Developing a type inference algorithm for The Ambient Calculus with Kill

Casper Jensen

June 5, 2013

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9220 Aalborg East
Telefon 99 40 99 40
http://cs.aau.dk

Title:

Developing a type inference algorithm for The Ambient Calculus with Kill

Projectperiod:
DAT10, Spring semester 2013

Projectgroup:
d106f13

Group members:
Casper Jensen
Cjens08@student.aau.dk

Supervisor:
Hans Hüttel

Circulation: 3

Pages: 43

Finished on June 5, 2013

Synopsis:

In this report we describe a slightly modified type system for The Ambient Calculus with Kill. We then describe a type inference algorithm for this type system. The algorithm works in three steps: "Constraint Generation", which takes a process P and generates the constraints needed to make it typeable. "Constraint Solving", which take the generated constraint and solve them, resulting in a pre-processed E and Δ and a set of group dependencies. And finally "Post-Processing" which solves the group dependencies and assures that any constraints on the overall typing is ensured, yielding our final E and Δ. Finally we prove that the E and Δ returned by our algorithm makes P well-typed and that Δ is minimal.

Contents

Chapter 1

Introduction

1.1 Type inference and Ambient Calculus with Kill

The Ambient Calculus on its own is able to precisely and fairly readable describe most distributed systems, and with the extension of kill n, from "Describing volunteer computing with departing and centralized nodes"[5], it is also able to describe volunteer computing systems and other peer-to-peer systems where a node may leave the system at a given time. In the original description of the Ambient Calculus with Kill[5], a type system, derived from the notion of groups, is also proposed. This system gives a security policy for a process P, describing where an ambient belonging to a given group may reside, and where and if it may be killed.

Where a type checker takes a given process P and a given typing E and concludes if P is well-typed under the typing assumptions of E, another approach is type inference where we assume that $E \vdash P$ is well-typed for some given process P and deduce what the typing of E must be to for fill this assumption. The typing of E would then be built from the context of P, where if we e.g. had kill $n.P$ we would assume that the name n is killable, and that the type of n would reflect or at least not contradict this assumption.

In this report we will take a slightly modified version of the proposed type system[5], and with it, build a type inference algorithm for the Ambient Calculus with Kill. The algorithm will work in three steps.

1. Given a process P generate a constraint $\mathbb{C}_E$ on the group environment and a constraint $\mathbb{C}_\Delta$ on the security policy.

2. Then solve the constraints $\mathbb{C}_E$ and $\mathbb{C}_\Delta$ yielding a pre-processed group environment E and security policy Δ

3. Some finishing post-processing assures that any constraints on the overall typing is ensured, yielding our final E and Δ.

1.2 Related Work

Type inference algorithms have been proposed for versions of the calculus of Mobile Ambients before. In "Type Inference for Mobile Ambients in Prolog [4]" the authors describe an implemented algorithm, for a version of Mobile Ambients without kill and open, but with communication instead, where $\langle M \rangle.P$ is an output process and $(x : W).P$ an input process.
The type system is, like the one described in this report, derived from the notion of groups, but where our type system purely describes where an ambient belonging to a certain group may reside or be killed, the system presented in [4] also concentrates on communication and what messages may be passed between groups. The group type is as follows: $g : gr(\mathcal{S}, \mathcal{C}, \mathcal{E}, T)$, where $\mathcal{S}$ are the groups that g may stay in, $\mathcal{C}$ are the groups that g may cross from an *in* or *out*, $\mathcal{E}$ are the groups that g may enter from an *to* and T is the fixed communications types within g.
The type inference algorithm is built on the principle of principal typing, where the algorithm iteratively goes through a process and for each subpart, generates the group environment Γ and variable environment Δ. The author also implements this type inference algorithm in Prolog, where the implementation elegantly follows the algorithm through Prolog clauses.

A common approach to type inference is that of generating and solving constraints. In "Type inference for a distributed π-calculus[6]" the authors describe a type inference algorithm for Dπ working in two steps

- given a term S and its initial typing context Γ, generate type constraints involving type variables.
- produce a substitution μ of type variables for types solving the constraints.

If the algorithm terminates, then the substitution applied to the initial context $\mu\Gamma$ is a valid typing context for S. Otherwise, S is not typeable. The author also describes how to handle subtyping and dependent types through the notion of binding relations, saying that if a binding relation (a, α) exists, then it is forbidden to substitute a type variable α by a type in which name a occurs.
Both the constraint generation and constraint solving are done through a rewriting system, where a constraint is defined as $(\mathcal{E}, \mathcal{I})_{\mathcal{B}}$, where $\mathcal{E}$ is the typing constraint, defined as a set of equations between type schemes, $\mathcal{I}$ is the subtyping constraint, defined as a set of inequations between location types, and $\mathcal{B}$ is a binding relation.
The constraint generation is defined as a reduction relation $\rightsquigarrow$ on tuples $(\mathcal{J}, \mathcal{E}, \mathcal{I})^{\mathcal{X}}_{\mathcal{B}}$ where $\mathcal{J}$ is a set of sequences involving context schemes, $\mathcal{X}$ is a set of type variables, and $(\mathcal{E}, \mathcal{I})_{\mathcal{B}}$ is the constraint being generated. The generation terminates when $\mathcal{J}$ is empty, so reductions for the solving phase are of the form:

$$(\{\Gamma \vdash S\}, \emptyset, \emptyset)^{var(\Gamma)}_{\emptyset} \rightsquigarrow^* (\emptyset, \mathcal{E}, \mathcal{I})^{\mathcal{X}}_{\mathcal{B}}$$

Where Γ is an initial context for the sequence S
The constraint solving is defined as a reduction relation $\rightsquigarrow$ on tuples $(\mathcal{E}, \mathcal{I}, \mu)^{\mathcal{X}}_{\mathcal{B}}$ where μ is a principal solution of the constraint $(\mathcal{E}, \mathcal{I})_{\mathcal{B}}$. The solving terminates when it either reaches a terminal configuration $(\emptyset, \mathcal{A}, \mu)^{\mathcal{Y}}_{\mathcal{B}}$ that can not reduce more, or the failure state $\perp$, so the solving goes like:

$$(\mathcal{E}, \mathcal{I}, \emptyset)^{var(\mathcal{E},\mathcal{I})}_{\mathcal{B}} \rightsquigarrow^* (\emptyset, \mathcal{A}, \mu)^{\mathcal{Y}}_{\mathcal{B}} \not\rightsquigarrow$$

or

$$(\mathcal{E}, \mathcal{I}, \emptyset)^{var(\mathcal{E},\mathcal{I})}_{\mathcal{B}} \rightsquigarrow^* \perp$$

1.3 Overview

The report is organized as follows. In chapter 2 we will revisit the Ambient Calculus with Kill, an extension to the original Calculus of Mobile Ambients, we will present its syntax and a possible type system for it.
Then in chapter 3 we will describe how constraints on the group environment E and the security policy Δ can be generated from each of the formation rules and an initial group $\mathcal{G}$.
Following in chapter 4, we will derive an algorithm for solving the different

constraints generated in the chapter before, yielding a pre-processed group environment E and security policy Δ.
In chapter 5 we take the pre-processed E and Δ and do any finalizing post-processing that could not be done before the full environments was generated, this includes solving group dependencies, assigning missing multiplicities and making sure our typing is minimal.
In chapter 6 we prove the correctness of our algorithm, saying that P is typeable in the resulting environment E and security policy Δ and that the Δ generated is minimal.
Finally in chapter 7 we will draw our conclusions and reflect on future works.

Chapter 2

The Ambient Calculus with Kill

The Ambient Calculus[3] is a process calculus, designed to describe concurrent system with mobility. It does this through the use of ambients, where an ambient is defined as "a bounded placed where computation happens". It also gives the possibility of moving ambients in and out of each other, making it able to describe mobile systems. The Ambient Calculus with Kill[5] extends this with a kill-capability, able to remove a named ambient with all of its containing processes.

2.1 The Syntax

The full syntax of the Ambient Calculus with Kill looks as follows.

n		Ambient name
$\mathcal{P}, \mathcal{Q}$ ::=		Process
	$(\nu n)\mathcal{P}$	restriction
	0	inactivity
	$\mathcal{P} \mid \mathcal{Q}$	composition
	$!\mathcal{P}$	replication
	$n[\mathcal{P}]$	ambient
	$\mathcal{M}.\mathcal{P}$	action
$\mathcal{M}$ ::=		Capability
	$\mathsf{in}\ n$	can enter n
	$\mathsf{out}\ n$	can exit n
	$\mathsf{open}\ n$	can open n
	$\mathsf{kill}\ n$	can kill n

Where n ranges over a countable infinite set of names $\mathcal{N}$. An ambient n is a bounded space in which a process P runs, by definition we say that our total process is contained in an invisible top-level ambient. *Restriction* creates a new unique name n for use in P, the new name is a reference to an ambient and may be used to operate on that ambient by name. *Inactivity* is the process that does nothing. *Composition* lets two processes P and Q run in parallel. *Replication* lets us create an unbounded supply of copies of the process P.
Finally *action* lets us use one of the three capabilities. $\mathsf{in}\ n$ which lets an ambient, with the in-capability in it, move to the named ambient n if it is parallel with the enclosing ambient. $\mathsf{out}\ n$ lets an enclosed ambient, with the out-capability in it, move out of the enclosing named ambient n. The capability, $\mathsf{open}\ n$, opens up an ambient, removing the ambient n and exposing the content of it. The $\mathsf{kill}\ n$ capability looks much like $\mathsf{open}\ n$, but instead of exposing the contained processes, the $\mathsf{kill}\ n$ also removes those.

The reduction rules then look as shown in figure 2.1

(Par)
$$\frac{P \to P'}{P \mid Q \to P' \mid Q}$$

(New)
$$\frac{P \to P'}{(\nu n)P \to (\nu n)P'}$$

(Out)
$$n[Q \mid m[\mathsf{out}\ n.P]] \to n[Q] \mid m[P]$$

(In)
$$m[\mathsf{in}\ n.P] \mid n[Q] \to n[Q \mid m[P]]$$

(Open)
$$\mathsf{open}\ n.P \mid n[Q] \to Q \mid P$$

(Kill)
$$\mathsf{kill}\ n.P \mid n[Q] \to P$$

(Rep)
$$!P \to !P \mid P$$

(Str)
$$\frac{P \equiv Q \quad Q \to Q' \quad Q' \equiv P'}{P \to P'}$$

Figure 2.1: Reduction rules

2.2 The Type System

The type system we use for the Ambient Calculus with Kill is a linear type system, where an ambient n belongs to a group $\mathcal{G}$. Where the type of the group describes the possible behaviour of ambients belonging to the group. The type system also introduces multiplicity, describes if the ambients belonging to the group is linear 1 or unlimited ω, that is if there may only occur exactly one ambient belonging to the group, or if there may be zero or more. The type system therefore acts as a security policy, describing where an ambient may reside, and what ambients may be killed or opened, and where.

The type system makes use of two environments. The first is a group environment E binding names to groups, e.g. $n_1 : \mathcal{G}_1, \ldots, n_m : \mathcal{G}_m$, where each name occurs only once. The second is the security policy Δ, binding each group $\mathcal{G}$ to its particular type $(\mathcal{C}, \mathcal{K}, m)$, where $\mathcal{C}$ is the set of groups which $\mathcal{G}$ can be contained in, $\mathcal{K}$ is the set of groups which may kill or open an ambient belonging to $\mathcal{G}$ and m is the multiplicity of $\mathcal{G}$. Giving us the type system shown in figure 2.2.

