Static Analysis of Services

Flemming Nielson

Technical University of Denmark

GLOBAN 2008

Abstract

- Static Analysis has its origins in improving the efficiency of interpreters and compiled code but is gaining increased importance due to its ability to validate important aspects of systems behaviour. For several years static analysis have been used to analyse process algebras for a variety of communication and mobility paradigms in order to establish communication invariants and correctness of communication protocols.
- A new challenge arises from the use of process algebras for modelling services - ensuring for example that service invocations do not interfere with each other. In these lectures we provide the foundations for performing static analysis for proces algebras including considerations of correctness, adequacy and complexity and touching upon the distinction between compositionality and global characteristic features.

- Thanks to Hanne Riis Nielson, Henrik Pilegaard, Han Gao.

Static Analysis of Services

Overview:

The Grand View

The Grand View

Eventually we want to analyse service oriented systems written in suitable programming languages for certain properties of interest.

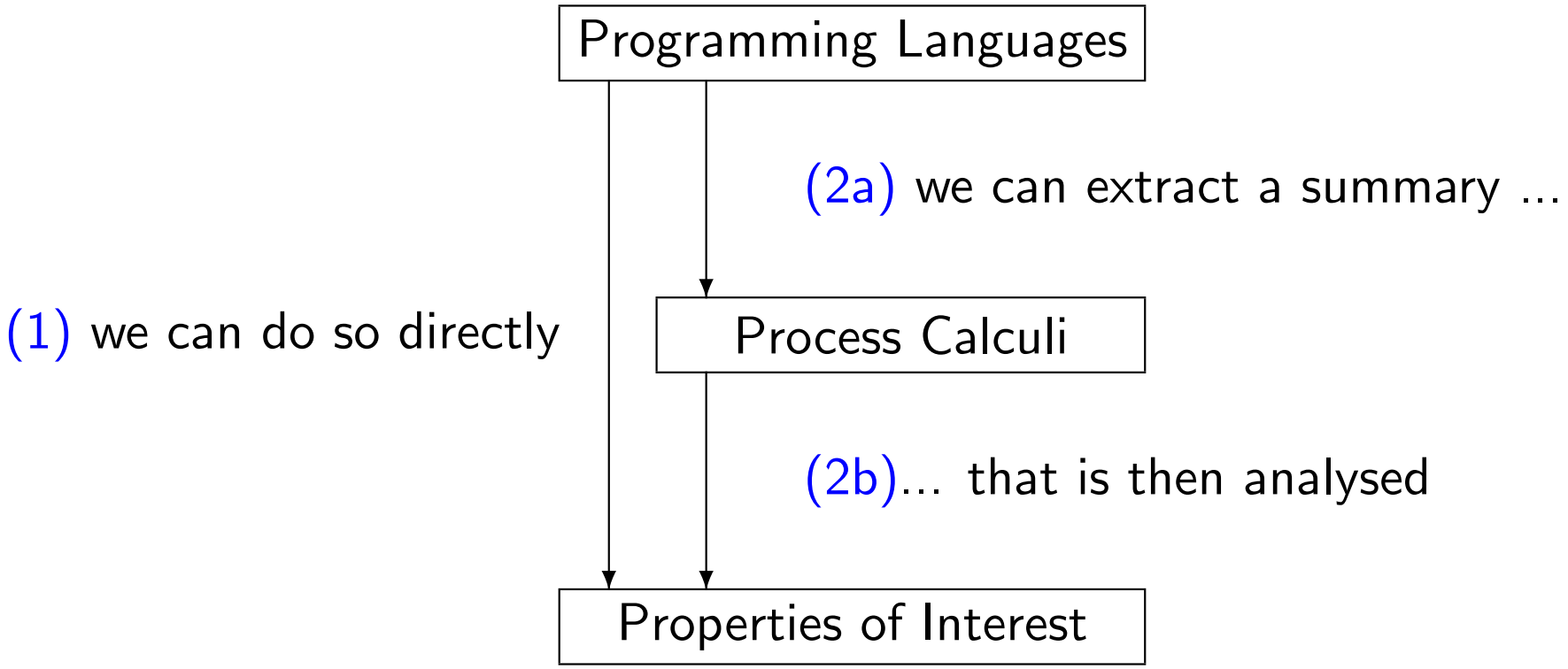
There are at least two approches to doing so:

(1) The traditional approach is to perform the analysis directly on the programming language – think optimizing compilers.

(2a) A more modern way first extracts a behaviour from the programming language, perhaps in the form of a term in a process calculus,

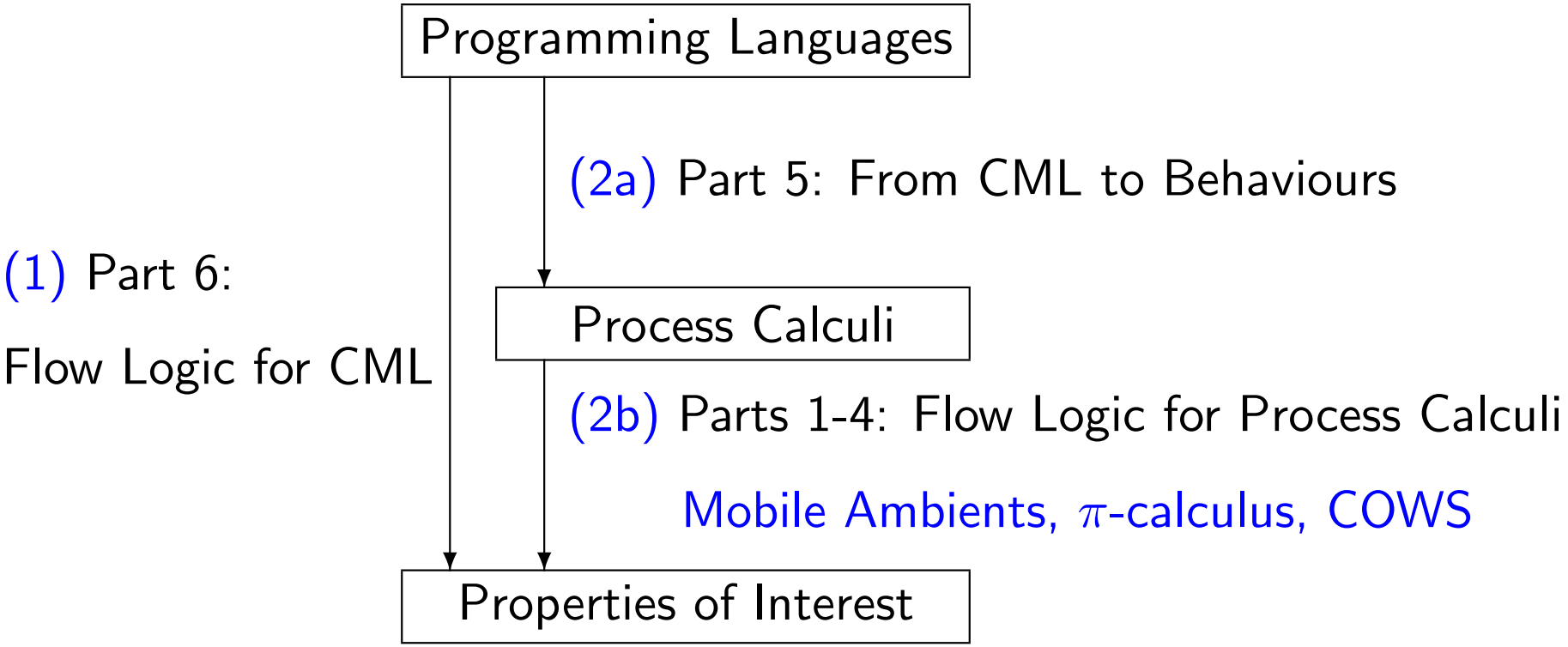
(2b) that is then analysed for the properties of interest.

The Grand View



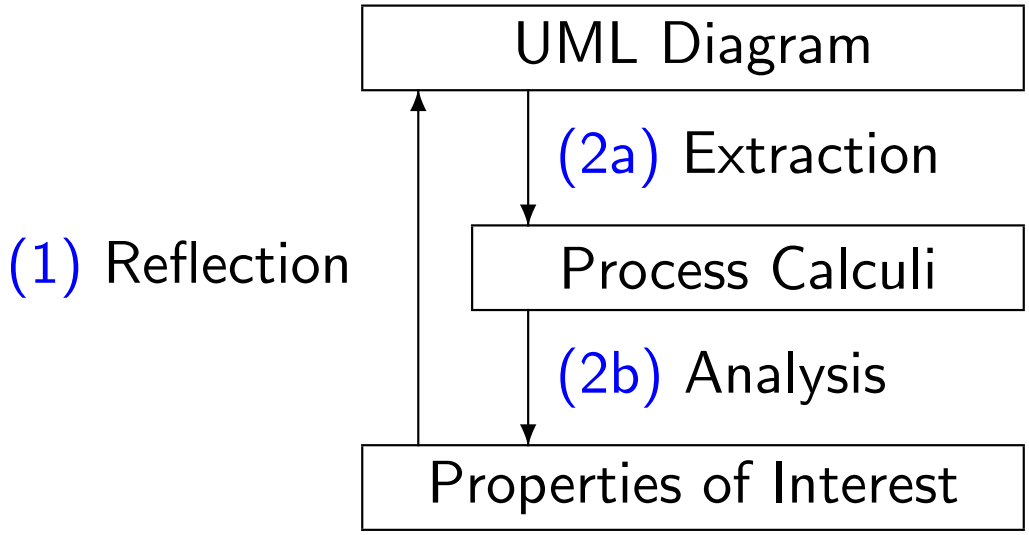
The Grand View

These lectures focus on the more modern approach while briefly illustrating the other:



The Grand View

The more modern approach works well with modern software development:
As shown in the DEGAS EU-project (leading up to SENSORIA) one can extract process calculi descriptions from UML diagrams, analyse them, and reflect the results back into the UML diagram.



Static Analysis of Services

Part 1:

Flow Logic for Mobile Ambients

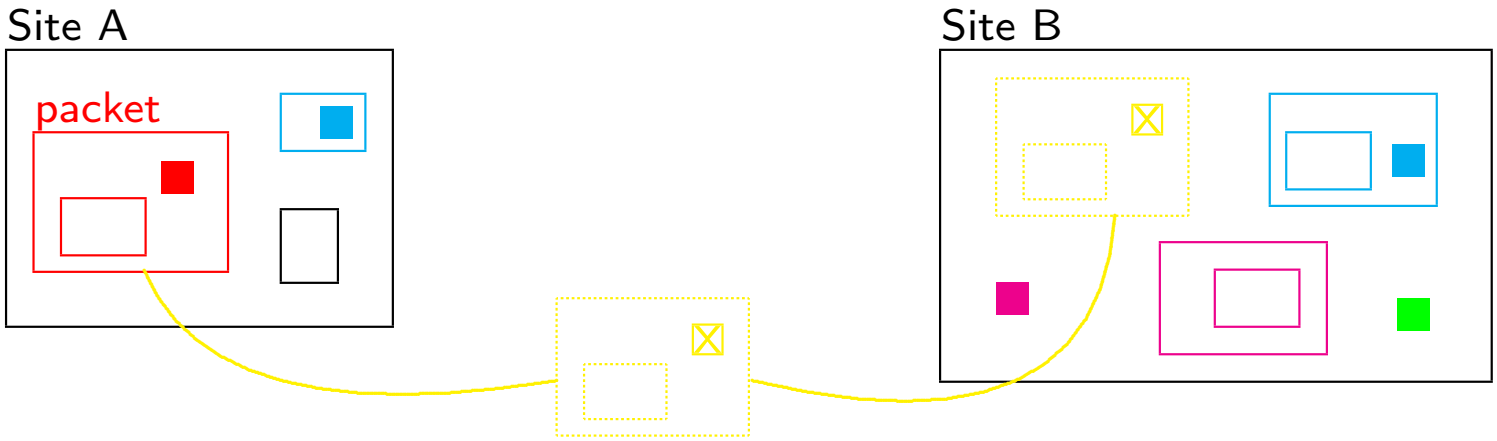
Mobile Ambients

- Overview: the ambient view of computation
- Syntax: processes and capabilities
- Semantics: structural congruence and transition relation

The ambient view of computation

An ambient is a bounded place where computations take place

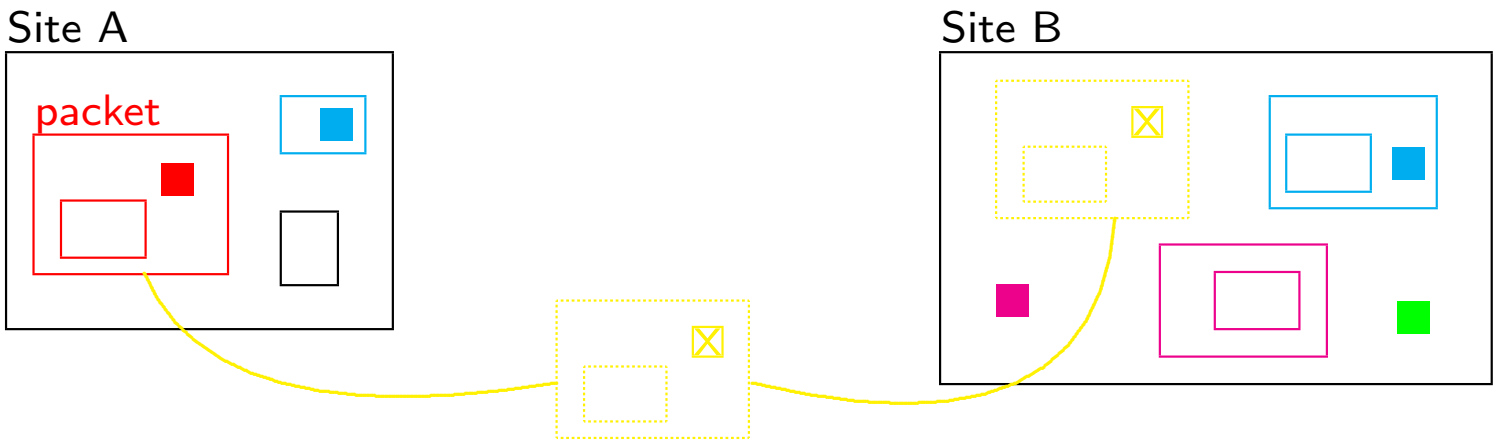
- the boundary determines what is inside and what is outside
- the ambient moves as a whole
- example ambients: applets, agents, laptops, $\cdots$



The ambient hierarchy

Ambients can be nested inside oneanother forming a tree structure

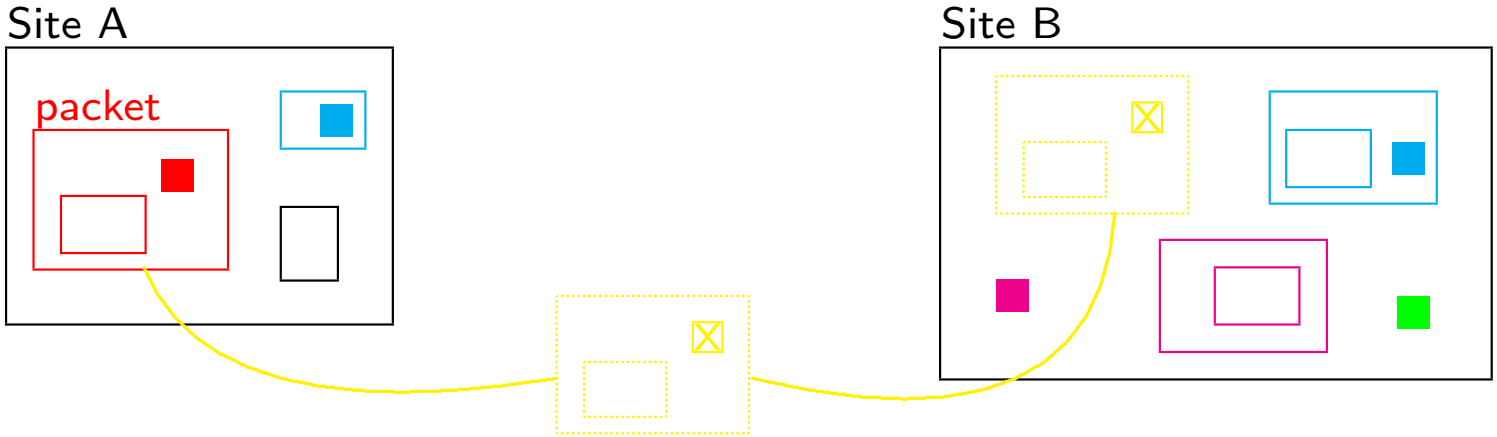
- mobility is represented as navigation within this hierarchy of ambients
- example: to move a packet from one site to another we must first remove it from the enclosing ambient and then insert it in the new enclosing ambient



Controlling the ambients

Each ambient contains a number of multi-threaded running processes

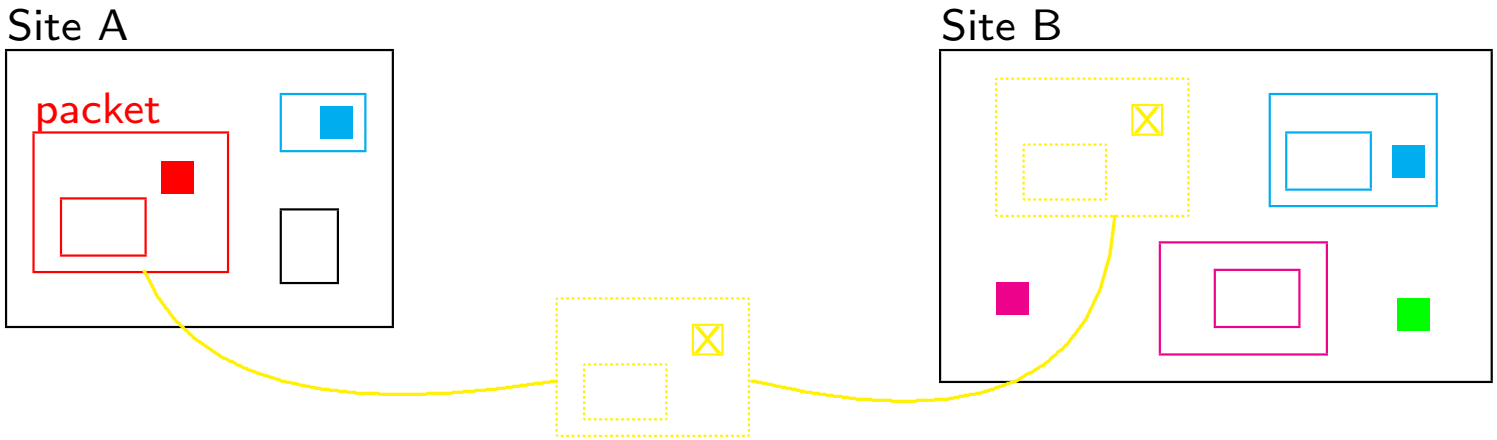
- the top-level processes of an ambient have direct control over it and can instruct it to move and thereby change the future behaviour of its processes and subambients
- the processes of subambients have no control over the parent
- processes continue running while being moved



