π without α

António Ravara
with Adrian Francalanza and Marco Giunti

NOVA-LINCS and Dep of Informatics
School of Science and Technology, NOVA University of Lisboa, PT
Univ of Malta, Malta

June 5, 2020

What are we aiming at

Static detection of locked channels

- find bugs, not prevent them
- automatic analysis to unearth real problems
- Identify problematic pieces of code and misuse of resources

Notice: when looking for bugs
(instead of aiming at avoiding them)

- correctness means all issues found are true positives
  when avoiding, correctness means no false negatives
- completeness means all bugs are found
  when avoiding, unachievable completeness implies some false positives

We want to be precise (although not exhaustive)

Motivation: our favorite framework

A Model of Distributed Systems

π-calculus featuring:

- replication
- linear /unrestricted names

Linear π-calculus, Kobayashi, Pierce, and Turner, 1999

Example

Let u, v be unrestricted in the following processe.

$$P = *u?(x).v?(y).x!\mathit{true}.y?(z).0 \parallel Q = *(\kappa\, c : \mathit{linear})(u!c \parallel v!c)$$

It may "continuously" produce deadlocks on fresh linear names

Motivation: informative program analysis

Let u, v be unrestricted in the following program.

$$
\begin{aligned}
P &= \; * \, u?(x).v?(y).x!\mathit{true}.y?(z).0 \\
Q &= \; * \, (\kappa \, c : \mathit{linear})(u!c \parallel v!c) \\
R &= P \parallel Q
\end{aligned}
$$

$$
\begin{aligned}
R \; &\longrightarrow^2 R \parallel (\kappa \, c_1 : \mathit{linear})(c_1!\mathit{true}.c_1?(z).0) \\
&\longrightarrow^2 R \parallel (\kappa \, c_1 : \mathit{linear})(c_1!\mathit{true}.c_1?(z).0) \\
&\qquad\quad \parallel (\kappa \, c_2 : \mathit{linear})(c_2!\mathit{true}.c_2?(z).0) \\
&\longrightarrow^2 R \; \cdots
\end{aligned}
$$

- Program analysis might detect the problem in name c_1
- It should report a problem in c, referring to the static code

A certified program analysis tool

Goals for the implementation

- to implement an efficient mechanism of capturing-avoiding substitution
- to ensure the absence of clashes on (bound) identifiers

Our approach: unique identifiers

Goals for the mechanisation

- proof correct exactly the implementation's code (not some idealised version of it)
- deal with name generation explicitly

In this talk

A reformulation of the linear pi-calculus

does not assume alpha-conversion

automatically book-keeps information regarding name scoping

The labelled transition system tracks the evolution of linear permissions

renaming of scoped names to avoid clashes performed with a total function generating natural numbers not used elsewhere.

A main concern: not to introduce non-determinism unnecessarily

type-splitting only in the parallel composition rule, showed determinisable

Language

Process and Type Syntax

$$
\begin{array}{rlll}
m \in \mathrm{M{\scriptsize UL}} ::= \omega & \text{(unrestricted)} & \mid\ \iota & \text{(linear)} \\
p, q \in \mathrm{P{\scriptsize OL}} ::= \updownarrow & \text{(input \& output)} & \mid\ \emptyset & \text{(empty)} \\
\mid\ \downarrow & \text{(input)} & \mid\ \uparrow & \text{(output)} \\
T, S \in \mathrm{T{\scriptsize YP}} ::= \mathsf{bs} & \text{(base)} & \mid\ p[T]^m & \text{(channel)} \\
P, Q, R \in \mathrm{P{\scriptsize ROC}} ::= \mathsf{nil} & \text{(inert)} & \mid\ P \parallel Q & \text{(composition)} \\
\mid\ u?x.P & \text{(input)} & \mid\ *u?x.P & \text{(replication)} \\
\mid\ u!v.P & \text{(output)} & \mid\ (\kappa\, n)P & \text{(hiding)}
\end{array}
$$

Remarks on the language

- "Normal" binders for variables of input prefixes – static scope
- No binders for names – dynamic scope
  We use a Church-style hiding construct, $(\kappa\, n)P$, that does *not* assume alpha-conversion.
- all hidden (restricted) names are disjoint from one another (no duplicates in hidden names) and also disjoint from visible (free) names.