$E(a)$ ::=		Group
	$\mathcal{G}_a$	
$\Delta(\mathcal{G}_a)$ ::=		Group Type
	$(\mathcal{C}, \mathcal{K}, m)$	
$\mathcal{C}$ ::=		The groups which $\mathcal{G}_a$ may be contained in
	$\mathcal{G}_1, \ldots, \mathcal{G}_n \mid \emptyset$	
$\mathcal{K}$ ::=		The groups that may kill $\mathcal{G}_a$, $\mathcal{K} \subseteq \mathcal{C}$
	$\mathcal{G}_1, \ldots, \mathcal{G}_n \mid \emptyset$	
m ::=		Group Multiplicity
	0	Zero, never occurs in the given environment
	1	Linear, occurs exactly once in the given environment
	ω	Unlimited, occurs any number of times in the given environment

Where if $\Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$ and $\exists \mathcal{G}_a \in \mathcal{C}, \Delta(\mathcal{G}_a) = (\mathcal{C}_a, \mathcal{K}_a, \omega)$ then $m = \omega$

Figure 2.2: The Type System

Where the condition at the end assures that no groups with linear multiplicity are inside an ambient which may risk being killed.

The type of the top-level ambients group $\mathcal{G}_{top}$ is $(\mathcal{C}, \mathcal{K}, \omega)$ where $\mathcal{C}$ and $\mathcal{K}$ is empty.

2.2.1 Group Operations

The sum of two groups $\mathcal{G}_1$ and $\mathcal{G}_2$, written $\mathcal{G}_1 \oplus \mathcal{G}_2$, where $\Delta(\mathcal{G}_1) = (\mathcal{C}_1, \mathcal{K}_1, m_1)$ and $\Delta(\mathcal{G}_2) = (\mathcal{C}_2, \mathcal{K}_2, m_2)$, is defined as follows:

$$(\mathcal{C}_1, \mathcal{K}_1, m) \oplus (\mathcal{C}_2, \mathcal{K}_2, m) = (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{K}_1 \cup \mathcal{K}_2, m) \quad m = \{0, \omega\}$$

$$(\mathcal{C}_1, \mathcal{K}_1, m_1) \oplus (\mathcal{C}_2, \mathcal{K}_2, m_2) = (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{K}_1 \cup \mathcal{K}_2, 1) \quad m_1, m_2 = \{0, 1\}, m_1 \neq m_2$$

In all the cases but the listed, $\mathcal{G}_1 \oplus \mathcal{G}_2$ is undefined.

And the sum of two Security policies Δ_1 and Δ_2, written $\Delta_1 \oplus \Delta_2$, being

defined as the security policy Δ given by $dom(\Delta) = dom(\Delta_1) \cup dom(\Delta_2)$. where for all $\mathcal{G} \in dom(\Delta)$ we get

$$\Delta(\mathcal{G}) = \begin{cases} \Delta_1(\mathcal{G}) \oplus \Delta_2(\mathcal{G}) & \text{if } \mathcal{G} \in dom(\Delta_1) \cap dom(\Delta_2) \\ \Delta_1(\mathcal{G}) & \text{if } \mathcal{G} \in dom(\Delta_1) \text{ and } \mathcal{G} \notin dom(\Delta_2) \\ \Delta_2(\mathcal{G}) & \text{if } \mathcal{G} \in dom(\Delta_2) \text{ and } \mathcal{G} \notin dom(\Delta_1) \end{cases}$$

2.3 The Typing Relation

The set of well-typed processes is defined using a typing relation, where the names are defined in a group environment E, which is a set of typing assumptions, $n : \mathcal{G}_n$, meaning n belongs to the group $\mathcal{G}_n$, and the type of our groups are defined in our security policy Δ.
The typing relation $E \vdash^{\Delta}_{\mathcal{G}} P$ indicates that if a Process P is in a group $\mathcal{G}$ and the free names have the types as defined in E and Δ, then P is well-typed. An instance of the typing relation is called a type judgement, where the validity of a typing judgement is shown by providing a typing derivation, constructed using a set of typing rules.
In Cardelli, Ghelli and Gordons original type system[2], without groups, we have the typing derivation:

$$\frac{E \vdash P : T \qquad \dfrac{E \vdash Q : T}{E \vdash n[Q] : T}}{E \vdash \mathsf{open}\ n.P \mid n[Q] : T}$$

where the reduction to $E \vdash P \mid Q : T$ is straight forward, as both P and Q is typed in the same environment. If we look at the same reduction in our type system, the typing derivation looks like:

$$\frac{E \vdash^{\Delta}_{\mathcal{G}} P \qquad \dfrac{E \vdash^{\Delta}_{\mathcal{G}_n} Q}{E \vdash^{\Delta}_{\mathcal{G}} n[Q]}}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{open}\ n.P \mid n[Q]}$$

which we would want to reduce to $E \vdash^{\Delta}_{\mathcal{G}} P \mid Q$, we have that P and Q is still typed in the same environment, but P is typed in the group $\mathcal{G}$ and Q in $\mathcal{G}_n$. We therefore define that if a process P is well-typed in some group $\mathcal{G}$, then it is also well-typed in any group the content of P may be in, in the current scope. That is $E \vdash^{\Delta}_{\mathcal{G}'} P$, where for any ambient $n[Q]$ in P, $E(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$, $\mathcal{G} \in \mathcal{C}$ and $\mathcal{G}' \in \mathcal{C}$

Lemma 1 (Consistency)**.** *If* $E \vdash^{\Delta}_{\mathcal{G}} P$ *and* $\forall \mathcal{G}_n \in Scope(E, P)$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ *where* $\mathcal{G} \in \mathcal{C}$ *and* $\mathcal{G}' \in \mathcal{C}$ *then* $E \vdash^{\Delta}_{\mathcal{G}'} P$

Where $Scope(E, P)$ is defined as:

$$\begin{aligned} Scope(E, 0) &= \emptyset \\ Scope(E, n[Q]) &= E(n) \\ Scope(E, !Q) &= Scope(E, Q) \\ Scope(E, M.Q) &= Scope(E, Q) \\ Scope(E, Q \mid R) &= Scope(E, Q) \cup Scope(E, R) \\ Scope(E, (\nu n : \mathcal{G}_n)Q) &= Scope(E, n : \mathcal{G}_n, Q) \end{aligned}$$

Proof. Which we prove by structural induction in P, most of the cases is alike, so we will only show a few.

Case $n[P]$
By induction we have that $E \vdash^{\Delta}_{\mathcal{G}'} P$, it must be the case that $E(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ where $\mathcal{G} \in \mathcal{C}$ and $\mathcal{G}' \in \mathcal{C}$ and $E \vdash^{\Delta}_{\mathcal{G}} P$

Case $!P$
By induction we have that $E \vdash^{\Delta}_{\mathcal{G}'} P$, it must be the case that $\forall \mathcal{G}_n \in Scope(E, P)$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ where $\mathcal{G} \in \mathcal{C}$ and $\mathcal{G}' \in \mathcal{C}$ and $E \vdash^{\Delta}_{\mathcal{G}} P$

Case $P \mid R$
By induction we have that $E \vdash^{\Delta}_{\mathcal{G}'} P \mid R$, it must be the case that $\forall \mathcal{G}_n \in Scope(E, P)$, $\Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n)$ where $\mathcal{G} \in \mathcal{C}_n$ and $\mathcal{G}' \in \mathcal{C}_n$, and $\forall \mathcal{G}_m \in Scope(E, R)$, $\Delta(\mathcal{G}_m) = (\mathcal{C}_m, \mathcal{K}_m, m_m)$ where $\mathcal{G} \in \mathcal{C}_m$ and $\mathcal{G}' \in \mathcal{C}_m$, finally we have $E \vdash^{\Delta}_{\mathcal{G}} P \mid R$ □

In this report we will only show the typing rules which differs from the original report[5], the full set can be seen in appendix A

Ambient

An ambient in our environment belongs to group $\mathcal{G}_n$, where the multiplicity of $\mathcal{G}_n$ must be 1 or ω and $\mathcal{G}_n$ must be in $\mathcal{C}$ of the surrounding group $\mathcal{G}$, we also have that all groups that may kill the surrounding group $\mathcal{G}$ must be in

$\mathcal{C}_n$

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}_n} P}{E \vdash^{\Delta}_{\mathcal{G}} n[P]} \qquad \text{(E-Amb)}$$

$$\text{where } \Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m) \text{ and } \Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n),$$
$$\mathcal{G} \in \mathcal{C}_n,\ m \neq 0 \text{ and } \mathcal{K} \subseteq \mathcal{C}_n$$

In Capability

The in capability is checking that our group $\mathcal{G}_a$ may move in to the group of n $\mathcal{G}_n$ and that all group that may kill $\mathcal{G}_n$ must be in $\mathcal{C}$

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{in}\ n.P} \qquad \text{(E-In)}$$

$$\text{where } \Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m) \text{ and } \Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n), \mathcal{G}_n \in \mathcal{C} \text{ and } \mathcal{K}_n \subseteq \mathcal{C}$$

Out Capability

Out is similar to in but instead of checking if $\mathcal{G}_a$ may be contained in $\mathcal{G}_n$, we need to check if $\mathcal{G}_a$ may be in what ever group $\mathcal{G}_n$ is in. We also checks that all groups that may kill any group we end up in, is in $\mathcal{C}$

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{out}\ n.P} \qquad \text{(E-Out)}$$

$$\text{where } \Delta(\mathcal{G}) = (\mathcal{C},\ \mathcal{K}, m), \Delta(\mathcal{G}_n) = (\mathcal{C}_n,\ \mathcal{K}_n, m_n) \text{ and } \forall \mathcal{G}_a \in \mathcal{C}_n,$$
$$\Delta(\mathcal{G}_a) = (\mathcal{C}_a, \mathcal{K}_a, m_a),\ \mathcal{G} \in \mathcal{C}_a \text{ and } \mathcal{K}_a \subseteq \mathcal{C}$$

2.4 Minimal and Most Permissive Typing

An interesting result of the type system, is that for any process P, we can produce a most permissive typing, where all names in P can be bound to a group, where both $\mathcal{C}$ and $\mathcal{K}$ contains each and every group of the type system, and the multiplicity is unlimited, ω. This means that any and all processes can be typed with at least this most permissive typing.

Definition 1 (Most Permissive Typing)**.** *A security policy* Δ *is most permissive iff:*
$\forall \mathcal{G} \in dom(\Delta)$
$\quad \Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, \omega)$
Where $\mathcal{C} = dom(\Delta)$ *and* $\mathcal{K} = dom(\Delta)$

We say that a security policy Δ is minimal iff for any group $\mathcal{G} \in dom(\Delta)$, there exist no group $\mathcal{G}_1 \in dom(\Delta)$ so that $\Delta(\mathcal{G}) \not\equiv \Delta(\mathcal{G}_1)$, that is: Δ is injective, and for all possible security policies Δ', any group in Δ is a subset of the same group in Δ'.
The most permissive and minimal typing then becomes $\forall n \in dom(E), E(n) = \mathcal{G}$ where $\Delta(\mathcal{G}) = (\{\mathcal{G}_{top}, \mathcal{G}\}, \{\mathcal{G}_{top}, \mathcal{G}\}, \omega)$

Definition 2 (Minimal Typing)**.** *A security policy* Δ *is minimal iff:*
$\forall \mathcal{G} \in dom(\Delta). \nexists \mathcal{G}_1 \in dom(\Delta) \Rightarrow \Delta(\mathcal{G}) \not\equiv \Delta(\mathcal{G}_1)$
$\forall \Delta'. \Delta \leq \Delta'$
Where $\Delta \leq \Delta'$ *if* $\forall \mathcal{G}. \Delta(\mathcal{G}) \subseteq \Delta'(\mathcal{G})$

Theorem 1 (Typeability)**.** *Any process* P *can be typed with the most permissive typing.*

Proof. Which we prove by induction on the typing relation. All cases are mostly trivia, so we will only show a few.