Naming ambients

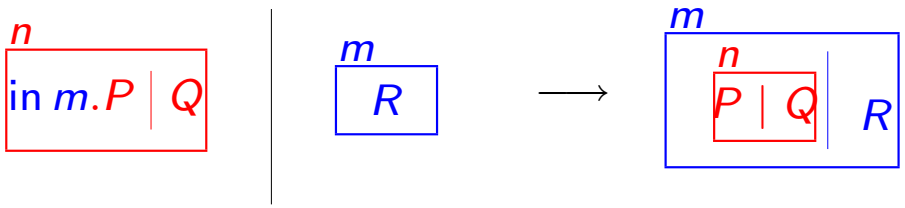
Each ambient has a name

- only the name can be used to control the access to the ambient: entry, exit, communication, etc.
- ambient names are unforgeable

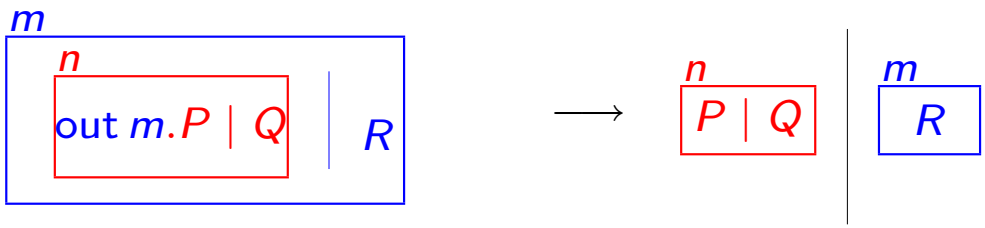


Mobility primitives

Move into an ambient:

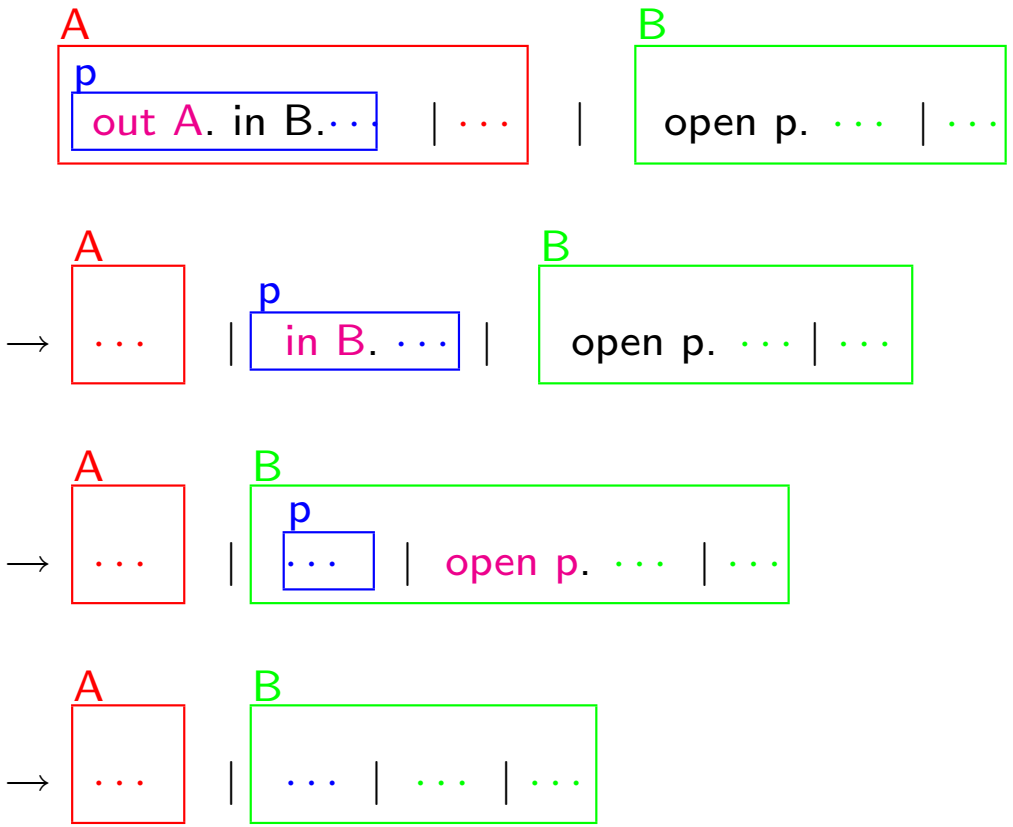


Move out of an ambient:



Dissolve an ambient:

Example: a packet on a network



the packet p moves out of site A

... and then into site B

... where it is dissolved

Example: kidnapping an ambient

$(\nu\ \mathsf{k})(\ \overset{\mathsf{k}}{\boxed{}}\ \mid \mathsf{go}\ (\mathsf{in}\ \mathsf{n}).\ \mathsf{in}\ \mathsf{k}\ ) \mid \overset{\mathsf{n}}{\boxed{P}}$

the ambient in k moves into n

$\rightarrow (\nu\ \mathsf{k})\ (\ \overset{\mathsf{k}}{\boxed{}}\ \mid \overset{\mathsf{n}}{\boxed{\mathsf{in}\ \mathsf{k}\ \mid\ P}}\ )$

the ambient n moves into k

$\rightarrow (\nu\ \mathsf{k})\ (\ \overset{\mathsf{k}}{\boxed{\overset{\mathsf{n}}{\boxed{P}}}}\ )$

the name k is private so nobody can interact with n

Syntax of Mobile Ambients

Processes:

P	$::=$	$(\nu\, n\colon \mu)\, P$	a process with private name n in group μ
	$\mid$	$(\nu\, \mu)\, P$	a new group named μ with its scope
	$\mid$	0	inactive process
	$\mid$	$P_1 \mid P_2$	two concurrent processes
	$\mid$	$!P$	any number of occurrences of P
	$\mid$	$n\,[P]$	ambient named n
	$\mid$	$M.P$	a capability M followed by P

Capabilities

M	$::=$	in n	move the enclosing ambient into a sibling named n
	$\mid$	out n	move the enclosing ambient out of a parent named n
	$\mid$	open n	dissolve a sibling ambient named n

Semantics of Mobile Ambients

- Structural congruence relation: $P \equiv Q$

  Examples: $!P \equiv !P \mid P$

  $P \equiv Q \quad \Rightarrow \quad n\,[P] \equiv n\,[Q]$

- Transition relation: $P \rightarrow Q$

  Examples: $n\,[\text{in}\ m.\ P \mid Q] \mid m\,[R] \quad \rightarrow \quad m\,[n\,[P \mid Q] \mid R]$

  $P \equiv P' \quad \wedge \quad P' \rightarrow Q' \quad \wedge \quad Q' \equiv Q \quad \Rightarrow \quad P \rightarrow Q$

Structural congruence relation

$P \equiv P$
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$
$P \equiv Q \Rightarrow Q \equiv P$

$P \equiv Q \Rightarrow (\nu\, n\colon \mu)\, P \equiv (\nu\, n\colon \mu)\, Q$
$P \equiv Q \Rightarrow (\nu\, \mu)\, P \equiv (\nu\, \mu)\, Q$
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$
$P \equiv Q \Rightarrow !P \equiv !Q$
$P \equiv Q \Rightarrow n\,[P] \equiv n\,[Q]$
$P \equiv Q \Rightarrow m.\, P \equiv m.\, Q$

$P \mid Q \equiv Q \mid P$
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
$P \mid \mathbf{0} \equiv P$

$!P \equiv P \mid !P$
$!\mathbf{0} \equiv \mathbf{0}$
$(\nu\, n\colon \mu)\, \mathbf{0} \equiv \mathbf{0}$
$(\nu\, \mu)\, \mathbf{0} \equiv \mathbf{0}$

$(\nu\, n\colon \mu)\,(\nu\, n'\colon \mu')\, P \equiv (\nu\, n'\colon \mu')\,(\nu\, n\colon \mu)\, P$ if $n \neq n'$
$(\nu\, \mu)\,(\nu\, \mu')\, P \equiv (\nu\, \mu')\,(\nu\, \mu)\, P$
$(\nu\, n\colon \mu)\,(\nu\, \mu')\, P \equiv (\nu\, \mu')\,(\nu\, n\colon \mu)\, P$ if $\mu \neq \mu'$

$(\nu\, n\colon \mu)\,(P \mid Q) \equiv P \mid (\nu\, n\colon \mu)\, Q$ if $n \notin \underline{\mathrm{fn}}(P)$
$(\nu\, \mu)\,(P \mid Q) \equiv P \mid (\nu\, \mu)\, Q$ if $\mu \notin \underline{\mathrm{fg}}(P)$

$(\nu\, n' : \mu)\,(n\,[P]) \equiv n\,[(\nu\, n' : \mu)\, P]$ if $n \neq n'$
$(\nu\, \mu)\,(n\,[P]) \equiv n\,[(\nu\, \mu)\, P]$

$(\nu\, n\colon \mu)\, P \equiv (\nu\, n' : \mu)\,(P\{n \leftarrow n'\})$ if $n' \notin \underline{\mathrm{fn}}(P)$
AVOID $(\nu\, \mu)\, P \equiv (\nu\, \mu')\,(P\{\mu \leftarrow \mu'\})$ if $\mu' \notin \underline{\mathrm{fg}}(P)$

Transition relation

$$P \rightarrow Q \Rightarrow (\nu\, n\colon \mu)\, P \rightarrow (\nu\, n\colon \mu)\, Q$$

$$P \rightarrow Q \Rightarrow (\nu\, \mu)\, P \rightarrow (\nu\, \mu)\, Q$$

$$P \rightarrow Q \Rightarrow n\,[P] \rightarrow n\,[Q]$$

$$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$$

$$P \equiv P' \wedge P' \rightarrow Q' \wedge Q' \equiv Q \Rightarrow P \rightarrow Q$$

$$n\,[\mathsf{in}\ m.\ P \mid Q] \mid m\,[R] \ \rightarrow\ m\,[n\,[P \mid Q] \mid R]$$

$$m\,[n\,[\mathsf{out}\ m.\ P \mid Q] \mid R] \ \rightarrow\ n\,[P \mid Q] \mid m\,[R]$$

$$\mathsf{open}\ n.\ P \mid n\,[Q] \ \rightarrow\ P \mid Q$$

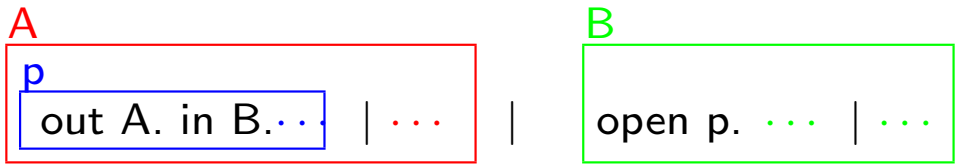
Static Analysis for Mobile Ambients

- The aims of the analysis
- The nature of approximation

The aims of the analysis

1. Which ambients may turn up inside other ambients during the execution?
2. Which capabilities might be possessed by an ambient during the execution?

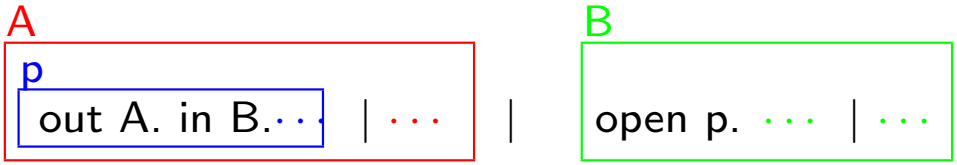
Example:



The exact answer (for 1):

– p will turn up inside A — holds initially
– p will turn up inside B — holds after two steps
but
– A and B never turns up inside p

The analysis in terms of groups



Groups:

p: $\mathbb{P}$acket

A, B: $\mathbb{S}$ite

The exact answer:

– p may turn up inside A
– p may turn up inside B
but
– A and B never turns up inside p

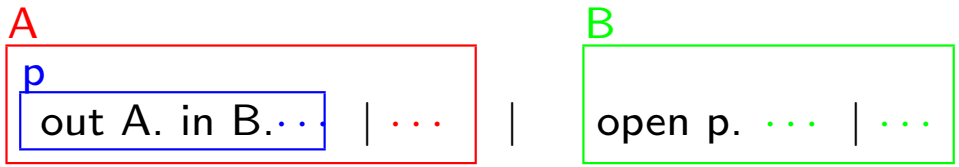
The analysis result with groups:

– $\mathbb{P}$ may turn up inside $\mathbb{S}$
but
– $\mathbb{S}$ never turns up inside $\mathbb{P}$

Assumptions

- groups cannot be renamed — they carry the analysis information
- groups are only allowed at the top-level — they are global

The nature of approximation



An acceptable and precise analysis result

– $\mathbb{P}$ may turn up inside $\mathbb{S}$
but
– $\mathbb{S}$ never turns up inside $\mathbb{P}$

An acceptable but imprecise analysis result

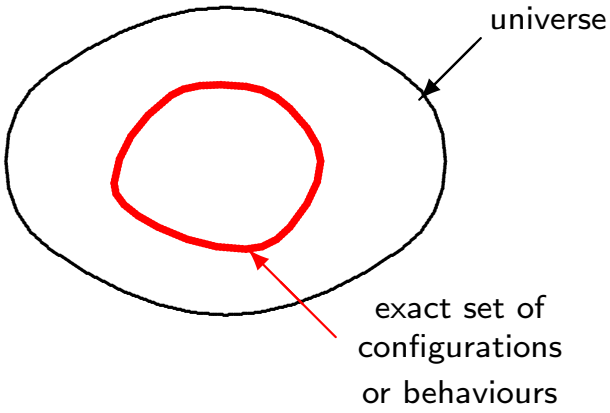
– $\mathbb{P}$ may turn up inside $\mathbb{S}$
– $\mathbb{S}$ may turn up inside $\mathbb{P}$

An unacceptable analysis result

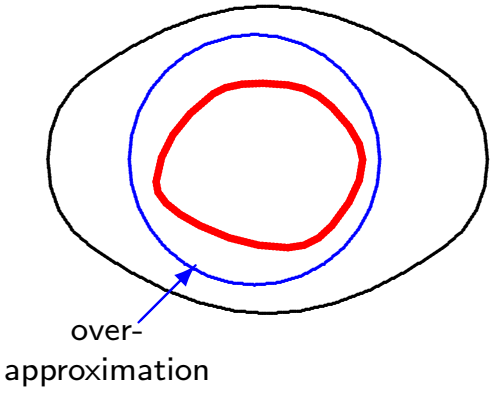
– $\mathbb{S}$ may turn up inside $\mathbb{P}$
but
– $\mathbb{P}$ never turns up inside $\mathbb{S}$

The nature of approximation

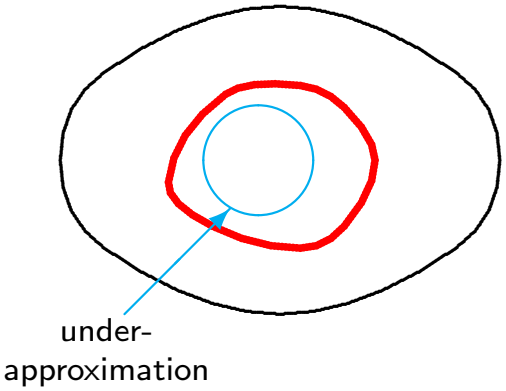
The exact world



Over-approximation



Under-approximation



Our analysis of mobile ambients: over-approximation

Flow Logic for Mobile Ambients

- Analysis estimates
- Analysis judgements
- The Flow Logic approach
- Syntax-directed definition
  - of the analysis of processes
  - of the analysis of capabilities: in, out, and open

Analysis estimates

The analysis estimate

$$\mathcal{I} : \textbf{Group} \rightarrow \mathcal{P}(\textbf{Group} \cup \textbf{Cap})$$

tells us for each ambient group $\mu \in \textbf{Group}$:

- which ambient groups may be inside an ambient in group μ
- which group capabilities may be possessed by an ambient in group μ

A group capability $M \in \textbf{Cap}$ is given by

$$M ::= \text{in}\,\mu \mid \text{out}\,\mu \mid \text{open}\,\mu$$

Analysis judgements

$$\mathcal{I} \models^{\mu}_{\Gamma} P$$

means that

$$\mathcal{I} : \textbf{Group} \rightarrow \mathcal{P}(\textbf{Group} \cup \textbf{Cap})$$

is an acceptable analysis estimate for the process P when it occurs inside an ambient from the group μ and when the ambients are in the groups specified by the group environment

$$\Gamma : \textbf{Name} \rightarrow \textbf{Group}$$

Hence $\mathcal{I} \models^{\mu}_{\Gamma} P$ is either true or false.

