Flow Logic is an approach to statically determining the behaviour of programs and processes.

It borrows methods and techniques from Abstract Interpretation, Data Flow Analysis and Constraint Based Analysis while presenting the analysis in a style more reminiscent of Type Systems. Traditionally developed for programming languages, this paper provides a tutorial development of the approach of Flow Logic for process calculi based on a decade of research.

We first develop a simple analysis for the $\pi$-calculus; this consists of the specification, semantic soundness (in the form of subject reduction and adequacy results), and a Moore Family result showing that a least solution always exists, as well as providing insights on how to implement the analysis. We then show how to strengthen the analysis technology by introducing reachability components, interaction points, and localised environments and, finally, we extend it to a relational analysis.

A Flow Logic is a program logic – in the same sense that a Hoare’s logic is. We conclude with an executive summary presenting the highlights of the approach from this perspective including a discussion of theoretical properties as well as implementation considerations.

The electronic supplements present an application of the analysis techniques to a version of the $\pi$-calculus incorporating distribution and code mobility; also the proofs of the main results can be found in the electronic supplements.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification – Formal Methods; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs – Assertions, Invariants, Logics of Programs, Mechanical Verification, Specification Techniques; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages – Operational Semantics, Process Models, Program Analysis; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic – Logic and Constraint Programming, Mechanical Theorem Proving; I.2.2 [Artificial Intelligence]: Automatic Programming – Program Verification

General Terms: Algorithms, Design, Documentation, Languages, Reliability, Theory, Verification

Additional Key Words and Phrases: Static Analysis, Flow Logic, Process Calculi, Moore Family, Subject Reduction, Adequacy

1. INTRODUCTION

Given a process in a process calculus, or a program in some programming language, one might be interested in knowing certain properties that will hold in all executions. Examples include knowing where values originate from, what are the
possible values that expressions might evaluate to, what are the combination of
values possible at a given program point.

Static analysis encompasses a number of approaches aiming at providing safe
approximations to the dynamic behaviour of the processes or programs of interest.
In [Nielson et al. 1999] we cover four of the main ones: Data Flow Analysis, Con-
straint Based Analysis, Abstract Interpretation, and Type and Effect Systems. All
of these approaches were originally developed for programming languages and then
subsequently some of these have been developed also for process calculi.

Flow Logic bridges the gap between these approaches and is a robust approach
able to deal with a wide variety of programming languages and calculi of com-
putation including calculi with functional, imperative, object-oriented, concurrent,
distributed and mobile features; we refer to the survey paper [Nielson and Nielson
2002] for an overview of Flow Logic for programming languages.

In short, Flow Logic uses the methods and techniques prevalent in Data Flow
Analysis, Constraint Based Analysis, and Abstract Interpretation while orchestrat-
ing the development using the mindset from Type Systems. Indeed, Flow Logic
makes a clear distinction between (i) the specification of the analysis, (ii) whether
or not a proposed analysis result is indeed sound with respect to the semantics, and
(iii) the computation of the best analysis result. It is a program logic in much the
same sense that a Hoare’s logic is; but it does not rely on a clear separation between
pre- and post-condition. The logical format used for presenting specifications fo-
cuses on ensuring the implementability of the analyses — often in low polynomial
(cubic) time.

Approaches based on type and effect systems (including session types) have a
long history of dealing with process calculi whereas the development of Flow Logic
for process calculi has happened during the last decade and is the subject of the
present paper. We introduce and highlight the basic concept of Flow Logic for
process calculi as developed in a decade of work and also provide concrete examples
embracing these concepts.

Overview. We introduce the basic approach of Flow Logic in Section 2. Here
we consider the polyadic π-calculus [Milner 1999] and develop a simple analysis
tracking whether there might be arity mismatches in communications. We illustrate
a number of applications of the analysis and also develop the relevant semantic
correctness results: a subject reduction result and a number of adequacy results. We
next prove the Moore Family result which ensures that “there is no non-determinism
in our analysis”, i.e. that there always is a least analysis result and we develop an
algorithm for finding it efficiently. We conclude with a few pointers to the literature;
the novice to Flow Logic might want to stop here on a first reading.

In Section 3 we demonstrate a number of analysis techniques for improving the
precision of the analysis. In Subsection 3.1 we add a reachability component where
we make sure not to analyse unreachable code, e.g. parts of processes following a
communication that has no matching counterpart. Next in Subsection 3.2 we show
how a careful analysis of the potential interaction points can improve the precision
of the analysis result; basically this is the technique we use to differentiate between
the parallel composition and the non-deterministic choice of processes. Then in
Subsection 3.3 we show how to make use of localised environments that are able to
differentiate between different binding occurrences of the same name. The analyses presented in this paper generally use a rather straightforward approach to name bindings, where their relationship is not taken into account, but in Subsection 3.4 we develop a more complex analysis able to take relations between name bindings into account. We conclude with a few pointers to the literature.

We raise the level of abstraction in Section 4 where we give a general characterisation of what a Flow Logic for a process calculus is. This section is not intended for the novice to Flow Logic and is not required for straightforward adaptations of Flow Logic to other calculi. It highlights the main points that have been developed in the previous sections and provide firm guidance on how to adapt Flow Logic specifications to more demanding scenarios without jeopardising the theoretical and practical properties that are the hallmark of Flow Logic.

We provide our concluding remarks in Section 5.

The paper is supplemented with two electronic appendices. Appendix A shows the ease with which the Flow Logic approach is able to deal with new linguistic primitives; this takes the form of developing a Flow Logic along the lines of Section 2 for the D\pi calculus [Hennessy 2007]. Finally, Appendix B contains the proofs of the main results.

2. THE BASICS OF FLOW LOGIC

In this section we introduce the basic approach of Flow Logic for the polyadic \pi-calculus [Milner 1999]. We first introduce the process calculus and then develop the Flow Logic specification and go on to demonstrate the main properties of the specification. Semantic soundness is established by means of a subject reduction result and several adequacy results for several applications of the analysis. We give a theoretical treatment of how to obtain a solution by establishing a Moore Family result (as briefly mentioned in the Introduction) and then we give a more pragmatic treatment by outlining how a solver might be constructed. We conclude with a few pointers to the literature.

2.1 The Process Calculus

Syntax. We shall consider the following version of the \pi-calculus where the syntax of processes and actions is given by:

\[
P \ ::= \ (\text{new } n)P \mid P_1 \mid P_2 \mid \! P \mid \Sigma_{i \in I} \pi_i . P_i
\]

\[
\pi \ ::= \ u(\vec{v}) \mid u(\vec{x}) \mid \tau
\]

As in [Milner 1999] processes \( P \in \text{Proc} \) can be constructed using process restriction \( (\text{new } n)P \), parallel composition \( P_1 \mid P_2 \), replication \( \! P \) and guarded sums \( \Sigma_{i \in I} \pi_i . P_i \). In its general form, the latter makes use of a finite index set \( I \); nullary sums are written \( 0 \), unary sums are written \( \pi . P \) whereas binary sums are written \( \pi_1 . P_1 + \pi_2 . P_2 \). There are three different kinds of actions \( \pi \in \text{Act} \), namely polyadic output \( u(\vec{v}) \), polyadic input \( u(\vec{x}) \) and silent actions \( \tau \). In examples we often dispense with writing the continuation process when it is \( 0 \).

A central component of the \pi-calculus is that of names and we shall use \( u, v \in \text{Name} \) to range over names. A bound name \( u \) introduced by a restriction \( (\text{new } u) \) will be called a constant and we generally use \( n, m \in \text{Const} \) to denote this. Similarly;
Abelian monoid laws for parallel:

\[(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)\]
\[P \parallel Q \equiv Q \parallel P\]
\[P \parallel 0 \equiv P\]

Unfolding of replication:
\[!P \equiv P \parallel !P\]

Summands can be permuted in \(\Sigma_{i \in I} \pi_i P_i\)

Scope laws:

\[(\text{new } n)(\text{new } m)P \equiv (\text{new } m)(\text{new } n)P\]
\[(\text{new } n)0 \equiv 0\]
\[(\text{new } n)(P \parallel Q) \equiv (\text{new } n)P \parallel Q \quad \text{if } n \notin \text{fc}(Q)\]

\[P \equiv Q \quad \text{if } P \equiv Q\]

Table I. Structural congruence: \(P \equiv Q\).

bound name \(u\) introduced by an input \(v(\bar{u})\) will be called a variable and we generally use \(x, y \in \text{Var}\) to denote this. This distinction between constants and variables is common in programming languages and will be a useful guide when explaining the detailed operation of the Flow Logic; clearly it does not limit the expressivity of the process calculus.

In contrast to [Milner 1999] we shall find it helpful for our presentation to formally partition the name space \(\text{Name}\) into the disjoint union of \(\text{Const}\) and \(\text{Var}\), both of which provide infinite supplies of names that are constants resp. variables, and to incorporate this into the syntax as shown above; to be pedantic, \(\text{Name} = \text{Const} \cup \text{Var}\) and \(\text{Const} \cap \text{Var} = \emptyset\). We use vector notation for sequences of such entities and write \(|\bar{u}|\) for the length of the sequence \(\bar{u}\). Finally, \(\text{fc}(P)\) and \(\text{fv}(P)\) denote the free constants, resp. free variables, of \(P\); the definitions are straightforward and therefore omitted. We shall write \(P\) for the main process of interest.

\[\text{Semantics.}\] We shall follow the approach of [Milner 1999] and define the semantics by a structural congruence relation \(\equiv\) and a reduction relation \(\rightarrow\). The structural congruence is defined by the laws in Table I together with the classical laws for an equivalence relation and the laws for replacement in context. The definition makes use of a notion of \(\alpha\)-renaming for names:

\[(\text{new } n)P \equiv_{\alpha} (\text{new } m)P[m/n] \quad \text{if } m \notin \text{fc}(P)\]

Here \(P[u/v]\) is a process that is as \(P\) except that all free occurrences of the name \(v\) has been replaced by the name \(u\) while avoiding name capture. We shall later replace this notion of \(\alpha\)-renaming with a notion of disciplined \(\alpha\)-renaming that, without loss of generality, disciplines the choice of fresh names.

The reduction relation is specified in Table II. The axiom for communication checks whether the arities of the input and output match and it makes use of the substitution operation \(P[\bar{u}/\bar{v}]\) that is the generalisation of the operation \(P[u/v]\) explained above obtained by taking \(P[\bar{u}/\bar{v}] = (P[u/v])[\bar{u}/\bar{v}]\) for non-empty sequences. As usual we have rules for reduction in the context of restriction and parallel composition and we have a rule incorporating the structural congruence. We refer to [Milner 1999] for additional explanations.

Example 2.1. Consider the following system \(P\), consisting of three parallel processes exchanging messages using a global channel \(c\):

\[(\text{new } n)(c(n).c(n)) \parallel c(x).((\text{new } m)x(m)) \parallel c(y).y(x)\]

Let us assume that the two leftmost processes communicate in the first reduction.
\[(n\langle \vec{m} \rangle.P + P') \mid (n\langle \vec{x} \rangle.Q + Q') \rightarrow P \mid Q[\vec{m}/\vec{x}] \text{ if } |\vec{m}| = |\vec{x}|\]

\[
\begin{align*}
\text{[COM]} & \quad (n\langle \vec{m} \rangle.P + P') \mid (n\langle \vec{x} \rangle.Q + Q') \rightarrow P \mid Q[\vec{m}/\vec{x}] \\
\text{[RES]} & \quad P \rightarrow P' \\
\text{[EQ]} & \quad P \equiv Q \rightarrow Q' \equiv P' \\
\text{[PAR]} & \quad P \mid Q \rightarrow P' \mid Q
\end{align*}
\]

Table II. Reduction relation: \( P \rightarrow P' \).

step; then we get

\[
\begin{align*}
\text{(new } n) & \quad (c\langle n \rangle \mid (\text{new } m)n\langle m \rangle) \mid c(y).y(x)
\end{align*}
\]

reflecting not only that \( n \) has been substituted for \( x \) in the second process but also that the scope of \( n \) has been extended to include both processes. The leftmost and the rightmost processes communicate in the second reduction step thereby substituting \( n \) for \( y \) in the rightmost process:

\[
\begin{align*}
\text{(new } n) & \quad (0 \mid (\text{new } m)n\langle m \rangle) \mid n(x)
\end{align*}
\]

Finally, the two rightmost processes can communicate over the channel \( n \) and we are done.

A variant of the process called \( P'_* \) has a slightly different rightmost process:

\[
\begin{align*}
\text{(new } n) & \quad (c\langle n \rangle, c\langle n \rangle) \mid c(x).((\text{new } m)x\langle m \rangle) \mid c(x,y).x(y)
\end{align*}
\]

After the initial communication over \( c \) it becomes

\[
\begin{align*}
\text{(new } n) & \quad (c\langle n \rangle \mid (\text{new } m)n\langle m \rangle) \mid c(x,y).x(y)
\end{align*}
\]

but now the process is stuck: the communication over \( c \) is not possible since the arities of the output and input do no match. \( \Box \)

Our choice of semantics restricts our attention to closed processes, that is, processes with no free names and hence processes that cannot interact with an unknown environment – thus the main process \( P_* \) of interest will satisfy \( \text{fv}(P_*) = \text{fc}(P_*) = \emptyset \).

