Progress as Compositional Lock-Freedom

Ornela Dardha
(joint work with Marco Carbone and Fabrizio Montesi)

School of Computing Science
University of Glasgow

June 3, 2014

Gentle Intro

- Concurrent/Distributed Systems using Session Types.

Gentle Intro

- Concurrent/Distributed Systems using Session Types.
- Progress is a fundamental property of safe processes.

Gentle Intro

- Concurrent/Distributed Systems using Session Types.
- Progress is a fundamental property of safe processes.
- A program having progress does not get "stuck", i.e., a state that is not designated as a final value and that the language semantics does not tell how to evaluate further.

Gentle Intro: Comparing Properties of Communication

- Deadlock-Freedom: communications eventually succeed, *unless* the whole process diverges. (Standard π)
- Lock-Freedom: communications eventually succeed *even if* the whole process diverges. (Standard π)
- Progress: In-session communications eventually succeed, provided that a suitable context can be found. (Session π)

Deadlock-freedom vs. lock-freedom

- Consider the process:

$$P = (\boldsymbol{\nu} x_1 x_2)(\boldsymbol{\nu} y_1 y_2)\big(\ x_1?(z).y_1!\langle z\rangle \ \mid\ y_2?(w).x_1!\langle w\rangle\ \big)$$

  It is deadlocked and hence locked!

Deadlock-freedom vs. lock-freedom

- Consider the process:

$$P = (\boldsymbol{\nu} x_1 x_2)(\boldsymbol{\nu} y_1 y_2)(\ x_1?(z).y_1!\langle z\rangle \ \mid\ y_2?(w).x_1!\langle w\rangle\ )$$

  It is deadlocked and hence locked!

- Consider the process:

$$Q = (\boldsymbol{\nu} x_1 x_2)(x_1?(z) \mid \Omega)$$

  It is deadlock-free but locked!

What about progress?

- Deadlock- and lock-freedom checked for *closed* systems.

What about progress?

- Deadlock- and lock-freedom checked for *closed* systems.
- (Session-based) systems may be *open-ended*: participants missing; join the system dynamically.

What about progress?

- Deadlock- and lock-freedom checked for *closed* systems.
- (Session-based) systems may be *open-ended*: participants missing; join the system dynamically.
- *Compositional* formulation of progress for open-ended systems.

What about progress?

- Deadlock- and lock-freedom checked for *closed* systems.
- (Session-based) systems may be *open-ended*: participants missing; join the system dynamically.
- *Compositional* formulation of progress for open-ended systems.
- Intuitively: an (open) process has progress if it can reduce within all adequate execution contexts, called *catalysers*, providing the missing participants.

Research Question

Compositionality of progress leads back to lock-freedom; both inspect the behaviour of subprocesses.

What is the relationship between lock-freedom and progress, in particular for open-ended systems?

How to achieve progress?

- Progress through typed closure
- Progress through untyped closure
- Progress through lock-freedom typing discipline

Progress through typed closure

Progress for closed processes

Theorem (Lock-freedom $\Leftrightarrow$ Closed Progress)

Let P be a well-typed, closed *process.*
P is lock-free if and only if P has progress.

Intuition:

- A closed lock-free process reduces on $\rightarrow$ has progress.
- A closed process with progress has all its participants $\rightarrow$ is locked-free.

Progress for open processes

- Progress and lock-freedom do not coincide for *open* processes.
- Define catalysers by using characteristic processes.
- Wrap an open process using catalysers, until all session communications are closed.
- We call this procedure: typed closure.

Catalyesers and Characteristic Processes

$$
\begin{array}{rcl}
\mathcal{C}[\cdot] & = & (\boldsymbol{\nu} xy)([\cdot] \mid P) \\
P & = & y \triangleleft \{l_1.\mathbf{0},\ l_2.y!\langle\mathtt{false}\rangle.\mathbf{0}\}
\end{array}
$$

$\mathcal{C}[\cdot]$ is a catalyser by composing the characteristic process P of session type $T = \oplus\{l_1 : \mathtt{end}, l_2 : !\mathtt{Bool}.\mathtt{end}\}$

Example of typed closure

Consider

$$P = x!\langle\texttt{true}\rangle.x?(z).\mathbf{0}$$

P can be typed under $\Gamma = x : !\texttt{Bool}.?\texttt{Bool}.\texttt{end}$.
Its typed closure is

$$\texttt{tclose}(P) = (\boldsymbol{\nu} xy)(P \mid y?(w).y!\langle\texttt{true}\rangle.\mathbf{0})$$

Progress for open processes

Theorem (Progress $\Leftrightarrow$ Closed Lock-Free)

If P *is* well-typed *then*
P *has progress if and only if* $\texttt{tclose}(P)$ *is lock-free.*

Intuition:

1. $\texttt{tclose}(P)$ is lock-free if and only if $\texttt{tclose}(P)$ has progress.
2. $\texttt{tclose}(P)$ has progress if and only if P has progress.