Example

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}\ ]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$$

holds for

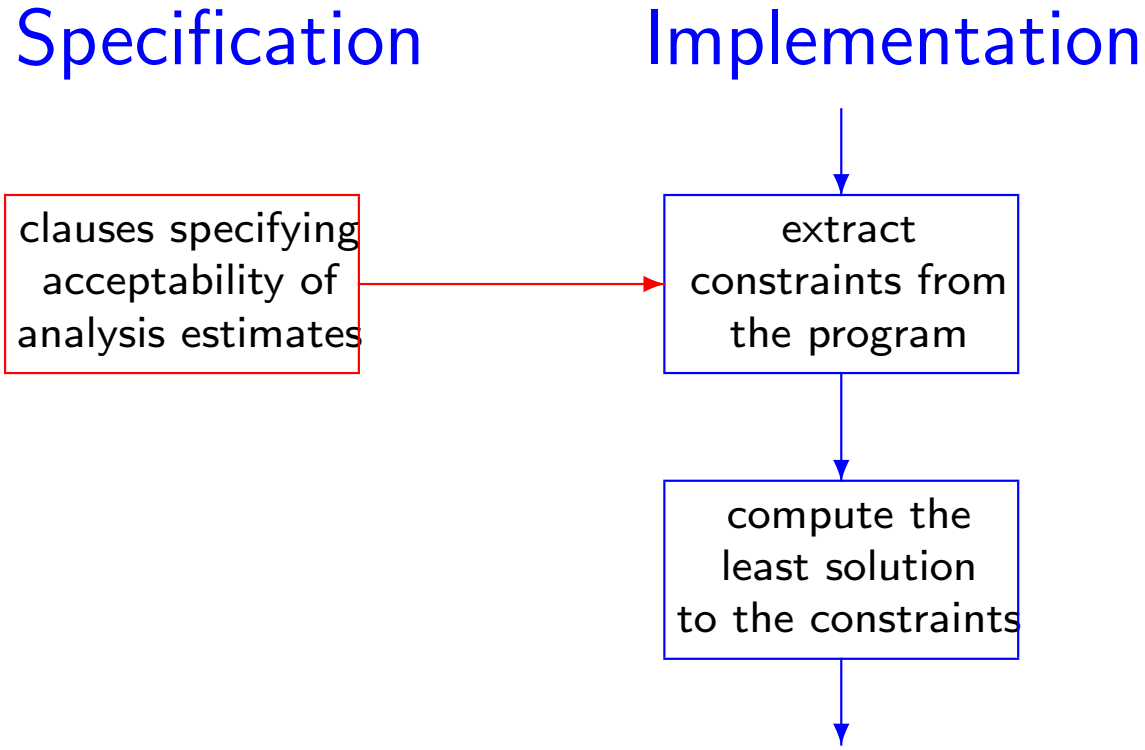
Γ	**Name** $\rightarrow$ **Group**
A	$\mathbb{S}$
B	$\mathbb{S}$
p	$\mathbb{P}$

$\mathcal{I}$	**Group** $\rightarrow \mathcal{P}$(**Group** $\cup$ **Cap**)
$\star$	$\{\mathbb{S}, \mathbb{P}\}$
$\mathbb{S}$	$\{\mathbb{P}, \mathbb{S}, \text{in } \mathbb{S}, \text{out } \mathbb{S}, \text{open } \mathbb{P}\}$
$\mathbb{P}$	$\{\text{in } \mathbb{S}, \text{out } \mathbb{S}\}$

The analysis result shows that

- $\mathbb{P}$ may turn up inside $\mathbb{S}$ — and so may $\mathbb{S}$
- $\mathbb{S}$ will never turn up inside $\mathbb{P}$ — and neither will $\mathbb{P}$

The Flow Logic approach



Acceptable analysis results

- Each acceptable analysis estimate for a composite program must also be an acceptable analysis estimate for its sub-programs; perhaps more imprecise than need be.

  syntax directed analysis of processes

- Each acceptable analysis estimate must mimick the semantics: if the semantics maps one configuration into another then it must be reflected in the analysis estimate.

  analysis of capabilities

Analysis of processes

$\mathcal{I} \models^{\star}_{\Gamma} (\nu\, n\colon \mu)\, P$	iff	$\mathcal{I} \models^{\star}_{\Gamma[n \mapsto \mu]} P$	update group environment; check process
$\mathcal{I} \models^{\star}_{\Gamma} (\nu\, \mu)\, P$	iff	$\mathcal{I} \models^{\star}_{\Gamma} P$	check process
$\mathcal{I} \models^{\star}_{\Gamma} \mathbf{0}$	iff	true	nothing to check
$\mathcal{I} \models^{\star}_{\Gamma} P_1 \mid P_2$	iff	$\mathcal{I} \models^{\star}_{\Gamma} P_1 \;\wedge\; \mathcal{I} \models^{\star}_{\Gamma} P_2$	check both branches
$\mathcal{I} \models^{\star}_{\Gamma} !P$	iff	$\mathcal{I} \models^{\star}_{\Gamma} P$	check process; ignore multiplicity
$\mathcal{I} \models^{\star}_{\Gamma} n\,[P]$	iff	$\mu \in \mathcal{I}(\star) \;\wedge\; \mathcal{I} \models^{\mu}_{\Gamma} P$ where $\mu = \Gamma(n)$	μ is inside $\star$; check process

Example: subambient

Checking

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}\ ]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$$

involves checking

$$\mathbb{S} \in \mathcal{I}(\star) \text{ and } \mathcal{I} \models^{\mathbb{S}}_{\Gamma} \mathsf{p}\,[\mathsf{out\ A.\ in\ B}\ ]$$

Γ	**Name** $\rightarrow$ **Group**
A	$\mathbb{S}$
B	$\mathbb{S}$
p	$\mathbb{P}$

$\mathcal{I}$	**Group** $\rightarrow \mathcal{P}$(**Group** $\cup$ **Cap**)
$\star$	$\{\mathbb{S}, \mathbb{P}\}$
$\mathbb{S}$	$\{\mathbb{P}, \mathbb{S}, \text{in } \mathbb{S}, \text{out } \mathbb{S}, \text{open } \mathbb{P}\}$
$\mathbb{P}$	$\{\text{in } \mathbb{S}, \text{out } \mathbb{S}\}$

Analysis of in-capability

$$
\begin{aligned}
\mathcal{I} \models^{\star}_{\Gamma} \text{in } n.\ P \ \underline{\text{iff}} \quad & \text{in } \mu \in \mathcal{I}(\star) \ \wedge \ \mathcal{I} \models^{\star}_{\Gamma} P \ \wedge \\
& \forall\ \mu^a, \mu^p : \ \text{in } \mu \in \mathcal{I}(\mu^a) \ \wedge \\
& \qquad \mu^a \in \mathcal{I}(\mu^p) \ \wedge \\
& \qquad \mu \in \mathcal{I}(\mu^p) \\
& \qquad \Rightarrow \ \mu^a \in \mathcal{I}(\mu) \\
& \text{where } \mu = \Gamma(n)
\end{aligned}
$$

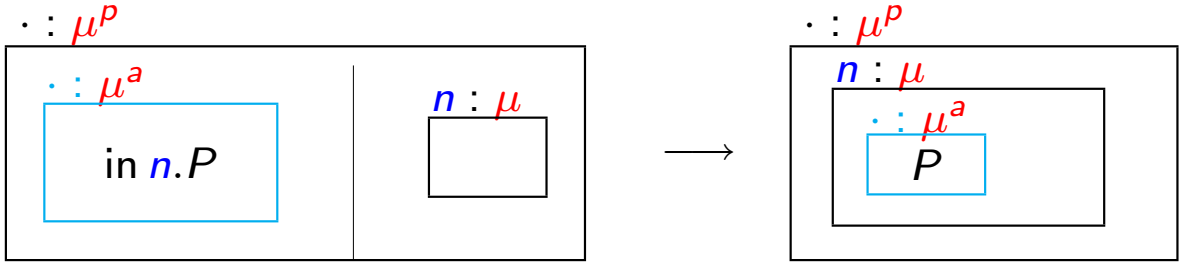
μ^a has the capability in μ

μ^p is parent of μ^a

μ^a has a sibling in group μ

μ^a may move into μ

Mimicking the semantics:



Example: in-capability

Checking

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}\ ]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$$

involves checking

$$\mathcal{I} \models^{\mathbb{P}}_{\Gamma} \mathsf{in\ B}$$

Γ	**Name** $\rightarrow$ **Group**
A	$\mathbb{S}$
B	$\mathbb{S}$
p	$\mathbb{P}$

$\mathcal{I}$	**Group** $\rightarrow \mathcal{P}$(**Group** $\cup$ **Cap**)
$\star$	$\{\mathbb{S}, \mathbb{P}\}$
$\mathbb{S}$	$\{\mathbb{P}, \mathbb{S}, \mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \mathsf{open}\ \mathbb{P}\}$
$\mathbb{P}$	$\{\mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}\}$

which holds because $\mathsf{in}\,\mathbb{S} \in \mathcal{I}(\mathbb{P})$ and

$$\mathsf{in}\,\mathbb{S} \in \mathcal{I}(\mu^a) \wedge \mu^a \in \mathcal{I}(\mu^p) \wedge \mathbb{S} \in \mathcal{I}(\mu^p) \Rightarrow \mu^a \in \mathcal{I}(\mathbb{S})$$

holds for all $(\mu^a, \mu^p) \in \{(\mathbb{S}, \star), (\mathbb{S}, \mathbb{S}), (\mathbb{P}, \star), (\mathbb{P}, \mathbb{S})\}$

Analysis of out-capability

$$
\begin{aligned}
\mathcal{I} \models^{\star}_{\Gamma} \text{out } n.\ P \ \underline{\text{iff}} \quad & \text{out } \mu \in \mathcal{I}(\star) \ \wedge \ \mathcal{I} \models^{\star}_{\Gamma} P \ \wedge \\
& \forall\ \mu^a, \mu^g : \ \text{out } \mu \in \mathcal{I}(\mu^a) \wedge \\
& \qquad \mu^a \in \mathcal{I}(\mu) \ \wedge \\
& \qquad \mu \in \mathcal{I}(\mu^g) \\
& \qquad \Rightarrow \ \mu^a \in \mathcal{I}(\mu^g) \\
& \text{where } \mu = \Gamma(n)
\end{aligned}
$$

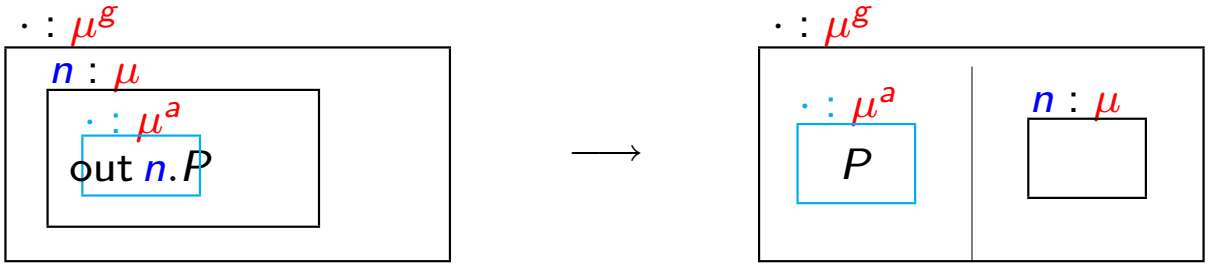
μ^a has the capability out μ

μ is parent of μ^a

μ^g is grandparent of μ

μ^a may move out of μ

Mimicking the semantics:



Example: out-capability

Checking

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out}\,\mathsf{A}.\ \mathsf{in}\,\mathsf{B}]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open}\,\mathsf{p}]$$

involves checking

$$\mathcal{I} \models^{\mathbb{P}}_{\Gamma} \mathsf{out}\,\mathsf{A}.\ \mathsf{in}\,\mathsf{B}$$

Γ	**Name** $\rightarrow$ **Group**
A	$\mathbb{S}$
B	$\mathbb{S}$
p	$\mathbb{P}$

$\mathcal{I}$	**Group** $\rightarrow \mathcal{P}$(**Group** $\cup$ **Cap**)
$\star$	$\{\mathbb{S}, \mathbb{P}\}$
$\mathbb{S}$	$\{\mathbb{P}, \mathbb{S}, \mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \mathsf{open}\ \mathbb{P}\}$
$\mathbb{P}$	$\{\mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}\}$

which holds because $\mathcal{I} \models^{\mathbb{P}}_{\Gamma} \mathsf{in}\,\mathsf{B}$ and $\mathsf{out}\,\mathbb{S} \in \mathcal{I}(\mathbb{P})$ and

$$\mathsf{out}\,\mathbb{S} \in \mathcal{I}(\mu^a) \wedge \mu^a \in \mathcal{I}(\mathbb{S}) \wedge \mathbb{S} \in \mathcal{I}(\mu^g) \Rightarrow \mu^a \in \mathcal{I}(\mu^g)$$

holds for all $(\mu^a, \mu^g) \in \{(\mathbb{S}, \star), (\mathbb{S}, \mathbb{S}), (\mathbb{P}, \star), (\mathbb{P}, \mathbb{S})\}$

Analysis of open-capability

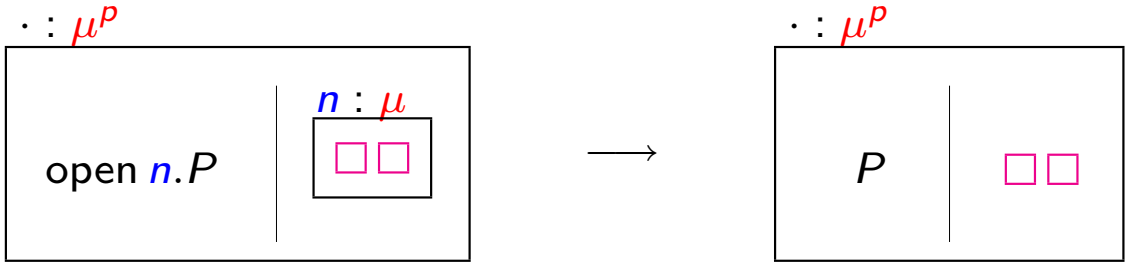
$$\mathcal{I} \models^{\star}_{\Gamma} \text{open } n.\ P \ \underline{\text{iff}} \quad \begin{array}{l} \text{open } \mu \in \mathcal{I}(\star) \ \wedge \ \mathcal{I} \models^{\star}_{\Gamma} P \ \wedge \\ \forall\ \mu^p : \ \text{open } \mu \in \mathcal{I}(\mu^p) \ \wedge \\ \qquad \mu \in \mathcal{I}(\mu^p) \\ \qquad \Rightarrow \ \mathcal{I}(\mu) \subseteq \mathcal{I}(\mu^p) \\ \text{where } \mu = \Gamma(n) \end{array}$$

μ^p has the capability open μ

μ is sibling

anything in μ may appear in μ^p

Mimicking the semantics:



Example: open-capability

Checking

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out}\,\mathsf{A}.\ \mathsf{in}\,\mathsf{B}\ ]]\ \mid\ \mathsf{B}\,[\mathsf{open}\,\mathsf{p}]$$

involves checking

$$\mathcal{I} \models^{\mathbb{S}}_{\Gamma} \mathsf{open}\,\mathsf{p}$$

Γ	**Name** $\rightarrow$ **Group**
A	$\mathbb{S}$
B	$\mathbb{S}$
p	$\mathbb{P}$

$\mathcal{I}$	**Group** $\rightarrow \mathcal{P}$(**Group** $\cup$ **Cap**)
$\star$	$\{\mathbb{S}, \mathbb{P}\}$
$\mathbb{S}$	$\{\mathbb{P}, \mathbb{S}, \mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \mathsf{open}\ \mathbb{P}\}$
$\mathbb{P}$	$\{\mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}\}$

which holds because $\mathsf{open}\,\mathbb{P} \in \mathcal{I}(\mathbb{S})$ and

$$\mathsf{open}\,\mathbb{P} \in \mathcal{I}(\mu^p) \wedge \mathbb{P} \in \mathcal{I}(\mathbb{S}) \Rightarrow \mathcal{I}(\mathbb{P}) \subseteq \mathcal{I}(\mu^p)$$

holds for $\mu^p = \mathbb{S}$

Properties of the analysis

- Semantic correctness
  - We err on the safe side!
- Moore family property
  - We have best analysis results!

Semantic correctness

The analysis estimate is preserved during the execution:

$$
\begin{array}{ccccccccc}
P_1 & \rightarrow & P_2 & \rightarrow & \cdots & \rightarrow & P_i & \rightarrow & \cdots \\
\Updownarrow \models^\star_\Gamma & & \Updownarrow \models^\star_\Gamma & & & & \Updownarrow \models^\star_\Gamma & & \\
\mathcal{I} & & \mathcal{I} & & \cdots & & \mathcal{I} & & \cdots
\end{array}
$$

Subject reduction result:

If $\mathcal{I} \models^\star_\Gamma P$ *and* $P \rightarrow^* Q$ *then* $\mathcal{I} \models^\star_\Gamma Q$

Familiar from type systems

Proof of subject reduction result

Lemma: The analysis is invariant under the structural congruence:

$$\text{If} \quad P \equiv Q \quad \text{then} \quad \mathcal{I} \models^{\star}_{\Gamma} P \text{ if and only if } \mathcal{I} \models^{\star}_{\Gamma} Q$$

The proof is by induction on the inference of $P \equiv Q$

Theorem: The analysis is preserved under the transition relation:

$$\text{If} \quad P \rightarrow Q \quad \text{and} \quad \mathcal{I} \models^{\star}_{\Gamma} P \quad \text{then} \quad \mathcal{I} \models^{\star}_{\Gamma} Q$$

The proof is by induction on the inference of $P \rightarrow Q$

Moore family property

All processes can be analysed and has a least (best) analysis result:

The set $\{\mathcal{I} \mid \mathcal{I} \models^{\star}_{\Gamma} P\}$ *is a Moore family*

If $\mathcal{Y} \subseteq \{\mathcal{I} \mid \mathcal{I} \models^{\star}_{\Gamma} P\}$ then $\sqcap\mathcal{Y} \models^{\star}_{\Gamma} P$ where $(\sqcap\mathcal{Y})(\mu) = \bigcap\{\mathcal{I}(\mu) \mid \mathcal{I} \in \mathcal{Y}\}$

The proof is by structural induction on P.
Corollaries:

- All processes can be analysed:
  — a Moore family cannot be empty
- All processes has a least (best) analysis result:
  — a Moore family has a least element

Suggested Reading

- H. Riis Nielson, F. Nielson, M. Buchholtz: Security for Mobility. In *Foundations of Security Analysis and Design II*, SLNCS 2946, pages 207 – 266, Springer, 2004.

  Section 2 contains the "semantics-directed" way of "reading off" the Flow Logic from the Operational Semantics that was illustrated here.

  Sections 3, 4 and 5 cover Discretionary Access Control, Mandatory Access Control and Cryptographic Protocols – all in terms of Mobile Ambients and Flow Logic.