This will simplify the presentation in the rest of the paper but it is in no way inherent to our approach; indeed in previous work (e.g. [Bodei et al. 2001a; 1998; 1999]) we have performed similar developments based on the late semantics of [Parrow 2001] thereby catering for open processes as well.

### 2.2 The Flow Logic Specification

**Abstract domains.** The overall aim of our analysis is to determine whether the process \( P_* \) of interest might enter a stuck configuration because of a mismatch between the arities in a communication step. In order to do so the analysis will capture which sequences of constants that might be communicated over the various channels – and in order to do that it will also need to track the bindings of constants to variables. This motivates introducing the following abstract domains:

\[-\rho : \text{Var} \rightarrow \wp(\text{Const}) \text{ is the abstract environment that maps a variable to the set of constants that it might be bound to.}\]

\[-\kappa : \text{Const} \rightarrow \wp(\text{Const}^*) \text{ is the abstract channel environment that maps a (channel) constant to the set of sequences of constants that may be communicated over it.}\]

Hanne Riis Nielson et al

$\psi : \wp(\text{Const})$ is the error component that records the set of (channel) constants where there may be an arity mismatch in a communication.

These domains can be turned into complete lattices by extending the subset ordering $\subseteq$ on $\wp(\text{Const})$ in a point-wise manner. For ease of presentation we extend the abstract environment to $\rho : (\text{Const} \cup \text{Var}) \rightarrow \wp(\text{Const})$ by setting $\rho(n) = \{n\}$ for all constants $n$; we prefer this to simply setting $\rho : \text{Name} \rightarrow \wp(\text{Name})$ as this would allow having e.g. $\rho(m) = \{x\}$ for a name $m$ and variable $x$ which is semantically impossible. Also we extend $\rho$ to sequences $\vec{u} = (u_1, \ldots, u_k)$ of constants and variables in a point-wise manner so $\rho(\vec{u}) = \rho(u_1) \times \cdots \times \rho(u_k) = \{(m_1, \ldots, m_k) \mid \land_{i=1}^k m_i \in \rho(u_i)\}$.

Judgements. In Flow Logic the analysis is specified by logical judgements and in our analysis of the $\pi$-calculus they take the form:

$$\rho, \kappa \vdash P : \psi$$

This judgement expresses that $\rho$, $\kappa$ and $\psi$ provide a valid analysis result for the behaviour of $P$ – note that the judgement does not say that it is the best such analysis result nor does it give any guidelines for how to compute the analysis result itself.

The idea is that $\rho$ and $\kappa$ contain global information that captures information about the behaviour of the overall process $P$, of interest whereas $\psi$ captures local information of interest primarily for the process $P$. It is important to notice that the analysis result captures information about $P$ as well as the potential processes it may evolve into.

The judgement is defined by a number of clauses: the first four clauses of Table III (to be explained below). We have exactly one clause for each of the syntactic forms of processes in the $\pi$-calculus. In the case of guarded sums we shall make use of a judgement defined for actions. It happens to have the same form as that for processes

$$\rho, \kappa \vdash_A \pi : \psi$$

and expresses that $\rho$, $\kappa$ and $\psi$ together form a valid analysis result for the action $\pi$. Again we have exactly one clause for each of the syntactic forms of actions: the last three clauses of Table III (to be explained below).

Clauses. We shall now explain the form of the individual clauses of Table III. We begin with the clause $\text{[RES]}$ for $(\text{new } n)P$: it expresses that $\rho$, $\kappa$ and $\psi$ constitute a valid analysis result for $(\text{new } n)P$ if and only if they do so for $P$ alone; thus the clause (and hence the analysis) is oblivious to the scoping of constants.

The clause $\text{[PAR]}$ for parallel composition states that $\rho$, $\kappa$ and $\psi$ constitute a valid analysis result for $P_1 \mid P_2$ whenever we can find valid analysis judgements for $P_1$ and $P_2$ using the same global components $\rho$ and $\kappa$ but possibly different local components $\psi_1$ and $\psi_2$; however, it must be the case that $\psi_1 \cup \psi_2 \subseteq \psi$ reflecting that arity mismatches recorded for $P_1$ and $P_2$ also should be recorded for $P_1 \mid P_2$.

Turning to the clause $\text{[REP]}$ for replication we immediately observe that it is oblivious to the replication operator. For simple analyses, like checking for potential mismatches of arities in communications, this is quite sufficient.
[RES] \( \rho, \kappa \vdash \rho \uparrow \sum \) \( \text{new } n \) \( P : \psi \) \iff \( \rho, \kappa \vdash P : \psi \)

[PAR] \( \rho, \kappa \vdash P_1 \mid P_2 : \psi \) \iff \( \rho, \kappa \vdash P_1 : \psi_1 \land \rho, \kappa \vdash P_2 : \psi_2 \land \psi_1 \cup \psi_2 \subseteq \psi \)

[REP] \( \rho, \kappa \vdash !P : \psi \) \iff \( \rho, \kappa \vdash P : \psi \)

[SUM] \( \rho, \kappa \vdash \sum \pi \in I \pi_i : \psi \) \iff \( \forall i \in I : ( \rho, \kappa \vdash \sum \pi_i : \psi_i \land \psi_i \subseteq \psi \lor \rho, \kappa \vdash P_i : \psi_i \land \psi_i \subseteq \psi ) \)

[OUT] \( \rho, \kappa \vdash A \mathop{out} \psi \) \iff \( \forall n \in \rho(u) : \rho(n) \subseteq \kappa(n) \)

[IN] \( \rho, \kappa \vdash A \mathop{in} \psi \) \iff \( \forall n \in \rho(u) : \kappa(n) \cap \text{Const}[\vec{x}] \subseteq \rho(\vec{x}) \land \kappa(n) \setminus \text{Const}[\vec{x}] 

\neq \emptyset \Rightarrow n \in \psi \)

[Tau] \( \rho, \kappa \vdash \tau : \psi \) \iff \( \text{true} \)

Table III. Flow Logic: \( \rho, \kappa \vdash P : \psi \) and \( \rho, \kappa \vdash \pi : \psi \).

The fourth clause [SUM] is for the guarded sum and let us first take a look at unary sums where the clause specialises to:

\( \rho, \kappa \vdash \pi \uparrow \psi \) \iff \( \rho, \kappa \vdash \pi : \psi' \land \psi' \subseteq \psi \land \rho, \kappa \vdash P : \psi'' \land \psi'' \subseteq \psi \)

This expresses that \( \rho \) and \( \kappa \) must be valid analysis results for the action \( \pi \) as well as its continuation \( P \) and that the corresponding error components must be included in the error component associated with \( \pi \uparrow \psi \). The clause for \( \sum \pi_i. P_i \) of Table III generalises this to hold for each of the summands \( \pi_i. P_i \).

The remaining clauses of Table III define the judgement for actions and it is in these clauses that we impose conditions on the information contained in \( \rho, \kappa \) and \( \psi \). Let us consider the clause [OUT] for the output action \( u(\vec{v}) \). In the case where \( u \) as well as \( \vec{v} \) are constants it amounts to

\( \rho, \kappa \vdash A \mathop{out} \psi \) \iff \( \vec{m} \in \kappa(n) \)

and this simply mimics that \( \vec{m} \) might be communicated over the channel \( n \) and hence this must be recorded in \( \kappa \). Now \( u \) as well as \( \vec{v} \) could contain variables so the right hand side of the clause has to be generalised and here we use \( \rho \) to capture all the possibilities as expressed by \( \forall n \in \rho(u) : \rho(n) \subseteq \kappa(n) \). Note that the clause does not impose any conditions on the error component \( \psi \) as the output action on its own cannot reveal any arity mismatch in communication.

To explain the clause [IN] for the input action \( u(\vec{x}) \) let us first consider the case where \( u \) is a constant; here the clause specialises to:

\( \rho, \kappa \vdash A \mathop{in} \psi \) \iff \( \kappa(n) \cap \text{Const}[\vec{x}] \subseteq \rho(\vec{x}) \land \kappa(n) \setminus \text{Const}[\vec{x}] 

\neq \emptyset \Rightarrow n \in \psi \)

The set \( \kappa(n) \) contains all the sequences of constants \( \vec{m} \) that might be output to \( n \) somewhere in the process, and, from the point of view of the analysis, any of them could be input over \( n \) at this particular point in the process. Provided that \( \vec{m} \) and \( \vec{x} \) have the same length, the \( i \)’th constant of \( \vec{m} \) will be a potential value of the \( i \)’th variable of \( \vec{x} \); this is expressed by the condition

\( \kappa(n) \cap \text{Const}[\vec{x}] \subseteq \rho(\vec{x}) \)
and assuming that $\vec{x} = (x_1, \ldots, x_k)$ this amounts to:

$$\forall (m_1, \ldots, m_k) \in \kappa(n) \cap \text{Const}^k : m_1 \in \rho(x_1) \land \cdots \land m_k \in \rho(x_k)$$

However, $\kappa(n)$ might also contain sequences with lengths different from $|\vec{x}|$ and then

the communication might fail and we shall then require that $n \notin \psi$; this is captured

by the condition $\kappa(n) \setminus \text{Const}^{|\vec{x}|} \neq \emptyset \Rightarrow n \in \psi$. The clause of Table III generalises these requirements to the case where $u \in \text{Const} \cup \text{Var}$.

**Example 2.2.** Let us return to the process $P_\star$ of Example 2.1. Taking $\rho$, $\kappa$ and $\psi$ as follows

$$\rho = \begin{bmatrix} x & y \\ \{n,m\} & \{n\} \end{bmatrix} \quad \kappa = \begin{bmatrix} c & n & m \\ \{n\} & \{m\} & \{m\} \end{bmatrix} \quad \psi = \emptyset$$

we obtain an acceptable analysis result as can easily be verified from the clauses of Table III. This is the most precise analysis result for $P_\star$ and we can see that it correctly captures that $x$ may be bound to $n$ as well as $m$. However, it fails to notice that $m$ will not be used for communication and obviously this is because the two defining occurrences of $x$ are mixed up. It is easy to check that we can obtain a more precise analysis result if we avoid this and replace $y(x)$ by, say, $y(z)$ in the rightmost process.

For the process $P'_\star$ of Example 2.1 we have the following acceptable analysis result

$$\rho = \begin{bmatrix} x & y \\ \{n\} & \{m\} \end{bmatrix} \quad \kappa = \begin{bmatrix} c & n & m \\ \{n\} & \{m\} & \emptyset \end{bmatrix} \quad \psi = \{c\}$$

reflecting that the process indeed will become stuck when trying to communicate over $c$. \(\square\)

It is characteristic for Flow Logic that there is exactly one clause for each syntactic construct. Thus we would never introduce a general “subtyping” clause stating for example that it is always possible to enlarge the error component of an analysis result. Such results, however, can often be established as formal consequences of the specification:

**Fact 2.3 Monotonicity.** If $\rho, \kappa \vdash P : \psi$ and $\psi \subseteq \psi'$ then $\rho, \kappa \vdash P : \psi'$.

**Well-definedness.** It is easy to argue that the analysis specified by Table III is well-defined as it is syntax-directed: the validity of an analysis result for a composite construct always relies on the validity of analysis results for syntactically smaller constructs.

### 2.3 The Subject Reduction Result

Semantic correctness amounts to ensuring that the judgement $\rho, \kappa \vdash P : \psi$ correctly captures the behaviour of the process $P$. We shall express this as a subject reduction result meaning that if we have an analysis result for $P$, and $P$ evolves into some process $Q$, then the very same analysis result is also valid for $Q$.

Now if $P$ evolves into $Q$ then it also evolves into any process $Q'$ that is structurally congruent to $Q$. In particular, this means that an analysis result that is valid for $P$ also must be valid for $Q$ and all processes that are obtained from $Q$ by $\alpha$-renaming. However, the analysis results $\rho$, $\kappa$ and $\psi$ are closely tailored to the constants and
variables occurring in $P$ and as a consequence it is impossible to establish the required connection.

The solution to this is to introduce equivalence classes of constants and make use of **disciplined $\alpha$-renaming** as expressed by:

$$(\text{new } n)P \equiv_\alpha (\text{new } m)P[m/n] \quad \text{if } m \notin \text{fc}(P) \land \lfloor n \rfloor = \lfloor m \rfloor$$

Here $\lfloor n \rfloor$ is the equivalence class corresponding to $n$; it is also called the **canonical name** of $n$ and is similar to the **sorts** of [Milner 1999]. Clearly, we assume that there is an infinite supply of names within each equivalence class. Without loss of generality, we shall in the following replace the law for $\alpha$-renaming in Table I with the above rule for disciplined $\alpha$-renaming.

The clauses specifying the analysis judgements refer to the constants of the initial process $P_\star$ of interest. We shall therefore take $\lfloor n \rfloor = n$ for all constants $n$ occurring in $P_\star$. By taking $\lfloor x \rfloor = x$ for all variables $x$, we can lift the operation $\lfloor \cdot \rfloor$ to actions and processes in a point-wise manner. Clearly, for the initial process we will then have $P_\star = \lfloor P_\star \rfloor$ whereas this does not necessarily hold for the derivatives of $P_\star$.