Well-formed processes

A process P is *well-formed* iff $\mathsf{noDup}(\mathsf{hid}(P))$ and $\mathsf{hid}(P) \cap \mathsf{vis}(P) = \emptyset$.

where: $\mathsf{hid}(P)$ is the multi-set of the hidden names of P, $\mathsf{vis}(P)$ is the multi-set of the names of P that are not hidden, and
$\mathsf{noDup}(M) = \nexists M', n \cdot M = M' \uplus \{n, n\}$

Operations on Types

Type Operations

$$\updownarrow \sqcup p \triangleq \updownarrow \qquad \emptyset \sqcup p \triangleq p \qquad \downarrow \sqcup \downarrow \triangleq \downarrow \qquad \uparrow \sqcup \uparrow \triangleq \uparrow \qquad \downarrow \sqcup \uparrow \triangleq \updownarrow$$

$$\emptyset \uplus p \triangleq p \qquad \downarrow \uplus \uparrow \triangleq \updownarrow$$

$$\mathsf{bs}+\mathsf{bs} \triangleq \mathsf{bs} \qquad p[T]^\iota + q[T]^\iota \triangleq p \uplus q[T]^\iota \qquad p[T]^\omega + q[T]^\omega \triangleq p \sqcup q[T]^\omega$$

Type subtraction

$$\mathsf{bs} - \mathsf{bs} = \mathsf{bs} \qquad p[T]^\iota - p[T]^\iota = \emptyset[T]^\iota \qquad \updownarrow [T]^\iota - \downarrow [T]^\iota = \uparrow [T]^\iota$$

$$\updownarrow [T]^\iota - \uparrow [T]^\iota = \downarrow [T]^\iota \qquad p[T]^\omega - p[T]^\omega = p[T]^\omega$$

$$\updownarrow [T]^\omega - \downarrow [T]^\omega = \updownarrow [T]^\omega \qquad \updownarrow [T]^\omega - \uparrow [T]^\omega = \updownarrow [T]^\omega$$

$$p[T]^m - \emptyset[T]^m = p[T]^m$$

Type Environments

Flags

- *hidden* ($\mathtt{h}$, under a scope declaration)
- *illegal* ($\mathtt{i}$, not usable by the process)
- *visible* ($\mathtt{v}$, neither scoped nor illegal).

Flag Combination

$$\mathtt{h} + \mathtt{i} = \mathtt{h} = \mathtt{i} + \mathtt{h} \qquad \mathtt{v} + \mathtt{v} = \mathtt{v} \qquad \mathtt{i} + \mathtt{i} = \mathtt{i}$$

Environment Splitting Relation

$$\frac{\Gamma_1 + \Gamma_2 = \Gamma_3}{(\Gamma_1, u{:}(\mathtt{f}, p[T]^{\iota})) + (\Gamma_2, u{:}(\mathtt{g}, q[T]^{\iota})) = \Gamma_3, u{:}(\mathtt{f} + \mathtt{g}, p \uplus q[T]^{\iota})}$$

$$\frac{\Gamma_1 + \Gamma_2 = \Gamma_3}{(\Gamma_1, u{:}(\mathtt{f}, p[T]^{\omega})) + (\Gamma_2, u{:}(\mathtt{g}, p[T]^{\omega})) = \Gamma_3, u{:}(\mathtt{f} + \mathtt{g}, p[T]^{\omega})}$$

Type System

Consumed predicate cons($\emptyset$)

$$\frac{\textbf{cons}(\Gamma)}{\textbf{cons}(\Gamma, u\colon (\mathtt{f}, \mathrm{bs}))} \qquad \frac{\textbf{cons}(\Gamma) \qquad \mathtt{f} \neq \mathtt{i}}{\textbf{cons}(\Gamma, u\colon (\mathtt{f}, p[T]^{\omega}))} \qquad \frac{\textbf{cons}(\Gamma)}{\textbf{cons}(\Gamma, u\colon (\mathtt{f}, \emptyset[T]^{m}))}$$

