldots, n_k : \tau_k \vdash_g P : b \\
\Delta, E \vdash_g (n_1 \ldots n_k : \tau_1 \ldots \tau_k).P : \text{get}(\times(\tau_1, \ldots, \tau_k)).b
\end{align*}
\]
Motivation: we want the derived rule

\[
E \vdash n : \text{amb}^q_H[g_0 : b_0] \\
\hline
\Delta, E \vdash_{g_0} P : b \\
\hline
\Delta, E \vdash_{g_0} \text{open} \, n.P : g_0 \text{open}(g).(b_0 \mid b)
\]
Selected Clauses, II

Motivation: we want the derived rule

\[
E \vdash n : \text{amb}_H^g [g_0 : b_0] \\
\Delta, E \vdash_{g_0} P : b \\
\Delta, E \vdash_{g_0} \text{open} n.P : g_0 \text{open}(g).(b_0 \parallel b)
\]

Therefore we define

\[
E \vdash M : \text{amb}_H^g \{g_i : b_i\}_{i \in I} \\
E \vdash \text{open} M : \text{cap}[g_i \text{open}(g).(b_i \parallel \Box)]
\]

\[
E \vdash M : \text{cap}[B] \\
\Delta, E \vdash_{g} P : b \\
\Delta, E \vdash_{g} M.P : B[b]
\]
Motivation: we want the derived rule

\[
E \vdash n : \text{amb}^g_H[g_0 : b_0]
\]
\[
E \vdash \text{open } n : \text{cap}[g_0 \text{open}(g).(b_0 | \Box)] \quad \Delta, E \vdash_{g_0} P : b
\]
\[
\Delta, E \vdash_{g_0} \text{open } n.P : g_0 \text{open}(g).(b_0 | b)
\]

Therefore we define

\[
E \vdash M : \text{amb}^g_H[\{g_i : b_i\}_{i \in I}]
\]
\[
E \vdash \text{open } M : \text{cap}[g_i \text{open}(g).(b_i | \Box)]
\]
\[
E \vdash M : \text{cap}[B] \quad \Delta, E \vdash_{g} P : b
\]
\[
\Delta, E \vdash_{g} M.P : B[b]
\]
Selected Clauses, II

Motivation: we want the derived rule

\[
E \vdash n : \text{amb}^g_H [g_0 : b_0]
\]

\[
E \vdash \text{open } n : \text{cap}[g_0 \text{open}(g). (b_0 \mid \square)] \quad \Delta, E \vdash_{g_0} P : b
\]

\[
\Delta, E \vdash_{g_0} \text{open } n. P : g_0 \text{open}(g). (b_0 \mid b)
\]

Therefore we define

\[
E \vdash M : \text{amb}^g_H [\{g_i : b_i\}_{i \in I}]
\]

\[
E \vdash \text{open } M : \text{cap}[g_i \text{open}(g). (b_i \mid \square)]
\]

\[
E \vdash M : \text{cap}[B] \quad \Delta, E \vdash_g P : b
\]

\[
\Delta, E \vdash_g M. P : B[b]
\]

\[
E \vdash M : \text{amb}^g_H [br]
\]

\[
E \vdash \text{in } M : \text{cap}^H[\text{enter}(g). \square]
\]
Clause for Ambients

\[ E \vdash M : \text{amb} \quad \Delta, E \vdash P : \]  
\[ \Delta, E \vdash_\_ M[P] : \varepsilon \]  

provided
Clause for Ambients

\[
E \vdash M : \text{amb}^g \begin{array}{c} \Delta, E \vdash_g P : b \\ \Delta, E \vdash_\_ M[P]^\xi : \varepsilon \end{array}
\]

provided

1. \( \text{group}(\xi) = g \)
2. \( \Delta(\xi) = b \)
Clause for Ambients

\[
E \vdash M : \text{amb}^g [ \quad ] \quad \Delta, E \vdash_g P : b
\]

\[
\Delta, E \vdash \_ M[P]\xi : \varepsilon
\]

provided

- \(\text{group}(\xi) = g\)
- \(\Delta(\xi) = b\)

