Name Groups and Group Creation:

Semantics and Applications

A. D. Gordon (Microsoft)

Based on joint work with L. Cardelli (Microsoft), S. Dal Zilio (Microsoft), and G. Ghelli (Pisa)

MFPS 16, Hoboken, April 2000

Just as the **new-name** construct of the π-calculus models dynamic generation of values, a **new-group** construct models dynamic generation of types, as in, for instance, Tofte and Talpin's region-based memory management.

A Fundamental Abstraction: Pure Names

A **pure name** is "*nothing but a bit pattern that is an identifier, and is only useful for comparing for identity with other bit patterns*" (Needham 1989).

A **nominal calculus** is a computational formalism that includes a set of pure names and allows the dynamic generation of fresh, unguessable names.

The π-calculus of Milner, Parrow, and Walker (1989) is the most prominent example: atomic names are its only data.

The idea of my talk is that it's useful to consider pure names and name generativity at the level of types as well as data.

The π-Calculus and its Applications

The π-calculus is a parsimonious formalism for describing the essential semantics of concurrent systems.

A running π-program is an assembly of concurrent processes, communicating on named channels.

Applications include programming semantics, design of programming languages, and analysis of crypto protocols.

Names model a wide variety of value-level identifiers: machine addresses, communication channels, object references, crypto keys.

The π-calculus is extremely expressive, but, until now, the curious phenomenon of **type generativity** has been beyond its scope.

Type Generativity and its Applications

A language has **type generativity** when type equivalence is by name (that is, when types with different names but the same structure are not equivalent), and when type names can be generated dynamically.

Constructs for type generativity introduce a fresh type with a lexical scope, and, usually, enable static guarantees of confinement:

- **datatype** in Standard ML
- **letregion** in Tofte and Talpin's region calculus (ML)
- **runST** in Launchbury and Peyton Jones' state threads (Haskell)
- **newlock** in Flanagan and Abadi's types for safe locking (Java)

Example: Tofte and Talpin's Region Calculus

A compiler intermediate language enabling an implementation of SML with no garbage collector, the ML Kit compiler (Tofte and Talpin 1994).

Heap memory is partitioned into a stack of regions, and each boxed value is annotated with a region ρ.

The construct *letregion* ρ *in* b manages the allocation and de-allocation of regions.

It means: "Allocate a fresh, empty region, denoted by the region variable ρ; evaluate the expression b; de-allocate ρ."

The type of a value stored in ρ takes the form V *at* ρ; such types are generated dynamically by *letregion* ρ.

A Problem and a Solution

The π-calculus is based on **names**: pure names at the level of data.

Name-creation $(\nu x)P$ introduces a fresh name x.

Problem: Standard type systems for nominal calculi (such as π) cannot model type generativity, and its confinement properties, because name-creation does not generate a new type.

Solution: We introduce **groups** : pure names at the level of types. Each name belongs to a group, added to its type. Groups are like Milner's sorts. **Group-creation** $(\nu G)P$ introduces a fresh group G, and hence can model type generativity.

Rest of the Talk

How adding groups and new-group to the π-calculus allows static guarantees of secrecy.

How extending this calculus with an effect system allows us to model type generativity in the region calculus; region de-allocation is an instance of a general garbage collection principle.

How these ideas apply to other nominal calculi.

A Secrecy Theorem via Groups

Controlling a Resource in the Untyped π-Calculus

Client: requests a virtual printer c; then uses it

$$\overbrace{(\nu c)}^{\text{new channel}} \quad (\overbrace{\overline{client}\langle c \rangle}^{\text{tell server}} \mid \overbrace{\overline{c}\langle job \rangle}^{\text{send job}}$$

Server: controls physical printer; creates virtual printers

$$(\nu p)(\overbrace{printer\{p\}}^{\text{driver code}} \mid \overbrace{!client(c).}^{\text{listen for client}} \quad \overbrace{!c(x).\overline{p}\langle x \rangle}^{\text{virtual printer}})$$

Conventional type systems for π cannot check that the direct channel to the printer driver, p, is confined to the driver and virtual printer code.

A Type System with Groups

A **group**, G, is drawn from an infinite set of pure names disjoint from the set of names used as channels. A **type**, T, for a polyadic channel is given by $\mathrm{T} ::= \mathrm{G}[\mathrm{T}_1, \ldots, \mathrm{T}_n]$.