Case $P \mid Q$

From the typing relation we have:

$$\frac{E_1 \vdash^{\Delta}_{\mathcal{G}} P \quad E_2 \vdash^{\Delta}_{\mathcal{G}} Q}{E \vdash^{\Delta}_{\mathcal{G}} P|Q} \qquad E = E_1 \oplus E_2 \qquad \text{(E-Par)}$$

By induction we have that $E_1 \vdash^{\Delta}_{\mathcal{G}} P$ and $E_2 \vdash^{\Delta}_{\mathcal{G}} Q$ is well-typed, and typed with the most permissive typing. The joining of the environments are straight forward, Δ must be the same in both environments, as such any names in both E_1 and E_2 will have exactly the same type, the most permissive type.

Case $n[P]$

From the typing relation we have:

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}_n} P}{E \vdash^{\Delta}_{\mathcal{G}} n[P]} \qquad \text{(E-Amb)}$$

where $\Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$ and $\Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n)$,
$\mathcal{G} \in \mathcal{C}_n,\ m \neq 0$ and $\mathcal{K} \subseteq \mathcal{C}_n$

By induction we have that $E \vdash^{\Delta}_{\mathcal{G}_n} P$ is well-typed, and typed with the most permissive typing. From the definition of the most permissive typing we have that $\forall \mathcal{G} \in dom(\Delta),\ \Delta(\mathcal{G}) = (dom(\Delta), dom(\Delta), \omega)$, thereby $\mathcal{G}$ and $\mathcal{G}_n$ also have this most permissive type. The first part of the side condition holds true, as $\mathcal{G}$ is in $dom(\Delta)$ and therefore in $\mathcal{C}_n$. The multiplicity of $\mathcal{G}$ is ω and thereby not equal to 0, meaning the second part of the side condition also holds true. The third part of the side condition also holds true, as $\mathcal{K} = \mathcal{C}_n = dom(\Delta)$, and therefore $\mathcal{K} \subseteq \mathcal{C}_n$.

The remaining cases are similar to the case of $n[P]$ □

2.5 Type Soundness

In proving the soundness of our type system, we use a annotated reduction semantics $P \xrightarrow{\alpha} Q$, where α is either $cap\, n$, $kill\, n$ or 1. Where $cap\, n$ is a $\mathsf{in}\ n$ or $\mathsf{out}\ n$ reduction, $kill\, n$ is a $\mathsf{open}\ n$ or $\mathsf{kill}\ n$ reduction, and 1 is a restricted or replicated reduction. The full annotated reduction semantics is given in Appendix B
Using this annotated reduction semantics we get Theorem 2. The theorem tells us that a process P in group $\mathcal{G}$ may reduce to a process P' in group $\mathcal{G}'$ where $\mathcal{G}'$ may or may not be the same as $\mathcal{G}$ and if the annotation is $\mathsf{kill}\ n$ then n must be killable in $\mathcal{G}$ and n must have multiplicity ω. In this report we will only prove the theorem for any of the reductions rules altered by the changes in our typing relation. The full proof can be seen in Appendix C

Theorem 2 (Type Soundness)**.** *If* $E \vdash^{\Delta}_{\mathcal{G}} P$ *and* $P \xrightarrow{\alpha} P'$ *then*

(1) $E \vdash^{\Delta}_{\mathcal{G}'} P'$ *and*

(2) if $\alpha = kill\ n$ *where* $E(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ *then* $\mathcal{G} \in \mathcal{K}$ *and* $m = \omega$

Proof. Which we prove by induction in the annotated reduction rules

Case $n[Q \mid m[\textbf{out}\ n.P]] \overset{cap\,n}{\rightarrow} n[Q] \mid m[P]$

Starting with $E \vdash^{\Delta}_{\mathcal{G}} n[Q \mid m[\mathsf{out}\ n.P]]$, we get $E(n) = \mathcal{G}_n$ and $E \vdash^{\Delta}_{\mathcal{G}_n} Q \mid m[\mathsf{out}\ n.P]$, this was concluded using (E-Amb). Then using (E-Par) we get $E_1 \vdash^{\Delta_1}_{\mathcal{G}_n} Q$ and $E_2 \vdash^{\Delta_2}_{\mathcal{G}_n} m[\mathsf{out}\ n.P]$. Using (E-Amb) again we get $E_2 \vdash^{\Delta_2}_{\mathcal{G}_m} \mathsf{out}\ n.P$, then using E-Out we get $E(n) = \mathcal{G}_n$ and $E_2 \vdash^{\Delta_2}_{\mathcal{G}_m} P$ where $\Delta_2(\mathcal{G}_m) = (\mathcal{C}_m,\ \mathcal{K}_m, m_m), \Delta_2(\mathcal{G}_n) = (\mathcal{C}_n,\ \mathcal{K}_n, m_n)$ and $\forall \mathcal{G}_a \in \mathcal{C}_n, \Delta_2(\mathcal{G}_a) = (\mathcal{C}_a, \mathcal{K}_a, m_a),\ \mathcal{G}_m \in \mathcal{C}_a$ and $\mathcal{K}_a \subseteq \mathcal{C}_m$, where some $\mathcal{G}_a$ must be $\mathcal{G}$

Then using induction, Lemma 1 and (E-Par) we get our result $E \vdash^{\Delta}_{\mathcal{G}} n[Q] \mid m[P]$ where $E = E_1 \oplus E_2$

Case $m[\textbf{in}\ n.P] \mid n[Q] \overset{cap\,n}{\rightarrow} n[Q \mid m[P]]$

Starting with $E \vdash^{\Delta}_{\mathcal{G}} m[\mathsf{in}\ n.P] \mid n[Q]$, we get $E_1 \vdash^{\Delta_1}_{\mathcal{G}} m[\mathsf{in}\ n.P]$ $E_2 \vdash^{\Delta_2}_{\mathcal{G}} n[Q]$ where $E = E_1 \cup E_2, \Delta = \Delta_1 \oplus \Delta_2$, this was concluded using (E-Par). Then using (E-Amb) we get $E(m) = \mathcal{G}_m$ and $E_1 \vdash^{\Delta_1}_{\mathcal{G}_m} \mathsf{in}\ n.P$. Using (E-In) we get $E(n) = \mathcal{G}_n$ and $E_1 \vdash^{\Delta_1}_{\mathcal{G}_m} P$ where $\Delta_1(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$ and $\Delta_1(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n), \mathcal{G}_n \in \mathcal{C}$ and $\mathcal{K}_n \subseteq \mathcal{C}$

Then using induction, Lemma 1 and (E-Amb) we get our result $E \vdash^{\Delta}_{\mathcal{G}} n[Q \mid m[P]]$

Case $\textbf{open}\ n.P \mid n[Q] \overset{kill\,n}{\rightarrow} Q \mid P$

It must be the case that there exist Δ_1, E_1 and Δ_2, E_2 such that $\Delta = \Delta_1 \oplus \Delta_2, E = E_1 \cup E_2$, $E_1 \vdash^{\Delta_1}_{\mathcal{G}} \mathsf{open}\ n.P$ and $E_2 \vdash^{\Delta_2}_{\mathcal{G}} n[Q]$. Using (E-Amb) we get $E_2(n) = \mathcal{G}_n$, $\Delta_2(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ where it must be the case that $m \neq 0$ and $E_2 \vdash^{\Delta_2}_{\mathcal{G}_n} Q$. And using (E-Open) we get $E_1(n) = \mathcal{G}_n$, $\Delta_1(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ where it must be the case that $\mathcal{G} \in \mathcal{K}$ and $m = \omega$ and $E_1 \vdash^{\Delta_1}_{\mathcal{G}} P$.

By using Lemma 1 we get $E_2 \vdash^{\Delta_2}_{\mathcal{G}} Q$ and by joining with $E_1 \vdash^{\Delta_1}_{\mathcal{G}} P$ we get $E \vdash^{\Delta}_{\mathcal{G}} P \mid Q$ where $\Delta = \Delta_1 \oplus \Delta_2, E = E_1 \cup E_2$

□

Chapter 3

Generating Constraints

In this chapter we describe how we can generate constraints for a processes P. We do this by taking a process P and a group $\mathcal{G}$, where $\mathcal{G}$ is the group that P is currently in, and it returns a constraint $(\mathbb{C}_E, \mathbb{C}_\Delta)$ where $\mathbb{C}_E$ is the set of constraints on the group environment E

$$\mathbb{C}_E ::= \ \mathbb{C}_E^1 \cup \mathbb{C}_E^2 \mid n : \mathcal{G}_n \mid new\, n : \mathcal{G}_n$$

and $\mathbb{C}_\Delta$ is the set of constraints on the security policy Δ.

$$\begin{aligned}\mathbb{C}_\Delta ::= \mathbb{C}_\Delta^1 \wedge \mathbb{C}_\Delta^2 \mid \mathcal{G} \in C_{\mathcal{G}_n} \mid \mathcal{G} \in K_{\mathcal{G}_n} \mid m_\mathcal{G} = \omega \mid m_\mathcal{G} \neq 0 \mid \\ \forall n \in \{n, \ldots, m\}.\mathbb{C}_{\Delta_2} \mid \forall \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n}.\mathbb{C}_{\Delta_2} \mid \mathcal{K}_\mathcal{G} \subseteq \mathcal{C}_{\mathcal{G}_n}\end{aligned}$$

In the following we assume the Barendregt-convention[1], that is: that all bound variables are pairwise distinct and distinct from all free variables. In practice this is achieved by alpha-converting the process before we generate the constraints on it.

Parallel Composition

The first process we will look at is parallel composition. From the typing rule we can see, that we need to type each of the parallel processes separately, and that the resulting environment is gained from the typing of the separate processes.

$$\frac{E_1 \vdash_\mathcal{G}^{\Delta_1} P \quad E_2 \vdash_\mathcal{G}^{\Delta_2} Q}{E \vdash_\mathcal{G}^{\Delta} P|Q} \qquad E = E_1 \cup E_2, \Delta = \Delta_1 \oplus \Delta_2 \qquad \text{(E-Par)}$$

So the constraint on E is the union of the constraint from the two processes, and the constraint on Δ is the constraints from the two processes.

$$\frac{(\mathcal{G}, P_1) \rightarrow_g (\mathbb{C}_E^1, \mathbb{C}_\Delta^1) \qquad (\mathcal{G}, P_2) \rightarrow_g (\mathbb{C}_E^2, \mathbb{C}_\Delta^2)}{(\mathcal{G}, P_1 | P_2) \rightarrow_g (\mathbb{C}_E^1 \cup \mathbb{C}_E^2,\ \mathbb{C}_\Delta^1 \wedge \mathbb{C}_\Delta^2)} \quad \text{(C-Par)}$$

Ambient

The next process is the ambient. Here we see that the containing process must be typed in the group of the ambient. We also have that the current group $\mathcal{G}$ must be in C of the group of the ambient $\mathcal{G}_n$, and that the multiplicity of $\mathcal{G}_n$ must be different from 0, we also have that $\mathcal{K}$ of $\mathcal{G}$ must be in in $\mathcal{C}$ of $\mathcal{G}_n$.

