Secrecy and Group Creation

Luca Cardelli

Microsoft Research

with Giorgio Ghelli (University of Pisa)
and Andrew Gordon (Microsoft Research)

Cork, 2000-07-20

Introduction

- *Group creation* is a new general construct that can be added to virtually any language of formalism.
- In its simplest form, it is a natural extension of the sort-based type systems developed for the π-calculus:

  $(\nu G)\ (\nu n{:}G)\ (\nu m{:}G)\ \ldots$
  create a new group (i.e., unstructured type or collection) *G*,
  and populate it with new elements *n*, *m*, ...

- While studying it for other purposes (control of mobility), we noticed an interesting and subtle connection with secrecy. A secret like *n* can never escape from the initial scope of *G*, as a simple matter of typechecking.

Leaking Secrets

- Consider a configuration, with a *player* P and an *opponent* O:

$$O \mid P$$

- P wants to create a fresh secret x not shared by the opponent. In the π-calculus this is obtained by letting P evolve into a configuration:

$$O \mid (\nu x)P'$$

  which means: create a new x to be used in the scope of P'; with the intent that x should remain forever private to P'.

- Such a privacy policy is violated if the system then evolves into a situation such as this, for a public channel p:

$$p(y)O' \mid (\nu x)(p\langle x\rangle \mid P'')$$

  where $p\langle x\rangle$ may happen accidentally or maliciously.

- For the actual communication, and leakage, to happen, the π-calculus provides *extrusion* rules for (νx), so that the system “automatically” becomes:

$$(\nu x)\ (p(y)O' \mid p\langle x\rangle \mid P'')$$

- After which, the opponents obtains the secret:

$$(\nu x)\ (O'\{y \leftarrow x\} \mid P'')$$

Preventing Leakage

- The secret x has been leaked by a combination of an output $p\langle x\rangle$ (possibly a bug) and extrusion of (νx). Can leakage be prevented?
  - Restricting output: it is not easy to say that $p\langle x\rangle$ should not happen because p could be obtained dynamically. (Flow analysis...)
  - Restricting extrusion: technically difficult, because extrusion is needed for legitimate communication.
- One can look for solutions based on typed π-calculi:
  - Declare x to be *Private*. Unfortunately, in standard sorting systems all sorts are global, so the opponent can be typechecked:

$$p(y{:}Private).O' \quad | \quad (\nu x{:}Private)\,(p\langle x\rangle \mid P'')$$

  - Hint: we want the secret x to belong to a sort G which is itself secret...

Group Creation

- In general, we want the ability to create fresh groups (sorts) on demand, and then to create fresh elements of those groups.
  - Creating arbitrary new groups does not restrict us to a rigid distinction between public and private groups, which works only between two parties.
- We extend the sorted π-calculus with an operator, $(\nu G)P$, to dynamically create new a new group G in a scope P. This is a dynamic operator; it can be used to create a fresh group after an input:

  $$q(z{:}T).(\nu G)P.$$

  Still, the group information can be tracked statically to ensure that names of different groups are not confused.
- N.B.: (νG) extrudes very much like $(\nu x{:}T)$, for the same reasons.

- Leakage to a well-typed opponent is now prevented simply lexical scoping and typing rules (where $G[]$ is the type of channels of group G that carry no values):

  $$p(y{:}T).O' \quad | \quad (\nu G)\ (\nu x{:}G[])\ (p\langle x\rangle \mid P'')$$

  This term cannot be typed: the type T would have to mention G, which is out of scope.

Untyped Opponents

- Problem: opponents cheat. Suppose the opponent is untyped, or not well-typed (e.g.: running on an untrusted machine):

$$(\nu p{:}U) \quad (p(y).O' \quad | \quad (\nu G)\,(\nu x{:}G[\,])\,(p\langle x\rangle \mid P''))$$

untrusted	untrusted	trusted locally typechecked
name server	opponent	player

  Will an untyped opponent, by cheating on the type of the public channel *p*, be able to acquire secret information?