We can now establish the following auxiliary results:

**Lemma 2.4 Structural Congruence.** If $P \equiv Q$ then $\rho, \kappa \vdash [P] : \psi$ if and only if $\rho, \kappa \vdash [Q] : \psi$.

**Lemma 2.5 Substitution.** If $\rho, \kappa \vdash [P] : \psi$ then $\rho, \kappa \vdash [P[m/y]] : \psi$ provided that $\lfloor m \rfloor \in \rho(y)$.

We are now ready to formalise and prove the subject reduction result:

**Theorem 2.6 Subject Reduction.** If $P \rightarrow Q$ and $\rho, \kappa \vdash [P] : \psi$ then $\rho, \kappa \vdash [Q] : \psi$.

### 2.4 Adequacy Results

We shall now give a few applications of the analysis developed above and present the associated adequacy results. We already noticed that there are processes that are **stuck** because the arities of the actions in a communication do not match and we now want to exclude them from consideration.

**Well-behaved processes.** To formalise this we shall first define a notion of **dynamic well-behavedness** based on the semantics and then a notion of **static well-behavedness** based on the Flow Logic. Our adequacy result will then show that static well-behavedness implies dynamic well-behavedness.

To express dynamic well-behavedness we shall first introduce the standard concept of a context; a **context** $C$ is formally given by

$$C ::= (\text{new } n)C \mid C \mid P \mid [\cdot]$$

and essentially is a process with a “hole” $[\cdot]$ where another process can be inserted; we write $C[P]$ when for the process obtained by inserting $P$ in the hole of $C$. (Unlike substitution there is no $\alpha$-renaming to avoid name capture when “plugging the hole” in a context.)

We shall say that the process $P_\star$ is **dynamically well-behaved** if whenever $P_\star$ evolves into a process that is structurally congruent to one of the form $C[(n\langle \bar{m} \rangle).R + \cdots]$. 

---

Adequacy for well-behaved processes

If the process \( P_* \) is \textit{statically well-behaved} if there exist \( \rho \) and \( \kappa \) such that \( \rho, \kappa \vdash P_* : \emptyset \); that is, there is a valid analysis result for \( P_* \) where the error component \( \psi \) is empty. We now have:

\textbf{Theorem 2.7 Adequacy for \textit{well-behaved processes}.} If the process \( P_* \) is \textit{statically well-behaved} then it is also \textit{dynamically well-behaved}.

\textit{Well-sorted processes.} We may go one step further and introduce a notion of well-sorted programs as suggested in [Milner 1999]. Also in this case our analysis gives the necessary information for statically imposing well-sortedness on processes. Our approach is similar to above: first we define a notion of dynamic well-sortedness based on the semantics, then we define a notion of static well-sortedness based on the analysis and finally we prove an adequacy result expressing that the static notion of well-sortedness implies the dynamic one.

Following Milner we shall first introduce a set of \textit{sorts} and require that each constant \( n \) has a sort \( \sigma(n) \in \text{Sort} \). The sorts are preserved by disciplined \( \alpha \)-renaming so if \( |n| = |m| \) then also \( \sigma(n) = \sigma(m) \). We extend \( \sigma \) in a component-wise manner to work on tuples as well as sets of tuples. A \textit{sorting} is then defined as a function \( \Sigma : \text{Sort} \rightarrow \text{Sort}^* \) that for each sort specifies a sequence of sorts describing not only the arity of the constants with that sort but also the sorts of the constants that may be communicated over it.

We shall now define that \( P_* \) is \textit{dynamically well-sorted} if whenever \( P_* \) evolves into a process that is structurally congruent to one of the form \( C[(n(\vec{m}).R + R') | (n(\vec{x}).Q + Q')] \) then \( |\vec{m}| = |\vec{x}| \) as well as \( \Sigma(\sigma(n)) = \sigma(\vec{m}) \).

We shall say that the process \( P_* \) is \textit{statically well-sorted} if there exist \( \rho \) and \( \kappa \) such that \( \rho, \kappa \vdash P_* : \emptyset \) and furthermore \( \sigma(\kappa(n)) \subseteq \{ \Sigma(\sigma(n)) \} \) for all constants \( n \). We then have:

\textbf{Theorem 2.8 Adequacy for \textit{well-sorted processes}.} If the process \( P_* \) is \textit{statically well-sorted} then it is also \textit{dynamically well-sorted}.

\textit{Non-leaking processes.} As a final example we shall assign \textit{security levels} to the channels and use the analysis to impose a confidentiality policy on the processes ensuring that they do not send constants with a high security level on channels with a lower security level; this is along the line of the development of [Bodei et al. 2001a; 1998; 1999]. For simplicity we shall only impose two levels, \textit{low} and \textit{high}, and we want to ensure that only \textit{low} information is communicated on \textit{low} channels.

The idea is to introduce a mapping \( \Lambda : \text{Const} \rightarrow \text{LEVEL} \) that to each constant associates a security level; we shall require that \( |n| = |m| \) ensures that \( \Lambda(n) = \Lambda(m) \) so that the security level will be preserved by disciplined \( \alpha \)-renaming. Let us write \( \text{LEVEL} = \{ \text{low}, \text{high} \} \) and define an ordering \( \subseteq \) by \( \text{low} \subseteq \text{high} \).

Following the above approach we shall say that the process \( P_* \) is \textit{dynamically non-leaking} if whenever \( P_* \) evolves into a process that is structurally congruent to one of the form \( C[(n(\vec{m}).R + R') | (n(\vec{x}).Q + Q')] \) then \( |\vec{m}| = |\vec{x}| \) as well as \( \Lambda(\vec{m}) \subseteq \Lambda(\vec{n})^{\mid \vec{m} \mid} \) – in the case where \( \Lambda(n) = \text{low} \) this ensures that all the constants of \( \vec{m} \) will be of \textit{low} security level and when \( \Lambda(n) = \text{high} \) it does not impose any restrictions.
Flow Logic for Process Calculi

INPUT: a Flow Logic specification \( F \) (e.g. the one in Table III) and a process \( P \)
OUTPUT: a logical formula \( \Phi \) such that \( (\rho,\kappa,\psi) \models \Phi \) iff \( \rho,\kappa \vdash P : \psi \)
METHOD: initialise \( \Phi \) to be \( \rho,\kappa \vdash P : \psi \)
while \( \Phi \) contains an occurrence of \( \vec{R} \vdash_{P'} P' : \vec{T} \)
and there is a clause in \( F \) of the form \( \alpha \iff \beta \)
and a substitution \( \theta \) such that \( \theta\alpha \) equals the occurrence \( \vec{R} \vdash_{P'} P' : \vec{T} \)
do replace the occurrence of \( \vec{R} \vdash_{P'} P' : \vec{T} \) with \( \theta\beta \) in \( \Phi \)

Table IV. Unfolding of the Flow Logic Specification.

We shall say that the process \( P \) is statically non-leaking if there exist \( \rho \) and \( \kappa \) such that \( \rho,\kappa \vdash P : \emptyset \) and furthermore \( \kappa \) satisfies \( \forall \vec{m} \in \kappa(n) : \Lambda(\vec{m}) \sqsubseteq \Lambda(n)^{|\vec{m}|} \) for all constants \( n \). We then have:

**Theorem 2.9 Adequacy for non-leaking processes.** If the process \( P \) is statically non-leaking then it is also dynamically non-leaking.

2.5 The Moore Family Result

The theorems presented so far have been concerned with interpreting the meaning of a valid analysis result for a process \( P \) in relation to its semantics and some properties of interest. However, can we be sure that \( P \) has any valid analysis results at all? And if it has more than one valid analysis result, is there a “best” one? These questions will be answered in the affirmative by the Moore Family result, also known as the model intersection property.

Formally, a Moore Family is a subset \( \mathcal{Y} \) of a complete lattice that is closed under greatest lower bounds, that is, it satisfies \( \forall \mathcal{Y} \subseteq \mathcal{Y} : \cap \mathcal{Y} \in \mathcal{Y} \). We have:

**Proposition 2.10 Moore Family.** The set \( \{(\rho,\kappa,\psi) \mid \rho,\kappa \vdash P : \psi\} \) is a Moore Family for all processes \( P \).

This result has the interesting corollary that all processes have a least, or best, analysis result. This follows by observing that the set

\[
\{(\rho,\kappa,\psi) \mid \rho,\kappa \vdash P : \psi\}
\]

is indeed a subset of the Moore Family and therefore \( \cap \{(\rho,\kappa,\psi) \mid \rho,\kappa \vdash P : \psi\} \) will be a valid analysis result for \( P \) and clearly it is the least such analysis result for \( P \).

2.6 Implementation

The Moore Family result shows the existence of a least analysis result. To obtain an algorithm for constructing it we merely need to change our viewpoint.

Given a process \( P \) and the Flow Logic specification in Table III we can unfold the defining clauses in Table III according to the syntax of \( P \) to obtain a formula \( \Phi \) such that the judgement \( \rho,\kappa \vdash P : \psi \) is equivalent to \( \Phi \). As a simple example,
in the case where \( P \) is \( c(n) \cdot 0 \mid c(x) \cdot 0 \) the formula \( \Phi \) becomes
\[
\{\forall c' \in \rho(c) : \rho(n) \subseteq \kappa(c')\} \land \\
\{\forall c' \in \rho(c) : \kappa(c') \cap \text{Const} \subseteq \rho(x)\} \land \\
\{\forall c' \in \rho(c) : \kappa(c') \setminus \text{Const} \neq \emptyset \Rightarrow c' \in \psi\}
\]
which can be further simplified to
\[
\{\{n\} \subseteq \kappa(c)\} \land \\
\{\kappa(c) \cap \text{Const} \subseteq \rho(x)\} \land \\
\{\kappa(c) \setminus \text{Const} \neq \emptyset \Rightarrow c' \in \psi\}
\]
using our extension of \( \rho \) to constants, e.g. \( \rho(c) = \{c\} \). Next we can solve the “constraints” generated by computing the least values of the predicates used in \( \Phi \) such that \( \Phi \) holds. Continuing the example, we need to compute the least values of \( \kappa(x) \) and \( \psi \) that satisfy the constraints; clearly this gives \( \kappa(x) = \{n\} \) and \( \psi = \emptyset \).

Taking a somewhat more general approach, the acceptability judgement
\[
\rho, \kappa \vdash_P P : \psi \iff \Phi
\]
associates each process \( P \) with a logical formula \( \Phi \), such that an analysis estimate \( (\rho, \kappa, \psi) \) is acceptable for \( P \) if and only if \( (\rho, \kappa, \psi) \) constitutes a model of \( \Phi \). This suggests a two-step algorithm. First we unfold the Flow Logic specification for a given process \( P^* \) into a formula \( \Phi^* \) of an appropriate formal system. Next we compute the least model \( (\rho, \kappa, \psi) \) of \( \Phi^* \) using an appropriate solver for the formal system. We now sketch the development of this two-step algorithm.

**Unfolding the Flow Logic specification.** Algorithmically we perform the unfolding by the iterative algorithm outlined in Table IV. It takes as input a Flow Logic specification \( F \) (e.g. the one in Table III) and produces the formula \( \Phi^* \) by repeated replacement of instances of lefthand sides with instances of righthand sides; this is formalised in the algorithm by the explicit use of substitutions.

The algorithm can be made deterministic by fixing a strategy for finding candidate judgements for unfolding, e.g. choosing the first occurrence in a depth-first traversal of the logical formula, but the details will be of no importance for us because the resulting formula can be freely rearranged using, for example, the commutative and associative laws for conjunction. Termination of the algorithm, and the finiteness of the logical formula \( \Phi^* \), follows directly from the finiteness of the syntactic representation of \( P^* \) whenever the Flow Logic is syntax-directed as is the case for the one in Table III. (More complex techniques are necessary when this is not the case.)