Typing Rules

$$\frac{\textbf{cons}(\Gamma)}{\Gamma \vdash \mathsf{nil}} \qquad \frac{\Gamma, n{:}(\mathtt{v}, T) \vdash P}{\Gamma, n{:}(\mathtt{h}, T) \vdash (\kappa\, n)\,(P)} \qquad \frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \parallel P_2}$$

$$\frac{\Gamma, u{:}(\mathtt{v}, p[T]^m - \uparrow[T]^m), v{:}(\mathtt{v}, T' - T) \vdash P \qquad \uparrow \in p}{\Gamma, u{:}(\mathtt{v}, p[T]^m), v{:}(\mathtt{v}, T') \vdash u!v.P}$$

$$\frac{\Gamma, u{:}(\mathtt{v}, p[T]^m - \downarrow[T]^m), x{:}(\mathtt{v}, T) \vdash P \qquad \downarrow \in p}{\Gamma, u{:}(\mathtt{v}, p[T]^m) \vdash u?x.P}$$

$$\frac{\Gamma, u{:}(\mathtt{v}, p[T]^{\omega}), x{:}(\mathtt{v}, T) \vdash P \qquad \downarrow \in p \qquad \textbf{cons}(\Gamma)}{\Gamma, u{:}(\mathtt{v}, p[T]^{\omega}) \vdash {*u?x.P}}$$

Type Soundness

Linearity Violation

A configuration $\Gamma \triangleright P$ *violates linearity* if there exists an evaluation context and a channel n such that $\Gamma(n) = (\mathtt{f}, p[T]^{\iota})$, and either:

1. $P = \mathcal{E}[n?x_1.Q_1, n?x_2.Q_2]$; or
2. $P = \mathcal{E}[n!v_1.Q_1, n!v_2.Q_2]$; or
3. $P = \mathcal{E}[*n?x.Q]$.

Immediate Race-freedom

If $\Gamma \vdash P$ then $\Gamma \triangleright P$ does not violate linearity.

Labelled Transition System – I

Relabelling

Replication comports that a transition spawns a *relabelled* copy of the input prefix, to preserve the (representation of the) Barendregt convention.

Typed Transitions (Selected rules)

$$\frac{T'' = T' - T}{\Gamma, a{:}(\mathtt{v}, \uparrow[T]^{\iota}), n{:}(\mathtt{v}, T') \triangleright a!n.P \xrightarrow{a!n} \Gamma, a{:}(\mathtt{v}, \emptyset[T]^{\iota}), n{:}(\mathtt{v}, T'') \triangleright P}$$

$$\frac{T'' = T' + T \qquad \mathtt{f} \neq \mathtt{h}}{\Gamma, a{:}(\mathtt{v}, \downarrow[T]^{\iota}), n{:}(\mathtt{f}, T') \triangleright a?x.P \xrightarrow{a?n} \Gamma, a{:}(\mathtt{v}, \emptyset[T]^{\iota}), n{:}(\mathtt{v}, T'') \triangleright P[{}^{n}\!/x]}$$

$$\frac{\downarrow \in p \quad T'' = T' + T \quad \mathtt{f} \neq \mathtt{h} \quad (\Gamma', P') = \mathrm{relabelling}((\Gamma, c{:}(\mathtt{v}, p[T]^{\omega}), n{:}(\mathtt{f}, T')), P)}{\Gamma, c{:}(\mathtt{v}, p[T]^{\omega}), n{:}(\mathtt{f}, T') \triangleright *c?x.P \xrightarrow{c?n} \Gamma, c{:}(\mathtt{v}, p[T]^{\omega}), n{:}(\mathtt{v}, T''), \Gamma' \triangleright P[{}^{n}\!/x] \parallel *c?x.P'}$$

If $\mathrm{relabelling}(\Gamma, P) = (\Gamma', P')$ then $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \emptyset$