- Fortunately, no. The fact that the player is well-typed is sufficient to ensure secrecy, even in presence of untyped opponents. Essentially because $p\langle x\rangle$ must be locally well-typed.

- We do not even need to trust the type of the public channel *p*, obtained from a potentially untrusted name server.

Secrecy

- Summary: a player creating a fresh group G cannot communicate channels of group G to an opponent outside of the initial scope of G:
  - either because a (well-typed) opponent cannot name G to receive the message,
  - or because a well-typed player cannot use public channels to communicate G information to an (untyped) opponent.
- Programmer's reference manual:

  Channels of group G remain *secret*, forever,
  outside the initial scope of (νG).

- Thus, secrecy is reduced to scoping and typing restrictions. But the situation is still fairly subtle because of
  - the extrusion rules associated with scoping,
  - the fact that scoping restrictions in the ordinary π-calculus do not prevent leakage,
  - and the possibility of untyped opponents.
- The only essential requirement is that the player must be typechecked with respect to a global (untrusted) environment within a trusted context. This seems reasonable. This is all our secrecy theorem needs to assume.

A Typed Pi-Calculus with Groups

Good Environments

$$\frac{}{\varnothing \vdash \Diamond} \qquad \frac{E \vdash T \quad x \notin dom(E)}{E, x{:}T \vdash \Diamond} \qquad \frac{E \vdash \Diamond \quad G \notin dom(E)}{E', G \vdash \Diamond}$$

Good Types

$$\frac{G \in dom(E) \quad E \vdash T_1 \quad \ldots \quad E \vdash T_k}{E \vdash G[T_1, \ldots, T_k]}$$

Good Messages

$$\frac{E', x{:}T, E'' \vdash \Diamond}{E', x{:}T, E'' \vdash x : T}$$

Good Processes

$$\frac{E, G \vdash P}{E \vdash (\nu G)P} \qquad \frac{E, x{:}T \vdash P}{E \vdash (\nu x{:}T)P}$$

$$\frac{E \vdash \diamond}{E \vdash \mathbf{0}} \qquad \frac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q} \qquad \frac{E \vdash P}{E \vdash !P}$$

$$\frac{E \vdash x : G[T_1, \ldots, T_k] \qquad E, y_1{:}T_1, \ldots, y_k{:}T_k \vdash P}{E \vdash x(y_1{:}T_1, \ldots, y_k{:}T_k).P}$$

$$\frac{E \vdash x : G[T_1, \ldots, T_k] \qquad E \vdash y_1{:}T_1 \;\; \ldots \;\; E \vdash y_k{:}T_k}{E \vdash x\langle y_1, \ldots, y_k\rangle}$$

Structural Congruence and Reduction

$P \equiv P$	(Struct Refl)
$P \equiv Q \quad \Rightarrow \quad Q \equiv P$	(Struct Symm)
$P \equiv Q,\, Q \equiv R \quad \Rightarrow \quad P \equiv R$	(Struct Trans)
$P \equiv Q \quad \Rightarrow \quad (\nu G)P \equiv (\nu G)Q$	(Struct GRes)
$P \equiv Q \quad \Rightarrow \quad (\nu x{:}T)P \equiv (\nu x{:}T)Q$	(Struct Res)
$P \equiv Q \quad \Rightarrow \quad P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \quad \Rightarrow \quad !P \equiv !Q$	(Struct Repl)
$P \equiv Q \quad \Rightarrow$ $x(y_1{:}T_1, \ldots, y_k{:}T_k).P \equiv x(y_1{:}T_1, \ldots, y_k{:}T_k).Q$	(Struct Input)
$P \mid \mathbf{0} \equiv P$	(Struct Par Zero)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!P \equiv P \mid !P$	(Struct Repl Par)