Progress through untyped closure

Co-process vs. Catalyser

- *Typing* is useful for defining adequate contexts for checking progress, i.e., catalysers.
- Adequate contexts can be defined without a typing discipline.
- Based on the structure of the process, as opposed to the typing environment for catalysers.
- We build a co-process; define untyped closure.

Co-process and untyped closure

$$co[x!\langle v\rangle.P]_f = \begin{cases} co[P]_f & \text{if } x \notin \mathrm{dom}(f) \\ f_x?(y).co[P]_f & \text{otherwise} \end{cases}$$

$$co[(\boldsymbol{\nu} xy)P]_f = co[P]_f \quad x, y \notin \mathtt{dom}(f)$$

$$co[P \mid Q]_f = co[P]_f \mid co[Q]_f$$

The untyped closure of P, $\mathtt{uclose}(P)$, is:

$$(\boldsymbol{\nu} \widetilde{x f_x})(P \mid co[P]_f)$$

where $\mathtt{dom}(f) = \mathtt{fn}(\mathtt{P})$.

Progress through untyped closure: adequacy of `uclose`

Theorem

Let P be well-typed,
$\texttt{uclose}(P)$ *is lock-free if and only if* $\texttt{tclose}(P)$ *is lock-free.*

Corollary

Let P be well-typed.
$\texttt{uclose}(P)$ *is lock-free if and only if P has progress.*

Untyped closure is a conservative extension of typed closure: preserves the connection of progress and lock-freedom.

Progress through types for lock-freedom

Static-analysis for progress

- Checking progress reduces to checking whether the closure (typed or untyped) is lock-free.
- Static analysis for lock-freedom lifted to static analysis for progress.
- E.g., we use Kobayashi's typing discipline for lock-freedom in the standard π- calculus.
- We hence use an encoding of session π- calculus to the standard typed π- calculus.

Typing Progress

Theorem (Typing Progress)

Let P be a well-typed process in the π-calculus with sessions. If $\emptyset \vdash_{\mathtt{LF}} [\![\mathtt{uclose}(P)]\!]_f$, then P has progress.

Progress in Practice: "Bad" Process

Consider

$$(\boldsymbol{\nu} x_1 x_2)(\boldsymbol{\nu} y_1 y_2)(\; x_1?(z).y_1!\langle z\rangle \;\mid\; y_2?(w).x_1!\langle w\rangle \;)$$

Progress in Practice: "Bad" Process

Consider

$$(\nu x_1 x_2)(\nu y_1 y_2)\big(\ x_1?(z).y_1!\langle z\rangle \ \mid\ y_2?(w).x_1!\langle w\rangle\ \big)$$

By encoding we obtain the process:

$$(\nu x)(\nu y)\big(x?(z).y!\langle z\rangle \ \mid\ y?(w).x!\langle w\rangle\ \big)$$

The type system for lock-freedom rejects it!

Progress in Practice: "Good" Process

Consider the process

$$(\boldsymbol{\nu} ab)(\boldsymbol{\nu} cd)\Big(a?(x).c!\langle x\rangle.c?(y).a!\langle y\rangle \mid b!\langle 1\rangle.d?(z).d!\langle 1\rangle.b?(z)\Big)$$

Progress in Practice: "Good" Process

Consider the process

$$(\boldsymbol{\nu} ab)(\boldsymbol{\nu} cd)\Big(a?(x).c!\langle x\rangle.c?(y).a!\langle y\rangle \mid b!\langle 1\rangle.d?(z).d!\langle 1\rangle.b?(z)\Big)$$

By the encoding we obtain the process:

$$(\boldsymbol{\nu} k)(\boldsymbol{\nu} l)\left(\begin{array}{l} k?(x,c_1).\ (\boldsymbol{\nu} c_2)\Big(l!\langle x,c_2\rangle.\ c_2?(y).\ c_1!\langle y\rangle\ \Big) \mid \\ (\boldsymbol{\nu} c_1)\Big(k!\langle 1,c_1\rangle.\ l?(z,c_2).\ c_2!\langle 1\rangle.\ c_1?(z)\Big) \end{array}\right)$$

The type system for lock-freedom accepts it!

Conclusions and Future Work

- Relating progress to lock-freedom in π- calculus with session types.
- Progress as *compositional* form of lock-freedom.
- Progress obtained through:
  - typed closure, using *catalysers*
  - untyped closure, using *co-processes*
  - types and type system for lock-freedom in standard typed π-calculus
- Examples show we have a more accurate analysis for progress.
- Extend progress analysis to multiparty session types; extend encoding first!