**Example 2.11.** Continuing Example 2.1 the algorithm of Table IV can be used to unfold the judgement \( \rho, \kappa \vdash_P P^* : \psi \) into a conjunction of the following six judgements for actions and eight set inclusions:
\[
\begin{align*}
\rho, \kappa \vdash_A c(n) : \psi_{11} & \quad \rho, \kappa \vdash_A c(x) : \psi_{211} & \quad \rho, \kappa \vdash_A c(x, y) : \psi_{221} \\
\rho, \kappa \vdash_A c(n) : \psi_{12} & \quad \rho, \kappa \vdash_A x(m) : \psi_{212} & \quad \rho, \kappa \vdash_A x(y) : \psi_{222} \\
\psi_1 \cup \psi_2 \subseteq \psi & \quad \psi_{11} \subseteq \psi_1 & \quad \psi_{211} \subseteq \psi_{21} & \quad \psi_{221} \subseteq \psi_{22} \\
\psi_{21} \cup \psi_{22} \subseteq \psi & \quad \psi_{12} \subseteq \psi_1 & \quad \psi_{212} \subseteq \psi_{21} & \quad \psi_{222} \subseteq \psi_{22}
\end{align*}
\]
The six judgements on actions can be further expanded using a combination of set inclusions and first order logic while leaving the eight set inclusions unchanged:

\[ \forall n' \in \rho(c) : \rho(n) \subseteq \kappa(n') \]
\[ \forall n' \in \rho(c) : \rho(n) \subseteq \kappa(n') \]
\[ \forall n' \in \rho(c) : \kappa(n') \cap \text{Const} \subseteq \rho(x) \land (\kappa(n') \setminus \text{Const} \neq \emptyset) \Rightarrow n' \in \psi_{211} \]
\[ \forall n' \in \rho(x) : \rho(m) \subseteq \kappa(n') \]
\[ \forall n' \in \rho(c) : \kappa(n') \cap \text{Const}^2 \subseteq \rho(x,y) \land (\kappa(n') \setminus \text{Const}^2 \neq \emptyset) \Rightarrow n' \in \psi_{221} \]
\[ \forall n' \in \rho(x) : \kappa(n') \cap \text{Const} \subseteq \rho(y) \land (\kappa(n') \setminus \text{Const} \neq \emptyset) \Rightarrow n' \in \psi_{222} \]
\[ \psi_1 \cup \psi_2 \subseteq \psi \]
\[ \psi_1 \cup \psi_2 \subseteq \psi \]
\[ \psi_2 \cup \psi_2 \subseteq \psi_2 \]
\[ \psi_2 \cup \psi_2 \subseteq \psi_2 \]

The formula \( \Phi_\ast \) is the conjunction of these constraints.

**Solving the constraints.** Having obtained the logical formula \( \Phi_\ast \), the next problem is to find its least (or best) model \((\rho, \kappa, \psi)\). One obvious approach is to develop a special purpose constraint solver handling exactly the combination of set inclusions and first order logic that we have used; essentially this is the approach taken in Table 3.7 of [Nielson et al. 1999]. An alternative, and more general approach, is to transform the constraints into an appropriate fragment of first order logic and then use an off-the-shelf solver. We shall illustrate this approach in the following example where we recast the constraints within Datalog [Chandra and Harel 1980; Apt et al. 1988] and then use The Succinct Solver [Nielson et al. 2002b] for computing the least solution – we shall discuss this approach in more detail in Subsection 4.3.

**Example 2.12.** Returning to the constraints of Example 2.11 the idea is that each of the analysis components \( \rho, \kappa \) and \( \psi \) are turned into predicates and the constraints are then rephrased using first order logic. Let us first consider the eight subset inclusions. They take one of two forms as exemplified by \( \psi_1 \subseteq \psi_1 \) and \( \psi_1 \cup \psi_2 \subseteq \psi_1 \) and they will be rewritten as:

\[ \forall n' : \psi_{11}(n') \Rightarrow \psi_1(n') \]
\[ \forall n' : \psi_1(n') \lor \psi_2(n') \Rightarrow \psi(n') \]

Next consider the constraints generated for the three output actions. A typical case is \( \forall n' \in \rho(x) : \rho(m) \subseteq \kappa(n') \) that becomes:

\[ \forall n' : \rho(x, n') \Rightarrow (\forall m' : \rho(m, m') \Rightarrow \kappa(n', m')) \]

Finally consider the constraints generated for the three input actions. Here we exploit that a simple inspection of the process \( P'_s \) shows that communication only occurs with arity one and two. First let us consider one of the unary input actions, namely \( \forall n' \in \rho(x) : (\kappa(n') \cap \text{Const} \subseteq \rho(y)) \land ((\kappa(n') \setminus \text{Const} \neq \emptyset) \Rightarrow n' \in \psi_{222}) \).

Here we re write the constraint as follows:

\[ \forall n' : \rho(x, n') \Rightarrow (\forall m' : \kappa(n', m') \Rightarrow \rho(y, m')) \land (\exists m'_1 : \exists m'_2 : \kappa(n', m'_1, m'_2) \Rightarrow \psi_{222}(n')) \]

For the binary input action we have \( \forall n' \in \rho(c) : \kappa(n') \cap \text{Const}^2 \subseteq \rho(x,y) \land (\kappa(n') \setminus \text{Const}^2 \neq \emptyset) \Rightarrow n' \in \psi_{221} \) and we rewrite it as

\[ \forall n' : \rho(c, n') \Rightarrow (\forall m'_1 \cap \forall m'_2 : \kappa(n', m'_1, m'_2) \Rightarrow \rho(x, m'_1) \land \rho(y, m'_2)) \land (\exists m' : \kappa(n', m') \Rightarrow \psi_{221}(n')) \]
The conjunction of the formulae obtained in this way is in Datalog and the Succinct Solver [Nielson et al. 2002b] will produce the solution already presented in Example 2.2.

2.7 Bibliographical Notes

As discussed in the Introduction, the development of Flow Logic for process calculi involves transferring methods and techniques from Data Flow Analysis, Constraint Based Analysis and Abstract Interpretation from the world of programming languages to the world of process calculi. The differences between programming languages and process calculi have presented a few obstacles that have not always been solved as elegantly as presented in the present paper. We now give an overview of some of these issues.

Names versus constants and variables. The distinction between constants and variables is quite standard in programming languages but is absent in many process calculi. We do consider it useful when developing the static analysis although it is not formally necessary to do so. First of all, our presentation in Subsection 2.1 clearly shows that the distinction can always be made implicitly based on the binding occurrence. Secondly, we would obtain exactly the same least solution regardless of whether we use $\rho : (\text{Const} \cup \text{Var}) \to \wp(\text{Const})$ or $\rho : \text{Name} \to \wp(\text{Name})$ as spurious content like $\rho(m) = \{x\}$ for a name $m$ and variable $x$ would never arise in the least solution. However, maintaining a clear distinction serves as a useful “typing discipline” that facilitates a less error prone development of the Flow Logic specification — a lesson we learned the hard way in the early work of [Bodei et al. 2001a; 1998; 1999].

Structural congruence and canonical names. Starting with the $\pi$-calculus many process calculi are given a semantics that has two components. One is a structural congruence that allows rearranging processes in a way that should clearly be invariant under any semantic observations. The other is a reduction relation, whether in the form of a reaction semantics as used here or a labelled transition system, for modelling the actual computation steps. One can dispense with the structural congruence, at the price of a more complex specification of the reduction relation, and indeed this is the approach of early process calculi and of programming languages.

One of the complications offered by the structural congruence is the ability to perform $\alpha$-renaming of names — usually only for constants. Since the analyses generally track the way constants are propagated through the process as part of the computation steps it is clear that names cannot be used directly in carrying analysis information. The solution presented here is the use of canonical names and disciplined $\alpha$-renaming and in such a way that we only analyse processes where all names are indeed canonical (as clearly displayed in Theorem 2.6).

The early work of [Bodei et al. 2001a; 1998; 1999] takes a slightly different route. Here each defining occurrence is annotated with a marker $\chi$ as in $(\text{new } n^\chi)P$. The markers are nothing but pointers into the process and thus remain stable under $\alpha$-renaming; hence they can be used to carry analysis information but the presentation is syntactically heavier.

A similar approach is taken in the early work on analysing the ambient calculus [Nielson et al. 2002]. The terminology is that markers for constants are called stable
names whereas markers for variables are called binders. The later work of [Nielsen et al. 2004] abandons the α-renaming of variables and hence the binders but then it introduces a notion of groups as a replacement for the stable names. Each defining occurrence of a constant is now associated with a group as in \((\text{new } n : \mu) P\) and then an additional construct \((\text{new } \mu) P\) is used to introduce the group names. Essentially this means that the groups are sorts in the sense of Subsection 2.4.

The notion of canonical names was introduced for the LySa calculus in [Bodei et al. 2005]; clearly the canonical names play a role similar to that of markers, groups, and stable names. However, the presentation in [Bodei et al. 2005] lacks the idea of only analysing processes where all names are indeed canonical (as in Theorem 2.6).

Reaction semantics versus labelled transition systems. As already mentioned, for process calculi we often have the choice between using a reaction semantics (as in this paper) or using a labelled transition system [Pilegaard et al. 2006a]. Both approaches are equally amenable to the development of a Flow Logic and have been illustrated in many papers.

In our later papers we have generally favoured the use of reaction semantics because we find that it gives a somewhat more concise definition of the subject reduction result and of the dynamic notions captured in the adequacy results. The reason quite simply is that accounting for the labels on transitions, some of which are input and others are output, sometimes lead to formulations that are more than three times more verbose than is the case for reaction semantics; this in turn gives rise to more complex proofs (as found in e.g. [Bodei et al. 2001a; 1998; 1999]).

Replication versus recursion. There are at least two ways to equip process calculi with the ability to perform recursive invocations. One is the use of replication, as performed in this paper, which seems to be the approach favoured by many researchers and which is technically rather easy to deal with. Another is the use of explicit recursion operators, either in the form of named constants that are unfolded, or in the form of an explicit recursion construct embedded in the syntax. We refer to [Nielsen et al. 2007; Nielsen et al. 2004; Tolstrup et al. 2007] for Flow Logics taking this approach. In the case where the recursion constants take parameters the development becomes more demanding.

3. EXTENDING THE ANALYSIS TECHNOLOGY

In Section 2 we have shown how to perform a full development of a Flow Logic for the π-calculus. In this section we illustrate a few techniques that can increase the precision for the examples considered in Section 2. The techniques are presented in the order of increasing sophistication and are mainly orthogonal so that they can be combined in different ways; the analysis presented in Subsection 3.4 may be omitted on a first reading. For each of the extensions we shall establish a subject reduction result; when the proofs go beyond straightforward extensions of previous proofs we give the details in the Appendix. We shall refrain from going into the details about adequacy results and Moore Family results just as we omit discussions about how to implement the analyses — these parts of the development can be carried out much as in Section 2 and hence do not provide essential new insights. We conclude
with a few pointers to the literature.

3.1 Reachability

A process $P$ is dead if it can never be executed. We might want to use our analysis to identify dead subprocesses in the process $P_\star$ of interest, and furthermore, we might want the analysis result of $P_\star$ not to be polluted by the analysis result of dead subprocesses. It turns out that it is very simple to modify the analysis of Section 2 to achieve this.

The idea is to extend the judgements of actions to contain yet another component:

$$\rho, \kappa \vdash A \pi : \psi \& \delta$$

Here $\delta \subseteq \{\bullet\}$ is a reachability component: if $\delta = \emptyset$ then it means that neither the action $\pi$, nor its continuation, can ever be executed. On the other hand, if $\delta = \{\bullet\}$ then both $\pi$ and its continuation might be executed. This information can be used in the analysis of guarded sums; as an example consider the clause for unary sums

$$\rho, \kappa \vdash \pi, P : \psi \iff (\rho, \kappa \vdash A \pi : \psi' \& \delta' \& \psi' \subseteq \psi \land \delta' \neq \emptyset \Rightarrow (\rho, \kappa \vdash P : \psi'' \land \psi'' \subseteq \psi))$$

where the reachability component $\delta$ from the analysis of the action $\pi$ is used to determine whether or not the analysis result needs to be acceptable for the continuation $P$. This clause is generalised to guarded sums in Table V; the remaining clauses for processes are as in Table III.

The clauses defining the analysis of actions are also listed in Table V. In the clause [OUT] for output $u(\vec{v})$ we inspect whether the sequence $\vec{v}$ indeed denotes anything; if it does not then the action can never be part of an interaction and we refrain from imposing that $\bullet \in \delta$. In the clause [IN] for input $u(\vec{z})$ we consult the abstract channel environment $\kappa$ to determine whether there are any suitable sequences that might be communicated over the channel; if not then the action will never be part of an interaction and again we shall not impose that $\bullet \in \delta$. The silent action $\tau$ is always possible so the clause [TAU] always imposes that $\bullet \in \delta$.

---

Table V. Flow Logic with a reachability component: $\rho, \kappa \vdash A \pi : \psi \& \delta$. 

---
\[
\begin{align*}
&\text{IP}(\text{new}\ n\ P) = \text{IP}(P) \\
&\text{IP}(P_1 | P_2) = \text{IP}(P_1) \cup \text{IP}(P_2) \cup \\
&(\text{Lab}_{oa}(P_1) \times \text{Lab}_{ia}(P_2)) \cup (\text{Lab}_{oa}(P_2) \times \text{Lab}_{ia}(P_1)) \\
&\text{IP}(\text{!}P) = \text{Lab}_{oa}(P) \times \text{Lab}_{ia}(P) \\
&\text{IP}(\Sigma_{i \in I} \pi_i. P_i) = \bigcup_{i \in I} \text{IP}(P_i)
\end{align*}
\]

Table VI. Sets of interaction points in \(P\): \(\text{IP}(P)\).

**Example 3.1.** The following values of \(\rho\), \(\kappa\) and \(\psi\) now constitute an acceptable analysis result for the process \(P'\) of Example 2.1:

\[
\rho = \begin{bmatrix} x & y \\ \{n\} & \emptyset \end{bmatrix} \quad \kappa = \begin{bmatrix} c & n & m \\ \{n\} & \{m\} & \emptyset \end{bmatrix} \quad \psi = \{c\}
\]

Compared to Example 2.2 we are now able to discover that neither the action \(c(x, y)\) nor its continuation \(x(y)\) will ever be executed and therefore we can have \(\rho(y) = \emptyset\). □

The correctness result of Theorem 2.6 carries over to the present setting:

**Theorem 3.2** Subject Reduction. If \(P \rightarrow Q\) and \(\rho, \kappa \vdash_P [P] : \psi\) then \(\rho, \kappa \vdash_P [Q] : \psi\).