$(\nu x{:}T)(\nu x'{:}T')P \equiv (\nu x'{:}T')(\nu x{:}T)P \quad \text{if } x \neq x'$	(Struct Res Res)
$(\nu x{:}T)(P \mid Q) \equiv P \mid (\nu x{:}T)Q \quad \text{if } x \notin fn(P)$	(Struct Res Par)
$(\nu G)(\nu G')P \equiv (\nu G')(\nu G)P$	(Struct GRes GRes)
$(\nu G)(\nu x{:}T)P \equiv (\nu x{:}T)(\nu G)P \quad \text{if } G \notin fn(T)$	(Struct GRes Res)
$(\nu G)(P \mid Q) \equiv P \mid (\nu G)Q \quad \text{if } G \notin fn(P)$	(Struct GRes Par)

$$x\langle z_1, \ldots, z_k\rangle \mid x(y_1{:}T_1, \ldots, y_k{:}T_k).P \longrightarrow P\{y_1 \leftarrow z_1, \ldots, y_k \leftarrow z_k\}$$

$$P \longrightarrow Q \quad \Rightarrow \quad P \mid R \longrightarrow Q \mid R$$

$$P \longrightarrow Q \quad \Rightarrow \quad (\nu G)P \longrightarrow (\nu G)Q$$

$$P \longrightarrow Q \quad \Rightarrow \quad (\nu x{:}T)P \longrightarrow (\nu x{:}T)Q$$

$$P' \equiv P,\, P \longrightarrow Q,\, Q \equiv Q' \;\Rightarrow\; P' \longrightarrow Q'$$

Secrecy by Subject Reduction

- For well-typed opponents, subject reduction already has secrecy implications.

  Theorem (Subject Reduction)

  If $E \vdash P$ and $P \equiv Q$ then $E \vdash Q$.

  If $E \vdash P$ and $P \longrightarrow Q$ then $E \vdash Q$.

  Corollary (No Leakage)

  Let $P = (\nu G)(\nu x{:}G[T_1, \ldots, T_k])P'$. If $E \vdash P$ for some E then there is no Q', Q'', $C\{\text{-}\}$ such that $P \equiv (\nu G)(\nu x{:}G[T_1, \ldots, T_k])Q'$ and $Q' \longrightarrow Q''$ and $Q'' \equiv C\{p\langle x\rangle\}$ where p and x are not bound by $C\{\text{-}\}$.

- But we need to consider also untyped opponents.

A Secrecy Theorem

Theorem (Secrecy):

> Suppose that $E \vdash (\nu G)(\nu x{:}T)P$ where $G \in fg(T)$. Let S be the names occurring in $dom(E)$. Then the type erasure $(\nu x)erase(P)$ of $(\nu G)(\nu x{:}T)P$ preserves the secrecy of the restricted name x from S.

Where “preserves the secrecy of x from the environment S” is defined in terms of a relation that traces the execution of the process, keeping track of the names that the process exchanges with the environment. (Similar to Abadi’s definition.) The opponent process here is idealized as a set of names known to the environment. See paper for details.

Instances and Applications

- The notion of group creation has surfaced in many places. We think we have identified a particularly pure version of this notion. Many connections are still vague:
  – *type generativity* in languages such as SML.
  – *newlock* in Flanagan and Abadi's types for safe locking.
  – *runST* in Lauchbury and Peyton Jones' state threads.
- At least a couple of applications have been worked out in detail:
  – Groups model Tofte and Talpin's *regions*. Andy Gordon and Silvano Dal-Zilio offer an embedding of a typed λ-calculus with regions into a typed π-calculus with groups and effects.
  – Groups are used in a type system for the Ambient Calculus to statically restrict mobility (L.Cardelli, G.Ghelli, A.D.Gordon).

Conclusions

- A new programming construct that allows programmers to easily state some secrecy intentions, as well as other intentions of the kind “this will not escape” (e.g.: pointers).
- Most suitable for groups of “unstructured” entities with little more than equality on them: channels, keys, nonces, pointers.
- While groups are similar to the *sorts* used in typed versions of the π-calculus, a *new sort* operator does not seem to have been considered in the π-calculus literature.
- The basic idea can be adapted to any language: concurrent, functional or imperative.
- Easily statically checkable by routine typechecking (no flow analysis, information flow, etc.).