- for all \(tr_1 \diamond tr_2 \in b\) such that \(tr_1\) is feasible from \(g^\uparrow\) the following properties hold with \(H_1 = \text{Dest}(g^\uparrow, tr_1)\):
Clause for Ambients

\[ E \vdash M : \text{amb}_H^g \quad \Delta, E \vdash g \quad P : b \]

\[ \Delta, E \vdash _M[P]_\xi : \varepsilon \]

provided

- \( \text{group}(\xi) = g \)
- \( \Delta(\xi) = b \)
- for all \( tr_1 \vdash tr_2 \in b \) such that \( tr_1 \) is feasible from \( g \uparrow \) the following properties hold with \( H_1 = \text{Dest}(g \uparrow, tr_1) \):
  - \( H_1 \subseteq H \)
Clause for Ambients

\[
\begin{align*}
E \vdash M : \text{amb}^g_H[\{g_i : b_i\}_{i \in I}] & \quad \Delta, E \vdash_g P : b \\
\Delta, E \vdash_\_ M[P]^{\xi} : \varepsilon
\end{align*}
\]

provided

- \( \text{group}(\xi) = g \)
- \( \Delta(\xi) = b \)

for all \( tr_1 \diamond tr_2 \in b \) such that \( tr_1 \) is feasible from \( g^\uparrow \) the following properties hold with \( H_1 = \text{Dest}(g^\uparrow, tr_1) \):

- \( H_1 \subseteq H \)
- for all \( i \in I : g_i \in H_1 \) implies \( \{tr_2\} \subseteq b_i \)
Clause for Ambients

\[
E \vdash M : \text{amb}^g_H[\{g_i : b_i\}_{i \in I}] \quad \Delta, E \vdash g \ P : b
\]

\[
\Delta, E \vdash_- M[P]^{\xi} : \varepsilon
\]

provided

- \(\text{group}(\xi) = g\)
- \(\Delta(\xi) = b\)
- for all \(tr_1 \diamond tr_2 \in b\) such that \(tr_1\) is feasible from \(g^\uparrow\) the following properties hold with \(H_1 = Dest(g^\uparrow, tr_1)\):
  - \(H_1 \subseteq H\)
  - for all \(i \in I\): \(g_i \in H_1\) implies \(\{tr_2\} \leq b_i\)
  - if \(tr_2\) takes the form \(\text{put}(\sigma_1) \ get(\sigma_2) \diamond tr_3\) then \(\sigma_1 \leq \sigma_2\)
  - if \(tr_2\) takes the form \(G\text{open}(\_\_) \diamond tr_3\) then \(g \in G\)
Semantic Soundness

Suppose that (for uniquely tagged processes)

\[ P \xrightarrow{\xi: \text{enter}} Q \]
Suppose that (for uniquely tagged processes)

\[ P \xrightarrow{\xi: \text{enter}} \chi \quad Q \]

\[ \Delta, E \vdash_g P : b \]
Semantic Soundness

Suppose that (for uniquely tagged processes)

\[ P \xrightarrow{\xi:\text{enter}} Q \]

\[ \Delta, E \vdash_g P : b \]

Then there exists \( \Delta' \) such that

\[ \Delta', E \vdash_g Q : b \]
Semantic Soundness

Suppose that (for uniquely tagged processes)

\[ P \xrightarrow{\xi;\text{enter} \chi} Q \]
\[ \Delta, E \vdash_g P : b \]

Then there exists \( \Delta' \) such that

\[ \Delta', E \vdash_g Q : b \]
\[ \Delta' \text{ agrees with } \Delta \text{ on } \text{dom}(\Delta) \setminus \{\xi\} \]
Semantic Soundness

Suppose that (for uniquely tagged processes)

\[ P \xrightarrow{\xi: \text{enter } \chi} Q \]
\[ \Delta, E \vdash_g P : b \]

Then there exists \( \Delta' \) such that

\[ \Delta', E \vdash_g Q : b \]
\[ \Delta' \text{ agrees with } \Delta \text{ on } \text{dom}(\Delta) \setminus \{\xi\} \]
\[ ^0\text{enter}(\text{group}(\chi)).\Delta'(\xi) \leq \Delta(\xi) \]
Semantic Soundness