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}_n} P}{E \vdash_{\mathcal{G}} n[P]}^{\Delta} \quad \text{(E-Amb)}$$

$$\text{where } \Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m) \text{ and } \Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n),$$
$$\mathcal{G} \in \mathcal{C}_n,\ m \neq 0 \text{ and } \mathcal{K} \subseteq \mathcal{C}_n$$

So the constraint on E is the union of the constraint of the process P typed in group $\mathcal{G}_n$ and the binding of n with the group $\mathcal{G}_n$. The constraint on Δ is the constraint of the process P in $\mathcal{G}_n$, and the constraints that $\mathcal{G}$ must be in $\mathcal{C}_{\mathcal{G}_n}$and that $m_{\mathcal{G}_n}$ must be different from 0, we also have that $\mathcal{K}_{\mathcal{G}}$ must be in in $\mathcal{C}_{\mathcal{G}_n}$.

$$\frac{(\mathcal{G}_n, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)}{(\mathcal{G}, n[P]) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} \neq 0 \wedge \mathcal{K}_{\mathcal{G}} \subseteq C_{\mathcal{G}_n})} \quad \text{(C-Amb)}$$

Null

The typing of the null process is trivial.

$$\frac{}{E \vdash_{\mathcal{G}} 0}^{\Delta} \quad \text{(E-Null)}$$

As expected, null being trivial is reflected in it neither returning any constraints on E or Δ.

$$\frac{}{(\mathcal{G}, 0) \rightarrow_g (\emptyset, \emptyset)} \quad \text{(C-Null)}$$

New

Due to the Barendregt-convention, the new process is rather simple. From the typing rule we have that all we need to do, is type P with the restricted name n and its type $\mathcal{G}_n$ added to the environment E.

$$\frac{E, n : \mathcal{G}_n \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} (\nu n : \mathcal{G}_n)P} \tag{E-New}$$

So the constraints on E are simple the constraints of P and the binding of a new n with $\mathcal{G}_n$, and the constraints on Δ is simply the constraints on P.

$$\frac{(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E \; \mathbb{C}_\Delta)}{(\mathcal{G}, (\nu n : \mathcal{G}_n)P) \rightarrow_g (\mathbb{C}_E \cup new\; n : \mathcal{G}_n,\; \mathbb{C}_\Delta)} \tag{C-New}$$

In Capability

From the typing rule of the in capability we have, that the group $\mathcal{G}_n$ of n must be in $\mathcal{C}$ of the current group $\mathcal{G}$, we also have that $\mathcal{K}$ of $\mathcal{G}_n$ must be in $\mathcal{C}$ of $\mathcal{G}$.

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{in}\; n.P} \tag{E-In}$$

where $\Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$ and $\Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n), \mathcal{G}_n \in \mathcal{C}$ and $\mathcal{K}_n \subseteq \mathcal{C}$

This is reflected in the constraint, where the constraint on E is the union of the constraint from P and the binding of n to the group $\mathcal{G}_n$, and the constraint on Δ is the constraint on P and the constraint that $\mathcal{G}_n$ must be in $\mathcal{C}$ of $\mathcal{G}$, we also have that $\mathcal{K}_{\mathcal{G}_n}$ must be in in $\mathcal{C}_\mathcal{G}$.

$$\frac{(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)}{(\mathcal{G}, \mathsf{in}\; n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\; \mathbb{C}_\Delta \wedge \mathcal{G}_n \in \mathcal{C}_\mathcal{G} \wedge \mathcal{K}_{\mathcal{G}_n} \subseteq \mathcal{C}_\mathcal{G})} \tag{C-In}$$

Out Capability

The out capability is a little more complicated. From the side condition of the typing rule, we have that $\mathcal{C}$ of the group $\mathcal{G}_n$ must be in $\mathcal{C}$ of $\mathcal{G}$, our current group, we also have that forall $\mathcal{G}_a$ in $\mathcal{C}$ of $\mathcal{G}_n$, $\mathcal{K}$ of $\mathcal{G}_a$ must be in $\mathcal{C}$ of $\mathcal{G}$.

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{out}\; n.P} \tag{E-Out}$$

$$\text{where } \Delta(\mathcal{G}) = (\mathcal{C},\ \mathcal{K}, m), \Delta(\mathcal{G}_n) = (\mathcal{C}_n,\ \mathcal{K}_n, m_n) \text{ and } \forall \mathcal{G}_a \in \mathcal{C}_n,$$
$$\Delta(\mathcal{G}_a) = (\mathcal{C}_a, \mathcal{K}_a, m_a),\ \mathcal{G} \in \mathcal{C}_a \text{ and } \mathcal{K}_a \subseteq \mathcal{C}$$

This is reflected in the constraint, where the constraint on E is the union of the constraint from P and the binding of n to the group $\mathcal{G}_n$, and the constraint on Δ is the constraint on P and the constraint that $\mathcal{C}_{\mathcal{G}_n}$ must be in $\mathcal{C}_{\mathcal{G}}$, we also have that for all groups $\mathcal{G}_a$ in $\mathcal{C}_n$, $\mathcal{K}_{\mathcal{G}_a}$ must be in in $\mathcal{C}_{\mathcal{G}}$.

$$\frac{(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)}{(\mathcal{G}, \mathsf{out}\ n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \forall \mathcal{G}_a \in \mathcal{C}_{\mathcal{G}_n}.\mathcal{G} \in C_{\mathcal{G}_a} \wedge \mathcal{K}_{\mathcal{G}_a} \subseteq C_{\mathcal{G}})} \quad \text{(C-Out)}$$

Open Capability

From the typing rule of open we have, that the current group $\mathcal{G}$ must be in $\mathcal{K}$ of $\mathcal{G}_n$, the groups that may kill $\mathcal{G}_n$, and that the multiplicity of $\mathcal{G}_n$ the group of n must be unlimited, ω.

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{open}\ n.P} \quad \text{(E-Open)}$$
$$\text{where } \Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m),\ \mathcal{G} \in \mathcal{K},\ m = \omega$$

This is reflected in the constraint, where the constraint on E is the union of the constraint from P and the binding of n to the group $\mathcal{G}_n$, and the constraint on Δ is the constraint on P and the constraint that $\mathcal{G}$ must be in $\mathcal{K}_{\mathcal{G}_n}$ and that $m_{\mathcal{G}_n}$ must be ω.

$$\frac{(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)}{(\mathcal{G}, \mathsf{open}\ n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{K}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} = \omega)} \quad \text{(C-Open)}$$

Kill Capability

The typing rule of the kill capability is identical to the open capability.

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{kill}\ n.P} \quad \text{(E-Kill)}$$
$$\text{where } \Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m),\ \mathcal{G} \in \mathcal{K},\ m = \omega$$

And so is its constraints.

$$\frac{(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)}{(\mathcal{G}, \mathsf{kill}\ n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{K}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} = \omega)} \quad \text{(C-Kill)}$$

Replication

From the formation rule for replication we have that any name n in the process P must be unlimited, that is that m of $\mathcal{G}_n$, where $\mathcal{G}_n$ is the group of n, must be ω. We find all the names in P using the $n(P)$, which gives us all names in P.

$$\frac{E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} !P} \qquad \text{(E-Rep)}$$

$$\text{where } \forall n \in n(P),\ E(n) = \mathcal{G}_n, \Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m),\ m = \omega$$

This is reflected in the constraint, where the constraint on E is the constraint from P, and the constraint on Δ is the constraint on P and the constraint that for all n in $n(P)$, $m_{\mathcal{G}_n}$ must be ω.

$$\frac{(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)}{(\mathcal{G},\ !P) \rightarrow_g (\mathbb{C}_E,\ \mathbb{C}_\Delta \wedge \forall n \in n(P).m_{\mathcal{G}_n} = \omega)} \qquad \text{(C-Rep)}$$

Chapter 4

Solving Constraints

In this chapter we will describe how to solve the constraints generated in the previous section, the process works in two steps:

1. First we generate our initial and final group environment E_{init} and E and initial security policy Δ_{init}.
2. Then using E_{init} and Δ_{init} we solve our constraint on Δ and generates group dependencies, through an iterative process on the possible rules of $\mathbb{C}_\Delta$.

4.1 Generating E and Δ_{init}

First we generate our group environment E, which is simply E_{init} as defined by the domain of our constraint $\mathbb{C}_E$, without all bindings on the form $new\ n : \mathcal{G}$

$$E_{init} ::= dom(\mathbb{C}_E)$$
$$\forall new\ n \in E_{init}.E ::= E_{init} \setminus n$$

Where $E \setminus n$ is defined as:

$$E \setminus n = E' \qquad If\ E ::= E', n : \mathcal{G}$$

Next we generate our initial security policy Δ_{init}, where we initialize all the possible groups, as defined by the bindings in E_{init}, to the type $(\emptyset, \emptyset, \bot)$

$$\Delta_{init} ::= \forall \mathcal{G} \in E_{init}.\mathcal{G} \mapsto (\emptyset, \emptyset, \bot)$$

Where $\bot$ means the multiplicity is not yet determined.

4.2 Solving our Δ-constraint

We then solve our Δ-constraint using the following transition:

$$E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A)$$

Saying: The constraint $\mathbb{C}_\Delta$ yields a security policy Δ and a dependency set A, given a initial group environment E_{init} and initial security policy Δ_{init}. Where A is the arc-set of a directed graph telling which part of a group ($\mathcal{C}$ or $\mathcal{K}$) is dependent on which part of other groups. So if the arc $(\mathcal{C}_\mathcal{G}, \mathcal{C}_{\mathcal{G}_1})$ exists, we have that $\mathcal{C}_\mathcal{G} \subseteq \mathcal{C}_{\mathcal{G}_1}$.
From the previous chapter we have that our security policy constraint is a conjunction $\bigwedge_{i=1}^{k} \mathbb{C}_{\Delta_i}$ where $\mathbb{C}_\Delta$ is defined as follows

$$\begin{aligned}\mathbb{C}_\Delta ::= \emptyset \mid \mathcal{G} \in C_{\mathcal{G}_n} \mid \mathcal{G} \in K_{\mathcal{G}_n} \mid m_\mathcal{G} = \omega \mid m_\mathcal{G} \neq 0 \mid \\ \forall n \in \{n, \ldots, m\}.\mathbb{C}_{\Delta_2} \mid \forall \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n}.\mathbb{C}_{\Delta_2} \mid \mathcal{K}_\mathcal{G} \subseteq \mathcal{C}_{\mathcal{G}_n}\end{aligned}$$