For example, $\mathit{client} : \mathit{Client}[\mathit{Request}[\mathit{Job}]]$ and $\mathrm{p} : \mathit{Printer}[\mathit{Job}]$ for some groups *Client*, *Printer*, and *Request*, and some type *Job*.

Name-creation, $(\nu x{:}\mathrm{T})\mathrm{P}$, generates a fresh, typed channel name with scope P. **Group-creation**, $(\nu \mathrm{G})\mathrm{P}$, generates a fresh group with scope P.

For example, $(\nu \mathit{Printer})(\nu \mathrm{p}{:}\mathit{Printer}[\mathit{Job}])(\mathit{printer}\{\mathrm{p}\} \mid \cdots)$.

Operational Semantics with Groups

Reduction, $P \rightarrow Q$, and the auxiliary congruence, $P \equiv Q$, are defined much as for the untyped π-calculus.

The most interesting equations are:

$(\nu G)(\nu H)P \equiv (\nu H)(\nu G)P$

$(\nu G)(\nu x{:}T)P \equiv (\nu G)(\nu x{:}T)P$ if G not free in T.

$(\nu G)(P \mid Q) \equiv (\nu G)P \mid (\nu G)Q$

$(\nu G)P \equiv P$ if G not free in P

Groups make no difference to operational behaviour:

Theorem (Untyped Erasure) If $P \rightarrow Q$ then $\mathit{erase}(P) \rightarrow \mathit{erase}(Q)$.

If $\mathit{erase}(P) \rightarrow U$ there is Q such that $P \rightarrow Q$ and $\mathit{erase}(Q) \equiv U$.

Typing Judgments with Groups

The type system consists of four judgments:

Typing Judgments:

$\mathcal{E} \vdash \diamond$	good environment
$\mathcal{E} \vdash \mathcal{T}$	good channel type $\mathcal{T}$
$\mathcal{E} \vdash x : \mathcal{T}$	good name x of channel type $\mathcal{T}$
$\mathcal{E} \vdash \mathcal{P}$	good process $\mathcal{P}$

The operational semantics preserves good typing:

Theorem (Subject Reduction)

If $\mathcal{E} \vdash \mathcal{P}$ and either $\mathcal{P} \longrightarrow \mathcal{Q}$ or $\mathcal{P} \equiv \mathcal{Q}$ then $\mathcal{E} \vdash \mathcal{Q}$.

Typing Rules:

$$\frac{}{\varnothing \vdash \diamond} \qquad \frac{E \vdash T \quad x \notin \mathit{dom}(E)}{E, x{:}T \vdash \diamond} \qquad \frac{E \vdash \diamond \quad G \notin \mathit{dom}(E)}{E, G \vdash \diamond}$$

$$\frac{G \in \mathit{dom}(E) \quad E \vdash T_1 \quad \cdots \quad E \vdash T_n}{E \vdash G[T_1, \ldots, T_n]} \qquad \frac{E', x{:}T, E'' \vdash \diamond}{E', x{:}T, E'' \vdash x : T}$$

$$\frac{E, G \vdash P}{E \vdash (\nu G)P} \qquad \frac{E, x{:}T \vdash P}{E \vdash (\nu x{:}T)P} \qquad \frac{E \vdash \diamond}{E \vdash \mathbf{0}} \qquad \frac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q} \qquad \frac{E \vdash P}{E \vdash !P}$$

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

$$\frac{E \vdash x : G[T_1, \ldots, T_n] \qquad E \vdash y_1 : T_1 \quad \cdots \quad E \vdash y_n : T_n}{E \vdash \overline{x}\langle y_1, \ldots, y_n \rangle}$$

Interactions between Untyped Processes

To state our secrecy theorem, we consider processes $(\nu\vec{x})(O \mid P)$, that represent an encounter between an opponent O and a player P, who share the private names $\vec{x}$.