Suppose that (for uniquely tagged processes)

\[ P \xrightarrow[\xi:]{} \text{enter } \chi \quad Q \]

\[ \Delta, E \vdash_g P : b \]

Then there exists \( \Delta' \) such that

\[ \Delta', E \vdash_g Q : b \]

\[ \Delta' \text{ agrees with } \Delta \text{ on } \text{dom}(\Delta) \setminus \{\xi\} \]

\[ \emptyset \text{enter}(\text{group}(\chi)).\Delta'(\xi) \leq \Delta(\xi) \]

Similarly for other constructs
Security properties

If $P$ is typable, the following properties hold:

**Static** Assume that inside $P$ an ambient of group $g_0$ is directly enclosed in an ambient of group $g$.

Then $\mathcal{I}(g_0, g)$ does hold.
Security properties

If \( P \) is typable, the following properties hold:

**Static** Assume that inside \( P \) an ambient of group \( g_0 \) is directly enclosed in an ambient of group \( g \).

Then \( \mathcal{I}(g_0, g) \) does hold.

**Dynamic** Assume that \( P \xrightarrow{\xi: \text{open}} Q \).

Then \( \mathcal{O}(\text{group}(\chi), \text{group}(\xi)) \).
Type Checking

\[ b_1 \leq b_2 \text{ is decidable} \]
Type Checking

\( b_1 \leq b_2 \) is decidable

Construct generalized difference automaton \( b_1 \setminus b_2 \) and check for emptiness
Type Checking

\[ b_1 \leq b_2 \text{ is decidable} \]

\[ t_1 \leq t_2 \text{ is decidable} \]
Type Checking

\( b_1 \leq b_2 \) is decidable

\( t_1 \leq t_2 \) is decidable

Key result: given \( B_1 \) and \( B_2 \), there exists \( b_0 \) such that

\[
\text{cap}[B_1] \leq \text{cap}[B_2] \iff \forall b : B_1[b] \leq B_2[b] \\
\iff B_1[b_0] \leq B_2[b_0]
\]
Type Checking

\( b_1 \leq b_2 \) is decidable

\( t_1 \leq t_2 \) is decidable

\((H_0, b) \xrightarrow{g} (H, gbs)\) is decidable
Type Checking

\[ b_1 \leq b_2 \text{ is decidable} \]

\[ t_1 \leq t_2 \text{ is decidable} \]

\[ (H_0, b) \xrightarrow{g} (H, gbs) \text{ is decidable} \]

Construct automaton, annotated such that if state \( q \) is reachable via \( tr \) then the annotation of \( q \) includes \( \text{Dest}(H_0, tr) \)
Future work

Type Inference
Future work

Type Inference

A type reconstruction algorithm exists, succeeding for a large class of processes.
Future work

Type Inference

A type reconstruction algorithm exists, succeeding for a large class of processes, but requiring certain information from the user.
Future work

Type Inference

A type reconstruction algorithm exists, succeeding for a large class of processes, but requiring certain information from the user, and may suffer from state explosion.
Future work

Type Inference

A type reconstruction algorithm exists, succeeding for a large class of processes, but requiring certain information from the user, and may suffer from state explosion.

Implementation
Future work

Type Inference

A type reconstruction algorithm exists, succeeding for a large class of processes, but requiring certain information from the user, and may suffer from state explosion.

Implementation

Is the average case also exponential? If so, what will be suitable trade-offs?
Future work

Type Inference

A type reconstruction algorithm exists, succeeding for a large class of processes, but requiring certain information from the user, and may suffer from state explosion.

Implementation

Is the average case also exponential? If so, what will be suitable trade-offs?

Relationship to other approaches
Future work

Type Inference

A type reconstruction algorithm exists, succeeding for a large class of processes, but requiring certain information from the user, and may suffer from state explosion.

Implementation

Is the average case also exponential? If so, what will be suitable trade-offs?

Relationship to other approaches

Can we construct formal embeddings? Or prove that they are incomparable?