Where in all we define $\Delta_1 \wedge \Delta_2$ as:

$$\Delta_1 \wedge \Delta_2 = \Delta \text{ where } \Delta(\mathcal{G}) = \begin{cases} \Delta_1(\mathcal{G}) & \text{if } \mathcal{G} \in dom(\Delta_1) \text{ and } \mathcal{G} \notin dom(\Delta_2) \\ \Delta_2(\mathcal{G}) & \text{if } \mathcal{G} \in dom(\Delta_2) \text{ and } \mathcal{G} \notin dom(\Delta_1) \\ \Delta_1(\mathcal{G}) \otimes \Delta_2(\mathcal{G}) & \text{if } \mathcal{G} \in dom(\Delta_2) \text{ and } \mathcal{G} \in dom(\Delta_1) \end{cases}$$

Where $\otimes$ is defined as:

$$(\mathcal{C}_1, \mathcal{K}_1, m_1) \otimes (\mathcal{C}_2, \mathcal{K}_2, m_2) = (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{K}_1 \cup \mathcal{K}_2, m)$$

Where m is defined as:

$$m = \begin{cases} \omega & \text{if } m_1 = \omega \vee m_2 = \omega \vee (m_1 = \top \wedge m_2 = \top) \\ \top & \text{if } (m_1 = \top \vee m_2 = \top) \wedge m_1 \neq m_2 \wedge m_1 \neq \omega \wedge m_2 \neq \omega \\ \bot & \text{if } m_1 = \bot \wedge m_2 = \bot \end{cases}$$

We then have that the transition for each of the rules is as follows:

Solving $(\emptyset)$

For the case of $\mathbb{C}_\Delta = (\emptyset)$ we get, to no surprise, that it changes nothing in our security policy.

$$\frac{}{E_{init}, \Delta_{init} \vdash (\emptyset) \rightarrow_s (\emptyset, \emptyset)}$$

Solving $(\mathcal{G} \in C_{\mathcal{G}_n})$

For the case of $\mathbb{C}_\Delta = (\mathcal{G} \in C_{\mathcal{G}_n})$ we remap $\mathcal{G}_n$ to also contain $\mathcal{G}$ in its $\mathcal{C}$-component, we also solve any remaining constraint $\mathbb{C}_\Delta$ with this changed group.

$$\frac{E_{init}, \Delta_{init}[\mathcal{G}_n \mapsto (\mathcal{C} \cup \{\mathcal{G}\}, \mathcal{K}, m)] \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge (\mathcal{G} \in C_{\mathcal{G}_n}) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\mathcal{C} \cup \{\mathcal{G}\}, \mathcal{K}, m), A)}$$

$$where\ E_{init}(n) = \mathcal{G}_n\ and\ \Delta_{init}(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$$

Solving $(\mathcal{G} \in K_{\mathcal{G}_n})$

For the case of $\mathbb{C}_\Delta = (\mathcal{G} \in K_{\mathcal{G}_n})$ we remaps $\mathcal{G}_n$ to also contain $\mathcal{G}$ in its $\mathcal{K}$-component, we also solve any remaining constraint $\mathbb{C}_\Delta$ with this changed group.

$$\frac{E_{init}, \Delta_{init}[\mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K} \cup \{\mathcal{G}\}, m)] \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge (\mathcal{G} \in K_{\mathcal{G}_n}) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K} \cup \{\mathcal{G}\}, m), A)}$$

$$where\ E_{init}(n) = \mathcal{G}_n\ and\ \Delta_{init}(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$$

Solving $(\mathcal{K}_\mathcal{G} \subseteq \mathcal{C}_{\mathcal{G}_n})$

For the case of $\mathbb{C}_\Delta = (\mathcal{K}_\mathcal{G} \subseteq \mathcal{C}_{\mathcal{G}_n})$ we add the arc $(\mathcal{K}_\mathcal{G}, \mathcal{C}_{\mathcal{G}_n})$ to A to be solved in the post-processing.

$$\frac{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge ((\mathcal{K}_\mathcal{G} \subseteq \mathcal{C}_{\mathcal{G}_n}) \rightarrow_s (\Delta, A \cup (\mathcal{K}_\mathcal{G}, \mathcal{C}_{\mathcal{G}_n}))}$$

Solving $(m_{\mathcal{G}_n} = \omega)$

For the case of $\mathbb{C}_\Delta = (m_{\mathcal{G}_n} = \omega)$ we remaps $\mathcal{G}_n$ to have a multiplicity of ω, we also solve any remaining constraint $\mathbb{C}_\Delta$ with this changed group.

$$\frac{E_{init}, \Delta_{init}[\mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K}, \omega)] \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge (m_{\mathcal{G}_n} = \omega) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K}, \omega), A)}$$

$$where\ E_{init}(n) = \mathcal{G}_n\ and\ \Delta_{init}(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$$

Solving $(m_{\mathcal{G}_n} \neq 0)$

For the case of $\mathbb{C}_\Delta = (m_{\mathcal{G}_n} \neq 0)$ we remaps $\mathcal{G}_n$ to have a multiplicity of $\top$ if it was $\bot$.

$$\frac{E_{init}, \Delta_{init}[\mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K}, \top)] \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge (m_{\mathcal{G}_n} \neq 0) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K}, \top), A)}$$

$$where\ E_{init}(n) = \mathcal{G}_n\ and\ \Delta_{init}(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, \bot)$$

And remaps $\mathcal{G}_n$ to have a multiplicity of ω if it was $\top$ or ω., in all cases we also solve any remaining constraint $\mathbb{C}_\Delta$ with this changed group.

$$\frac{E_{init}, \Delta_{init}[\mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K}, \omega)] \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge (m_{\mathcal{G}_n} \neq 0) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\mathcal{C}, \mathcal{K}, \omega), A)}$$

$$where\ E_{init}(n) = \mathcal{G}_n\ and\ \Delta_{init}(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, \top)\ or\ \Delta_{init}(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, \omega)$$

Solving $(\forall n \in \{n, \ldots, m\}.\mathbb{C}_{\Delta_2})$

For the case $\mathbb{C}_\Delta = (\forall n \in \{n, \ldots, m\}.\mathbb{C}_{\Delta_2})$ we solve $\mathbb{C}_\Delta$ in the current environment and forall n in $\{n, \ldots, m\}$ we solve $\mathbb{C}_{\Delta_2}$ with the prime variable renamed to n.

$$\frac{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A) \qquad \forall n \in \{n, \ldots, m\}.E_{init}, \Delta_{init} \vdash (\mathbb{C}_{\Delta_2}(n)) \rightarrow_s (\Delta_1, A_1) \ldots (\Delta_n, A_n)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge (\forall n \in \{n, \ldots, m\}.\mathbb{C}_{\Delta_2}) \rightarrow_s (\Delta \wedge \Delta_1 \wedge \cdots \wedge \Delta_n, A \cup A_1 \cup \cdots \cup A_n)}$$

Solving $(\forall \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n}.\mathbb{C}_{\Delta_2})$

For the case $\mathbb{C}_\Delta = (\forall \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n}.\mathbb{C}_{\Delta_2})$ we solve $\mathbb{C}_\Delta$ in the current environment and forall $\mathcal{G}$ in $\mathcal{C}_{\mathcal{G}_n}$ we solve $\mathbb{C}_{\Delta_2}$ with the prime variable renamed to $\mathcal{G}$.

$$\frac{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \rightarrow_s (\Delta, A) \qquad \forall \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n}.E_{init}, \Delta_{init} \vdash (\mathbb{C}_{\Delta_2}(\mathcal{G})) \rightarrow_s (\Delta_1, A_1) \ldots (\Delta_n, A_n)}{E_{init}, \Delta_{init} \vdash \mathbb{C}_\Delta \wedge (\forall \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n}.\mathbb{C}_{\Delta_2}) \rightarrow_s (\Delta \wedge \Delta_1 \wedge \cdots \wedge \Delta_n, A \cup A_1 \cup \cdots \cup A_n)}$$

Chapter 5

Post-processing

In this chapter we will take the E, Δ and A produced in the previous section and with it generated our final group environment E and security policy Δ. This post-processing will work in four steps:

1. First we solve the group dependencies of A, finalising the $\mathcal{C}$ of all types.
2. Next we assign multiplicities to any group which has not yet been assigned one.
3. With all groups assigned a multiplicity, we ensure that no linear group may end up inside an unlimited group, where it risk being killed.
4. Finally we ensure that no two groups have the same type, ensuring that our typing is minimal.

5.1 Solving group dependencies

The first thing we do in our post-processing is solving our group dependency A, we solve our dependency by evaluating the reduction $\Delta \vdash (A) \to \Delta'$ where $A = (\mathcal{K}_{\mathcal{G}_a}, \mathcal{C}_{\mathcal{G}_1}), \ldots, (\mathcal{K}_{\mathcal{G}_z}, \mathcal{C}_{\mathcal{G}_n})$.

We have two cases of A, the first have the first element in the list A being a dependency on the $\mathcal{K}$-set of a group $\mathcal{G}$.

$$\frac{\Delta \vdash (D) \to \Delta'}{\Delta \vdash ((\mathcal{K}_{\mathcal{G}}, \mathcal{C}_{\mathcal{G}_n}), D) \to \Delta' \otimes \mathcal{G}_n \mapsto (\mathcal{C} \cup \mathcal{K}_{\mathcal{G}}, \mathcal{K}, m)} \qquad \text{where } \Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$$

And the last where A is empty.

$$\frac{}{\Delta \vdash (\emptyset) \to \Delta}$$

5.2 Remaining post-processing

The transitions in the remaining post-processing are all off the form $E, \Delta \to E', \Delta'$, taking the generated group environment E and security policy Δ, and assigning any missing multiplicities, securing that no linear group may be killed and finally that the typing is minimal.

5.2.1 Assign missing multiplicities

The first step is assigning a multiplicity of 1 to any group not yet assigned a multiplicity, $\bot$, or assigned a multiplicity of $m \neq 0$, $\top$.

$$\frac{\forall \mathcal{G} \in dom(\Delta).(m = \top \vee m = \bot)}{E, \Delta \to E, \Delta(\mathcal{G} \mapsto (\mathcal{C}, \mathcal{K}, 1))} \qquad where\ \Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$$

5.2.2 Ensure that no linear group may be killed

Next we check every group $\mathcal{G}$ in our security policy, to see if there exists a group, $\mathcal{G}_1$, with multiplicity $m_1 = \omega$ in the groups that $\mathcal{G}$ may be contained in, if such a group exists, we assign the multiplicity of $\mathcal{G}$ to ω

$$\frac{\forall \mathcal{G} \in dom(\Delta).\exists(\mathcal{G}_1 \in \mathcal{C} \wedge m_1 = \omega)}{E, \Delta \to E, \Delta(\mathcal{G} \mapsto (\mathcal{C}, \mathcal{K}, \omega))}$$

$$where\ \Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)\ and\ \Delta(\mathcal{G}_1) = (\mathcal{C}_1, \mathcal{K}_1, m_1)$$

5.2.3 Ensure minimal typing

Finally we assure that our type system is minimal. We do this by checking for all names n bound to group $\mathcal{G}_1$, if there exists a name m bound to group $\mathcal{G}_2$, so that $\mathcal{G}_1$ and $\mathcal{G}_2$ are equal. If such a name exists, we rebind the names n and m to a new group $\mathcal{G}$ in E, and add the new group $\mathcal{G}$ to Δ with the type of the original group, finally we also remove $\mathcal{G}_1$ and $\mathcal{G}_2$ from Δ.

$$\frac{\forall E(n) = \mathcal{G}_1.\exists(E(m) = \mathcal{G}_2 \wedge \mathcal{G}_1 \equiv \mathcal{G}_2)}{E, \Delta \to E, n : \mathcal{G}, m : \mathcal{G}, \Delta[\mathcal{G} \mapsto (\mathcal{C}, \mathcal{K}, m)] \setminus \{\mathcal{G}_1, \mathcal{G}_2\}}$$

$$where\ \Delta(\mathcal{G}_1) = (\mathcal{C}, \mathcal{K}, m)$$

where $\Delta \setminus \mathcal{G}$ is defined as:

$$\Delta \setminus \mathcal{G} = \Delta' \qquad if\ \Delta ::= \Delta'[\mathcal{G} \mapsto (\mathcal{C}, \mathcal{K}, m)]$$

Chapter 6

Correctness of the Algorithm

In this chapter we prove that given a process P, our algorithm returns a E and Δ, that make P well-typed under the typing assumptions of the group environment E and security policy Δ, and that our Δ is minimal.