Static Analysis of Services

Part 2:

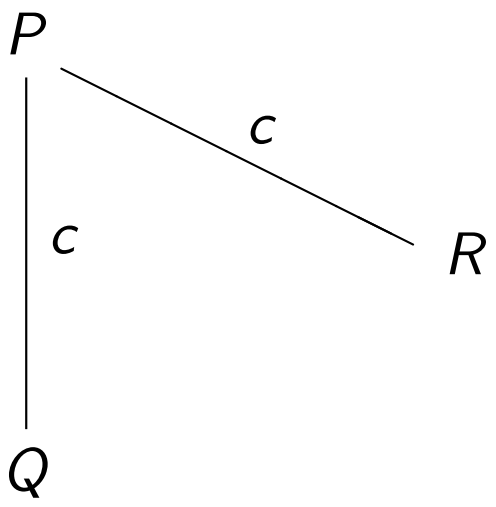
Flow Logic for π-Calculus

The π-calculus

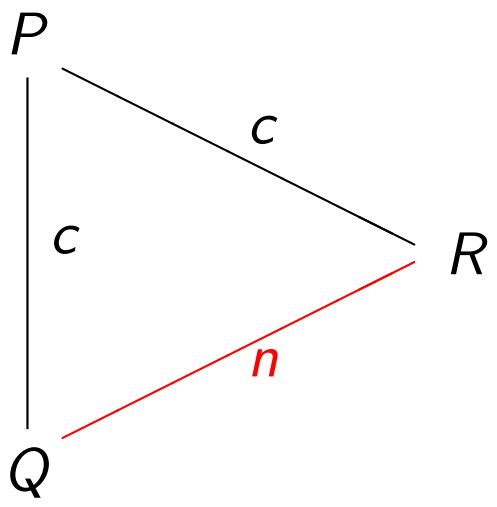
Communication model:

- a number of processes exchanging names by communicating over channels
- this can change the connectivity of the processes

Initially:



Eventually:



Syntax and Semantics of π-calculus

Syntax:

Processes: $P \quad ::= \quad (\text{new } n)P \mid P_1 \mid P_2 \mid !P \mid \Sigma_{i \in I} \pi_i . P_i$

Actions: $\pi \quad ::= \quad \underbrace{u\langle \vec{v} \rangle}_{\text{output}} \mid \underbrace{u(\vec{x})}_{\text{input}} \mid \underbrace{\tau}_{\text{internal}}$

Semantics:

- Structural congruence relation: $P \equiv Q$
- Reduction relation: $P \rightarrow Q$

Structural congruence relation

Abelian monoid laws for parallel: $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
$P \mid Q \equiv Q \mid P$
$P \mid \mathbf{0} \equiv P$

unfolding of replication: $!P \equiv P \mid !P$

summands can be permuted in $\Sigma_{i \in I} \pi_i.P_i$

scope laws: $(\text{new } n)(\text{new } m)P \equiv (\text{new } m)(\text{new } n)P$
$(\text{new } n)\mathbf{0} \equiv \mathbf{0}$
$(\text{new } n)(P \mid Q) \equiv (\text{new } n)P \mid Q$ if $n \notin \text{fn}(Q)$

α-renaming: $(\text{new } n)P \equiv (\text{new } m)P[m/n]$ if $m \notin \text{fn}(P)$

Reduction Relation

$$(n\langle\vec{m}\rangle.P + P') \mid (n(\vec{x}).Q + Q') \rightarrow P \mid Q[\vec{m}/\vec{x}] \qquad \text{if } |\vec{m}| = |\vec{x}|$$

OBS: stuck if $|\vec{m}| \neq |\vec{x}|$

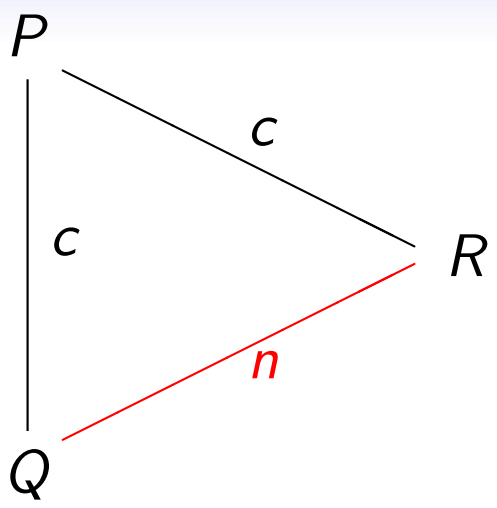
$$\tau.P + Q \rightarrow P$$

$$\frac{P \rightarrow P'}{(\text{new } n)P \rightarrow (\text{new } n)P'} \qquad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$$

$$\frac{P \equiv Q \quad Q \rightarrow Q' \quad Q' \equiv P'}{P \rightarrow P'}$$

Successful example

- P creates the channel n and distributes it to Q and R using the channel c
- Q sends the message m to R using the new channel n

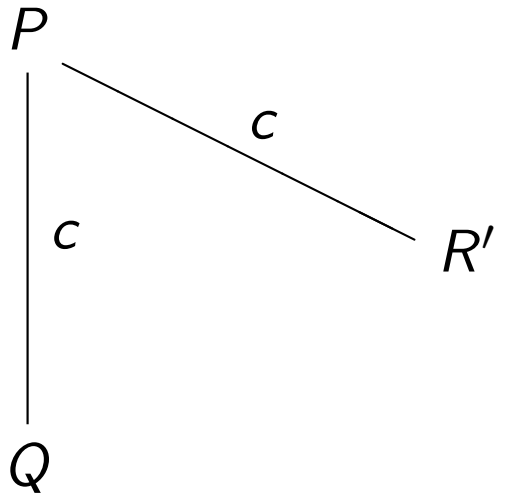


$$\underbrace{((\text{new } n)c\langle n\rangle.\, c\langle n\rangle)}_{P} \;\mid\; \underbrace{c(x).\,((\text{new } m)x\langle m\rangle)}_{Q} \;\mid\; \underbrace{c(y).\,y(x)}_{R}$$

$$\begin{aligned}
&\longrightarrow\; (\text{new } n)(\; c\langle n\rangle \;\mid\; (\text{new } m)n\langle m\rangle \;) \;\mid\; c(y).\,y(x)\\
&\longrightarrow\; (\text{new } n)(\; \mathbf{0} \;\mid\; (\text{new } m)n\langle m\rangle \;\mid\; n(x) \;)\\
&\longrightarrow\; (\text{new } n)(\; \mathbf{0} \;\mid\; (\text{new } m)\; \mathbf{0} \;\mid\; \mathbf{0} \;)
\end{aligned}$$

Unsuccessful example

- P creates the channel n and distributes it to Q and R' using the channel c – but R' fails to receive it



$$\underbrace{((\text{new } n)c\langle n\rangle.\, c\langle n\rangle)}_{P} \;\mid\; \underbrace{c(x).\,((\text{new } m)x\langle m\rangle)}_{Q} \;\mid\; \underbrace{c(x,y).\,x(y)}_{R'}$$

$$\rightarrow \quad (\text{new } n)(\; c\langle n\rangle \;\mid\; (\text{new } m)(n\langle m\rangle)\;) \;\mid\; c(x,y).\,x(y)$$

$\not\rightarrow$ stuck!

The aims of the analysis

Can we guarantee that the process will **never** encounter a configuration where a communication fails because of arity mismatch?

- which sequences of names can be **communicated** over the various channels?
- which names may be **bound** to which variables?

Analysis estimates

The analysis estimate has three components

- $\rho : \mathsf{Var} \rightarrow \wp(\mathsf{Name})$ is the **abstract environment** that maps a variable to the set of names that it might be bound to.
- $\kappa : \mathsf{Name} \rightarrow \wp(\mathsf{Name}^*)$ is the **abstract channel environment** that maps a (channel) name to the set of sequences of names that may be communicated over it.
- $\psi : \wp(\mathsf{Name})$ is the **error component** that records the set of (channel) names where there may be an arity mismatch in a communication.

Analysis judgements

For processes:

$$\rho, \kappa \vdash_{\mathsf{P}} P : \psi$$

means that ρ, κ and ψ is an acceptable analysis estimate for the process P.

For actions:

$$\rho, \kappa \vdash_{\mathsf{A}} \pi : \psi$$

means that ρ, κ and ψ is an acceptable analysis estimate for the action π.

Analysis of processes

$$\rho, \kappa \vdash_{\mathsf{P}} (\text{new } n)P : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash_{\mathsf{P}} P : \psi$$

$$\rho, \kappa \vdash_{\mathsf{P}} P_1 \mid P_2 : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash_{\mathsf{P}} P_1 : \psi_1 \ \wedge \ \rho, \kappa \vdash_{\mathsf{P}} P_2 : \psi_2 \ \wedge \ \psi_1 \cup \psi_2 \subseteq \psi$$

$$\rho, \kappa \vdash_{\mathsf{P}} !P : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash_{\mathsf{P}} P : \psi$$

$$\rho, \kappa \vdash_{\mathsf{P}} \Sigma_{i \in I} \pi_i.P_i : \psi \quad \underline{\text{iff}} \quad \forall i \in I : (\ \rho, \kappa \vdash_{\mathsf{A}} \pi_i : \psi'_i \ \wedge \ \psi'_i \subseteq \psi \ \wedge \ \rho, \kappa \vdash_{\mathsf{P}} P_i : \psi_i \ \wedge \ \psi_i \subseteq \psi \ )$$

Special case for prefixed actions:

$$\rho, \kappa \vdash_{\mathsf{P}} \pi.P : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash_{\mathsf{A}} \pi : \psi' \ \wedge \ \psi' \subseteq \psi \ \wedge \ \rho, \kappa \vdash_{\mathsf{P}} P : \psi'' \ \wedge \ \psi'' \subseteq \psi$$

Analysis of actions

$\rho, \kappa \vdash_{\mathsf{A}} u\langle\vec{v}\rangle : \psi$ <u>iff</u> $\forall n \in \rho(u) : \ \rho(\vec{v}) \subseteq \kappa(n)$

if n is a possible value of u then
all values of $\vec{v}$ may be communicated over n

$\rho, \kappa \vdash_{\mathsf{A}} u(\vec{x}) : \psi$ <u>iff</u> $\forall n \in \rho(u) : \ \kappa(n) \cap \mathsf{Name}^{|\vec{x}|} \subseteq \rho(\vec{x}) \ \wedge$
$\kappa(n) \setminus \mathsf{Name}^{|\vec{x}|} \neq \emptyset \Rightarrow n \in \psi$

if n is a possible value of u then
all sequences recorded in κ of length $|\vec{x}|$
are possible values for $\vec{x}$ and
if there are sequences of different lengths
then there might be an error

$\rho, \kappa \vdash_{\mathsf{A}} \tau : \psi$ <u>iff</u> true

Examples

Successful process:

$$((\text{new } n)c\langle n\rangle.\, c\langle n\rangle) \;\mid\; c(x).\,((\text{new } m)x\langle m\rangle) \;\mid\; c(y).\, y(x)$$

Acceptable analysis result:

$\rho =$

x	y
$\{n, m\}$	$\{n\}$

$\kappa =$

c	n	m
$\{n\}$	$\{m\}$	$\{m\}$

$\psi = \emptyset$

Unsuccessful process:

$$((\text{new } n)c\langle n\rangle.\, c\langle n\rangle) \;\mid\; c(x).\,((\text{new } m)x\langle m\rangle) \;\mid\; c(x, y).\, x(y)$$

Acceptable analysis result:

$\rho =$

x	y
$\{n\}$	$\{m\}$

$\kappa =$

c	n	m
$\{n\}$	$\{m\}$	$\emptyset$

$\psi = \{c\}$

Properties of the analysis

- Semantic correctness
  - we err on the safe side
- Adequacy results
  - Well-behaved processes
  - Well-sorted processes
  - Non-leaking processes
- Moore family property
  - we have best analysis results

Semantic Correctness

Idea: ρ, κ and ψ must capture the behaviour of P and **all** the processes it may evolve into.

Disciplined α-renaming:

$$(\text{new } n)P \equiv_\alpha (\text{new } m)P[m/n] \quad \text{if } m \notin \text{fn}(P) \wedge \lfloor n \rfloor = \lfloor m \rfloor$$

Canonical name: $\lfloor n \rfloor$ is the equivalence class for n

Semantic Correctness

Structural Congruence Lemma:
If $P \equiv Q$ then $\rho, \kappa \vdash_{\mathsf{P}} \lfloor P \rfloor : \psi$ if and only if $\rho, \kappa \vdash_{\mathsf{P}} \lfloor Q \rfloor : \psi$.

The proof is by induction on the inference of $P \equiv Q$.

Substitution Lemma:
If $\rho, \kappa \vdash_{\mathsf{P}} \lfloor P \rfloor : \psi$ then $\rho, \kappa \vdash_{\mathsf{P}} \lfloor P[m/y] \rfloor : \psi$ provided that $\lfloor m \rfloor \in \rho(y)$.

The proof is by structural induction on P.

Subject Reduction Theorem:
Assume $P \rightarrow Q$ and $\rho, \kappa \vdash_{\mathsf{P}} \lfloor P \rfloor : \psi$. Then $\rho, \kappa \vdash_{\mathsf{P}} \lfloor Q \rfloor : \psi$.

The proof is by induction on the inference of $P \rightarrow Q$.

Adequacy Result: Well-behaved processes

$P_\star$ is **dynamically well-behaved** if whenever $P_\star \longrightarrow^* P$ and $P \equiv C[(n\langle\vec{m}\rangle.R + R') \mid (n(\vec{x}).Q + Q')]$ for some C then $|\vec{m}| = |\vec{x}|$.
Here C is a characteristic context:

$$C ::= (\text{new } n)C \;\mid\; C \mid P \;\mid\; [\cdot]$$

$P_\star$ is **statically well-behaved** if there exists ρ and κ such that $\rho, \kappa \vdash_{\mathsf{P}} P_\star : \emptyset$.

Adequacy Theorem for Well-behaved Processes:
If $P_\star$ is statically well-behaved then it is also dynamically well-behaved.

Well-sorted processes

Idea:

- each name n has a sort $\sigma(n) \in \mathsf{Sort}$
- a sorting is a mapping

$$\Sigma : \mathsf{Sort} \rightarrow \mathsf{Sort}^*$$

that for each sort specifies a sequence of sorts describing

- the arity of the names with that sort and
- the sorts of the names that may be communicated over it.

Adequacy Result: Well-sorted processes

$P_\star$ is **dynamically well-sorted** if whenever $P_\star \longrightarrow^* P$ and $P \equiv C[(n\langle\vec{m}\rangle.R + R') \mid (n(\vec{x}).Q + Q')]$ for some C then $|\vec{m}| = |\vec{x}|$ as well as $\Sigma(\sigma(n)) = \sigma(\vec{m})$.

$P_\star$ is **statically well-sorted** if there exists ρ and κ such that $\rho, \kappa \vdash_{\mathsf{P}} P_\star : \emptyset$ and furthermore $\sigma(\kappa(n)) \subseteq \{\Sigma(\sigma(n))\}$ for all n.

Adequacy for well-sorted processes

If the process $P_\star$ is statically well-sorted then it is also dynamically well-sorted.

Non-leaking processes

Idea:

- $\mathsf{Level} = \{\mathsf{low}, \mathsf{high}\}$ is a set of **security levels** ordered by low $\sqsubseteq$ high.
- a mapping

$$\Lambda : \mathsf{Name} \to \mathsf{Level}$$

  assigns a security level to each name; we require that if $\lfloor n \rfloor = \lfloor m \rfloor$ then $\Lambda(n) = \Lambda(m)$

Confidentiality policy: processes are not allowed to send names with higher security level on channels with lower security level

Adequacy Result: Non-leaking processes

$P_\star$ is **dynamically non-leaking** if whenever $P_\star \rightarrow^* P$ and $P \equiv C[(n\langle\vec{m}\rangle.R + R') \mid (n(\vec{x}).Q + Q')]$ for some C then $|\vec{m}| = |\vec{x}|$ as well as $\Lambda(m) \sqsubseteq \Lambda(n)^{|\vec{m}|}$.

$P_\star$ is **statically non-leaking** if there exists ρ and κ such that $\rho, \kappa \vdash_{\mathsf{P}} P_\star : \emptyset$ and furthermore $\forall \vec{m} \in \kappa(n) : \Lambda(\vec{m}) \sqsubseteq \Lambda(n)^{|\vec{m}|}$ for all n.

Adequacy for non-leaking processes

If the process $P_\star$ is statically non-leaking then it is also dynamically non-leaking.

Moore family property

All processes can be analysed and has a least (best) analysis result:

The set $\{(\rho, \kappa, \psi) \mid \rho, \kappa \vdash_{\mathsf{P}} P : \psi\}$ *is a Moore family*

If $\mathcal{Y} \subseteq \{\mathcal{I} \mid \mathcal{I} \models^{\star}_{\Gamma} P\}$ then $\sqcap\mathcal{Y} \models^{\star}_{\Gamma} P$ where
$(\sqcap\mathcal{Y})(\mu) = \bigcap\{\mathcal{I}(\mu) \mid \mathcal{I} \in \mathcal{Y}\}$

The proof is by structural induction on P.

Corollaries:

- All processes can be analysed:
  — a Moore family cannot be empty
- All processes has a least (best) analysis result:
  — a Moore family has a least element

Interaction points

Observation: two actions only interact when in different threads

This is not captured in the analysis:

$$\rho, \kappa \vdash_{\mathsf{P}} \pi_1.P_1 + \pi_2.P_2 : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash_{\mathsf{P}} \pi_1.P_1 \mid \pi_2.P_2 : \psi$$

Example: for $n\langle m\rangle.n(x)$ we get $m \in \kappa(n)$

Extended syntax: adds labels to output and input actions:

$$\pi \quad ::= \quad u\langle \vec{v}\rangle^{\ell} \mid u(\vec{x})^{\ell} \mid \tau$$

Semantics:

unchanged — the labels are just pointers into the syntax

Set of interaction points

Output labels: $\mathsf{Lab}_o(P)$

Input labels: $\mathsf{Lab}_i(P)$

Interaction points: $\mathsf{IP}(P)$ contains pairs (ℓ_o, ℓ_i) of (labels of) output and input actions that may interact

$$
\begin{aligned}
\mathsf{IP}((\mathsf{new}\, n)P) &= \mathsf{IP}(P) \\
\mathsf{IP}(P_1 \mid P_2) &= \mathsf{IP}(P_1) \cup \mathsf{IP}(P_2) \cup \\
&\quad (\mathsf{Lab}_o(P_1) \times \mathsf{Lab}_i(P_2)) \cup (\mathsf{Lab}_o(P_2) \times \mathsf{Lab}_i(P_1)) \\
\mathsf{IP}(!P) &= \mathsf{IP}(P) \cup (\mathsf{Lab}_o(P) \times \mathsf{Lab}_i(P)) \\
\mathsf{IP}(\Sigma_{i \in I} \pi_i.P_i) &= \bigcup_{i \in I} \mathsf{IP}(P_i)
\end{aligned}
$$

Revised analysis domains

We can improve the precision of the analysis by taking:

- $\rho : \mathsf{Var} \rightarrow \wp(\mathsf{Name})$ is as before: it maps a variable to the set of names that it might be bound to.
- $\kappa : \mathsf{Name} \times \mathsf{Lab} \rightarrow \wp(\mathsf{Name}^*)$ is extended to record which sequences of names that are communicated over a given (channel) name at a given output label.
- $\psi : \wp(\mathsf{Name} \times \mathsf{Lab} \times \mathsf{Lab})$ is extended to record the set of triples of (channel) names and interaction points where there may be an arity mismatch in a communication.

Revised analysis

Actions:

$$\rho, \kappa \vdash_{\mathsf{A}} u\langle \vec{v} \rangle^{\ell_o} : \psi \quad \underline{\text{iff}} \quad \forall n \in \rho(u) : \rho(\vec{v}) \subseteq \kappa(n, \ell_o)$$

$$\begin{aligned} \rho, \kappa \vdash_{\mathsf{A}} u(\vec{x})^{\ell_i} : \psi \quad \underline{\text{iff}} \quad & \forall n \in \rho(u) : \forall \ell_o : (\ell_o, \ell_i) \in \mathsf{IP}(P_\star) \Rightarrow \\ & (\ \kappa(n, \ell_o) \cap \mathsf{Name}^{|\vec{x}|} \subseteq \rho(\vec{x}) \ \wedge \\ & \kappa(n, \ell_o) \setminus \mathsf{Name}^{|\vec{x}|} \neq \emptyset \Rightarrow \\ & (n, \ell_o, \ell_i) \in \psi \ ) \end{aligned}$$

$$\rho, \kappa \vdash_{\mathsf{A}} \tau : \psi \quad \underline{\text{iff}} \quad \text{true}$$

Processes:

$\rho, \kappa \vdash_{\mathsf{P}} P : \psi$ as before

Tests

Extended syntax:

$$P \quad ::= \quad (\text{new } n)P \;\mid\; P_1 \mid P_2 \;\mid\; !P \mid \Sigma_{i \in I} \pi_i.P_i \;\mid\; [u = v]P$$

Extended semantics:

$$[n = n]P \rightarrow P$$

Extended analysis:

$$\rho, \kappa \vdash_{\mathsf{P}} [u = v]P : \psi \quad \underline{\text{iff}} \quad \rho(u) \cap \rho(v) \neq \emptyset \Rightarrow \rho, \kappa \vdash_{\mathsf{P}} P : \psi$$

Note:

- the continuation P is only analysed if it **may** be executed
- when P is analysed it is **not** enforced that $\rho(u) = \rho(v)$

Localised environment

Extended syntax: add labels to tests and input actions

$$P \quad ::= \quad (\text{new } n)P \;\mid\; P_1 \mid P_2 \;\mid\; !P \;\mid\; \Sigma_{i \in I} \pi_i.P_i \;\mid\; [u = v]^{\ell} P$$

$$\pi \quad ::= \quad u\langle \vec{v} \rangle \;\mid\; u(\vec{x})^{\ell} \;\mid\; \tau$$

Notation:

$$\text{lab}(\pi, \ell) = \begin{cases} \ell & \text{if } \pi = u\langle \vec{v} \rangle \\ \ell_i & \text{if } \pi = u(\vec{x})^{\ell_i} \\ \ell & \text{if } \pi = \tau \end{cases}$$

Semantics: unchanged — the labels are just pointers into the syntax

Revised analysis domains

We can improve the precision of the analysis by taking:

- $\rho : \mathsf{Lab} \times \mathsf{Var} \rightarrow \wp(\mathsf{Name})$ is the **localised abstract environment** that given a label maps a variable to the set of names that it can be bound to in the context described by the label.
- $\kappa : \mathsf{Name} \rightarrow \wp(\mathsf{Name}^*)$ is as before: it maps a (channel) name to the set of sequences of names that may be communicated over it.
- $\psi : \wp(\mathsf{Name} \times \mathsf{Lab})$ is the error component that now records set of pairs of (channel) names and (input) labels indicating where an arity mismatch may occur.