Thank You!!

The Model

Terms:

$$
\begin{array}{rcll}
P, Q & ::= & x!\langle v \rangle.P & \textit{(output)} \\
& & x?(y).P & \textit{(input)} \\
& & x \triangleleft \{l_i.P_i\}_{i \in I} & \textit{(selection)} \\
& & x \triangleright \{l_i : P_i\}_{i \in I} & \textit{(branching)} \\
& & P \mid Q & \textit{(parallel)} \\
& & (\boldsymbol{\nu} xy)P & \textit{(restriction)} \\
& & \mathbf{0} & \textit{(inaction)} \\
& & \mathbf{rec} X.P & \textit{(rec)} \\
& & X & \textit{(rec var)}
\end{array}
$$

The Model

Types:

$$
\begin{array}{rcll}
q & ::= & \texttt{lin} & \textit{(linear)} \\
 & & \texttt{un} & \textit{(unrestricted)} \\[1ex]
p & ::= & !T.U & \textit{(send)} \\
 & & ?T.U & \textit{(receive)} \\
 & & \oplus\{l_i : T_i\}_{i\in I} & \textit{(select)} \\
 & & \&\{l_i : T_i\}_{i\in I} & \textit{(branch)} \\[1ex]
T, U & ::= & q\ p & \textit{(qualified pretype)} \\
 & & \texttt{end} & \textit{(termination)} \\
 & & \mu\mathbf{t}.T & \textit{(recursive type)} \\
 & & \mathbf{t} & \textit{(rec var)}
\end{array}
$$

Lock-Freedom

Definition (Lock-Freedom for Sessions [3])

A process P_0 is *lock-free* if for any fair reduction sequence $P_0 \to P_1 \to P_2 \to \ldots$, we have that

1. $P_i \equiv (\boldsymbol{\nu}\widetilde{xy})(x!\langle v\rangle.Q \mid R)$, for $i \geq 0$, implies that there exists $n \geq i$ such that $P_n \equiv (\boldsymbol{\nu}\widetilde{x'y'})(x!\langle v\rangle.Q \mid y?(z).R_1 \mid R_2)$ and $P_{n+1} \equiv (\boldsymbol{\nu}\widetilde{x'y'})(Q \mid R_1[v/z] \mid R_2)$;
2. $P_i \equiv (\boldsymbol{\nu}\widetilde{xy})(x \triangleleft l_j.Q \mid R)$, for some $i \geq 0$, implies that there exists $n \geq i$ such that $P_n \equiv (\boldsymbol{\nu}\widetilde{x'y'})(x \triangleleft l_j.Q \mid y \triangleright \{l_k : R_k\}_{k \in I \cup \{j\}} \mid S)$ and $P_{n+1} \equiv (\boldsymbol{\nu}\widetilde{x'y'})(Q \mid R_j \mid S)$.

Definition (Characteristic Process [1])

Given a type T, its *characteristic process* $(\!| T |\!)^x_g$ is inductively defined on the structure of T as:

$$
\begin{array}{rl}
(\textsc{InVal}) & (\!| q?\mathbf{1}.U |\!)^x_g = x?(y).(\!| U |\!)^x_g \\
(\textsc{OutVal}) & (\!| q!\mathbf{1}.U |\!)^x_g = x!\langle \texttt{Unit} \rangle.(\!| U |\!)^x_g \\
(\textsc{InSess}) & (\!| q'?(qp).U |\!)^x_g = x?(y).((\!| U |\!)^x_g \mid (\!| qp |\!)^y_g) \\
(\textsc{OutSess}) & (\!| q'!(qp).U |\!)^x_g = (\boldsymbol{\nu} zw)(x!\langle z \rangle.((\!| U |\!)^x_g \mid (\!| \overline{qp} |\!)^w_g)) \\
(\textsc{InSum}) & (\!| q\&\{l_i : (q_i p_i)_i\}_{i\in I} |\!)^x_g = x \triangleright \{l_i : (\!| q_i p_i |\!)^x_g\}_{i\in I} \\
(\textsc{OutSum}) & (\!| q \oplus \{l_i : (q_i p_i)_i\}_{i\in I} |\!)^x_g = x \triangleleft \{l_i : (\!| q_i p_i |\!)^x_g\}_{i\in I} \\
(\textsc{End}) & (\!| \mathsf{end} |\!)^x_g = \mathbf{0} \\
(\textsc{RecVar}) & (\!| \mathbf{t} |\!)^x_g = g(\mathbf{t}) \\
(\textsc{Rec}) & (\!| \mu\mathbf{t}.T |\!)^x_g = \mathbf{rec}X.(\!| T |\!)^x_{g,\{\mathbf{t}\mapsto X\}}
\end{array}
$$