Theorem 3 (Correctness of the Algorithm)**.** *Given a process P we have that after the Constraint Generation $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$, Constraint Solving $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$, Solving Group Dependencies $\Delta \vdash (dom(A)) \rightarrow \Delta'$, and Post-Processing $E, \Delta \rightarrow^* E', \Delta'$.*
E' and Δ' will be subject to the following:

1. $E' \vdash_{\mathcal{G}}^{\Delta'} P$
2. Δ' *is minimal*

Proof. The proof proceeds by induction in the structure of P. In each case of the induction we prove the second half of the theorem (minimality) by contradiction as follows: Suppose Δ' is not minimal, then there must exist some Δ'', $\Delta'' \sqsubset \Delta'$, where Δ'' is minimal. From the post-processing we have that $\forall \mathcal{G} \in dom(\Delta')$ there exist no $\mathcal{G}' \in dom(\Delta')$ so that $\mathcal{G}$ and $\mathcal{G}'$ is injective, so if $\Delta'' \sqsubset \Delta'$ then there must exist some $\mathcal{G} \in dom(\Delta'') \cap dom(\Delta')$ where $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$, that is $\mathcal{G}$ contains fewer groups in either the $\mathcal{C}$ or $\mathcal{K}$ set.

Case $P \mid Q$

1. By induction we have that $E_1' \vdash_{\mathcal{G}}^{\Delta_1'} P$ and $E_2' \vdash_{\mathcal{G}}^{\Delta_2'} Q$. By using C-Par we have that $(\mathcal{G}, P \mid Q) \rightarrow_g (\mathbb{C}_{E_1} \cup \mathbb{C}_{E_2}, \mathbb{C}_{\Delta_1} \wedge \mathbb{C}_{\Delta_2})$ where $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_{E_1}, \mathbb{C}_{\Delta_1})$ and $(\mathcal{G}, Q) \rightarrow_g (\mathbb{C}_{E_2}, \mathbb{C}_{\Delta_2})$. From $\mathbb{C}_{E_1}$ we can get E_{init_1}, E_1 and Δ_{init_1}, and from $\mathbb{C}_{E_2}$ we can get E_{init_2}, E_2 and Δ_{init_2}. We

can then solve the constraint by the reductions $E_{init_1}, \Delta_{init_1} \vdash (\mathbb{C}_{\Delta_1}) \rightarrow_s (\Delta_1, A_1)$ and $E_{init_2}, \Delta_{init_2} \vdash (\mathbb{C}_{\Delta_2}) \rightarrow_s (\Delta_2, A_2)$. Finally after the post-processing we can use (E-Par) and get $E' \vdash^{\Delta'}_{\mathcal{G}} P \mid Q$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. This group must exist in either the P or Q process, but from the induction hypothesis we have that $E'_1 \vdash^{\Delta'_1}_{\mathcal{G}} P$ and $E'_2 \vdash^{\Delta'_2}_{\mathcal{G}} Q$ where both Δ'_1 and Δ'_2 is minimal, Δ' must thus already by minimal.

Case $n[P]$

1. By induction we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$. By then using C-Amb we have that $(\mathcal{G}, n[P]) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} \neq 0 \wedge \mathcal{K}_\mathcal{G} \subseteq C_{\mathcal{G}_n})$ where $(\mathcal{G}_n, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$. From $\mathbb{C}_E \cup n : \mathcal{G}_n$ we get E_{init}, E and Δ_{init} where atleast the binding $n : \mathcal{G}_n$ is in both E-environments. We can then recursively solve our constraint, $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{C}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} \neq 0 \wedge \mathcal{K}_\mathcal{G} \subseteq \mathcal{C}_{\mathcal{G}_n}) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\{\mathcal{G}\}, \emptyset, \top), A \cup (\mathcal{K}_\mathcal{G}, \mathcal{C}_{\mathcal{G}_n}))$ where $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$. By solving the group dependencies we get $\Delta(\mathcal{G}_n) \mapsto (\{\mathcal{G}\} \cup \{\mathcal{K}_\mathcal{G}\}, \emptyset, \top)$, and by assigning missing multiplicities we get $\Delta'(\mathcal{G}_n) \mapsto (\{\mathcal{G}\} \cup \{\mathcal{K}_\mathcal{G}\}, \emptyset, m)$ where $m = 1$ unless something in P changes that. Finally we have that the types given by our algorithm is consistence with (E-Par), giving us $E' \vdash^{\Delta'}_{\mathcal{G}} n[P]$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. From our algorithm we have that $\Delta'(\mathcal{G}_n) \mapsto (\{\mathcal{G}\} \cup \{\mathcal{K}_\mathcal{G}\}, \emptyset, m)$, we could either remove the group $\mathcal{G}$ or a group from the collection $\mathcal{K}_\mathcal{G}$, but in doing so we would have $E' \nvdash^{\Delta''}_{\mathcal{G}} n[P]$, from the induction hypothesis we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$ where Δ' is already minimal, Δ' must thus overall already by minimal.

Case 0

1. From C-Null we have that $(\mathcal{G}, 0) \rightarrow (\emptyset, \emptyset)$, and by solving it we get $\emptyset, \emptyset \vdash (\emptyset) \rightarrow_s (\emptyset, \emptyset)$, meaning 0 generates the everywhere undefined E and Δ environment. Looking at (E-Null) we have that $\emptyset \vdash^{\emptyset}_{\mathcal{G}} 0$ as (E-Null) have no side conditions to for fill.

2. As the 0-process generates the everywhere undefined E and Δ environment, the environments for 0 must thus already be minimal.

Case $(\nu n : \mathcal{G}_n)P$

1. By induction we have that $E' \vdash^{\Delta'}_{\mathcal{G}} P$. By then using C-New we have that $(\mathcal{G}, (\nu n : \mathcal{G}_n)P) \rightarrow_g (\mathbb{C}_E \cup new\ n : \mathcal{G}_n,\ \mathbb{C}_\Delta)$ where $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$. From $\mathbb{C}_E \cup new\ n : \mathcal{G}_n$ we get E_{init}, E and Δ_{init} where atleast the binding $n : \mathcal{G}_n$ is in E_{init}. We can then solve our constraints $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$ which is the constraint of P. Finally after the post-processing we use (E-New) and gets $E' \vdash^{\Delta'}_{\mathcal{G}} (\nu n : \mathcal{G}_n)P$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. If no such group exists in the P-process, so that $E' \vdash^{\Delta''}_{\mathcal{G}} (\nu n : \mathcal{G}_n)P$, but from the induction hypothesis we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$ where Δ' is already minimal, Δ' must thus overall already by minimal.

Case $\mathsf{in}\ n.P$

1. By induction we have that $E' \vdash^{\Delta'}_{\mathcal{G}} P$. By then using C-In we have that $(\mathcal{G}, \mathsf{in}\ n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \mathcal{G}_n \in \mathcal{C}_\mathcal{G} \wedge \mathcal{K}_{\mathcal{G}_n} \subseteq \mathcal{C}_\mathcal{G})$ where $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$. From $\mathbb{C}_E \cup n : \mathcal{G}_n$ we get E_{init}, E and Δ_{init} where atleast the binding $n : \mathcal{G}_n$ is in both E-environments. We can then recursively solve our constraint, yielding us $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta \wedge \mathcal{G}_n \in \mathcal{C}_\mathcal{G} \wedge \mathcal{K}_{\mathcal{G}_n} \subseteq \mathcal{C}_\mathcal{G}) \rightarrow_s (\Delta \wedge \mathcal{G} \mapsto (\{\mathcal{G}_n\}, \emptyset, \bot), A \cup (\mathcal{K}_{\mathcal{G}_n}, \mathcal{C}_\mathcal{G}))$ where $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$. By solving the group dependencies we get $\Delta(\mathcal{G}) \mapsto (\{\mathcal{G}_n\} \cup \{\mathcal{K}_{\mathcal{G}_n}\}, \emptyset, \bot)$, and by assigning missing multiplicities we get $\Delta'(\mathcal{G}) \mapsto (\{\mathcal{G}_n\} \cup \{\mathcal{K}_{\mathcal{G}_n}\}, \emptyset, m)$ where $m = 1$ unless something in P changes that. Finally we have that the types given by our algorithm is consistence with (E-In), giving us $E' \vdash^{\Delta'}_{\mathcal{G}} \mathsf{in}\ n.P$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. From our algorithm we have that $\Delta'(\mathcal{G}) \mapsto (\{\mathcal{G}_n\} \cup \{\mathcal{K}_{\mathcal{G}_n}\}, \emptyset, m)$, we could either remove the group $\mathcal{G}$ or a group from the collection $\mathcal{K}_{\mathcal{G}_n}$, but in doing so we would have $E' \not\vdash^{\Delta''}_{\mathcal{G}} \mathsf{in}\ n.P$, such a group must then exists in the

P-process, so that $E' \vdash^{\Delta''}_{\mathcal{G}} \mathsf{in}\ n.P$, but from the induction hypothesis we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$ where Δ' is already minimal, Δ' must thus overall already by minimal.