Revised analysis

Processes: $\rho, \kappa \vdash^{\ell}_{\mathsf{P}} P : \psi$

$$\rho, \kappa \vdash^{\ell}_{\mathsf{P}} (\mathsf{new}\ n)P : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash^{\ell}_{\mathsf{P}} P : \psi$$

$$\rho, \kappa \vdash^{\ell}_{\mathsf{P}} P_1 \mid P_2 : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash^{\ell}_{\mathsf{P}} P_1 : \psi_1 \ \wedge\ \rho, \kappa \vdash^{\ell}_{\mathsf{P}} P_2 : \psi_2 \ \wedge \ \psi_1 \cup \psi_2 \subseteq \psi$$

$$\rho, \kappa \vdash^{\ell}_{\mathsf{P}} !P : \psi \quad \underline{\text{iff}} \quad \rho, \kappa \vdash^{\ell}_{\mathsf{P}} P : \psi$$

$$\rho, \kappa \vdash^{\ell}_{\mathsf{P}} \Sigma_{i \in I} \pi_i.P_i : \psi \quad \underline{\text{iff}} \quad \forall i \in I : (\ \rho, \kappa \vdash^{\ell}_{\mathsf{A}} \pi_i : \psi'_i \ \wedge\ \psi'_i \subseteq \psi \ \wedge \ \rho, \kappa \vdash^{\mathsf{lab}(\pi_i, \ell)}_{\mathsf{P}} P_i : \psi_i \ \wedge\ \psi_i \subseteq \psi\ )$$

$$\rho, \kappa \vdash^{\ell}_{\mathsf{P}} [u = v]^{\ell_t} P : \psi \quad \underline{\text{iff}} \quad \rho(u, \ell) \cap \rho(v, \ell) \neq \emptyset \Rightarrow \forall x \notin \{u, v\} : \rho(x, \ell) \subseteq \rho(x, \ell_t) \ \wedge \ \forall x \in \{u, v\} : \rho(u, \ell) \cap \rho(v, \ell) \subseteq \rho(x, \ell_t) \ \ \rho, \kappa \vdash^{\ell_t}_{\mathsf{P}} P : \psi$$

Revised analysis

Actions: $\rho, \kappa \vdash^{\ell}_{\mathsf{A}} \pi : \psi$

$$\rho, \kappa \vdash^{\ell}_{\mathsf{A}} u\langle \vec{v} \rangle : \psi \quad \underline{\text{iff}} \quad \forall n \in \rho(u, \ell) : \rho(\vec{v}, \ell) \subseteq \kappa(n)$$

$$\begin{aligned} \rho, \kappa \vdash^{\ell}_{\mathsf{A}} u(\vec{x})^{\ell_i} : \psi \quad \underline{\text{iff}} \quad & \forall n \in \rho(u, \ell) : \\ & \kappa(n) \cap \mathsf{Name}^{|\vec{x}|} \subseteq \rho(\vec{x}, \ell_i) \wedge \\ & \kappa(n) \setminus \mathsf{Name}^{|\vec{x}|} \neq \emptyset \Rightarrow (n, \ell_i) \in \psi \wedge \\ & \forall y \notin \{\vec{x}\} : \rho(y, \ell) \subseteq \rho(y, \ell_i) \end{aligned}$$

$$\rho, \kappa \vdash^{\ell}_{\mathsf{A}} \tau : \psi \quad \underline{\text{iff}} \quad \text{true}$$

Suggested Reading

- H. Riis Nielson, F. Nielson, H. Pilegaard: Flow Logic for Process Calculi: a Tutorial.

  Section 2 develops a Flow Logic for the π-calculus.

  Sections 3 and 4 gives an overview of stronger analysis techniques as well as other calculi — not in your hand-out.

Static Analysis of Services

Part 3:

An Executive Summary of Flow Logic

The setting for Flow Logic

- Programs P in some programming language or process algebra
- Semantics $P \rightarrow P'$ often in the form of a Structural Operational Semantics or Reaction Semantics
- Analysis estimates $\mathcal{I}$ ususally elements of some complete lattice $(L, \sqsubseteq)$
- A validity judgement $\mathcal{I} \models P$ defined by clauses
- A subject reduction result $\mathcal{I} \models P \wedge P \rightarrow P' \Rightarrow \mathcal{I} \models P'$ indicating that the validity is preserved under reduction
- A Moore Family result showing that $\mathcal{I}_\star = \sqcap\{\mathcal{I} \mid \mathcal{I} \models P_\star\}$ defines a unique least solution

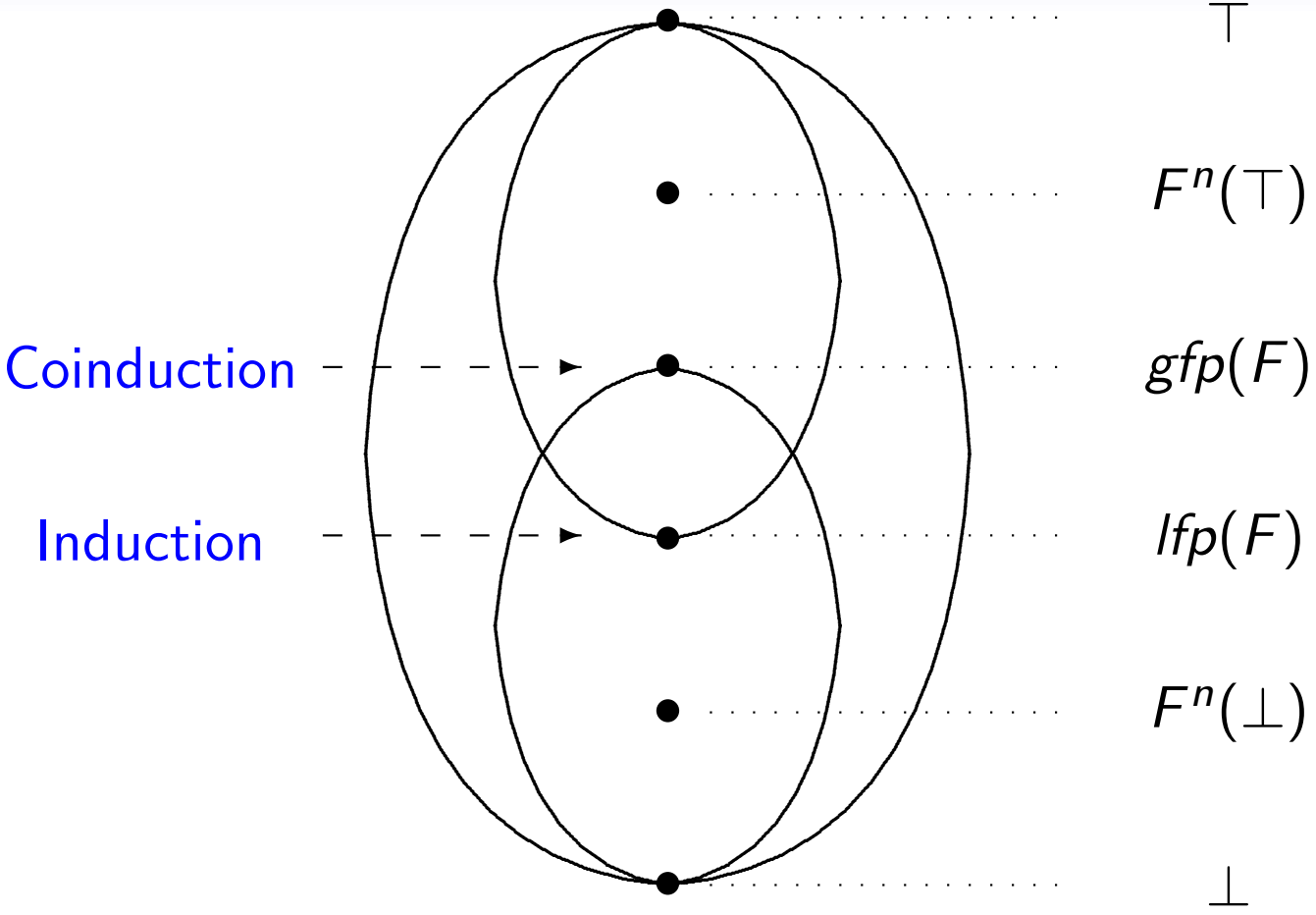
The validity judgement

- Usually there is one clause for each syntactic construct ϕ of the programming language to which P belongs
- Each clause takes the form

$$\mathcal{I} \models \phi(\cdots P_i \cdots) \ \underline{\text{iff}} \ (\text{some formula } \Psi \text{ with } \mathcal{I} \models P')$$

  for various subprograms P'
- The first challenge is to ensure that this constitutes a well-defined definition.
  - Sometimes an inductive definition suffices; this is the case if each P' is some P_i (see above).
  - Otherwise a coinducitive definition is called for; this works if the formula Ψ gives rise to a monotonic functional F.

The structure of fixed points (Tarski)



Induction versus Coinduction

- Sometimes induction and coinduction agrees. Then a clause

$$\mathcal{I} \models \phi(P_1, P_2) \ \underline{\text{iff}} \ (\Psi \wedge \mathcal{I} \models P_1 \wedge \mathcal{I} \models P_1)$$

may be written in a more familiar form as an inference rule

$$\frac{\begin{array}{c}\Psi \\ \mathcal{I} \models P_1 \\ \mathcal{I} \models P_2\end{array}}{\mathcal{I} \models \phi(P_1, P_2)}$$

- When induction and coinduction differs we always take the coinductive definition of the validity judgement.
  - This is not so often the case for process algebras.

Semantic correctness

Lemma: The analysis is invariant under the structural congruence:

$$\text{If} \quad P \equiv Q \quad \text{then} \quad \mathcal{I} \models P \text{ if and only if } \mathcal{I} \models Q$$

The proof is by induction on the inference of $P \equiv Q$:

- Using the <u>iff</u> form of the definition of $\models$ to freely fold and unfold formulae (regardles of which fixpoint used).
- Using a notion of canonical names (invariant under α-conversion) if names are collected in the analysis ($\mathcal{I}$).
- Sometimes the unfolding of recursion allowed by $A(y) \equiv P[y/x]$ (if we have $A(x) \stackrel{\Delta}{=} P$) creates severe complications!

Subject reduction result

Theorem: The analysis is preserved under the transition relation:

$$\text{If} \quad P \rightarrow Q \quad \text{and} \quad \mathcal{I} \models P \quad \text{then} \quad \mathcal{I} \models Q$$

The proof is by induction on the inference of $P \rightarrow Q$:

- Using the <u>iff</u> form of the definition of $\models$ to freely fold and unfold formulae (regardles of which fixpoint used).
- Being careful with substitution to get placeholder labels to work correctly (if present): $x^l[y^m/x] = y^l$.

Moore family property

Theorem: The set $\{\mathcal{I} \mid \mathcal{I} \models^{\star}_{\Gamma} P\}$ is a Moore family
This means that $\sqcap\mathcal{Y} \models^{\star}_{\Gamma} P$ whenever $\mathcal{Y} \subseteq \{\mathcal{I} \mid \mathcal{I} \models^{\star}_{\Gamma} P\}$ where
$(\sqcap\mathcal{Y})(\mu) \stackrel{\Delta}{=} \bigcap\{\mathcal{I}(\mu) \mid \mathcal{I} \in \mathcal{Y}\}$
The proof is by strutural induction on P.

It follows that $\mathcal{I}_\star = \sqcap\{\mathcal{I} \mid \mathcal{I} \models P_\star\}$ defines a unique least solution:

- we have $\mathcal{I}_\star \models P_\star$
- if $\mathcal{I} \models P_\star$ then $\mathcal{I}_\star \sqsubseteq \mathcal{I}$

Calculating the least solution

Generate $\mathcal{I}_\star = \sqcap\{\mathcal{I} \mid \mathcal{I} \models P_\star\}$ as a formula in a suitable format:

- Conditional constraints — with links to bitvector frameworks.
- Horn Clauses
  - Datalog
  - Alternating Least Fixed Point Logic *("The Succinct Solver")*
    The asymptotic time to check a solution equals the asymptotic time to calculate a solution.
  - H3, H1, $\cdots$ solvers

Ingredients for more advanced analyses

- Adding context information (from 0-CFA to k-CFA):

$$\mathcal{I} \models_c P \ \underline{\text{iff}} \ \cdots \mathcal{I} \models_{cc'} P' \cdots \wedge \cdots \{cc'\} \subseteq \mathcal{I}(c) \cdots$$

  Here c indicates the dynamic context in which the process P is encountered; this is useful for achieving more precise analyses of programming languages and process algebras.

- Making analyses data dependent.

  This is mainly used for imperative and object-oriented languages.

Abstract versus Compositional

One of the methodological choices of Flow Logic is when to use abstract and when to use compositional specifications.

- Abstract:
  - Abstract specifications place no demands on which processes may be used on the right-hand-sides of iff .
  - Usually processes communicated (in a higher-order process algebra) and recursive processes are analysed at each invocation.
  - This leads to treating **open systems** for free.
  - The coinductive interpretation (desired) may differ from the inductive one.
  - Often more pleasant to read and easier to understand.

Abstract versus Compositional

- Compositional:
  - Compositional specifications demand that only subprocesses of the process on the left-hand-side of an iff may be used on the right-hand-side.
  - Therefore processes communicated (in a higher-order process algebra) and recursive processes must be analysed once and for all at their point of definition.
  - Unless special care is taken this leads to analysing **closed systems** only.
  - The coinductive interpretation (desired) agrees with the inductive one.
  - Ususally necessary in order to implement the analysis.

Verbose versus Succinct

Another methodological choice of Flow Logic is when to use verbose and when to use succinct specifications.

- Verbose:
  - All the information of interest in collected in a global manner:

    $$\mathcal{I}, \mathcal{I}_I, \mathcal{I}_O \models P$$
  - Makes some forms of implementation easier to deal with.
  - Often requires placeholder labels in the syntax of processes.

Verbose versus Succinct

- Succinct:
  - Only some of the information of interest is collected in a global manner; other pieces of information may be
    - synthesized to describe the effect of local components:

      $$\mathcal{I} \models P : \triangleright\ \hat{O}$$

      which is useful for functional languages and process algebras.
    - record input- and output-dependencies

      $$\mathcal{I} \models P : \hat{I}\ \triangleright\ \hat{O}$$

      which is mainly useful for procedural languages.
  - Generally leads to simpler analysis domains.
  - Often more pleasant to read and easier to understand.

A word on notation

Some of the notation used has striking resemblance to (can be considered equivalent to) other well-known notations:

- Succinct Compositional Flow Logic

$$\mathcal{I} \models P : \hat{I} \ \triangleright \ \hat{O}$$

- Hoare Logic

$$\mathcal{I} \vdash \{\hat{I}\} \ P \ \{\hat{O}\}$$

Suggested Reading

- H. Riis Nielson, F. Nielson, H. Pilegaard: Flow Logic for Process Calculi: a Tutorial.

  Section 5 gives an executive summary of what Flow Logic is – concentrating on process calculi.

- H. Riis Nielson, F. Nielson. Flow Logic: a Multi-paradigmatic Approach to Static Analysis. In *The Essence of Computation: Complexity, Analysis, Transformation*, SLNCS 2566, pages 223 - 244, Springer, 2002.

  Contains a tutorial and executive summary of what Flow Logic is – concentrating on programming languages rather than process calculi.

Static Analysis of Services

Part 4:

Flow Logic for COWS

The Calculus for Orchestration of Web Services

Communication model:

- Service providers and users constitute loosely coupled agents
- Each service may require a number of request-response interactions that are strung together in **virtual sessions** using correlations
- A **correlation** contains (patterns of) values shared between the service provider and the user
- Correlations allow for a **flexible** communication model where interactions do not have to be performed in one fixed order
- Technically this is facilitated by an idea also found in the Fusion Calculus: the positions defining the contents of variables are seperated from the positions indicating the scope of variables
- This presents a challenge to static analysis

Syntax and semantics of COWS

Syntax:

Services: $s ::= u \bullet u'!\bar{v}^{\ell} \mid g \mid s \mid s \mid (n)s \mid [x]^{\ell}s \mid *s$

Input-guarded Choice: $g ::= \mathbf{0} \mid p \bullet o?\bar{v}^{\ell}.s \mid g + g$

Semantics:

- Structural congruence relation: $P \equiv Q$
- Labelled transition relation: $P \xrightarrow{\alpha} Q$

Structural congruence relation

Monoid laws for $\odot \in \{+, \mid\}$:

$$(P \odot Q) \odot R \equiv P \odot (Q \odot R)$$
$$P \odot Q \equiv Q \odot P$$
$$P \odot 0 \equiv P$$

Unfolding of replication:

$$\star s \equiv s \mid \star s$$
$$\star 0 \equiv 0$$

Scope laws:

$$(n)0 \equiv 0$$
$$(n)(m)s \equiv (m)(n)s$$
$$s_1 \mid (n)s_2 \equiv (n)(s_1 \mid s_2) \text{ if } n \notin fn(s_1)$$
$$[x](n)s \equiv (n)[x]s$$

α-renaming:

$$(n)P \equiv (m)P[^m/_n] \quad \text{if } m \notin \text{fn}(P)$$
$$[x]P \equiv [y]P[^y/_x] \quad \text{if } y \notin \text{fn}(P)$$

Pattern matching

Pattern matching is a **Partial** function:

$$\mathcal{M}(n, n) = \emptyset \qquad \mathcal{M}(x, n) = \{x \mapsto n\}$$

$$\frac{\mathcal{M}(u_1, n_1) = \sigma_1 \quad \mathcal{M}(\bar{u}_2, \bar{n}_2) = \sigma_2}{\mathcal{M}((u_1, \bar{u}_2), (n_1, \bar{n}_1)) = \sigma_1 \uplus \sigma_2}$$

Fails on $\mathcal{M}(n, m)$!