Definition (Catalyser)

A catalyser $\mathcal{C}[\cdot]$ is a context such that:

$$\mathcal{C}[\cdot] ::= \; [\cdot] \quad | \quad (\boldsymbol{\nu} xy)\mathcal{C}[\cdot] \quad | \quad \mathcal{C}[\cdot] \mid (\!| qp |\!)^{x}_{g}$$

Definition ($\bowtie$)

The *duality* $\bowtie_{\{x,y\}}$ is defined as follows:

$$x!\langle v\rangle.P \bowtie_{\{x,y\}} y?(z).Q$$
$$x \triangleleft \{l_i.P_i\}_{i\in I} \bowtie_{\{x,y\}} y \triangleright \{l_i : Q_i\}_{i\in I}$$

Definition (Evaluation Context)

$$\mathcal{E}[\cdot] ::= \quad [\cdot] \mid P \quad | \quad (\boldsymbol{\nu} xy)\mathcal{E}[\cdot] \quad | \quad \mathcal{E}[\cdot] \mid \mathcal{E}[\cdot] \quad | \quad \mathbf{rec} X.\mathcal{E}[\cdot]$$

Progress

Definition (Progress)

A process P has *progress* if for all catalysers $\mathcal{C}[\cdot]$ such that $\mathcal{C}[P]$ is well-typed, $\mathcal{C}[P] \to^* \mathcal{E}[R]$ (where R is an input or an output) implies that there exist $\mathcal{C}'[\cdot]$, $\mathcal{E}'[\cdot][\cdot]$ and R' such that $\mathcal{C}'[\mathcal{E}[R]] \to^* \mathcal{E}'[R][R']$ and $R \bowtie_{\{x,y\}} R'$ for some x and y such that $(\boldsymbol{\nu} xy)$ is a restriction in $\mathcal{C}'[\mathcal{E}[R]]$.

Kobayashi's types for lock-freedom

(actions)	$\alpha ::=$	$? \mid !$
(usage types)	$U ::=$	$\mathbf{0} \mid \alpha^{\circ}_{\mathtt{c}}.\mathtt{U} \mid \mathtt{U}_1 \mid \mathtt{U}_2 \mid \mathbf{t} \mid \mu\mathbf{t}.\mathtt{U}$
(channel types)	$T ::=$	$[\widetilde{T}]\ U \mid \langle l\_T \rangle_{i \in I} \mid \mathbf{1}$

Kobayashi's typing rules for lock-freedom

$$\frac{\Gamma, \widetilde{y} : \widetilde{T} \vdash_{\mathrm{LF}} P}{x : [\widetilde{T}]\ ?^{0}_{\mathsf{c}}\ ; \Gamma \vdash_{\mathrm{LF}} x?(\tilde{y}).P} \text{ (LF-IN)}$$

$$\frac{\Gamma, x : [\widetilde{T}]\ U \vdash_{\mathrm{LF}} P \quad \mathit{rel}(U)}{\Gamma \vdash_{\mathrm{LF}} (\boldsymbol{\nu} x)P} \text{ (LF-RES)}$$

Theorem (Lock-Freedom [3])

If $\Gamma \vdash_{\mathrm{LF}} P$ and $\mathit{rel}(\Gamma)$, then P is lock-free.

Encoding Sessions [2]

$$
\begin{array}{rcl}
[\![x!\langle v\rangle.P]\!]_f & = & (\boldsymbol{\nu} c) f_x!\langle v, c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x?(y).P]\!]_f & = & f_x?(y, c).[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x \triangleright \{l_i : P_i\}_{i\in I}]\!]_f & = & f_x?(y).\ \textbf{case}\ y\ \textbf{of}\ \{l_i\_c \triangleright [\![P_i]\!]_{f,\{x\mapsto c\}}\}_{i\in I} \\
[\![(\boldsymbol{\nu} xy)P]\!]_f & = & (\boldsymbol{\nu} c)[\![P]\!]_{f,\{x,y\mapsto c\}}
\end{array}
$$

References

M. Carbone and S. Debois.
A graphical approach to progress for structured communication in web services.
In *Proc. of ICE'10*, pages 13–27, 2010.

O. Dardha, E. Giachino, and D. Sangiorgi.
Session types revisited.
In *PPDP*, pages 139–150, 2012.

N. Kobayashi.
A type system for lock-free processes.
Inf. Comput., 177(2):122–159, 2002.