Case out $n.P$

1. By induction we have that $E' \vdash^{\Delta'}_{\mathcal{G}} P$. By then using C-Out we have that $(\mathcal{G}, \mathsf{out}\ n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \forall \mathcal{G}_a \in \mathcal{C}_{\mathcal{G}_n}.\mathcal{G} \in C_{\mathcal{G}_a} \wedge \mathcal{K}_{\mathcal{G}_a} \subseteq C_{\mathcal{G}})$ where $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$. From $\mathbb{C}_E \cup n : \mathcal{G}_n$ we get E_{init}, E and Δ_{init} where atleast the binding $n : \mathcal{G}_n$ is in both E-environments. We can then recursively solve our constraint, yielding us $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta \wedge \forall \mathcal{G}_a \in \mathcal{C}_{\mathcal{G}_n}.\mathcal{G} \in C_{\mathcal{G}_a} \wedge \mathcal{K}_{\mathcal{G}_a} \subseteq C_{\mathcal{G}}) \rightarrow_s (\Delta \wedge \Delta_1 \wedge \cdots \wedge \Delta_n, A \cup A_1 \cup \cdots \cup A_n \cup (\mathcal{C}_{\mathcal{G}_n}, \mathcal{C}_{\mathcal{G}}))$ where $\forall \mathcal{G}_a \in \mathcal{C}_{\mathcal{G}_n}.E_{init}, \Delta_{init} \vdash (\mathcal{G} \in C_{\mathcal{G}_a}) \rightarrow_s (\Delta_1, A_1) \ldots (\Delta_n, A_n)$ and $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$. By solving the group dependencies we get $\Delta(\mathcal{G}) \mapsto (\{\mathcal{K}_{\mathcal{G}_a}\}, \emptyset, \top)$, $\Delta(\mathcal{G}_a) \mapsto (\mathcal{G}, \emptyset, \top)$, and by assigning missing multiplicities we get $\Delta(\mathcal{G}) \mapsto (\{\mathcal{K}_{\mathcal{G}_a}\}, \emptyset, m)$, $\Delta(\mathcal{G}_a) \mapsto (\mathcal{G}, \emptyset, m)$ where $m = 1$ unless something in P changes that. Finally after the post-processing we have that $E' \vdash^{\Delta'}_{\mathcal{G}} P$ and using (E-Out) we have that $E' \vdash^{\Delta'}_{\mathcal{G}} \mathsf{out}\ n.P$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. From our algorithm we have that $\Delta(\mathcal{G}) \mapsto (\{\mathcal{K}_{\mathcal{G}_a}\}, \emptyset, m)$, $\Delta(\mathcal{G}_a) \mapsto (\mathcal{G}, \emptyset, m)$, we could either remove the group $\mathcal{G}$ or a group from the collection $\mathcal{K}_{\mathcal{G}_a}$, but in doing so we would have $E' \nvdash^{\Delta''}_{\mathcal{G}} \mathsf{out}\ n.P$, such group must then exists in the P-process, so that $E' \vdash^{\Delta''}_{\mathcal{G}} \mathsf{out}\ n.P$, but from the induction hypothesis we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$ where Δ' is already minimal, Δ' must thus overall already by minimal.

Case open $n.P$

1. By induction we have that $E' \vdash^{\Delta'}_{\mathcal{G}} P$. By then using C-Open we have $(\mathcal{G}, \mathsf{open}\ n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{K}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} = \omega)$ where $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$. From $\mathbb{C}_E \cup n : \mathcal{G}_n$ we get E_{init}, E and Δ_{init} where atleast the binding $n : \mathcal{G}_n$ is in both E-environments. We can then recursively solve our constraint, yielding us $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{K}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} = \omega) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\emptyset, \{\mathcal{G}\}, \omega), A)$ where $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$. Finally we have that the types given by our algorithm is consistence with (E-Open), giving us $E' \vdash^{\Delta'}_{\mathcal{G}} \mathsf{open}\ n.P$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. From our algorithm we have that $\Delta(\mathcal{G}_n) \mapsto (\emptyset, \{\mathcal{G}\}, \omega)$, we could remove the group $\mathcal{G}$, but in doing so we would have $E' \not\vdash^{\Delta''}_{\mathcal{G}}$ $\mathsf{open}\ n.P$, such a group must then exists in the P-process, so that $E' \vdash^{\Delta''}_{\mathcal{G}}$ $\mathsf{open}\ n.P$, but from the induction hypothesis we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$ where Δ' is already minimal, Δ' must thus overall already by minimal.

Case $\mathsf{kill}\ n.P$

1. By induction we have that $E' \vdash^{\Delta'}_{\mathcal{G}} P$. By then using C-Kill we have $(\mathcal{G}, \mathsf{kill}\ n.P) \rightarrow_g (\mathbb{C}_E \cup n : \mathcal{G}_n,\ \mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{K}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} = \omega)$ where $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$. From $\mathbb{C}_E \cup n : \mathcal{G}_n$ we get E_{init}, E and Δ_{init} where atleast the binding $n : \mathcal{G}_n$ is in both E-environments. We can then recursively solve our constraint, yielding us $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta \wedge \mathcal{G} \in \mathcal{K}_{\mathcal{G}_n} \wedge m_{\mathcal{G}_n} = \omega) \rightarrow_s (\Delta \wedge \mathcal{G}_n \mapsto (\emptyset, \{\mathcal{G}\}, \omega), A)$ where $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$. Finally we have that the types given by our algorithm is consistence with (E-Kill), giving us $E' \vdash^{\Delta'}_{\mathcal{G}}$ $\mathsf{kill}\ n.P$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. From our algorithm we have that $\Delta(\mathcal{G}_n) \mapsto (\emptyset, \{\mathcal{G}\}, \omega)$, we could remove the group $\mathcal{G}$, but in doing so we would have $E' \not\vdash^{\Delta''}_{\mathcal{G}}$ $\mathsf{kill}\ n.P$, such a group must then exists in the P-process, so that $E' \vdash^{\Delta''}_{\mathcal{G}}$ $\mathsf{open}\ n.P$, but from the induction hypothesis we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$ where Δ' is already minimal, Δ' must thus overall already by minimal.

Case $!P$

1. By induction we have that $E' \vdash^{\Delta'}_{\mathcal{G}} P$. By then using C-Rep we have $(\mathcal{G}, !P) \rightarrow_g (\mathbb{C}_E,\ \mathbb{C}_\Delta \wedge \forall n \in n(P).m_{\mathcal{G}_n} = \omega)$ where $(\mathcal{G}, P) \rightarrow_g (\mathbb{C}_E, \mathbb{C}_\Delta)$. From $\mathbb{C}_E$ we get E_{init}, E and Δ_{init}. We can then recursively solve our constraint, yielding us $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta \wedge \forall n \in n(P).m_{\mathcal{G}_n} = \omega) \rightarrow_s (\Delta \wedge \Delta_1 \wedge \cdots \wedge \Delta_n, A \cup A_1 \cup \cdots \cup A_n)$ where $E_{init}, \Delta_{init} \vdash (\mathbb{C}_\Delta) \rightarrow_s (\Delta, A)$, where $\Delta_1 \wedge \cdots \wedge \Delta_n$ each contains a group $\mathcal{G}_{1,\ldots,n} \mapsto (\emptyset, \emptyset, \omega)$. Finally we have that the types given by our algorithm is consistence with (E-Rep), giving us $E' \vdash^{\Delta'}_{\mathcal{G}} !P$

2. We have that if Δ' is not minimal then there must exists some Δ'' so that $\Delta'' \sqsubset \Delta'$, if such a security policy exist atleast one group $\mathcal{G} \in dom(\Delta') \cap dom(\Delta'')$ must exist so that $\Delta''(\mathcal{G}) \subset \Delta'(\mathcal{G})$. There are no types besides the ones in P, such a group must then exists in the P-process, so that $E' \vdash^{\Delta''}_{\mathcal{G}} \mathsf{open}\ n.P$, but from the induction hypothesis we have that $E' \vdash^{\Delta'}_{\mathcal{G}_n} P$ where Δ' is already minimal, Δ' must thus overall already by minimal.

□

Chapter 7

Conclusion

In this chapter we will draw our conclusion, summing up the results gained in this report and discuss the possible future work in regards to those results.

7.1 Results

In this report we presented a modified type system for the Ambient Calculus with Kill, which through Lemma 1 handles the problem of consistent typing between our type system and the one of Cardelli, Ghelli and Gordon[2], where e.g. the reduction of $\mathsf{open}\ n.P \mid n[Q]$ is typeable in [2] but not in our type system. It also presents three modified typing rules, which introduce the idea of group dependencies, where the type of one group may be dependent on the type of another group. We also present the notion of minimal typing, which is a typing where no group $\mathcal{G}$ have the same type as any other group in Δ, that is: Δ is injective. We also find the interesting result that any process P may be typed using atleast a most permissive typing, where each group in Δ have a type of $(dom(\Delta), dom(\Delta), \omega)$, that is: all groups may be everywhere, and be killed everywhere.

We then built a type inference algorithm on our modified type system, the algorithm works in three steps: constraint generation, constraint solving and post-processing.

The first step, constraint generation, takes a process P and a group $\mathcal{G}$ and generates a constraint $(\mathbb{C}_E, \mathbb{C}_\Delta)$, the constraint generation is done for each of the sub-processes in P, where each sub-process generates its own constraints

which together gives the over all constraint.

The next step, constraint solving, takes the constraint generated from the previous step, and with it generates E_{init}, E and Δ_{init}, which in turn is used to solve the constraint on Δ and generate group dependency, the solving is done through a iterative process solving the constraint for each sub-part of the over all constraint.

The final part of our algorithm is the post-processing, where we do some final adjustments to the E and Δ environments, this includes solving our group dependencies, assigning missing multiplicities, securing that no linear groups may accidentally be killed and that our typing is minimal.

Last but not least, we prove that our algorithm produces a typing which makes the process P well-typed, and that our final Δ environment is minimal.

7.2 Future Work

With a proven algorithm, an interesting possible next step would be to make an implementation of our algorithm. A possible implementation language could be Prolog as it nicely and easily support first order logic which our algorithm could easily be converted to, an implementation would then, more or less, just be the rules already stated. With an implementation at hand, it would also be interesting to use it on some larger and possible more complex systems.

When talking about implementing our algorithm, another interesting aspect to look at, is the time complexity of our algorithm. As the constraint is generated from P, we would assume that the generation is linear in P, same goes for the solving which would be linear in the constraints generated from P. As for the post-processing we have that solving group dependencies looks linear in the number of dependencies, and the remaining three is linear in the number of groups in our typing.

Other extensions of the ambient calculus introduces the notion of communication between processes, it would be interesting to look at what the addition of communication would do to our calculus with kill. It would then be possible to communicate the name of an ambient to be killed, or even to communicate a whole kill-process into an ambient where there where none before. This would present a number of considerations in respect to our type

inference algorithm, as we would have to consider what could be communicated in a given ambient, and incorporate that in to our typing.

Another interesting area to look in to, would be to compare the typing given by our algorithm with the predictions returned from a static analysis. Where our algorithm looks at where a given name is mentioned and from that deduce where the name might be contained, the prediction of a static analysis, like e.g. shape analysis[7], could be build so as to only return where a given name may actually end up from the possible actions. We would, with the static analysis, be able to more precisely see where an ambient $n[P]$ would end up with a process $\mathsf{kill}\ n.Q$, saying not only where it may be killed, but where it would be killed.