### 3.2 Interaction Points

It is clear from the semantics that two actions can only interact if they are in different threads; this is not captured in the current analysis where we happily record the result of a communication over \(c\) for a process like \(c\langle n\rangle.c(x)\). We shall now show how to improve the analysis in this respect.

We extend the syntax to have labels \(\ell \in \text{Lab}\) on input and output actions:

\[
\pi ::= u\langle \vec{v} \rangle^\ell \mid u(\vec{x})^\ell \mid \tau
\]

These labels have no semantic meaning – they merely serve as pointers into the syntax. We shall write \(\text{Lab}_{oa}(P)\) for the set of labels of output actions in \(P\) and similarly \(\text{Lab}_{ia}(P)\) for the set of labels of input actions in \(P\).

We can now determine the pairs of (labels of) output and input actions that might occur in parallel threads at some time during the execution of a process \(P\). Such pairs \((\ell_o, \ell_i)\) of labels will be called interaction points. The potential sets of interaction points \(\text{IP}(P) \subseteq \text{Lab}_{oa}(P) \times \text{Lab}_{ia}(P)\) of a process \(P\) can easily be determined by a simple syntax directed definition as shown in Table VI. Note that for parallel composition we include all the possibilities for the two threads to interact whereas this is not the case for sums. Replication \(!P\) corresponds to any number of parallel occurrences of \(P\) and hence all combinations of output and input labels from \(\text{Lab}_{oa}(P)\) and \(\text{Lab}_{ia}(P)\) are possible interaction points.

It is worth pointing out that \(\text{IP}(P)\) identifies a potential set of interaction points. For a process like \(c\langle n\rangle.c(x) | (c(y).c(y) + c(z).c(z))\) it will predict that the output \(c\langle n\rangle\) may interact with any of the inputs of the two branches of the sum and similarly that the input \(c(x)\) may interact with any of the two outputs of the sum – however, in a concrete execution one of the latter possibilities will be discarded depending on the first communication.
Assume that \( \emptyset \subseteq \kappa(n, \ell_o) \) for every output label \( \ell_o \) in \( \tau \). Let us add labels to the process \( \tau \): 

\[
\begin{array}{c|c}
\text{TABLE VII. Flow Logic with interaction points: } & \\
\hline
\rho, \kappa \vdash_\tau \pi : \psi & \text{true} \\
\end{array}
\]

We can now refine our analysis to use the following abstract domains:

\( \neg \rho : \text{Var} \rightarrow \varphi(\text{Const}) \) is as before: it maps a variable to the set of constants that it might be bound to.

\( \neg \kappa : \text{Const} \times \text{Lab} \rightarrow \varphi(\text{Const}^*) \) is extended to record which sequences of constants that are communicated over a given (channel) constant at a given output label.

\( \neg \psi : \varphi(\text{Const} \times \text{Lab} \times \text{Lab}) \) is extended to record the set of triples of (channel) constants and interaction points where there may be an arity mismatch in a communication.

The form of the judgements for processes as well as actions are as in Section 2 and so are the clauses for processes, that is, they are as in Table III. The clauses for actions are modified as shown in Table VII and now make use of the set of potential interaction points for the process \( P_* \) of interest. In the clause \( \text{[OUT]} \) for output \( u(\vec{x})^{\ell_o} \) we simply record the relevant information for \( \ell_o \). In the clause \( \text{[IN]} \) for the input action \( u(\vec{x})^{\ell_i} \) we restrict our attention to the interaction points of IP\( (P_*) \) involving \( \ell_i \), and furthermore, in the case of a potential mismatch of arity we require that the name of the channel as well as the interaction point be recorded in \( \psi \).

**Example 3.3.** Let us add labels to the process \( P'_* \) considered in Example 2.1:

\[
(\text{new } n)^1, c(n)^2 \mid c(x)^3, (\text{new } m)x(m)^4 \mid c(x, y)^5, x(y)^6
\]

The set of output labels is \( \{1, 2, 4\} \), the set of input labels is \( \{3, 5, 6\} \) and the set IP\( (P'_*) \) of interaction points contains all combinations of output and input labels except for \( (4, 3) \). The analysis result of Example 2.2 now becomes

\[
\rho = \begin{array}{c|c}
1 & \{n\} \\
2 & \{n\} \\
4 & \{m\} \\
\end{array}
\]

\[
\kappa = \begin{array}{c|c|c}
1 & \{\ell_o\} & \kappa(n, \ell_o) \\
2 & \{\ell_i\} & \emptyset \\
4 & \{\ell_o\} & \emptyset \\
\end{array}
\]

\[
\psi = \begin{array}{c}
\emptyset \\
(\emptyset) \\
\emptyset \\
\end{array}
\]

thereby giving a more precise account of where an error might arise.

**Correctness.** It is important to note that the analysis of a process \( P \) is relative to the set IP\( (P_*) \) of potential interaction points of the initial process – this is evident in the clause for \( \text{[IN]} \). So to be precise we really should write the clauses as \( \rho, \kappa \vdash_{\text{IP}(P_*)} P : \psi \) rather than \( \rho, \kappa \vdash P : \psi \). Using this notation we can establish subject reduction as follows:

**Theorem 3.4 Subject Reduction.** Assume that \( P_* \rightarrow \cdots \rightarrow P \). If \( P \rightarrow Q \) and \( \rho, \kappa \vdash_{\text{IP}(P_*)} [P] : \psi \) then \( \rho, \kappa \vdash_{\text{IP}(P_*)} [Q] : \psi \).
The analysis of processes is now specified by judgements of the form

\[ \rho, \kappa \models^\ell P : \psi \]

The proof is similar to that of Theorem 2.6 and the only complication arises for the communication axiom \((n\langle \bar{m}\rangle_{R_1} : R^1 + R'') \mid (n\langle \bar{x}\rangle_{\ell_1} : Q + Q') \rightarrow P \mid Q'\langle \bar{m}/\bar{x}\rangle\) (where \(|\bar{m}| = |\bar{x}|\)). Here we must additionally ensure that \((\ell_o, \ell_i) \in IP(P)\) as otherwise the Flow Logic would fail to capture the semantic behaviour. For this the following lemma is helpful:

**Lemma 3.5 Interaction Points.** If \(P \equiv Q\) then \(IP(P) = IP(Q)\). If \(P \rightarrow Q\) then \(IP(P) \supseteq IP(Q)\).

### 3.3 Localised Environments

So far we have used a global abstract environment \(\rho\) that is unable to differentiate between different binding occurrences of the same name. We shall now introduce localised environments as one solution to deal with this problem; this technique will prove useful in more complex analyses as will be illustrated later.

As in Subsection 3.2 we shall add labels \(\ell \in Lab\) to the actions; however, we only need the labels for the input actions as they are the only ones that introduce new bindings to variables:

\[ \pi ::= u(\bar{v}) \mid u(\bar{x})^\ell \mid \tau \]

We shall write \(lab(\pi, \ell)\) for the label of \(\pi\) in the case where \(\pi\) is an input action and otherwise it is simply \(\ell\).

We shall modify the analysis domains to use labels as indicated below:

- \(\rho : Lab \times Var \rightarrow \wp(\text{Const})\) is the localised abstract environment that given a label maps a variable to the set of constants that it can be bound to in the context described by the label.

- \(\kappa : \text{Const} \rightarrow \wp(\text{Const}^*)\) is as before: it maps a (channel) constant to the set of sequences of constants that may be communicated over it.

- \(\psi : \wp(\text{Const} \times Lab)\) is the error component that now records pairs of (channel) constants and (input) labels indicating where an arity mismatch might occur.

The analysis of processes is now specified by judgements of the form

\[ \rho, \kappa \models^\ell P : \psi \]

<table>
<thead>
<tr>
<th>Table VIII. Flow Logic with localised environments: (\rho, \kappa \models^\ell P : \psi) and (\rho, \kappa \models^\ell_X \pi : \psi).</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RES] (\rho, \kappa \vdash^\ell P : \psi) if (\forall i \in I : (\rho, \kappa \vdash^\ell\pi_i : \psi_i \land \psi_i \subseteq \psi) \land (\rho, \kappa \vdash_{lab}^\ell P_i : \psi_i \land \psi_i \subseteq \psi))</td>
</tr>
<tr>
<td>[PAR] (\rho, \kappa \vdash^\ell P_1</td>
</tr>
<tr>
<td>[REP] (\rho, \kappa \vdash^\ell !P : \psi) if (\rho, \kappa \vdash^\ell P : \psi)</td>
</tr>
<tr>
<td>[SUM] (\rho, \kappa \vdash^\ell \sum_{i \in I} \pi_i, P_i : \psi) if (\forall i \in I : (\rho, \kappa \vdash^\ell_{lab} \pi_i, P_i : \psi_i \land \psi_i \subseteq \psi))</td>
</tr>
<tr>
<td>[OUT] (\rho, \kappa \vdash^\ell_X u(\bar{v}) : \psi) if (\forall n \in \rho(\ell, u) : \rho(\ell, \bar{v}) \subseteq \kappa(n))</td>
</tr>
<tr>
<td>[IN] (\rho, \kappa \vdash^\ell_X u(\bar{x})^\ell : \psi) if (\forall n \in \rho(\ell, u) : (\kappa(n) \cap \text{Const}^{</td>
</tr>
<tr>
<td>[TAU] (\rho, \kappa \vdash^\ell_X \tau : \psi) if true</td>
</tr>
</tbody>
</table>
where \( \ell \) identifies the context in which \( P \) is being analysed; we shall assume that the initial process \( P_* \) is analysed in the context \( \ell_* \in \text{Lab} \). The analysis of actions is specified by judgements of the form:

\[
\rho, \kappa \vdash^\ell_X \pi : \psi
\]

Here \( X \subseteq \text{Var} \) is the set of variables occurring free in the continuation of \( \pi \). The clauses are specified in Table VIII and are explained below.

The clauses for processes are much as before except that the context information is propagated to the subcomponents and the context information may only change in the case of guarded sums. We shall explain two instances of this clause in detail.

Let us first consider a unary sum with an output action; here the clause \([\text{SUM}]\) specialises to

\[
\rho, \kappa \vdash^\ell_p u(\bar{v}).P : \psi \text{ if } \mathbf{f} \rho, \kappa \vdash^\ell_m \mathbf{f}(\mathbf{u}) \bar{v} : \psi' \wedge \psi' \subseteq \psi \wedge \rho, \kappa \vdash^\ell P : \psi'' \wedge \psi'' \subseteq \psi
\]

and we note that the continuation \( P \) is analysed in the same context \( \ell \) as \( u(\bar{v}).P \). The clause \([\text{OUT}]\) for output ensures that the binding information is used correctly in the given context; in particular, the values of \( u \) as well as \( \bar{v} \) are obtained from the current context \( \ell \) as expressed by the requirement \( \forall n \in \rho(\ell, u) : \rho(\ell, \bar{v}) \subseteq \kappa(n) \) of the clause. Here \( \rho(\ell, (v_1, \cdots, v_k)) \) is shorthand for \( \rho(\ell, v_1) \times \cdots \times \rho(\ell, v_k) \), i.e. \( \{(m_1, \cdots, m_k) | \wedge_{i=1}^k m_i \in \rho(\ell, v_i)\} \).

Let us next consider a unary sum with an input action; then the clause \([\text{SUM}]\) specialises to

\[
\rho, \kappa \vdash^\ell_p u(\bar{x}).P : \psi \text{ if } \mathbf{f} \rho, \kappa \vdash^\ell_m \mathbf{f}(\mathbf{u}) \bar{x} : \psi' \wedge \psi' \subseteq \psi \wedge \rho, \kappa \vdash^\ell P : \psi'' \wedge \psi'' \subseteq \psi
\]

and now we note that the continuation \( P \) is analysed in the new context \( \ell_i \) reflecting that new bindings are performed by the input action. The clause \([\text{IN}]\) for input ensures that the relevant bindings of variables in \( \ell \) are propagated to the context \( \ell_i \). For the bound variables of \( \bar{x} \) this is ensured by the requirement

\[
\forall n \in \rho(\ell, u) : \kappa(n) \cap \text{Const}^{\bar{x}} \subseteq \rho(\ell_i, \bar{x})
\]

which is a shorthand for

\[
\forall n \in \rho(\ell, u) : \forall(n_1, \cdots, n_k) \in (\kappa(n) \cap \text{Const}^{\bar{x}}) : \bigwedge_{j=1}^k \{n_j\} \subseteq \rho(\ell_i, x_j)
\]

where \( x_j \) is the \( j \)th element of \( \bar{x} \). For the other variables of \( \mathbf{f}(\mathbf{u}) \) the propagation of the relevant bindings is ensured by the requirement

\[
\forall y \in (\mathbf{f}(\mathbf{u}) \setminus \{\bar{x}\}) : \rho(\ell, y) \subseteq \rho(\ell_i, y)
\]

that expresses that these variables have the same potential values in the new and the old context. Finally, arity mismatches are recorded in \( \psi \).