Transition labels

The semantic transitions are labelled as follows:

$$\alpha ::= (p \bullet o) \lhd \bar{n}^{\ell} \;\mid\; (p \bullet o) \rhd \bar{u}^{\ell} \;\mid\; p \bullet o \lfloor \sigma \rfloor \bar{u}^{\ell_i} \bar{n}^{\ell_o}$$

where $\sigma \subseteq \mathcal{M}(\bar{u}^{\ell_i}, \bar{n}^{\ell_o})$.

Labelled transition relation

Rules that change transition labels:

$$p \bullet o!\bar{n}^{\ell} \xrightarrow{(p\bullet o)\lhd \bar{n}^{\ell}} 0 \qquad p \bullet o?\bar{u}^{\ell}.s \xrightarrow{(p\bullet o)\rhd \bar{u}^{\ell}} s$$

$$\frac{s_1 \xrightarrow{(p\bullet o)\rhd \bar{u}^{\ell_i}} s_1' \quad s_2 \xrightarrow{(p\bullet o)\lhd \bar{n}^{\ell_o}} s_2' \quad \mathcal{M}(\bar{u},\bar{n}) = \sigma}{s_1 \mid s_2 \xrightarrow{p\bullet o\lfloor\sigma\rfloor \bar{u}^{\ell_i}\bar{n}^{\ell_o}} s_1' \mid s_2'}$$

$$\frac{s \xrightarrow{p\bullet o\lfloor\sigma\uplus[x\mapsto n]\rfloor \bar{u}^{\ell_i}\bar{n}^{\ell_o}} s'}{[x]^{\ell}s \xrightarrow{p\bullet o\lfloor\sigma\rfloor \bar{u}^{\ell_i}\bar{n}^{\ell_o}} s' \cdot [x \mapsto n]}$$

A completed semantic transition has label $p \bullet o\lfloor\emptyset\rfloor \bar{u}^{\ell_i}\bar{n}^{\ell_o}$

Labelled transition relation

Rules that propagate transition labels:

$$\frac{g_1 \xrightarrow{\alpha} s}{g_1 + g_2 \xrightarrow{\alpha} s} \qquad \frac{s_1 \xrightarrow{\alpha} s}{s_1 \mid s_2 \xrightarrow{\alpha} s \mid s_2}$$

$$\frac{s \xrightarrow{\alpha} s' \quad n \notin d(\alpha)}{(n)s \xrightarrow{\alpha} (n)s'} \qquad \frac{s \xrightarrow{\alpha} s' \quad x \notin d(\alpha)}{(x)s \xrightarrow{\alpha} (x)s'}$$

$$\frac{s \equiv s_1 \quad s_1 \xrightarrow{\alpha} s_2 \quad s_2 \equiv s'}{s \xrightarrow{\alpha} s'}$$

where $d(\alpha)$ contains all names and variables addressed by the σ of α, if any.

Example: a car accident service

Cars

$$
\begin{aligned}
&*(\mathsf{self})\ *(\mathsf{gps})\ *(\mathsf{sid})\\
&\quad \mathsf{p}_s \bullet \mathsf{o}_{alarm}!\langle \mathsf{acc}, \mathsf{self}, \mathsf{sid}\rangle^1\\
&\mid\ (\mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^2.\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ok}, \mathsf{self}, \mathsf{sid}\rangle^3 \ +\\
&\qquad \mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^4.\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ko}, \mathsf{gps}, \mathsf{sid}\rangle^5)
\end{aligned}
$$

Service provider

$$
\begin{aligned}
&*[x_{info}]^6\ [x_{id}]^7\ [x_{reply}]^8\ [x_{sid}]^9\\
&\quad \mathsf{p}_s \bullet \mathsf{o}_{alarm}?\langle \mathsf{acc}, x_{id}, x_{sid}\rangle^{10}.x_{id} \bullet \mathsf{o}_{confirm}!\langle x_{sid}\rangle^{11}\\
&\mid\ \mathsf{p}_s \bullet \mathsf{o}_{confirm}?\langle x_{reply}, x_{info}, x_{sid}\rangle^{12}.\mathsf{p}_s \bullet \mathsf{o}_{Sos}!\langle x_{reply}, x_{info}, x_{sid}\rangle^{13}\\
&\mid\ \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ko}, x_{info}, x_{sid}\rangle^{14}.\mathsf{p}_{amb} \bullet \mathsf{o}_{Sos}!\langle x_{info}, x_{sid}\rangle^{15}\\
&\mid\ \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ok}, x_{info}, x_{sid}\rangle^{16}.0
\end{aligned}
$$

Example: a car accident service

Car

$$(\mathsf{self})\,(\mathsf{gps})\,(\mathsf{sid})$$
$$\boxed{\mathsf{p}_s \bullet \mathsf{o}_{alarm}!\langle \mathsf{acc}, \mathsf{self}, \mathsf{sid}\rangle^1}$$
$$\mid\ (\mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^2.\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ok}, \mathsf{self}, \mathsf{sid}\rangle^3 + \mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^4.\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ko}, \mathsf{gps}, \mathsf{sid}\rangle^5)$$

Service instance

$$[x_{info}]^6\ [x_{id}]^7\ [x_{reply}]^8\ [x_{sid}]^9$$
$$\boxed{\mathsf{p}_s \bullet \mathsf{o}_{alarm}?\langle \mathsf{acc}, x_{id}, x_{sid}\rangle^{10}.}\ x_{id} \bullet \mathsf{o}_{confirm}!\langle x_{sid}\rangle^{11}$$
$$\mid\ \mathsf{p}_s \bullet \mathsf{o}_{confirm}?\langle x_{reply}, x_{info}, x_{sid}\rangle^{12}.\mathsf{p}_s \bullet \mathsf{o}_{Sos}!\langle x_{reply}, x_{info}, x_{sid}\rangle^{13}$$
$$\mid\ \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ko}, x_{info}, x_{sid}\rangle^{14}.\mathsf{p}_{amb} \bullet \mathsf{o}_{Sos}!\langle x_{info}, x_{sid}\rangle^{15}$$
$$\mid\ \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ok}, x_{info}, x_{sid}\rangle^{16}.0$$

Example: a car accident service

Car

$$(\mathsf{self})\,(\mathsf{gps})\,(\mathsf{sid})$$
$$(\mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^{2}.\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ok}, \mathsf{self}, \mathsf{sid}\rangle^{3} \;+$$
$$\mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^{4}.\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ko}, \mathsf{gps}, \mathsf{sid}\rangle^{5})$$

Service instance

$$[x_{info}]^{6}\;[x_{reply}]^{8}$$
$$\mathsf{self} \bullet \mathsf{o}_{confirm}!\langle \mathsf{sid}\rangle^{11}$$
$$\mid\; \mathsf{p}_s \bullet \mathsf{o}_{confirm}?\langle x_{reply}, x_{info}, \mathsf{sid}\rangle^{12}.\mathsf{p}_s \bullet \mathsf{o}_{Sos}!\langle x_{reply}, x_{info}, \mathsf{sid}\rangle^{13}$$
$$\mid\; \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ko}, x_{info}, \mathsf{sid}\rangle^{14}.\mathsf{p}_{amb} \bullet \mathsf{o}_{Sos}!\langle x_{info}, \mathsf{sid}\rangle^{15}$$
$$\mid\; \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ok}, x_{info}, \mathsf{sid}\rangle^{16}.0$$

Example: a car accident service

Car

$$(\mathsf{self})\,(\mathsf{gps})\,(\mathsf{sid})$$
$$(\boxed{\mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^{2}.}\,\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ok}, \mathsf{self}, \mathsf{sid}\rangle^{3} \;+$$
$$\boxed{\mathsf{self} \bullet \mathsf{o}_{confirm}?\langle \mathsf{sid}\rangle^{4}.}\,\mathsf{p}_s \bullet \mathsf{o}_{confirm}!\langle \mathsf{ko}, \mathsf{gps}, \mathsf{sid}\rangle^{5})$$

Service instance

$$[x_{info}]^{6}\;[x_{reply}]^{8}$$
$$\boxed{\mathsf{self} \bullet \mathsf{o}_{confirm}!\langle \mathsf{sid}\rangle^{11}}$$
$$\mid\; \mathsf{p}_s \bullet \mathsf{o}_{confirm}?\langle x_{reply}, x_{info}, \mathsf{sid}\rangle^{12}.\mathsf{p}_s \bullet \mathsf{o}_{Sos}!\langle x_{reply}, x_{info}, \mathsf{sid}\rangle^{13}$$
$$\mid\; \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ko}, x_{info}, \mathsf{sid}\rangle^{14}.\mathsf{p}_{amb} \bullet \mathsf{o}_{Sos}!\langle x_{info}, \mathsf{sid}\rangle^{15}$$
$$\mid\; \mathsf{p}_s \bullet \mathsf{o}_{Sos}?\langle \mathsf{ok}, x_{info}, \mathsf{sid}\rangle^{16}.0$$

The aims of the analysis

Can we guarantee that the correlation of messages is strong enough to ensure **correct message delivery**?

- Which tuples may be **communicated** between the various endpoints?
- Which tuples of values may be **bound** at the various program points?

A note on correlation

- A **service instance** often engages in multiple concurrent conversations in order to orchestrate sub-services
- A **correlation set** is a set of message properties that allows each message to be uniquely associated with the appropriate instance
- Uses are often **security critical** but the concept is fragile
  - Similar in purpose to nonces in security protocols, yet more powerful and complicated to use
  - Similar in function to keys in relational databases, yet more dynamic and harder to control
- **Formal verification** is definitely worthwhile...

Auxiliary relations

The analysis relies on two auxiliary static relations

- $\mathsf{F} : \mathbf{Lab} \rightarrow \mathcal{P}(\mathbf{Lab})$ is the **flow relation** that maps maps a program point label, ℓ, to the labels of program points that may follow.
- $\mathsf{L} : \mathbf{Lab} \rightarrow (\mathbf{Var} \times \mathbf{Lab})^*$ is the **label environment** that maps a program point label, ℓ, to the variable scope that encloses it.

  E.g. $[x_{id}]^1\ [x_{sid}]^2 \mathsf{p}_s \bullet \mathsf{o}_{alarm}?\langle \mathsf{acc}, x_{id}, x_{sid}\rangle^3.x_{id} \bullet \mathsf{o}_{confirm}!\langle x_{sid}\rangle^4$
  where $\mathsf{L}.1 = \langle\rangle$, $\mathsf{L}.2 = \langle x^1_{id}\rangle$, and $\mathsf{L}.3 = \mathsf{L}.4 = \langle x^1_{id}, x^2_{sid}\rangle$

  Label lookup $x\#\langle x^{\ell_1}, \ldots, x^{\ell_k}\rangle = \ell_t$ if $t = \max\{i \mid x_i = x\}$

  E.g. $x_{sid}\#\mathsf{L}.3 = 2$

Analysis estimates

The analysis estimate has three components

- $\hat{\mathsf{K}} : (\mathbf{Name} \times \mathbf{Name}) \to \mathcal{P}(\mathbf{Name}^*)$ is the **abstract communication cache** that maps an endpoint, $p \bullet o$, to the set of tuples of names that may be send to it.
- $\hat{\mathsf{R}} : \mathbf{Lab} \to \mathcal{P}(\mathbf{Name}_\perp^*)$, where $\mathbf{Name}_\perp = \mathbf{Name} \cup \{\perp\}$, is the **abstract environment** that maps a program point, ℓ, to the set of tuples that may instantiate the variable scope, $\mathsf{L}.\ell$, enclosing it.

  E.g. if $\bar{w} = \langle n_1, \perp \rangle \in \hat{\mathsf{R}}.3$ (corresponding to $\mathsf{L}.3 = \langle x_{id}^1, x_{sid}^2 \rangle$) there may be an instance where x_{id} is bound to n_1 and x_{sid} is unbound.

  Value lookup $\Pi_{x@\langle x_1^{\ell_1}, \ldots, x_k^{\ell_k} \rangle}(\langle w_1, \ldots, w_k \rangle) = w_t \quad \text{if } t = \max\{i \mid x_i = x\}$

  E.g. $\Pi_{x_{id}@\mathsf{L}.3}(\bar{w}) = n_1$

  Note $\Pi_{x_{id}x_{sid}@\mathsf{L}.3}(\bar{w}) = \langle n_1, \perp \rangle$ and $\Pi_{x_{id}x_{sid}@\mathsf{L}.3}(\{\bar{w}\}) = \{\langle n_1, \perp \rangle\}$

Analysis judgements

Given a service s, with flow relation F and label environment L, the judgement

$$\hat{R}, \hat{K} \vdash_{L,F} s$$

expresses that $\hat{R}$ and $\hat{K}$ constitute an acceptable analysis estimate for s.

Analysis clauses

Recall Judgement defined by single clause for each syntactic construct.

The clauses for **unlabelled constructs** are particularly simple:

$$
\begin{array}{rcl}
\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} 0 & \underline{\text{iff}} & \text{true} \\
\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} \star s & \underline{\text{iff}} & \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s \\
\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} (n)s & \underline{\text{iff}} & \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s \\
\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s_1 \mid s_2 & \underline{\text{iff}} & \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s_1 \wedge \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s_2 \\
\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s_1 + s_2 & \underline{\text{iff}} & \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s_1 \wedge \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s_2
\end{array}
$$

Analysis clauses

The **labelled constructs** are much more interesting:

$\hat{R}, \hat{K} \vdash_{L,F} [x]^{\ell} s$ <u>iff</u> $\hat{R}, \hat{K} \vdash_{L,F} s \wedge$
$\forall \ell' \in F.\ell : \hat{R}.\ell \times \{\bot\} \subseteq \hat{R}.\ell'$

If ℓ' follows ℓ then every scope instantiation recorded for ℓ is extended by $\bot$ and recorded for ℓ'

$\hat{R}, \hat{K} \vdash_{L,F} u \bullet u'!\bar{v}^{\ell}$ <u>iff</u> $\Pi_{uu'\bar{v}@L.\ell}(\hat{R}.\ell) \cap \mathbf{Name}^* \subseteq \hat{K}$

Every suitable ($\bot$-free) instantiation, $\langle n, n', \bar{n} \rangle$, of u, u', and $\bar{v}$ recorded for ℓ is recorded in $\hat{K}$ as a potential communication

Analysis clauses

$$\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} p \bullet o?\bar{u}^{\ell}.s \quad \underline{\text{iff} } \quad (X \neq \emptyset \;\Rightarrow\; \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s) \wedge$$
$$(\forall \ell' \in \mathsf{F}.\ell : X \subseteq \hat{\mathsf{R}}.\ell') \wedge$$
$$(\forall x \in \{\bar{u}\} : \forall \ell_x \in (\mathsf{F}.(x\#\mathsf{L}.\ell)) : Y_x \subseteq \hat{\mathsf{R}}.\ell_x)$$

For any variable, x, bound by the input pattern, Y_x records the names, n, that may be bound to x by a communication that matches the pattern. This information flows to all labels, ℓ_x, that follow immediately after the introduction of the scope of x

$$Y_x = \{\bar{w}n \in \mathbf{Name}^*_{\perp} \mid \; \bar{w} \in \hat{\mathsf{R}}.(x\#\mathsf{L}.\ell) \wedge$$
$$\exists \bar{w}' \in \mathbf{Name}^*_{\perp} : \Pi_{po\bar{u}@\mathsf{L}.\ell}(\bar{w}n\bar{w}') \in \hat{\mathsf{K}} \;\}$$

All scope instantiations at the communication point, ℓ, that agree with a communicated tuple are recorded by X. This information flows to all labels that follow immediate after ℓ

$$X = \{\bar{w} \in \hat{\mathsf{R}}.\ell \mid \Pi_{po\bar{u}@\mathsf{L}.\ell}(\bar{w}) \in \hat{\mathsf{K}}\}$$

Properties of the analysis

- Semantic correctness
  - we err on the safe side
- Adequacy result
  - capture of observable transitions
- Moore family property
  - we have best analysis results

Semantic correctness

Idea: $\hat{\mathsf{R}}$ and $\hat{\mathsf{K}}$ must capture the behaviour of s and **all** the services it may evolve into.

Problem: The acceptability of analysis results is NOT preserved by semantic transitions.

Reason: The use of X for filtering that is natural in the context of localised environments and pattern matching becomes less restrictive as variables are bound:

$$\begin{array}{ll} [x]^1 & p \bullet o?x^2.\mathbf{0} \\ & \mid p \bullet o!\mathsf{a}^3 \\ & \mid p \bullet o?x^4.p \bullet o!\langle\rangle^5 \end{array} \xrightarrow{p \bullet o \lfloor \emptyset \rfloor x^2 \mathsf{a}^3} p \bullet o?\mathsf{a}^4.p \bullet o!\langle\rangle^5$$

An analysis estimate with $\hat{\mathsf{R}}.4 = \{a, \bot\}$ and $\hat{\mathsf{R}}.5 = \{a\}$ is acceptable before, but not after, the transition.

Semantic correctness

Idea: Define a sensible notion of correctness and ensure that this is preserved by semantic transitions.

exposed actions

$$
\begin{array}{lcl}
\mathcal{E}(\mathbf{0}) & = & \emptyset \\
\mathcal{E}(*s) & = & \mathcal{E}(s) \\
\mathcal{E}((n)s) & = & \mathcal{E}(s) \\
\mathcal{E}([x]^{\ell}s) & = & \mathcal{E}(s) \\
\mathcal{E}(s_1 \mid s_2) & = & \mathcal{E}(s_1) \cup \mathcal{E}(s_2) \\
\mathcal{E}(s_1 + s_2) & = & \mathcal{E}(s_1) \cup \mathcal{E}(s_2) \\
\mathcal{E}(u \bullet u'!\bar{v}^{\ell}) & = & \{u \bullet u'!\bar{v}^{\ell}\} \\
\mathcal{E}(p \bullet o?\bar{v}^{\ell}.s) & = & \{p \bullet o?\bar{v}^{\ell}.s\}
\end{array}
$$

exposed labels

$$
\begin{array}{lcl}
\mathsf{E}(\mathbf{0}) & = & \emptyset \\
\mathsf{E}(*s) & = & \mathsf{E}(s) \\
\mathsf{E}((n)s) & = & \mathsf{E}(s) \\
\mathsf{E}([x]^{\ell}s) & = & \{\ell\} \\
\mathsf{E}(s_1 \mid s_2) & = & \mathsf{E}(s_1) \cup \mathsf{E}(s_2) \\
\mathsf{E}(s_1 + s_2) & = & \mathsf{E}(s_1) \cup \mathsf{E}(s_2) \\
\mathsf{E}(u \bullet u'!\bar{v}^{\ell}) & = & \{\ell\} \\
\mathsf{E}(p \bullet o?\bar{v}^{\ell}.s) & = & \{\ell\}
\end{array}
$$

Semantic correctness

In the context of a distinguished initial service, $s_\star$, an estimate, $(\hat{\mathsf{R}}, \hat{\mathsf{K}})$, is correct for a service, s, written $\hat{\mathsf{R}}, \hat{\mathsf{K}} \models^{s_\star} s$, iff:

1. $\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s_\star$
2. $\forall \ell \in \mathsf{E}(s_\star) : \epsilon \in \hat{\mathsf{R}}.\ell$
3. For all $p \bullet o?\bar{u}^\ell.s' \in \mathcal{E}(s)$ there is a subexpression $p \bullet o?\bar{v}^\ell.s''$ of $s_\star$ s.t.
   - $\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} p \bullet o?\bar{v}^\ell.s''$ and
   - $\exists \bar{w} \in \hat{\mathsf{R}}.\ell : p \bullet o?\bar{u}^\ell.s' \equiv p \bullet o?\bar{v}^\ell.s''[\bar{w}/\mathsf{L}.\ell]$
4. For all $v \bullet v'!\bar{v}^\ell \in \mathcal{E}(s)$ there is a subexpression $u \bullet u'!\bar{u}^\ell$ of $s_\star$ s.t.
   - $\hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} u \bullet u'!\bar{u}^\ell$ and
   - $\exists \bar{w} \in \hat{\mathsf{R}}.\ell : v \bullet v'!\bar{v}^\ell \equiv u \bullet u'!\bar{u}^\ell[\bar{w}/\mathsf{L}.\ell]$

Semantic correctness

Structural Congruence Lemma:
If $s \equiv s'$ then $\hat{R}, \hat{K} \models^{s_\star} s$ if and only if $\hat{R}, \hat{K} \models^{s_\star} s'$.