Appendix

Appendix A

The Typing Relation

(E-Par)

$$\frac{E_1 \vdash^{\Delta}_{\mathcal{G}} P \quad E_2 \vdash^{\Delta}_{\mathcal{G}} Q}{E \vdash^{\Delta}_{\mathcal{G}} P|Q} \qquad E = E_1 \oplus E_2$$

(E-Amb)

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}_n} P}{E \vdash^{\Delta}_{\mathcal{G}} n[P]}$$

where $\Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$ and $\Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n)$, $\mathcal{G} \in \mathcal{C}_n,\ m \neq 0$ and $\mathcal{K} \subseteq \mathcal{C}_n$

(E-Null) (E-New)

$$\frac{}{E \vdash^{\Delta}_{\mathcal{G}} 0} \qquad \frac{E, n : \mathcal{G}_n \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} (\nu n : \mathcal{G}_n)P}$$

(E-In)

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{in}\ n.P}$$

where $\Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$ and $\Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n), \mathcal{G}_n \in \mathcal{C}$ and $\mathcal{K}_n \subseteq \mathcal{C}$

(E-Out)

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{out}\ n.P}$$

where $\Delta(\mathcal{G}) = (\mathcal{C},\ \mathcal{K}, m), \Delta(\mathcal{G}_n) = (\mathcal{C}_n,\ \mathcal{K}_n, m_n)$ and $\forall \mathcal{G}_a \in \mathcal{C}_n$,

$$\Delta(\mathcal{G}_a) = (\mathcal{C}_a, \mathcal{K}_a, m_a),\ \mathcal{G} \in \mathcal{C}_a \text{ and } \mathcal{C}_n \subseteq \mathcal{C}$$

(E-Open)

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{open}\ n.P} \quad \text{where } \Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m),\ \mathcal{G} \in \mathcal{K},\ m = \omega$$

(E-Kill)

$$\frac{E(n) = \mathcal{G}_n \quad E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} \mathsf{kill}\ n.P} \quad \text{where } \Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m),\ \mathcal{G} \in \mathcal{K},\ m = \omega$$

(E-Rep)

$$\frac{E \vdash^{\Delta}_{\mathcal{G}} P}{E \vdash^{\Delta}_{\mathcal{G}} !P} \quad \text{where } \forall n \in n(P),\ E(n) = \mathcal{G}_n, \Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m),\ m = \omega$$

Appendix B

Annotated Reduction

(R-Par)

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

(R-Rep)

$$!P \xrightarrow{1} !P \mid P$$

(R-New1)

$$\frac{P \xrightarrow{\alpha} P'}{(\nu n)P \xrightarrow{1} (\nu n)P'} \quad \text{Where } n(\alpha) = n$$

(R-New2)

$$\frac{P \xrightarrow{\alpha} P'}{(\nu n)P \xrightarrow{\alpha} (\nu n)P'} \quad \text{Where } n(\alpha) \neq n$$

(R-Out)

$$n[Q \mid m[\mathsf{out}\ n.P]] \xrightarrow{cap\, n} n[Q] \mid m[P]$$

(R-In)

$$m[\mathsf{in}\ n.P] \mid n[Q] \xrightarrow{cap\, n} n[Q \mid m[P]]$$

(R-Open)

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

(R-Kill)

$$\mathsf{kill}\ n.P \mid n[Q] \xrightarrow{kill\, n} P$$

(R-Str)

$$\frac{P \equiv Q \quad Q \xrightarrow{\alpha} Q' \quad Q' \equiv P'}{P \xrightarrow{\alpha} P'}$$

Appendix C

Type Soundness

Lemma 2 (Weakening)**.** *If* $E \vdash^{\Delta}_{\mathcal{G}} P$ *and* $E \oplus E'$ *is well-defined then* $E \oplus E' \vdash^{\Delta}_{\mathcal{G}} P$

Lemma 3 (Structural Group Preservation)**.** *If* $P \equiv Q$ *then* $E \vdash^{\Delta}_{\mathcal{G}} P$ *iff* $E \vdash^{\Delta}_{\mathcal{G}} Q$

Case $P \mid Q \xrightarrow{\alpha} P' \mid Q$ where $P \xrightarrow{\alpha} P'$

It must be the case that there exist E_1 and E_2 such that $E = E_1 \oplus E_2$, $E_1 \vdash^{\Delta}_{\mathcal{G}} P$ and $E_2 \vdash^{\Delta}_{\mathcal{G}} Q$. Using induction, we have that $E_1 \vdash^{\Delta}_{\mathcal{G}} P'$ and using (E-Par) we get $E \vdash^{\Delta}_{\mathcal{G}} P' \mid Q$
For the case where $\alpha = \mathit{kill}\, n$, $E(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ it must be the case that $\mathcal{G} \in \mathcal{K}$ and $m = \omega$

Case $(\nu n : \mathcal{G}_n)P \xrightarrow{1} (\nu n : \mathcal{G}_n)P'$ where $P \xrightarrow{\alpha} P'$ and $n(\alpha) = n$

It must be the case that $E, n : \mathcal{G}_n \vdash^{\Delta}_{\mathcal{G}} P$. Using induction we get $E, n : \mathcal{G}_n \vdash^{\Delta}_{\mathcal{G}} P'$ and using (E-New) we get $E \vdash^{\Delta}_{\mathcal{G}} (\nu n : \mathcal{G}_n)P'$

Case $(\nu n : \mathcal{G}_n)P \xrightarrow{\alpha} (\nu n : \mathcal{G}_n)P'$ where $P \xrightarrow{\alpha} P'$ and $n(\alpha) \neq n$

It must be the case that $E, n : \mathcal{G}_n \vdash^{\Delta}_{\mathcal{G}} P$. Using induction we get $E, n : \mathcal{G}_n \vdash^{\Delta}_{\mathcal{G}} P'$ and using (E-New) we get $E \vdash^{\Delta}_{\mathcal{G}} (\nu n : \mathcal{G}_n)P'$
For the case where $\alpha = \mathit{kill}\, n$, $E(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ it must be the case that $\mathcal{G} \in \mathcal{K}$ and $m = \omega$

Case $n[Q \mid m[\textbf{out}\ n.P]] \overset{cap\,n}{\rightarrow} n[Q] \mid m[P]$

Starting with $E \vdash^{\Delta}_{\mathcal{G}} n[Q \mid m[\mathsf{out}\ n.P]]$, we get $E(n) = \mathcal{G}_n$ and $E \vdash^{\Delta}_{\mathcal{G}_n} Q \mid m[\mathsf{out}\ n.P]$, this was concluded using (E-Amb). Then using (E-Par) we get $E_1 \vdash^{\Delta}_{\mathcal{G}_n} Q$ and $E_2 \vdash^{\Delta}_{\mathcal{G}_n} m[\mathsf{out}\ n.P]$. Using (E-Amb) again we get $E_2 \vdash^{\Delta}_{\mathcal{G}_m} \mathsf{out}\ n.P$, then using E-Out we get $E(n) = \mathcal{G}_n$ and $E_2 \vdash^{\Delta}_{\mathcal{G}_m} P$ where $\Delta(\mathcal{G}_m) = (\mathcal{C}_m,\ \mathcal{K}_m, m_m), \Delta(\mathcal{G}_n) = (\mathcal{C}_n,\ \mathcal{K}_n, m_n)$ and $\forall \mathcal{G}_a \in \mathcal{C}_n, \Delta(\mathcal{G}_a) = (\mathcal{C}_a, \mathcal{K}_a, m_a),\ \mathcal{G}_m \in \mathcal{C}_a$ and $\mathcal{K}_a \subseteq \mathcal{C}_m$, where some $\mathcal{G}_a$ must be $\mathcal{G}$
Then using induction, Lemma 1 and (E-Par) we get our result $E \vdash^{\Delta}_{\mathcal{G}} n[Q] \mid m[P]$ where $E = E_1 \oplus E_2$

Case $m[\textbf{in}\ n.P] \mid n[Q] \overset{cap\,n}{\rightarrow} n[Q \mid m[P]]$

Starting with $E \vdash^{\Delta}_{\mathcal{G}} m[\mathsf{in}\ n.P] \mid n[Q]$, we get $E_1 \vdash^{\Delta}_{\mathcal{G}} m[\mathsf{in}\ n.P]\ E_2 \vdash^{\Delta}_{\mathcal{G}} n[Q]$ where $E = E_1 \oplus E_2$, this was concluded using (E-Par). Then using (E-Amb) we get $E(m) = \mathcal{G}_m$ and $E_1 \vdash^{\Delta}_{\mathcal{G}_m} \mathsf{in}\ n.P$. Using (E-In) we get $E(n) = \mathcal{G}_n$ and $E_1 \vdash^{\Delta}_{\mathcal{G}_m} P$ where $\Delta(\mathcal{G}) = (\mathcal{C}, \mathcal{K}, m)$ and $\Delta(\mathcal{G}_n) = (\mathcal{C}_n, \mathcal{K}_n, m_n), \mathcal{G}_n \in \mathcal{C}$ and $\mathcal{K}_n \subseteq \mathcal{C}$
Then using induction, Lemma 1 and (E-Amb) we get our result $E \vdash^{\Delta}_{\mathcal{G}} n[Q \mid m[P]]$

Case $\textbf{open}\ n.P \mid n[Q] \overset{kill\,n}{\rightarrow} Q \mid P$

It must be the case that there exist E_1 and E_2 such that $E = E_1 \oplus E_2$, $E_1 \vdash^{\Delta}_{\mathcal{G}} \mathsf{open}\ n.P$ and $E_2 \vdash^{\Delta}_{\mathcal{G}} n[Q]$. Using (E-Amb) we get $E_2(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ where it must be the case that $m \neq 0$ and $E_2 \vdash^{\Delta}_{\mathcal{G}_n} Q$. And using (E-Open) we get $E_1(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ where it must be the case that $\mathcal{G} \in \mathcal{K}$ and $m = \omega$ and $E_1 \vdash^{\Delta}_{\mathcal{G}} P$.
By using Lemma 1 we get $E_2 \vdash^{\Delta}_{\mathcal{G}} Q$ and by joining with $E_1 \vdash^{\Delta}_{\mathcal{G}} P$ we get $E \vdash^{\Delta}_{\mathcal{G}} P \mid Q$ where $E = E_1 \oplus E_2$

Case $\textbf{kill}\ n.P \mid n[Q] \overset{kill\,n}{\rightarrow} P$

It must be the case that there exist E_1 and E_2 such that $E = E_1 \oplus E_2$, $E_1 \vdash^{\Delta}_{\mathcal{G}} \mathsf{kill}\ n.P$ and $E_2 \vdash^{\Delta}_{\mathcal{G}} n[Q]$. Using (E-Kill) we get $E_1(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ where it must be the case that $\mathcal{G} \in \mathcal{K}$ and $m = \omega$ and $E_1 \vdash^{\Delta}_{\mathcal{G}} P$.
Using Lemma 2 we get $E \vdash^{\Delta}_{\mathcal{G}} P$ where $E = E_1 \oplus E_2$.

Case $!P \xrightarrow{1} !P \mid P$

Using (E-Rep) we get $E \vdash^{\Delta}_{\mathcal{G}} P$ where $\forall n \in n(P)$, $E(n) = \mathcal{G}_n$, $\Delta(\mathcal{G}_n) = (\mathcal{C}, \mathcal{K}, m)$ and $m = \omega$.
Using (E-Par) we get $E \vdash^{\Delta}_{\mathcal{G}} !P \mid P$

Case $P \xrightarrow{\alpha} P'$ where $P \equiv Q$, $Q \xrightarrow{\alpha} Q'$ and $Q' \equiv P'$

Since structural congruence preserves typing (Lemma 3), we have that $E \vdash^{\Delta}_{\mathcal{G}_a} Q$ and using induction we get $E \vdash^{\Delta}_{\mathcal{G}_b} Q'$. Using Lemma 3 a second time we get $E \vdash^{\Delta}_{\mathcal{G}_b} P'$ as expected.

Bibliography

[1] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[2] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the ambient calculus, 2001.

[3] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *In Proceedings of POPL'98*. ACM Press, 1998.

[4] Elio Giovannetti. Type inference for mobile ambients in prolog, 2004.

[5] Casper Jensen. Describing volunteer computing with departing and centralized nodes. January 2013.

[6] Cedric Lhoussaine. Type inference for a distributed [pi]-calculus. *Science of Computer Programming*, 50(1-3):225–251, March 2004.

[7] Hanne Riis Nielson and Flemming Nielson. Shape analysis for mobile ambients. In *In POPL'00*, pages 142–154. ACM Press, 2000.