Example 3.6. Let us once again return to the process \( P_* \) of Example 2.1 and add labels to the input actions as follows:

\[
(\text{new } n)(c(n), c(n)) \mid c(x)^1 \cdot ((\text{new } m)x(m)) \mid c(x, y)^2 \cdot x(y)^3
\]
Using localised environments we obtain an acceptable analysis result when we take $\rho$, $\kappa$ and $\psi$ to be:

\[
\begin{array}{c|c|c}
\ell & \rho(\ell, x) & \rho(\ell, y) \\
\hline
\ell_* & \emptyset & \emptyset \\
1 & \{n\} & 0 \\
2 & 0 & \{m\} \\
3 & 0 & 0
\end{array}
\]

$\kappa = \begin{cases} c & n \\ m & \emptyset \end{cases}$

\[\psi = \{(c, 2)\}\]

This analysis is able to distinguish between the two defining occurrences of $y$ and it also records that the input action labelled 3 will never be executed. □

**Correctness.** As before the correctness of the analysis is expressed as a subject reduction result that requires a few auxiliary lemmas. The new substitution result reads as follows:

**Lemma 3.7 Substitution.** If $\rho, \kappa \vdash_\ell P : \psi$ then $\rho, \kappa \vdash_\ell P[m/y] : \psi$ provided that $\lfloor m \rfloor \in \rho(\ell, y)$.

The main complication in proving the subject reduction result arises for the communication axiom where we also need a relocation lemma:

**Lemma 3.8 Relocation.** If $\rho, \kappa \vdash_\ell P : \psi$ then $\rho, \kappa \vdash_\ell P : \psi$ provided that $\forall z \in \text{fv}(P) : \rho(\ell, z) \subseteq \rho(\ell', z)$.

We are now finally able to establish the subject reduction result:

**Theorem 3.9 Subject Reduction.** If $P \rightarrow Q$ and $\rho, \kappa \vdash_\ell P : \psi$ then $\rho, \kappa \vdash_\ell Q : \psi$.

### 3.4 Relational Analysis

The analyses presented so far are not able to record the relationship between the bindings of names. As an example consider the process:

\[c\langle n, m \rangle | c\langle m, n \rangle | c(x, y)\]

The analyses will correctly record that both $x$ and $y$ take values from the set $\{n, m\}$ but they fail to capture that $x$ and $y$ will be different. This gives rise to imprecision and we shall now see how to do better. Indeed, the analyses considered so far are *independent attribute* analyses [Nielson et al. 1999].

To improve on this aspect we shall now introduce a *relational analysis*. It will borrow ideas from the previous analyses but rather than recording the potential values of the variables by mappings $\text{Var} \rightarrow \wp(\text{Const})$ we shall fix the ordering of the variables in the process and record their potential values by sets of sequences of values of appropriate length. Each of these sequences corresponds to a potential environment where the $i$'th variable has the $i$'th value in the sequence. On top of this we shall localise the information much as in Subsection 3.3. Returning to the motivating example above we may fix the ordering of the variables as $(x, y)$ and then record their possible values by the set of pairs $\{(n, m), (m, n)\}$ – the first components of these pairs record the possible values for $x$ and the second components the possible values for $y$ and at the same time we have also recorded that $x$ and $y$ have different values.
The first step in our development is to add labels $\ell \in \text{Lab}$ to the actions and here we follow the approach of Subsection 3.2 and add labels to all actions:

$$\pi ::= u(\vec{v})^\ell | u(x)^\ell | \tau^\ell$$

We shall write $\text{lab}(\pi)$ for the label of $\pi$ and we shall write $\text{bv}(\pi)$ for the sequence of variables bound in $\pi$ (writing $\epsilon$ for the empty sequence).

We need to capture part of the control structure of the processes, and in particular, we need to fix the ordering of the variables at the various program points. To this effect let us consider the process $P_\star$ of interest and let us assume that all the actions in $P_\star$ are uniquely labelled. Furthermore we shall assume that $P_\star$ will be analysed in the context of the label $\ell_\star$ not occurring in $P_\star$. We shall now introduce the following functions $B$ and $L$ (or $B_\star$ and $L_\star$ to be precise):

$- B : \text{Lab} \hookrightarrow \text{Lab}$ is a (partial) function that maps a given label to the label in which scope it occurs within $P_\star$.

$- L : \text{Lab} \hookrightarrow \text{Var}^*$ is a (partial) function that maps a given label to the sequence of variables in scope after performing the bindings of the labelled action in $P_\star$.

In the sequel we shall find it helpful to write $B.\ell$ for $B(\ell)$ and similarly $L.\ell$ for $L(\ell)$.

Example 3.10. To illustrate the information captured by $B$ and $L$ let us consider the process $P_\star$ that is a variant of the running example where the channel $c$ is used to send pairs of names:

$$((\text{new } n)(\text{new } m)c(n,m)^1 . c(m,n)^2) | (c(x,y)^3 . x(y)^4) | (c(x,y)^5 . y(y)^6)$$

The auxiliary information recorded by $B$ and $L$ is as follows:

$$\begin{align*}
B &= \ell_\star | 1 | 2 | 3 | 4 | 5 | 6 \\
L &= \ell_\star | 1 | 2 | 3 | 4 | 5 | 6
\end{align*}$$

The mapping $B$ determines the scope of each of the labels: $B.1$ is the scope of the action $c(n,m)^1$ and it is equal to $\ell_\star$, $B.2$ is the scope of the action $c(m,n)^2$ and it is equal to 1, etc.

The mapping $L$, on the other hand, determines which variables are in scope after each of the labels. So we see that $L.1$ records the variables in scope after the action $c(n,m)^1$ has been performed and since there are none the result is $\epsilon$. More interestingly, $L.5$ records that after the action labelled 5 has been executed the variables $x$ and $y$ are in scope and they are ordered as $xy$. The action labelled 6 introduces a new binding occurrence of $y$ and this is recorded in $L.6$ by the sequence $xyy$ meaning that we will record the old as well as the new binding occurrence of the variable $y$.

To define the functions $B$ and $L$ we shall use the two auxiliary functions $B_\ell$ (for $\ell \in \text{Lab}$) and $L_{\vec{y}}$ (for $\vec{y} \in \text{Var}^*$) defined in Table IX and explained below. The function $B_\ell$ will, given a process $P$, compute the scoping information for the labels of $P$ under the assumption that the process itself is in the scope $\ell$. The definition is syntax directed and we use the function $\oplus$ to merge mappings with disjoint domains – the assumption that $P_\star$ is uniquely labelled ensures that the mappings constructed for the subprocesses indeed have disjoint domains. The only non-trivial
\[
B_ℓ((\text{new } n)P) = B_ℓ(P)
\]
\[
B_ℓ(P_1 \mid P_2) = B_ℓ(P_1) \oplus B_ℓ(P_2)
\]
\[
B_ℓ(IP) = B_ℓ(P)
\]
\[
B_ℓ(Σ_{i∈I} π_i.P_i) = \bigoplus_{i∈I}(B_{\text{lab}(π_i)}(P_i) \oplus [\text{lab}(π_i) → ℓ])
\]

\[
L_⃗y((\text{new } n)P) = L_⃗y(P)
\]
\[
L_⃗y(P_1 \mid P_2) = L_⃗y(P_1) \oplus L_⃗y(P_2)
\]
\[
L_⃗y(IP) = L_⃗y(P)
\]
\[
L_⃗y(Σ_{i∈I} π_i.P_i) = \bigoplus_{i∈I}(L_{⃗y\text{bv}(π_i)}(P_i) \oplus [\text{lab}(π_i) → ⃗y\text{bv}(π_i)])
\]

Table IX. Auxiliary Information for the Relational Analysis.

The clause of the definition is for sums; in the case of prefixing it amounts to
\[
B_ℓ(π.P) = B_{\text{lab}(π)}(P) \oplus [\text{lab}(π) → ℓ]
\]

thereby recording that the scope of (the label of) \(π\) is \(ℓ\) and that the continuation \(P\) is inspected in the scope of \(\text{lab}(π)\).

The function \(L_⃗y\) will, given a process \(P\), determine the sequences of variables that are in scope at the various labels in \(P\) under the assumption that the context of \(P\) already tells us that the variables of \(⃗y\) are in scope. Also here, only the case of sums is non-trivial; in the case of prefixing it amounts to
\[
L_⃗y(π.P) = L_{⃗y\text{bv}(π)}(P) \oplus [\text{lab}(π) → ⃗y\text{bv}(π)]
\]

Thus we inspect the continuation \(P\) in a context where the bound variables of \(π\) are appended to those of \(⃗y\) and we record that the same sequence of variables is of interest at the label \(\text{lab}(π)\).

Finally we shall define the functions \(B\) and \(L\) for the process \(P_⋆\) of interest by the equations
\[
B = B_{ℓ⋆}(P_⋆) \oplus [ℓ⋆ → ℓ⋆]
\]
\[
L = L_{ℓ⋆}(P_⋆) \oplus [ℓ⋆ → ℓ]
\]

where \(ℓ⋆\) is a label not occurring in \(P_⋆\).

We are now ready to define the relational analysis. The judgements for processes and actions take the rather familiar form
\[
ρ, κ ⊬ P : ψ \quad \text{and} \quad ρ, κ ⊏ A : ψ
\]

but some of the domains have been modified:

\(→ ρ : \text{Lab} → ϕ(\text{Const}^+)\) is the abstract environment that for each label contains sets of tuples of potential constants bound to the variables; we shall ensure that all tuples in \(ρ(ℓ)\) have the same length as \(L.ℓ\).

\(→ κ : ϕ(\text{Const}^+)\) contains non-empty tuples where the first component is the (channel) constant over which the remaining components have been communicated.

\(→ ψ : ϕ(\text{Const})\) is as before: it records the set of (channel) constants where there may be an arity mismatch in a communication.

The clauses are specified in Table X. First we observe that the clauses [RES], [PAR], [REP] and [SUM] are much as in Table III.
Before discussing the non-trivial clauses for actions we need to define the value \( \Pi_{u \in \mathcal{Z}}(\vec{m}) \) of a name \( u \) with respect to a “local environment” \( \vec{m} \) specifying the values of the variables \( \vec{z} \) (and hence satisfying \( |\vec{m}| = |\vec{z}| \)). Writing \( \vec{z} = (z_1, \cdots, z_h) \) and \( \vec{m} = (m_1, \cdots, m_h) \) we proceed by cases on whether the name \( u \) is a constant \( n \) or a variable \( y \):

\[
\begin{align*}
\Pi_{u \in \mathcal{Z}}(\vec{m}) &= n \\
\Pi_{y \in \mathcal{Z}}(\vec{m}) &= m_i \quad \text{when } i \text{ is maximal such that } z_i = y 
\end{align*}
\]

This means that if a variable \( y \) has several binding occurrences, that is, it occurs several times in \( \vec{z} \), then \( \Pi_{y \in \mathcal{Z}}(\vec{m}) \) returns the value in \( \vec{m} \) corresponding to the most recent (rightmost) binding of \( y \). As an example we have \( \Pi_{y \in \mathcal{Z}}(\vec{m}) = n \) reflecting that the value of \( y \) in the local environment \( nmn \) is \( n \) – given that the variables are ordered as \( xyy \) and that we are looking for the value of \( y \) in the most recent scope.

The clause [OUT] for an output action \( u(\vec{v})^\ell \) forwards the values in the existing environment into the new one. The new environment is given by the label \( \ell \) and we use the function \( B \) to determine the existing environment: it is \( B.\ell \) so we simply have \( \rho(B.\ell) \subseteq \rho(\ell) \). Furthermore, the clause records the values being output by considering each “local environment” \( \vec{m} \) of \( \rho(B.\ell) \) in turn. Here the corresponding variables are listed in \( L(B.\ell) \) and the values corresponding to \( u \) and \( \vec{v} \) must be extracted and recorded in \( \kappa \). To express this we make use of a slight generalisation of the above function

\[
\Pi_{u_1, \ldots, u_k \in \mathcal{Z}}(\vec{m}) = \Pi_{u_1 \in \mathcal{Z}}(\vec{m}) \cdots \Pi_{u_k \in \mathcal{Z}}(\vec{m})
\]

in order to collect the values in the right order; in the case of the [OUT] clause we require that \( \Pi_{u \in \mathcal{Z}}(\vec{m}) \in \kappa \) reflecting that the first value is the name of the channel and the remaining values are those being output.

The clause [IN] for an input action \( u(\vec{x})^\ell \) proceeds by considering each “local environment” \( \vec{m} \) in turn; the corresponding variables are obtained as \( L(B.\ell) \). It
Returning to Example 3.10 we have the following analysis result:

\[ n \in \{n \in \mathbb{N} \mid 6 \leq n \} \]

requiring us to check that \( n \in \{n \in \mathbb{N} \mid 6 \leq n \} \). In case some tuple \( \vec{o} \) has a length that differs from that of \( \vec{x} \), the corresponding channel \( n \) is a source of possible arity mismatch and \( n \) will be recorded in \( \psi \).