The proof is by induction on the inference of $s \equiv s'$.

Subject Reduction Theorem:
Assume $\hat{R}, \hat{K} \models^{s_\star} s$ and $s \xrightarrow{p \bullet o \lfloor \emptyset \rfloor \bar{u}^{\ell_i} \bar{n}^{\ell_o}} s'$.
Then $\hat{R}, \hat{K} \models^{s_\star} s'$.

The proof is by induction on the inference of $s \xrightarrow{p \bullet o \lfloor \emptyset \rfloor \bar{u}^{\ell_i} \bar{n}^{\ell_o}} s'$.

Adequacy Result: Capture of Observable Transitions:
If $(\hat{R}, \hat{K})$ is an acceptable analysis of $s_\star$
and if $s_\star \rightarrow^* s \xrightarrow{p \bullet o \lfloor \emptyset \rfloor \bar{u}^{\ell_o} \bar{n}^{\ell_i}} s'$
then $\langle p o \bar{n} \rangle \in \hat{K}$ and $\bar{n} \in \Pi_{\bar{u}@L.\ell_i}(\hat{R}.\ell_i)$.

Moore family property

All processes can be analysed and has a least (best) analysis result:

The set $\{(\hat{\mathsf{R}}, \hat{\mathsf{K}}) \mid \hat{\mathsf{R}}, \hat{\mathsf{K}} \vdash_{\mathsf{L},\mathsf{F}} s\}$ *is a Moore family*

If $\mathcal{Y} \subseteq \{\mathcal{I} \mid \mathcal{I} \models^{\star}_{\Gamma} s\}$ then $\sqcap\mathcal{Y} \models^{\star}_{\Gamma} s$ where $(\sqcap\mathcal{Y})(\mu) = \bigcap\{\mathcal{I}(\mu) \mid \mathcal{I} \in \mathcal{Y}\}$

The proof is by structural induction on P.

Corollaries:

- All processes can be analysed:
  — a Moore family cannot be empty
- All processes has a least (best) analysis result:
  — a Moore family has a least element

Suggested Reading

- F. Nielson, H. Riis Nielson, J. Bauer, C. R. Nielsen, H. Pilegaard: Relational Analysis for Delivery of Services. In **TGC 2007**, SLNCS 4912, pages 73-89, Springer, 2008.

  Performs a relational analysis for the π-calculus.

- J. Bauer, F. Nielson, H. Riis Nielson, H. Pilegaard: Relational analysis of Correlation. In **Proc. SAS'08**, SLNCS 5079, pages 32-46, Springer, 2008.

  Performs a relational analysis for COWS.

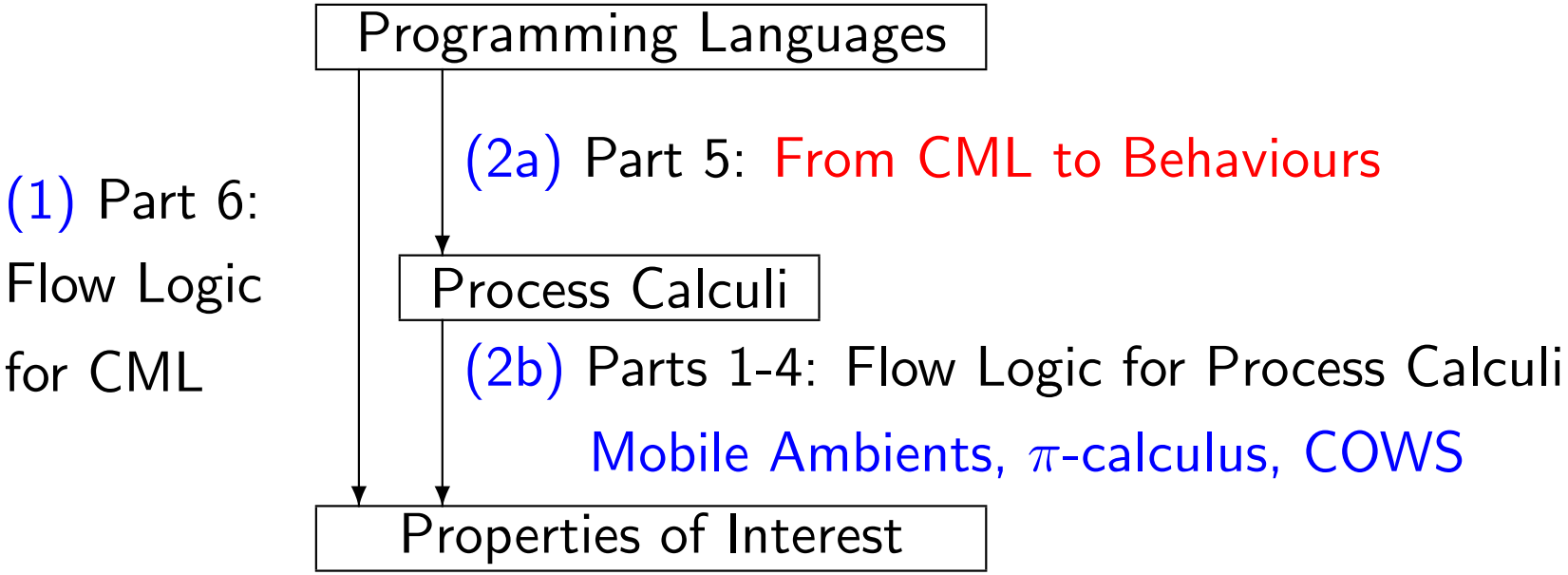
Static Analysis of Services

Part 5:

Extracting Process Calculus from Concurrent ML

The Grand View

Recall what we are doing:



Motivation

Concurrent ML

- functions and synchronous operations are first class values
- typed channels and processes are created dynamically

Hardware

- a finite number of channels and processors

Program analyses

- does the program have a finite communication topology
- what is the cost of the individual processors in static process allocation
- what is the cost of the individual processes in dynamic process allocation

CML subset

Expressions

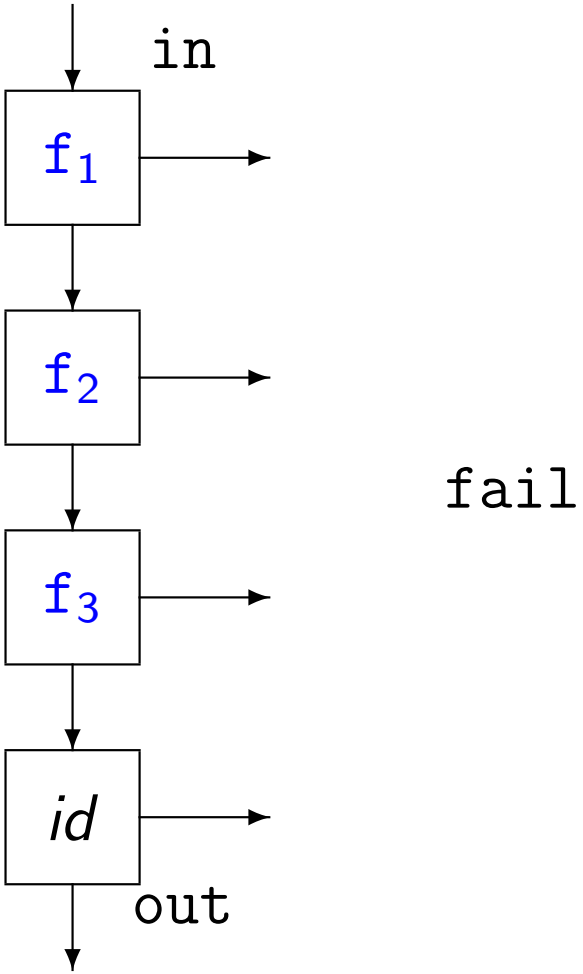
$$
\begin{array}{rcl}
e & ::= & c \mid x \mid \texttt{fn}\ x\ \texttt{=>}\ e \mid e_1\ e_2 \\
 & \mid & \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \\
 & \mid & \texttt{rec}\ f\ x\ \texttt{=>}\ e \\
 & \mid & \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2
\end{array}
$$

Constants:

$$
\begin{array}{rcl}
c & ::= & \texttt{()} \mid \texttt{true} \mid \texttt{false} \mid n \\
 & \mid & \texttt{pair} \mid \texttt{fst} \mid \texttt{snd} \\
 & \mid & \texttt{nil} \mid \texttt{cons} \mid \texttt{hd} \mid \texttt{tl} \mid \texttt{isnil} \\
 & \mid & \texttt{+} \mid \texttt{*} \mid \texttt{=} \mid \cdots \\
 & \mid & \texttt{send} \mid \texttt{receive} \mid \texttt{sync} \\
 & \mid & \texttt{fork}_\pi \mid \texttt{channel}_l \\
 & \mid & \texttt{choose} \mid \texttt{wrap}
\end{array}
$$

Example: pictorially

pipe $[f_1, f_2, f_3]$ in out



Example: CML program

```
let node =
   fn f => fn in => fn out =>
   fork_π
      (rec loop d =>
       sync (choose
               [wrap (receive in,
                  fn x =>
                  sync (send (out,
                                 f x));
                  loop d),
                send(fail,())]))
in rec pipe fs =>
   fn in => fn out =>
      if isnil fs
      then node (fn x => x) in out
      else let ch = channel_ch ()
            in (node (hd fs) in ch;
                 pipe (tl fs) ch out)
```

Example: behaviour

Assumptions:

$$\begin{array}{rcl} \texttt{f} & : & t_1 \longrightarrow^{b} t_2 \\ \texttt{fail} & : & \texttt{unit chan } r_0 \\ \texttt{in} & : & t_1 \texttt{ chan } r_1 \\ \texttt{out} & : & t_2 \texttt{ chan } r_2 \end{array}$$

$\texttt{node f in out:}\ \mathsf{FORK}_\pi\ (\mathsf{REC}\ \beta.\,(r_1?t_1;\ b;\ r_2!t_2;\ \beta + r_0!\mathsf{unit}))$

Fork a process that will
- read a value of type t_1 on a channel in r_1
- do the computation f with behaviour b
- write a value of type t_2 on a channel in r_2, and
- recurse

or
- write a value of type unit on a channel in r_0, and
- terminate

Example: behaviour

Assumptions:

$$\begin{array}{rcl} \texttt{fs} & : & (t \rightarrow^{b} t)\ \texttt{list} \\ \texttt{fail} & : & \texttt{unit chan}\ r_0 \\ \texttt{in} & : & t\ \texttt{chan}\ r_1 \\ \texttt{out} & : & t\ \texttt{chan}\ r_2 \end{array}$$

`pipe fs in out:`

$$\begin{array}{l} \mathsf{REC}\beta'.\ (\mathsf{FORK}_\pi\ (\mathsf{REC}\ \beta.(r_1?t;\ \epsilon;\ r_2!t;\ \beta + r_0!\mathsf{unit})) \\ \quad + \quad t\ \mathsf{CHAN}_{r_1}; \\ \qquad\ \ \mathsf{FORK}_\pi\ (\mathsf{REC}\ \beta.(r_1?t;\ b;\ r_2!t;\ \beta\ + r_0!\mathsf{unit}));\ \beta') \end{array}$$

- fork a process that ... or
- allocate a new channel,
- fork a process that ... and
- recurse

Typing system

Behaviours:

$$
\begin{array}{lcl}
b & ::= & \epsilon \mid r!t \mid r?t \mid t\ \mathrm{CHAN}_r \\
 & \mid & \mathrm{FORK}_\pi\ b \mid b_1;b_2 \mid b_1+b_2 \\
 & \mid & \mathrm{REC}\beta.b \mid \beta
\end{array}
$$

Types:

$$
\begin{array}{lcl}
t & ::= & \texttt{unit} \mid \texttt{bool} \mid \texttt{int} \mid \alpha \\
 & \mid & t_1 \rightarrow^b t_2 \mid t_1 \times t_2 \mid t\ \texttt{list} \\
 & \mid & t\ \texttt{chan}\ r \mid t\ \texttt{com}\ b
\end{array}
$$

Regions:

$$
\begin{array}{lcll}
r & ::= & l \mid r_1 + r_2 & \text{(sets of labels)} \\
 & \mid & \rho &
\end{array}
$$

Type schemes:

$$
ts \quad ::= \quad t \mid \forall\beta.ts \mid \forall\alpha.ts \mid \forall\rho.ts
$$

TypeOf(c)

send	$\forall \alpha, \rho.\ (\alpha\ \mathtt{chan}\ \rho) \times \alpha \rightarrow^{\epsilon} (\alpha\ \mathtt{com}\ \rho!\alpha)$
receive	$\forall \alpha, \rho.\ (\alpha\ \mathtt{chan}\ \rho) \rightarrow^{\epsilon} (\alpha\ \mathtt{com}\ \rho?\alpha)$
sync	$\forall \alpha, \beta.\ (\alpha\ \mathtt{com}\ \beta) \rightarrow^{\beta} \alpha$
$\mathtt{fork}_{\pi}$	$\forall \alpha, \beta.\ (\mathtt{unit} \rightarrow^{\beta} \alpha) \rightarrow^{\mathsf{FORK}_{\pi}\ \beta} \mathtt{unit}$
$\mathtt{channel}_{l}$	$\forall \alpha.\ \mathtt{unit} \rightarrow^{\alpha\ \mathsf{CHAN}_{l}} (\alpha\ \mathtt{chan}\ l)$
choose	$\forall \alpha, \beta.\ (\alpha\ \mathtt{com}\ \beta)\ \mathtt{list} \rightarrow^{\epsilon} (\alpha\ \mathtt{com}\ \beta)$
wrap	$\forall \alpha_1, \alpha_2, \beta_1, \beta_2.(\alpha_1\ \mathtt{com}\ \beta_1) \times (\alpha_1 \rightarrow^{\beta_2} \alpha_2) \rightarrow^{\epsilon} (\alpha_2\ \mathtt{com}\ \beta_1; \beta_2)$

To allow enlarging the behaviour:
replace $t_1 \rightarrow^{b} t_2$ by $t_1 \rightarrow^{\beta} t_2\ [\beta \geq b]$.

$\vdash e$: type & behaviour

$\vdash c : t \;\&\; \epsilon$ if TypeOf(c) $\succ t$

$[x \mapsto ts] \vdash x : t \;\&\; \epsilon$ if $ts \succ t$

$$\frac{[x \mapsto t] \vdash e : t' \;\&\; b}{\vdash \texttt{fn}\ x \Rightarrow e : t \rightarrow^{b} t' \;\&\; \epsilon}$$

$$\frac{\vdash e_1 : t \rightarrow^{b} t' \;\&\; b_1 \qquad \vdash e_2 : t \;\&\; b_2}{\vdash e_1\ e_2 : t' \;\&\; b_1; b_2; b}$$

$$\frac{\vdash e_1 : t_1 \;\&\; b_1 \qquad [x \mapsto ts] \vdash e_2 : t_2 \;\&\; b_2}{\vdash \texttt{let}\ x{=}e_1\ \texttt{in}\ e_2 : t_2 \;\&\; b_1; b_2} \quad \text{if } ts = \text{gen}(,b_1)t_1$$

$$\frac{\vdash e : t \;\&\; b}{\vdash e : t \;\&\; b'} \quad \text{if } b \sqsubseteq b'$$

Ignoring the type environment.