- $P \xrightarrow{\overline{c}} (\nu\vec{z})\langle x_1, \ldots, x_n\rangle P'$ means P is ready to write $\langle x_1, \ldots, x_n\rangle$ on channel c, where $\{\vec{z}\} \subseteq \{x_1, \ldots, x_n\}$ were private to P, and then proceed as P'.
- $P \xrightarrow{c} (y_1, \ldots, y_n)P'$ means P is ready to read a tuple off channel c into the variables $y_1, \ldots, y_n$, and then proceed as P'.
- $P \xrightarrow{\tau} P'$ means P evolves in one step to P'. (Fact: $P \longrightarrow P'$ iff $P \xrightarrow{\tau} \equiv P'$.)
- Specifically, $(\nu\vec{x})(O \mid P) \xrightarrow{\tau} (\nu\vec{x}')(O' \mid P')$ means $(\nu\vec{x})(O \mid P)$ evolves into $(\nu\vec{x}')(O' \mid P')$ either by O and P interacting, or by O or P evolving on its own.

Deriving τ-Labelled Transitions from $(\nu\vec{x})(O \mid P)$ Processes:

$$\frac{O \xrightarrow{\tau} O'}{O \mid P \xrightarrow{\tau} O' \mid P}$$

$$\frac{P \xrightarrow{\tau} P'}{O \mid P \xrightarrow{\tau} O \mid P'}$$

$$\frac{O \xrightarrow{\overline{c}} (\nu\vec{z})\langle\vec{x}\rangle O' \qquad P \xrightarrow{c} (\vec{y})P' \qquad \{\vec{z}\} \cap \mathit{fn}(P) = \varnothing}{O \mid P \xrightarrow{\tau} (\nu\vec{z})(O' \mid P'\{\vec{y}\leftarrow\vec{x}\})}$$

$$\frac{O \xrightarrow{c} (\vec{y})O' \qquad P \xrightarrow{\overline{c}} (\nu\vec{z})\langle\vec{x}\rangle P' \qquad \{\vec{z}\} \cap \mathit{fn}(O) = \varnothing}{O \mid P \xrightarrow{\tau} (\nu\vec{z})(O'\{\vec{y}\leftarrow\vec{x}\} \mid P')}$$

$$\frac{O \mid P \xrightarrow{\tau} (\nu\vec{y})(O' \mid P') \qquad \{\vec{x}\} \cap \{\vec{y}\} = \varnothing}{(\nu\vec{x})(O \mid P) \xrightarrow{\tau} (\nu\vec{x},\vec{y})(O' \mid P')}$$

A Secrecy Theorem for Typed Processes

Suppose $x_1{:}T_1, \ldots, x_n{:}T_n \vdash (\nu G)(\nu y{:}T)P$ where G free in T.

Consider any untyped opponent O who knows $\{x_1, \ldots, x_n\}$ but not y, that is, each $x_i \in \mathit{fn}(O)$ but $y \notin \mathit{fn}(O)$.

Then no series of interactions between the opponent O and the untyped erasure $\mathit{erase}(P)$ of P can reveal y to the opponent.

Formally, there is no transition sequence

$$O \mid \mathit{erase}(P) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (\nu \vec{x})(O' \mid P')$$

such that $y \in \mathit{fn}(O') - \{\vec{x}\}$.

Instance: Confining a Critical Channel

Client: requests a virtual printer c; then uses it

$$(\nu c{:}\mathit{Request}[\mathit{Job}])(\overline{\mathit{client}}\langle c\rangle \mid \overline{c}\langle \mathit{job}\rangle)$$

Server: controls physical printer; creates virtual printers

$$(\nu \mathit{Printer})(\nu p{:}\mathit{Printer}[\mathit{Job}])$$
$$(\mathit{printer}\{p\} \mid$$
$$!\mathit{client}(c{:}\mathit{Request}[\mathit{Job}]).\ !c(x{:}\mathit{Job}).\overline{p}\langle x\rangle)$$

By the Secrecy Theorem, if the driver code $\mathit{printer}\{p\}$ is well-typed, it cannot accidentally or maliciously leak p.

Discussion

We defined a type system with groups for π.

A Secrecy Theorem asserts that well-typed processes $(\nu G)(\nu x{:}G[T_1, \ldots, T_n])P$ cannot leak x to their environment, represented by an opponent O.

As in work on crypto protocols in the spi calculus, the opponent is simply an arbitrary untyped process—on the net, we cannot expect our opponent to be well-typed!

Our definition of leaking a name is inspired by a definition of Abadi for spi, but uses a standard operational semantics, τ-transitions.