The method used for forwarding “local environments” has some similarities to the one used in Subsection 3.3. In both cases we have new “local environments” for each labelled action; in Subsection 3.3 this was only the case for inputs and hence no forwarding was needed for outputs or silent actions. Forwarding in both of these cases means copying (and possibly modifying) the “local environment” from the label preceding the current action to the label of the current action. The method used in this subsection is slightly different, however; in Subsection 3.3 we remembered the preceding label on the judgement itself, whereas here we make use of the information provided by \( \mathcal{B} \). This is largely motivated by the more complex proof strategy needed for proving the relational analysis correct.

Example 3.11. Returning to Example 3.10 we have the following analysis result:

\[
\begin{array}{c|c|c|c|c|c|c}
\ell_* & 1 & 2 & 3 & 4 & 5 & 6 \\
\{\epsilon\} & \{\epsilon\} & \{n, m\} & \{n, m\} & \{n, m\} & \{n, m, n\} & \{n, m, n\} \\
\end{array}
\begin{array}{c}
\kappa = \{cm, m, n\} \\
\psi = \emptyset
\end{array}
\]

This turns out to be the least analysis result that satisfies both \( \rho, \kappa \vdash \nu \mathcal{P} \) and \( \epsilon \in \rho(\ell_*) \).

To convince ourselves that this is correct let us consider the output action \( x(y)^4 \). We have to check that:

\[ \rho(\mathcal{B}.4) \subseteq \rho(4) \]

\[ \{\Pi_{xy} \mathcal{B}.4(\vec{m}) \mid \vec{m} \in \rho(\mathcal{B}.4)\} \subseteq \kappa \]

The first inclusion clearly holds since \( \mathcal{B}.4 = 3 \) according to Example 3.10. To verify the second inclusion we first observe that \( L.\mathcal{B}.4 = xy \) and since \( \rho(3) = \{nm, mn\} \) we calculate \( \Pi_{xy} \mathcal{B}.4(nm) = nm \) and \( \Pi_{xy} \mathcal{B}.4(mn) = mn \) and thus the inclusion test amounts to checking \( \{nm, mn\} \subseteq \kappa \) which indeed holds.

Let us next consider the input action \( y(y)^6 \). Here we have to check the following two conditions:

\[ \{\vec{m} \mid \vec{m} \in \rho(\mathcal{B}.6) \land n' = \Pi_{xy} \mathcal{B}.6(\vec{m}) \land n' \in \kappa\} \subseteq \rho(6) \]

\[ \{n' \mid \vec{m} \in \rho(\mathcal{B}.6) \land n' = \Pi_{xy} \mathcal{B}.6(\vec{m}) \land n' \in \kappa \land |\vec{d}| \neq 1\} \subseteq \psi \]

Let us focus on the first of them. From Example 3.10 we see that \( \mathcal{B}.6 = 5 \) and \( L.5 = xy \). We know that \( \rho(5) \) contains the two sequences \( nm \) and \( mn \) and we now consider them one by one. In the case of \( nm \) the value of \( y \) \((n') \) in the clause above) is \( n \) and since \( nm \in \kappa \) we get that \( o = m \) and we therefore have to check that \( nmn \in \rho(6) \) (which clearly holds). In the case of \( mn \) we make similar calculations requiring us to check that \( mnm \in \rho(6) \).

The constraints imposed by the other actions can be checked in a similar way. We shall return to the importance of requiring \( \epsilon \in \rho(\ell_*) \) shortly. \( \square \)
Correctness. It is considerably more complex to prove the relational analysis correct than the previous analyses. Intuitively, this is due to the “mismatch” between the style of the static analysis and that of the semantics; the former uses explicit “local environments” whereas the latter uses a substitution based semantics (rather than one with explicit environments). As shown in the scenario of [Nielson et al. 2008] this means that one can sometimes find counterexamples to a straightforward generalisation of the previous subject reduction results. Consequently we shall follow [Nielson et al. 2008] in showing that a more complex notion of analysability is preserved under evaluation.

Let us fix the initial process $P_*$ of interest and define $B, L$ and $\ell_*$ as shown above. We shall then define the following analysability predicate $\rho, \kappa \vdash \vdash P \cap Q : \psi$
to hold if and only if:

For each exposed subprocess $\beta^\ell. Q'$ of $|Q|$, there exists a subprocess $\alpha^\ell. P'$ of $P_*$ and a “local environment” $\vec{m} \in \rho(\ell, \ell)$, corresponding to the variable sequence $L(\ell, \ell)$, such that $\beta^\ell.Q' \equiv (\alpha^\ell.P')[\vec{m}/L(\ell, \ell)]$ and $\rho, \kappa \vdash \vdash \alpha^\ell.P' : \psi$.

Here a subprocess $\beta^\ell. Q'$ is said to be exposed in $Q$ if it is not prefixed by any actions; this ensures that $\beta^\ell. Q'$ does not contain any free variables and this is essential for obtaining a congruence when replacing all variables in $L(\ell, \ell)$ with their values within $\alpha^\ell. P'$.

In Theorem 3.16 below we shall establish a subject reduction result using this notion of analysability. However, we shall first state a few auxiliary lemmas. The first two results clarify the properties of exposed subprocesses.

**Lemma 3.12.** Let $P$ and $Q$ be two processes such that $P \equiv Q$. For each exposed subprocess $\alpha^\ell. P'$ of $P_*$, there exists an exposed subprocess $\beta^\ell. Q'$ of $Q$ such that $\alpha^\ell. P' \equiv \beta^\ell. Q'$.

**Lemma 3.13.** If $\alpha^\ell. P'$ is an exposed subprocess of $P$ and $\rho, \kappa \vdash \vdash P \cap P : \psi$ then $\rho, \kappa \vdash \vdash \alpha^\ell. P' : \psi$.

The next auxiliary result shows how to establish analysability of the process $P_*$ of interest – and it explains the role of the condition $\epsilon \in \rho(\ell_*)$.

**Lemma 3.14 Initial Process.** Let $P_*, B, L$ and $\ell_*$ be as shown above. If $\rho, \kappa \vdash \vdash P_* : \psi$ and $\epsilon \in \rho(\ell_*)$ then $\rho, \kappa \vdash \vdash P_* : \psi$.

Next we clarify how the structural congruence interacts with analysability.

**Lemma 3.15 Structural Congruence.** Let $P$ and $Q$ be two processes such that $P \equiv Q$. If $\rho, \kappa \vdash \vdash P : \psi$ then $\rho, \kappa \vdash \vdash Q : \psi$.

Finally, we are able to establish our subject reduction result.

**Theorem 3.16 Subject Reduction.** If $P \rightarrow Q$ and $\rho, \kappa \vdash \vdash P : \psi$ then $\rho, \kappa \vdash \vdash Q : \psi$.
3.5 Bibliographical Notes

Many of the above techniques for improving the precision of the analysis of Section 2 were originally developed for other calculi. As already mentioned, the techniques are to a large extent orthogonal and hence can be combined with each other; we shall return to this in Section 5.

The idea of checking for reachability is present in the development of analyses for the $\lambda$-calculus (see e.g., [Nielson and Nielson 2002]). In [Bodei et al. 2001a; 1998] we exploit it when analysing the test construct $[x = y]P$: the continuation $P$ should be analysed only if the test $x = y$ might succeed, that is, if $\rho(x) \cap \rho(y) \neq \emptyset$. In [Bodei et al. 1999] we take this idea one step further and only require the continuation of an action to be analysed if the action indeed may succeed – this is along the lines of the analysis in Table V.

The idea of determining the potential interaction points originates from [Bodei et al. 2005] where logical addresses of subprocesses are used to estimate which subprocesses may interact – such pairs of addresses are said to be compatible. In [Bodei et al. 2005] the logical addresses are obtained from an enhanced operational semantics. In [Nielson et al. 2004] the same idea is developed for an analysis of BioAmbients but using a simple labelling schema to precompute which actions may occur in parallel branches that thereby may interact – just as we have seen in Table VI. The main point to notice is that only the ‘skeleton’ of the actions are used, in particular we do not use information about the actual or canonical names involved (as they are not preserved under evaluation). More complex analyses are needed in order to capture this; see e.g., [Nielson and Nielson 2009; Pilegaard et al. 2008; Nielson and Nielson 2007b].

The idea of using localised environments goes back to the control flow analysis (CFA) of functional programs [Shivers 1988] and is presented in [Nielson et al. 1999] for the $\lambda$-calculus. Here contexts record the call structure of the various functions of the program and in the so-called $k$-CFA analyses up to $k$ levels of such calls are recorded and it is possible to distinguish between the variable bindings in these contexts. As we already have seen the important actions are now the input actions because they are the binders of variables. For the sake of simplicity we have only recorded one level of context information – the generalisation to $k$ levels is straightforward. Another generalisation uses pairs of input and output labels as contexts.

The distinction between independent attribute analyses and relational analyses is classical in Static Analysis (see e.g., [Nielson et al. 1999]). The actual analysis presented above is based on [Nielson et al. 2008] but performs a few simplifications; they are partly due to the $\pi$-calculus considered here being less complex than the pattern matching $\pi$-calculus in [Nielson et al. 2008] and partly due to letting localised environments record the bindings after rather than before the labelled actions. A more complex development for a considerably more challenging language is performed in [Bauer et al. 2008]. Our approach is purely syntax directed unlike the (more powerful) approaches of [Venet 1998; Feret 2002] that essentially require the process calculus to be translated into the form of an abstract machine; as discussed in [Nielson et al. 2008; Bauer et al. 2008] we believe that this is likely to enable more researchers to perform static analyses that go beyond the simple independent
attribute analyses.

4. FLOW LOGIC AS A PROGRAM LOGIC

In this section we shall make the point that a Flow Logic is a Program Logic — in much the same way that a Hoare Logic [Apt 1981] is. We shall not give a formal definition of a Flow Logic because that would defeat our purposes, but we shall provide a set of normative guidelines based on more than a decade of research. The treatment in this section is not intended for the novice to Flow Logic nor is it required for straightforward adaptations of Flow Logic to other calculi; for the novice that wants to read more we recommend the development in the electronic Appendix A.

4.1 Flow Logic Judgements

Two notations parameterise a Flow Logic:

(1) the programming notation being analysed, and
(2) the language used to express the constraints imposed.

In this paper we have so far focussed on programming notations that are process calculi while [Nielson and Nielson 2002] has focussed on programming notations that are programming languages; we shall consider programming notations in more detail in Subsection 4.2. Similarly, the constraint language used has been a somewhat informal mathematical notation with ingredients from first order logic as well as inclusion constraints between analysis estimates; we shall consider the use of first order logic as a constraint language in Subsection 4.3.

For each syntactic category \( P \) of the programming notation the Flow Logic defines an analysis judgement of the form

\[ \vec{R} \vdash \gamma, \Gamma \vdash_p P : \vec{T} \]

for expressing the acceptability of the analysis information with respect to the syntactic entity in \( P \). To be specific, \( P \in \mathcal{P} \) is a meta-variable ranging over the syntactic category, \( \vec{R} \) is a (usually non-empty) list of analysis predicates expressing global analysis information, \( \vec{T} \) is a list of analysis predicates expressing local analysis information (and is often empty), and \( \gamma \) is an indicator of context information (and is often empty or absent).

The component \( \Gamma \) is an (often empty or absent) environment containing information of interest for the analysis. As an example it may contain information about higher-order syntactic entities; if our calculus is extended with a construct like \( \text{let } A(\vec{x}) = P \) in \( Q \) that defines the higher-order abbreviation \( A \) for a possible recursive process \( P \) that may be used within \( Q \); here we may take \( \Gamma(A) = (\vec{x})P \) to record the presence of the definition. The \( \Gamma \) component may also be used for mapping identifiers in the programming notation to the elements in the domains of discourse – as for example associating a name with a representation of its scope.

It should be clear that the many analyses developed in the previous sections define analysis judgements as described here. As an example the judgements \( \rho, \kappa \vdash_{F,A}^{\ell,X} \pi : \psi \) of Subsection 3.3 makes use of all components except \( \Gamma \).
4.2 Programming Notations

One of the main ingredients of a Flow Logic is the programming notation. A programming notation, as for example a process calculus, is characterised by a number of syntactic categories, \( P, P', \ldots \), (e.g. names, Name, actions, Act, and processes, Proc) as well a number of syntactic formation rules,

\[ P := \sigma(P_1, \cdots, P_n) \]

for creating syntactic terms; here each \( P_i \in P_i \) is a meta-variable ranging over a syntactic category. As an example, we may have \( P := \pi.Q \) and \( \pi := u(x) \) where \( P, Q \in \text{Proc} \), \( \pi \in \text{Act} \) and \( u, x \in \text{Name} \). We have chosen to formalise this as a context free grammar where the nonterminals correspond to the syntactic categories and the productions correspond to the formation rules. Alternatively, one could have used a many-sorted algebra where the sorts correspond to the syntactic categories and the operators correspond to the formation rules.