Ordering on behaviours

Axioms and rules that formally state:

- $\sqsubseteq$ is a pre-order
- $\sqsubseteq$ is a pre-congruence
- sequencing is associative
- sequencing distributes over join

$$(b_1 + b_2); b_3 \;\sqsubseteq\; b_1; b_3 + b_2; b_3$$

$$b_1; b_3 + b_2; b_3 \;\sqsubseteq\; (b_1 + b_2); b_3$$

- ϵ is left and right identity for sequencing
- join is least upper bound operation
- recursion can be unfolded

$$\text{REC } \beta.b \;\sqsubseteq\; b[\beta \mapsto \text{REC } \beta.b]$$

$$b[\beta \mapsto \text{REC } \beta.b] \;\sqsubseteq\; \text{REC } \beta.b$$

Example (1)

```
let node = fn_F f => fn_I inp => fn_O out =>
           spawn ((fun_H h d => let v = receive inp
                                in send (f v) on out; h d) ())
in ···
```

Type for node:

$$\forall\, {}'a, {}'b, {}'1, {}''1, {}''2.\ \underbrace{({}'a \xrightarrow{{}'1} {}'b)}_{\texttt{f}} \xrightarrow{\Lambda} \underbrace{({}'a\ \texttt{chan}\ {}''1)}_{\texttt{inp}} \xrightarrow{\Lambda} \underbrace{({}'b\ \texttt{chan}\ {}''2)}_{\texttt{out}} \xrightarrow{\varphi} \texttt{unit}$$

where $\varphi = \mathsf{spawn}(\mathsf{rec}\ {}'2.\ ({}''1?{}'a;\ {}'1;\ {}''2!{}'b;\ {}'2))$

Example (2)

```
let node = ···
in funP pipe fs => fnI inp => fnO out =>
        if isnil fs then node (fnX x => x) inp out
        else let ch = channelC
             in (node (hd fs) inp ch; pipe (tl fs) ch out)
```

Type for `pipe`:

$$\forall\, 'a, '1, ''1, ''2.$$

$$\underbrace{(('a \xrightarrow{'1} 'a)\ \texttt{list})}_{\texttt{fs}} \xrightarrow{\Lambda} \underbrace{('a\ \texttt{chan}\ (''1 \cup \{C\}))}_{\texttt{inp, ch}} \xrightarrow{\Lambda} \underbrace{('a\ \texttt{chan}\ ''2)}_{\texttt{out}} \xrightarrow{\varphi} \texttt{unit}$$

where $\varphi =$

$$\text{rec } '2.\ (\underbrace{\text{spawn}(\text{rec } '3.((''1 \cup \{C\})?'a;\ \Lambda;\ ''2!'a;\ '3))}_{\texttt{node (fn x => x) ...}}$$

$$+\ 'a\ \text{chan}\ C;\ \underbrace{\text{spawn}(\text{rec } '4.\ ((''1 \cup \{C\})?'a;\ '1;\ C!'a; '4))}_{\texttt{node (hd fs) ...}};\ '2)$$

Theory: overview

Structural operational semantics for

- CML
- behaviours

Subject Reduction Theorem:

- types are preserved by CML-evaluation steps
- steps in CML-semantics can be mimicked in behaviour semantics

Semantics of CML: Expressions

$$(\texttt{fn}_\pi\ x \texttt{ => } e)\ v \longrightarrow e[x \mapsto v]$$

$$\texttt{let } x \texttt{ = } v \texttt{ in } e \longrightarrow e[x \mapsto v]$$

$$v_1\ op\ v_2 \longrightarrow v \quad \text{if } v_1\ \mathbf{op}\ v_2 = v$$

$$\texttt{fun}_\pi\ f\ x \texttt{ => } e \longrightarrow (\texttt{fn}_\pi\ x \texttt{ => } e)[f \mapsto (\texttt{fun}_\pi\ f\ x \texttt{ => } e)]$$

$$\texttt{if true then } e_1 \texttt{ else } e_2 \longrightarrow e_1$$

$$\texttt{if false then } e_1 \texttt{ else } e_2 \longrightarrow e_2$$

$$v; e \longrightarrow e$$

Evaluation contexts:

$$\begin{array}{rcl} E & ::= & [\,] \mid E\ e \mid v\ E \mid \texttt{let } x \texttt{ = } E \texttt{ in } e \mid \texttt{if } E \texttt{ then } e_1 \texttt{ else } e_2 \mid E\ op\ e \\ & \mid & v\ op\ E \mid \texttt{send } E \texttt{ on } e \mid \texttt{send } v \texttt{ on } E \mid \texttt{receive } E \mid E; e \end{array}$$

Semantics of CML: Threads

$$CP, PP[p : E[e_1]] \Rightarrow CP, PP[p : E[e_2]]$$
$$\text{if } e_1 \rightarrow e_2$$

$$CP, PP[p : E[\texttt{channel}_\pi]] \Rightarrow CP \cup \{ch\}, PP[p : E[ch]]$$
$$\text{if } ch \notin CP$$

$$CP, PP[p : E[\texttt{spawn } e_0]] \Rightarrow CP, PP[p : E[()]][p_0 : e_0]$$
$$\text{if } p_0 \notin dom(PP) \cup \{p\}$$

$$CP, PP[p_1 : E_1[\texttt{send } v \texttt{ on } ch]][p_2 : E_2[\texttt{receive } ch]]$$
$$\Rightarrow CP, PP[p_1 : E_1[()]][p_2 : E_2[v]]$$
$$\text{if } p_1 \neq p_2$$

Semantics of CML: Summary

Sequential evaluation:

$$E[(\texttt{fn}\ x{=}{>}e)w] \rightarrow E[e[x \mapsto w]]$$

Concurrent evaluation:

$$\frac{(w_1, w_2) \overset{(ci!,ci?)}{\rightsquigarrow} (e_1, e_2)}{[pi_1 \mapsto E_1[\texttt{sync}\ w_1], pi_2 \mapsto E_2[\texttt{sync}\ w_2]]}$$

$$\longrightarrow^{(ci!,ci?)}_{pi_1,pi_2} [pi_1 \mapsto E_1[e_1], pi_2 \mapsto E_2[e_2]] \quad \text{if } pi_1 \neq pi_2$$

Annotation of $\longrightarrow^{ev}$:

$$\begin{array}{rcl} ev & ::= & \epsilon \mid (ci!, ci?) \\ & \mid & \mathrm{CHAN}_r\ ci \mid \mathrm{FORK}_\pi\ pi \end{array}$$

Matching:

$$(\langle \texttt{send}\langle \texttt{pair}ci\ w\rangle\rangle, \langle \texttt{receive}ci\rangle) \overset{(ci!,ci?)}{\rightsquigarrow} (w, w)$$

Semantics of behaviours

Sequential evolution:

$$\frac{b_1 \Rightarrow^{p} b_1'}{b_1; b_2 \Rightarrow^{p} b_1'; b_2} \qquad \frac{b_1 \Rightarrow^{p} \surd}{b_1; b_2 \Rightarrow^{p} b_2}$$

Annotations of $\Rightarrow^{p}$:

$$\begin{array}{rcl} p & ::= & \epsilon \mid r!t \mid r?t \\ & \mid & t\ \mathrm{CHAN}_r \mid \mathrm{FORK}_\pi\ b \end{array}$$

Concurrent evolution:

$$\frac{b_1 \Rightarrow^{r!t} b_1' \quad b_2 \Rightarrow^{r?t} b_2'}{[pi_1 \mapsto b_1, pi_2 \mapsto b_2] \Longrightarrow^{r!t?r}_{pi_1, pi_2} [pi_1 \mapsto b_1', pi_2 \mapsto b_2']} \quad \text{if } pi_1 \neq pi_2$$

Annotations on $\Longrightarrow^{a}$:

$$\begin{array}{rcl} a & ::= & \epsilon \mid r!t?r \\ & \mid & t\ \mathrm{CHAN}_r \mid \mathrm{FORK}_\pi\ b \end{array}$$

Simulation on behaviours

$\mathcal{S}$ is a simulation on (closed) behaviours if

- $\surd \; \mathcal{S} \; b$ if and only if $b = \surd$
- if $b_1 \Rightarrow^{p_1} b_1'$ and $b_1 \; \mathcal{S} \; b_2$ then there exists b_2 and p_2 such that
  - $b_2 \Rightarrow^{\widehat{p_2}} b_2'$,
  - $p_1 \; \mathcal{S}^{\partial} \; p_2$
  - $b_1' \; \mathcal{S} \; b_2'$.

where

$\mathcal{S}^{\partial} = \{(p,p) \mid p \in \{\epsilon, r!t, r?t, t \text{ CHAN}_r\}\} \cup \{(\text{FORK}_\pi \; b, \text{FORK}_\pi \; b') \mid b \; \mathcal{S} \; b'\}$

Define $\sqsubseteq_{\sim}$ as the largest simulation.

Lemma: Soundness of Ordering $\sqsubseteq$ is a simulation.

The ordering $\sqsubseteq_{\sim}$ is undecidable.

Subject reduction theorem

$$
\begin{tikzcd}
PP[pi \mapsto e] \arrow[r, "ev"', "ps"] & PP'[pi \mapsto e'] \\
\vdash e : t \;\&\; b \arrow[u, leftrightarrow] \arrow[d, leftrightarrow] & \vdash e' : t \;\&\; b' \arrow[u, leftrightarrow] \arrow[d, leftrightarrow, "b' \sqsubseteq b''"] \\
PB[pi \mapsto b] \arrow[r, Rightarrow, "\hat{a}"', "ps"] & PB'[pi \mapsto b'']
\end{tikzcd}
$$

where a is 'the translation of' ev

Ignoring channel identifiers and channel environments.

Algorithm

The algorithm is based on Milner's $\mathcal{W}$

- use unification for the types
- collect constraints for the behaviours

Problem: Constraints need not have principal solutions: incomparable solutions could be mixed.

$\mathcal{W}(e) = (s, d, C, S)$ where

s: simple type: only behaviour variables

d: simple behaviour: no REC behaviours

C: set of constraints

S: set of solution restrictions

Suggested Reading (1)

- T. Amtoft, H. Riis Nielson, F. Nielson: Behavior Analysis for Validating Communication Patterns. In **Journal on Software Tools for Technology Transfer**, vol.2, pages 13-28, Springer, 1998.

  Gives an overview of the development and of a computer system for carrying out the analysis.

- F. Nielson, H. Riis Nielson: Type and Effect Systems. In **Correct System Design**. Springer LNCS 1710, pages 114-136, Springer, 1999.

  Contains a survey of type and effect systems.

Suggested Reading (2)

- F. Nielson, H. Riis Nielson, C. L. Hankin: Principles of Program Analysis. Springer, 1999, revised 2005.

  Chapter 5 contains a tutorial on Type and Effect Systems. Section 5.5 summarizes the present development.

- T. Amtoft, F. Nielson, H. Riis Nielson: Type and Effect Systems: Behaviours for Concurrency. Imperial College Press, 1999.

  This book contains the full development.

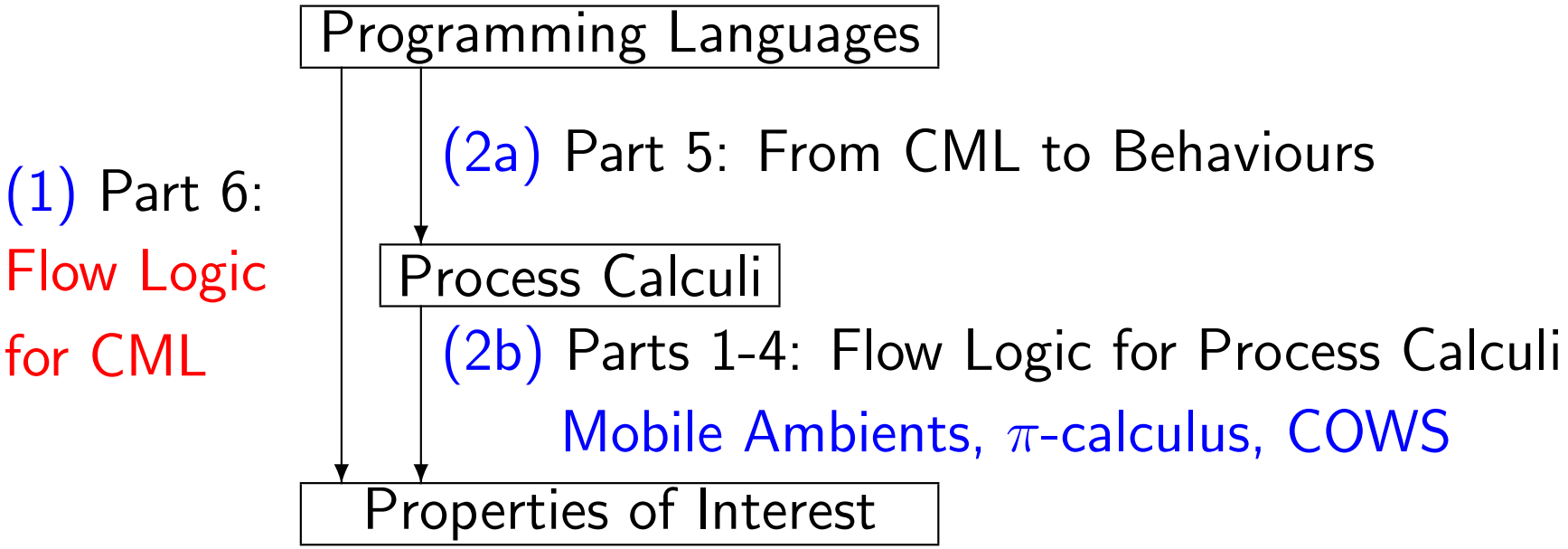
Static Analysis of Services

Part 6:

A Traditional Approach to Flow Logic for Concurrent ML

The Grand View

Recall what we are doing:



Combining paradigms: CML

$$
\begin{array}{rcl}
e & ::= & c \mid x \mid \texttt{fn}\ x_0 \texttt{=>} e_0 \mid \texttt{fun}\ f\ x_0 \texttt{=>} e_0 \mid e_1\ e_2 \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \cdots \\
& \mid & e_1 := e_2 \mid e_1; e_2 \mid \texttt{let}\ x = \texttt{ref}_r\ \texttt{in}\ e \mid \texttt{deref}\ e \mid \cdots \\
& \mid & \texttt{send}\ e_1\ \texttt{on}\ e_2 \mid \texttt{receive}\ e \mid \texttt{let}\ x = \texttt{chan}_n\ \texttt{in}\ e \mid \texttt{spawn}\ e \mid \cdots
\end{array}
$$

Judgements: $(\widehat{\rho}, \widehat{\sigma}, \widehat{\kappa}) \models e : \widehat{v}$

- $\widehat{\rho} : \mathbf{Var} \to \widehat{\mathbf{Val}}$ records the abstract values bound to variables
- $\widehat{\sigma} : \mathbf{Ref} \to \widehat{\mathbf{Val}}$ records the abstract values bound to reference cells
- $\widehat{\kappa} : \mathbf{Ch} \to \widehat{\mathbf{Val}}$ records the abstract values communicated over channels.

Abstract values are constants $\diamond$, abstract closures $\langle\!\langle \texttt{fn}\ x_0 \texttt{=>} e_0 \rangle\!\rangle$, recursive abstract closures $\langle\!\langle \texttt{fun}\ f\ x_0 \texttt{=>} e_0 \rangle\!\rangle$, reference cells r and channels n.

Functional constructs:

$$
\begin{array}{rcl}
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models c : \widehat{v} & \underline{\text{iff}} & \diamond \in \widehat{v} \\
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models x : \widehat{v} & \underline{\text{iff}} & \widehat{\rho}(x) \subseteq \widehat{v} \\
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{fn}\ x_0 \texttt{=>} e_0 : \widehat{v} & \underline{\text{iff}} & \langle\!\langle \texttt{fn}\ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{v} \\
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{fun}\ f\ x_0 \texttt{=>} e_0 : \widehat{v} & \underline{\text{iff}} & \langle\!\langle \texttt{fun}\ f\ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{v} \\
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_1\ e_2 : \widehat{v} & \underline{\text{iff}} & (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_1 : \widehat{v}_1 \ \wedge\ (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_2 : \widehat{v}_2 \ \wedge \\
 & & \forall \langle\!\langle \texttt{fn}\ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{v}_1 : \widehat{v}_2 \neq \emptyset \Rightarrow \\
 & & \quad [\widehat{v}_2 \subseteq \widehat{\rho}(x_0) \wedge (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_0 : \widehat{v}_0 \wedge \widehat{v}_0 \subseteq \widehat{v}] \\
 & & \forall \langle\!\langle \texttt{fun}\ f\ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{v}_1 : \widehat{v}_2 \neq \emptyset \Rightarrow \\
 & & \quad [\langle\!\langle \texttt{fun}\ f\ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{\rho}(f) \wedge \widehat{v}_2 \subseteq \widehat{\rho}(x_0) \wedge \\
 & & \quad\ (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_0 : \widehat{v}_0 \wedge \widehat{v}_0 \subseteq \widehat{v}]
\end{array}
$$

Functional constructs (cont.):

$$(\widehat{\rho}, \widehat{\sigma}, \widehat{\kappa}) \models \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \widehat{v}\ \underline{\text{iff}}\ (\widehat{\rho}, \widehat{\sigma}, \widehat{\kappa}) \models e_0 : \widehat{v}_0 \wedge \diamond \in \widehat{v}_0 \Rightarrow [(\widehat{\rho}, \widehat{\sigma}, \widehat{\kappa}) \models e_1 : \widehat{v}_1 \wedge \widehat{v}_1 \sqsubseteq \widehat{v} \wedge (\widehat{\rho}, \widehat{\sigma}, \widehat{\kappa}) \models e_2 : \widehat{v}_2 \wedge \widehat{v}_2 \sqsubseteq \widehat{v}]$$

Imperative constructs:

$$
\begin{array}{rcl}
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_1 := e_2 : \widehat{v} & \underline{\text{iff}} & (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_1 : \widehat{v}_1 \ \wedge \\
 & & (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_2 : \widehat{v}_2 \ \wedge \\
 & & \diamond \in \widehat{v} \ \wedge \ \forall r \in \widehat{v}_1 : \widehat{v}_2 \subseteq \widehat{\sigma}(r) \\
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{deref}\ e : \widehat{v} & \underline{\text{iff}} & (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e : \widehat{v}' \ \wedge \\
 & & \forall r \in \widehat{v}' : \widehat{\sigma}(r) \subseteq \widehat{v} \\
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{let}\ x = \texttt{ref}_r\ \texttt{in}\ e : \widehat{v} & \underline{\text{iff}} & r \in \widehat{\rho}(x) \ \wedge \ (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e : \widehat{v} \\
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_1; e_2 : \widehat{v} & \underline{\text{iff}} & (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_1 : \widehat{v}' \ \wedge \\
 & & (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_2 : \widehat{v}
\end{array}
$$

Concurrency constructs:

$$(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{send}\ e_1\ \texttt{on}\ e_2 : \widehat{v} \quad \underline{\text{iff}} \quad \begin{array}{l}(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_1 : \widehat{v}_1 \ \wedge \\ (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_2 : \widehat{v}_2 \ \wedge \\ \diamond \in \widehat{v} \ \wedge \ \forall n \in \widehat{v}_2 : \widehat{v}_1 \subseteq \widehat{\kappa}(n)\end{array}$$

$$(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{receive}\ e : \widehat{v} \quad \underline{\text{iff}} \quad \begin{array}{l}(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e : \widehat{v}' \ \wedge \\ \forall n \in \widehat{v}' : \widehat{\kappa}(n) \subseteq \widehat{v}\end{array}$$

$$(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{let}\ x = \texttt{chan}_n\ \texttt{in}\ e : \widehat{v} \quad \underline{\text{iff}} \quad n \in \widehat{\rho}(x) \ \wedge \ (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e : \widehat{v}$$

Concurrency constructs (cont.):

$$
\begin{array}{lll}
(\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models \texttt{spawn}\ e : \widehat{v} & \underline{\text{iff}} & \diamond \in \widehat{v} \ \wedge\ (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e : \widehat{v}' \ \wedge \\
& & \forall \langle\!\langle \texttt{fn}\ \ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{v}' : \\
& & \qquad \diamond \subseteq \widehat{\rho}(x_0) \wedge (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_0 : \widehat{v}_0 \\
& & \forall \langle\!\langle \texttt{fun}\ f\ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{v}' : \\
& & \qquad \langle\!\langle \texttt{fun}\ f\ x_0 \texttt{=>} e_0 \rangle\!\rangle \in \widehat{\rho}(f) \wedge \diamond \subseteq \widehat{\rho}(x_0) \wedge \\
& & \qquad\qquad (\widehat{\rho},\widehat{\sigma},\widehat{\kappa}) \models e_0 : \widehat{v}_0
\end{array}
$$

Suggested Reading

- H. Riis Nielson and F. Nielson. Flow Logic: a Multi-paradigmatic Approach to Static Analysis. In *The Essence of Computation: Complexity, Analysis, Transformation*, SLNCS 2566, pages 223 - 244, Springer, 2002.

  Section 4.3 gives a brief overview.

- K. L. Solberg Gasser and F. Nielson and H. Riis Nielson. Systematic Realisation of Control Flow Analyses for CML. In *Proc. ICFP'97*, pages 38 - 51, ACM Press, 1997.

  Contains the full development.

Static Analysis of Services

More on Flow Logic:

http://www.imm.dtu.dk/∼nielson/FlowLogic.html

MT-LAB - a VKR-Centre of Excellence:

http://www.MT-LAB.dk