A Garbage Collection Theorem via Groups and Effects

Region-Based Memory Management by Example

An ML term compiles via an intermediate λ-term such as:

$$
\begin{array}{l}
\textit{letregion}\ \rho'\ \textit{in} \\
\quad \textit{let}\ f : (\textit{Lit} \xrightarrow{\{\}} \textit{Lit})\ \textit{at}\ \rho' = (\lambda(x)x\ \textit{at}\ \rho')\ \textit{in} \\
\quad \textit{let}\ g : (\textit{Lit} \xrightarrow{\{\rho'\}} \textit{Lit})\ \textit{at}\ \rho = (\lambda(y)f(y)\ \textit{at}\ \rho)\ \textit{in}\ g(5) \\
:^{\{\rho\}}\ \textit{Lit}
\end{array}
$$

Evaluation creates a new region ρ', stores $f \mapsto \lambda(x)x$ in ρ', stores $g \mapsto \lambda(y)f(y)$ in ρ, computes $g(5)$, then de-allocates ρ'.

The type and effect judgment $E \vdash a :^{\{\rho_1,\ldots,\rho_n\}} A$ means all the reads and writes of a to global regions are within regions $\rho_1, \ldots, \rho_n$.

A Semantics for Regions in the π-Calculus

Ignoring typing information (like *letregion*), untyped π can represent the meaning of region terms using a standard CPS encoding:

$$(\nu f)(\nu g)(\overbrace{!f(x,k').\overline{k'}\langle x\rangle}^{f\mapsto\lambda(x)x\,at\,\rho'} \mid \overbrace{!g(y,k'').\overline{f}\langle y,k''\rangle}^{g\mapsto\lambda(y)f(y)\,at\,\rho} \mid \overbrace{\overline{g}\langle 5,k\rangle}^{g(5)})$$
$$\approx\ (\nu f)(\nu g)(!f(x,k').\overline{k'}\langle x\rangle \mid !g(y,k'').\overline{f}\langle y,k''\rangle \mid \overline{k}\langle 5\rangle)$$

Our goal is to generalize this to a typed encoding, and hence to give an abstract proof of the safety of region de-allocation.

In particular, by de-allocating ρ', the process above is equivalent to:

$$(\nu f)(\nu g)(!g(y,k'').\overline{f}\langle y,k''\rangle \mid \overline{k}\langle 5\rangle)$$

Adding Effects to the π-Calculus

An **effect**, **G** or **H**, is a finite set of groups.

The central judgment $E \vdash P : \{G_1, \ldots, G_n\}$ means the process P uses names according to their types and that all its external reads and writes are on channels in groups $G_1, \ldots, G_n$.

Type and Effect Rules for the π-Calculus:

$$\frac{E \vdash x : G[T] \quad E, y{:}T \vdash P : \mathbf{G}}{E \vdash x(y).P : \{G\} \cup \mathbf{G}} \qquad \frac{E \vdash x : G[T] \quad E \vdash y : T}{E \vdash \overline{x}\langle y \rangle : \{G\}} \qquad \frac{E \vdash \diamond}{E \vdash \mathbf{0} : \varnothing}$$

$$\frac{E, G \vdash P : \mathbf{H}}{E \vdash (\nu G)P : \mathbf{H} - \{G\}} \qquad \frac{E, x{:}T \vdash P : \mathbf{H}}{E \vdash (\nu x{:}T)P : \mathbf{H}} \qquad \frac{E \vdash P : \mathbf{G} \quad E \vdash Q : \mathbf{H}}{E \vdash P \mid Q : \mathbf{G} \cup \mathbf{H}}$$

A Typed Semantics of the Region Calculus

We obtain a typed semantics for the region calculus by interpreting regions as groups and extending the untyped semantics with the equation $[\![\textit{letregion}\ \rho\ \textit{in}\ b]\!]k = (\nu\rho)[\![b]\!]k$.