Labelled Transition System – II

Typed Transitions (Selected rules)

$$\frac{\Gamma \triangleright P_1 \xrightarrow{\alpha} \Gamma' \triangleright P_1'}{\Gamma \triangleright P_1 \parallel P_2 \xrightarrow{\alpha} \Gamma' \triangleright P_1' \parallel P_2}
\qquad
\frac{\Gamma, n{:}(\mathtt{v}, T) \triangleright P \xrightarrow{n'!n} \Gamma' \triangleright P'}{\Gamma, n{:}(\mathtt{h}, T) \triangleright (\kappa\, n)\,(P) \xrightarrow{n'!n} \Gamma' \triangleright P'}$$

$$\frac{\begin{array}{c} S' = S - T \qquad \Gamma, c{:}(\mathtt{v}, \updownarrow [T]^{\omega}, n{:}(\mathtt{v}, S)) \triangleright P_1 \xrightarrow{c!n} \Gamma_1 \triangleright P_1' \\ \Gamma, c{:}(\mathtt{v}, \updownarrow [T]^{\omega}), n{:}(\mathtt{v}, S') \triangleright P_2 \xrightarrow{c?n} \Gamma_2, c{:}(\mathtt{v}, \updownarrow [T]^{\omega}), n{:}(\mathtt{v}, S) \triangleright P_2' \end{array}}{\Gamma, c{:}(\mathtt{v}, \updownarrow [T]^{\omega}), n{:}(\mathtt{v}, S) \triangleright P_1 \parallel P_2 \xrightarrow{\tau} \Gamma_2, c{:}(\mathtt{v}, \updownarrow [T]^{\omega}), n{:}(\mathtt{v}, S) \triangleright P_1' \parallel P_2'}$$

$$\frac{\begin{array}{c} S' = S - T \qquad \Gamma, a{:}(\mathtt{v}, \uparrow [T]^{\iota}, n{:}(\mathtt{h}, S)) \triangleright P_1 \xrightarrow{a!n} \Gamma_1 \triangleright P_1' \\ \Gamma, a{:}(\mathtt{v}, \downarrow [T]^{\iota}), n{:}(\mathtt{i}, S') \triangleright P_2 \xrightarrow{a?n} \Gamma, a{:}(\mathtt{v}, \emptyset[T]^{\iota}), n{:}(\mathtt{v}, S) \triangleright P_2' \end{array}}{\Gamma, a{:}(\mathtt{v}, \updownarrow [T]^{\iota}), n{:}(\mathtt{h}, S) \triangleright P_1 \parallel P_2 \xrightarrow{\tau} \Gamma, a{:}(\mathtt{v}, \emptyset[T]^{\iota}), n{:}(\mathtt{h}, S) \triangleright (\kappa\, b)\,(P_1' \parallel P_2')}$$

Results

Well-formed Configuration

noDup(hid(P)) and $|\mathsf{vis}(P)| \subseteq \mathsf{vis}(\Gamma)$ and $|\mathsf{hid}(P)| \subseteq \mathsf{hid}(\Gamma)$.

Well-formed Subject-Reduction

If the configuration $\Gamma \triangleright P$ is well-formed and $\Gamma \triangleright P \xrightarrow{\alpha} \Gamma' \triangleright P'$ then $\Gamma' \triangleright P'$ is also well-formed.

Typeability implies Well-Formedness

If $\Gamma \vdash P$ then $\Gamma \triangleright P$ is well-formed.

Strong Subject-Reduction

$\Gamma_1 \vdash P$ and $\Gamma = \Gamma_1 + \Gamma_2$ and $\Gamma \triangleright P \xrightarrow{\alpha} \Gamma' \triangleright P'$ and $\Gamma_1 \triangleright P \xrightarrow{\alpha} \Delta_1 \triangleright P'$ imply $\Delta_1 \vdash P$.

Race Freedom

$\Gamma \vdash P$ and $\Gamma \triangleright P \xrightarrow{t} \Delta \triangleright Q$ implies $\Delta \triangleright Q$ does not violate linearity.