A subtlety: To encode the effect system, we need to add channel types $G[T_1, \ldots, T_n]\backslash\mathbf{H}$, where **H** is the **hidden effect** of the channel, an effect transferred from inputs to outputs:

$$\frac{E \vdash x : G[T]\backslash\mathbf{H} \quad E, y{:}T \vdash P : \mathbf{G} \quad \mathbf{G} \subseteq \mathbf{H}}{E \vdash x(y).P : \{G\}} \qquad \frac{E \vdash x : G[T]\backslash\mathbf{H} \quad E \vdash y : T}{E \vdash \overline{x}\langle y\rangle : \{G\} \cup \mathbf{H}}$$

Intuition: if there is no matching output, its safe to hide the effect of P.

A Garbage Collection Theorem for Typed Processes

Consider processes P and R such that R has effect $\{G\}$ but G is not in the effect of P.

R can only communicate on names in group G, while P cannot communicate on names in group G, so P and R cannot interact.

If P and R are the only sources of inputs or outputs in the scope of G, then R has no interactions whatsoever.

Hence R makes no difference to the behaviour of the whole process.

Formally, if $E, G, E' \vdash P : \mathbf{H}$ and $E, G, E' \vdash R : \{G\}$ with $G \notin \mathbf{H}$, then $(\nu G)(\nu E')(P \mid R) \approx (\nu G)(\nu E')P$.

Instance: Discarding Defunct Regions

With $E' = f : \rho'[L, K[L]]\backslash\{K\}, g : \rho[L, K[L]]\backslash\{K, \rho'\}$, we have:

$$k{:}K[L], \rho', E' \vdash !f(x, k').\overline{k'}\langle x\rangle : \{\rho'\}$$

$$k{:}K[L], \rho', E' \vdash !g(y, k'').\overline{f}\langle y, k''\rangle \mid \overline{k}\langle 5\rangle : \{\rho, K\}$$

and therefore:

$$(\nu\rho')(\nu E')(!f(x, k').\overline{k'}\langle x\rangle \mid !g(y, k'').\overline{f}\langle y, k''\rangle \mid \overline{k}\langle 5\rangle)$$

$$\approx\ (\nu\rho')(\nu E')(!g(y, k'').\overline{f}\langle y, k''\rangle \mid \overline{k}\langle 5\rangle)$$

This generalizes to a theorem asserting defunct regions make no difference to the behaviour of the encoding of any term. Its not an instance of standard garbage collection principles for π.

Discussion

We defined a type and effect system for the π-calculus.

We gave a typed translation of the region calculus into our π-calculus, that preserves the static and dynamic semantics.

Tofte and Talpin proved the safety of region de-allocation via a rather intricate co-inductive proof. Our construction shows safety to be an instance of a general confinement theorem.

Banerjee, Heintze, and Riecke (LICS'99) give an apparently unrelated abstract proof of safety based on denotational semantics.

We model region polymorphism by extending our π-calculus with group polymorphism.

Adding Groups to other Nominal Calculi

The **ambient calculus** models the mobility of both hardware devices and software agents using named ambients, containers holding running agents and nested subambients. A group-based type system controls the groups of ambients that may cross a particular ambient, allowing static checking of access rights.

See Cardelli, Ghelli, Gordon, *Ambient Groups and Mobility Types*, TCS2000, Sendai, Japan, August 2000.

The **spi calculus** extends π with crypto primitives to model crypto protocols. We are exploring applications of new-group to spi. It can help guarantee key secrecy, for example.

Conclusions

We obtained a new model of type generativity by adding groups (aka sorts), effects, and a new-group operator to the π-calculus.

A Secrecy Theorem shows that new-group prevents accidental or malicious leaks of private names.

A Garbage Collection Theorem supports an abstract proof of the safety of region-based memory management.

Our model unifies several existing applications of type generativity. We hope it will inspire the discovery of more.

Example: Protecting Encrypted Traffic

Suppose $atm, bank : Net[Un]$ are untrusted network links.

$$\overbrace{network\{atm, bank\}}^{\text{untrusted network}} \mid$$

$$(\nu\, PIN)(\nu p{:}PIN[])(\nu k{:}Key[PIN[], Request])$$

$$(\underbrace{\overline{atm}\langle\{p, r\}_k\rangle}_{\text{ATM request}} \mid \underbrace{bank(x{:}Un).case\ x\ of\ \{p, r\}_k\ in\ process\{p, r\}}_{\text{bank's server}})$$

Typing guarantees that neither the ATM nor the bank can accidentally or maliciously leak either the key or the PIN.