In a given syntactic formation rule, \( P := \sigma(P_1, \cdots, P_n) \), it is common to define some positions as defining positions, together with an indication of their scope. As an example, for a rule like \( P := \pi(x).P \) the parameter \( x \) is a defining occurrence and its scope is (just) the continuation \( P \); in contrast, \( \pi \) is not a defining position.

As has been illustrated we often annotate the syntactic terms with labels in order to more precisely identify subterms. This is also useful for providing context information that can be exploited in the analysis. For example in the construct \( \pi := u(x) \ell \) it might be useful to record that \( x \) is defined at label \( \ell \).

A programming notation should also have a semantics. For process calculi some form of operational semantics is quite common but nothing prevents the use of other forms of semantics.

4.3 First Order Logic as a Constraint Language

The other main ingredient of a Flow Logic is the language used to express the constraints imposed by the analysis. Here first order logic embodies a good part of the most commonly used mathematical notation needed and it will serve our purposes for the main part of this section.

A first order logic is usually characterised by a universe of discourse which we shall model as an algebra \( U \) (with just one sort). On top of this there is a number of predicate symbols; an \( n \)-ary predicate symbol \( R \) takes \( n \) parameters from the universe of discourse and yields a truth-value. The predicates are used to formalise the analysis domains of interest and example predicates from the previous sections are therefore \( \kappa, \rho \) and \( \psi \).

A logical formula \( \phi \) is built from the predicate symbols given their appropriate arguments (being terms of the algebra \( U \)), the usual logical connectives, and universal and existential quantification over variables ranging over elements of the universe of discourse. Sometimes we wish to be precise about the set of predicates allowed in a given formula; when only predicates in a given set \( R \) are allowed, we shall say that the logical formula is based on the predicates in \( R \). Unless otherwise stated, \( R \) will equal a fixed set \( R_B \) of base predicates and this set should contain the analysis predicates \( \bar{R} \) and \( \bar{T} \) of the judgements \( \bar{R} \vdash_{\rho, T} P : \bar{T} \) mentioned in Subsection 4.1.

We shall say that a logical subformula, e.g. an occurrence of a predicate, occurs
at top-level in the formula $\phi$, if there are no (explicit or implicit) negations on the path to it; here an occurrence to the left of an implication is considered an implicit use of negation. More generally, we shall say that a logical subformula occurs positively if there is an even number of (explicit or implicit) negations on the path to it; clearly all top-level occurrences are also positive ones. A predicate is said to occur at top-level, or positively, whenever this holds for all its occurrences.

The semantics of first order logic is standard and will only be briefly summarised here. An interpretation $I$ assigns to each $n$-ary predicate $R \in R$ a subset $I(R)$ of $U^n$ (or alternatively a function $I_R : U^n \rightarrow \{true, false\}$). The definition of validity of a closed formula $\phi$, i.e. one having no free variables, with respect to an interpretation $I$ is denoted by $I \models = \phi$ and yields a truth-value (i.e. $true$ or $false$). It is defined structurally in $\phi$ making use also of a valuation mapping variables to elements of $U$ in order to deal with logical formulae that are not closed.

In the subsequent development we will occasionally need to use existential quantification from second order logic; in particular, existential quantification over predicates. We shall not detail the semantics of this as it is entirely standard and not of primary focus.

Alternation-free Least Fixed Point Logic. We often find it useful to work with Alternation-free Least Fixed Point Logic (abbreviated ALFP) [Nielson et al. 2002b]. This is a fragment of a first order logic that restricts the formation of logical formulae according to certain well-formedness criteria; it is a generalization of Horn clauses as well as Datalog [Chandra and Harel 1980; Apt et al. 1988] that has proved to have a number of properties essential for our development.

The syntax of ALFP is presented in Table XI, where we write $c$ for analysis constants, $x$ for analysis variables, $v$ for analysis values, $f$ for analysis functions, $R$ for analysis predicates, $pre$ for preconditions, and $clause$ for clauses.

The clauses are interpreted over a universe $U$ of analysis constants; indeed, $c$ is an element of $U$, $f$ has arity $U^n \rightarrow U$ (for some $n$), and $x$ ranges over $U$. The interpretation is given in terms of satisfaction relations

$$(I, \iota) \models pre \text{ and } (I, \iota) \models clause$$

where $I$ is an interpretation of analysis predicates, $\iota$ is an interpretation of analysis variables (which we extend to operate on analysis values). The definition is standard and is shown in Table XII.

We shall often need to demand that the clauses in ALFP are stratified. This intuitively means that no predicate depends on the negation of itself. We refer to [Nielson et al. 2002b] for the details.

entities. For each syntactic formation rule for expressing the acceptability of analysis information with respect to syntactic an
totation, e.g. a process calculus, and a fragment of a first order logic, e.g. ALFP,
A First Order Flow Logic

4.4 First Order Flow Logic Specifications

A First Order Flow Logic \( F \) is a program logic relative to a choice of a programming
notation, e.g. a process calculus, and a fragment of a first order logic, e.g. ALFP,
for expressing the constraints between analysis predicates.

As already explained above, for each syntactic category, \( P \), the Flow Logic defines
an\(^1\) analysis judgement of the form

\[
R \vdash_{p}^{\gamma;\Gamma} P : T
\]

for expressing the acceptability of analysis information with respect to syntactic entities. For each syntactic formation rule \( P ::= \sigma(P_1, \ldots, P_n) \) the Flow Logic defines exactly one\(^2\) clause of the form

\[
R \vdash_{p}^{\gamma;\Gamma} \sigma(P_1, \ldots, P_n) : T \quad \text{iff} \quad \phi
\]

where \( \phi \) is a logical formula in the fragment of first order logic considered and that
is based on a set of predicates \( R \) to be detailed below. The previous sections give
several examples of such clauses – see Tables III, V, VII, VIII and X.

The set \( R \) of predicates used on the right hand side of these clauses, that is in \( \phi \), contains two different kinds of predicates. It can be a base predicate from the
set \( R_B \) already mentioned in Subsection 4.3; recall that we assume that this set
contains all of the predicates \( R \) and \( T \). Actually, \( \phi \) may use predicates in \( R_B \setminus \{ R, T \} \)
and they will implicitly be existentially quantified (using quantifiers of second order
logic) and special care needs to be taken in the subsequent development.

The set \( R \) of predicates used on the right hand side of the clauses may also contain analysis judgement corresponding to one of the syntactic categories of the
programming notations; we shall write \( R_I \) for these predicates. They have the general form

\[
R_I \vdash_{p}^{\gamma_i,\Gamma} \sigma'(P_1, \ldots, P_n) : T'\]

\(^1\)It simplifies the presentation to assume that there is just one analysis judgement for each syntactic
category, but it is not an essential assumption.

\(^2\)This is essential and differs from Type Systems where in general there will be at least one axiom
scheme or inference rule defined for each syntactic formation rule.

Table XII. Interpretation of Alternation-free Least Fixed Point Logic (ALFP).

\[
\begin{align*}
(\mathcal{I}, \iota) & \models R(v_1, \ldots, v_k) \quad \text{iff} \quad \sigma(v_1, \ldots, v_k) \in \mathcal{I}(R) \\
(\mathcal{I}, \iota) & \models \neg R(v_1, \ldots, v_k) \quad \text{iff} \quad \sigma(v_1, \ldots, v_k) \not\in \mathcal{I}(R) \\
(\mathcal{I}, \iota) & \models v_1 = v_2 \quad \text{iff} \quad \sigma(v_1) = \sigma(v_2) \\
(\mathcal{I}, \iota) & \models v_1 \neq v_2 \quad \text{iff} \quad \sigma(v_1) \neq \sigma(v_2) \\
(\mathcal{I}, \iota) & \models \text{pre}_1 \land \text{pre}_2 \quad \text{iff} \quad (\mathcal{I}, \iota) \models \text{pre}_1 \text{ and } (\mathcal{I}, \iota) \models \text{pre}_2 \\
(\mathcal{I}, \iota) & \models \text{pre}_1 \lor \text{pre}_2 \quad \text{iff} \quad (\mathcal{I}, \iota) \models \text{pre}_1 \text{ or } (\mathcal{I}, \iota) \models \text{pre}_2 \\
(\mathcal{I}, \iota) & \models \exists x : \text{pre} \quad \text{iff} \quad (\mathcal{I}, \iota[x \leftarrow a]) \models \text{pre} \text{ for some } a \in U \\
(\mathcal{I}, \iota) & \models \forall x : \text{clause} \quad \text{iff} \quad (\mathcal{I}, \iota[x \leftarrow a]) \models \text{clause} \text{ for all } a \in U \\
\end{align*}
\]
and will be subject to the following conditions:

1. $\sigma'(P_1, \cdots, P_n)$ is often some of the $P_i$ but may be a more complex syntactic term.
2. $\vec{R}'$ is usually $\vec{R}$ or a subsequence of $\vec{R}$.
3. $\gamma'$ is usually $\gamma$ or is obtained from $\gamma$ using information in $\sigma'(P_1, \cdots, P_n)$, $\vec{R}$, $\vec{T}$ or $\Gamma$.
4. $\Gamma'$ is usually $\Gamma$ or an extension of it where some of the defining positions of $\sigma'(P_1, \cdots, P_n)$ give rise to new entries.
5. $\vec{T}'$ is often empty or is otherwise some sequence of base predicates in $\mathcal{R}_B \setminus \{\vec{R}\}$.

It is easy to check that the clauses of Tables III, V, VII, VIII and X satisfy these conditions.

To distinguish between various formats of Flow Logic the following terminology has been introduced [Nielsen and Nielsen 2002]:

— A Flow Logic is **compositional** whenever all formulae of the form $\phi$ only contain judgements where the syntactic component is a meta-variable (and not a more complex term); it is **abstract** otherwise.

— A Flow Logic is **verbose** whenever all predicate sequences $\vec{T}$ of all judgements are empty; it is **succinct** otherwise.

The Flow Logics presented in this paper are all compositional and succinct.

### 4.5 Well-Definedness

So far our treatment of a First Order Flow Logic $\mathcal{F}$ has been purely syntactic. Clearly it is intended to have a meaning and to enjoy some properties useful for static analysis. One of the fundamental properties of Flow Logic, which it shares with Type Systems, is that there is a clear separation between determining

— whether or not a given analysis estimate $\vec{R}, \vec{T}$ is an acceptable description of a program $P$, and

— finding the best such description, usually the least acceptable description with respect to some partial order.

In Type Systems the first property is known as type checking and the second as type inference. In Flow Logic the first property ensures the well-definedness of the judgements $\vec{R} \vdash_{p,T} P : \vec{T}$, i.e. that they denote either true or false, and this is dealt with in this subsection while the second property is dealt with under the considerations of the Moore Family property in Subsection 4.8.

Assume that we have a judgement $\vec{R} \vdash_{p,T} P : \vec{T}$, a program fragment $\sigma(P_1, \cdots, P_n)$ of the corresponding syntactic category, and an interpretation $\mathcal{I}$ of $\mathcal{R}_B$ (or at least of $\{\vec{R}, \vec{T}\}$). We shall then define what it means for $\mathcal{I}(\vec{R}, \vec{T})$ to be an **admissible analysis estimate** for $\sigma(P_1, \cdots, P_n)$ according to the Flow Logic $\mathcal{F}$.

**Definition 4.1. Admissibility of analysis estimates.** The interpretation $\mathcal{I}$ is admissible for $\vec{R} \vdash_{p,T} \sigma(P_1, \cdots, P_n) : \vec{T}$, written

$$\mathcal{I} \text{ sat } \vec{R} \vdash_{p,T} \sigma(P_1, \cdots, P_n) : \vec{T}$$

whenever there exists an interpretation $\mathcal{J}$ of $\mathcal{R}_B \cup \mathcal{R}_J$ such that
Flow Logic for Process Calculi

$J \left( \vec{R} \vdash \gamma^\Gamma \sigma(P_1, \ldots, P_n) : \vec{T} \right) = \text{true}$, and

$J(R) = I(R)$ for all $R \in \mathcal{R}_B$ (abbreviated $J \upharpoonright_{\mathcal{R}_B} = I$), and

$-J \text{ SAT } \mathcal{F}$

where the latter condition expresses that the interpretation $J$ satisfies the Flow Logic $\mathcal{F}$ as defined by:

**Definition 4.2. Satisfaction of Flow Logic.** The interpretation $J$ satisfies the Flow Logic $\mathcal{F}$, written

$J \text{ SAT } \mathcal{F}$

whenever for each clause $\vec{R} \vdash \gamma^\Gamma \sigma(P_1, \ldots, P_n) : \vec{T}$ iff $\phi$ in $\mathcal{F}$, and for each program $\sigma(P'_1, \ldots, P'_n)$ in the syntactic category:

$J \left( \vec{R} \vdash \gamma^\Gamma \sigma(P'_1, \ldots, P'_n) : \vec{T} \right) \iff J \models (\phi[P'_1/P_1 \ldots P'_n/P_n])$

Thus Definition 4.1 merely says that $J$ satisfies all clauses in $\mathcal{F}$. Definition 4.2 is particularly pleasant to work with if there is an optimal way of constructing $J$ from $I$; we explore this below.

**Definition 4.3. Well-defined Flow Logic.** We shall say that $\mathcal{F}$ is well-defined whenever $[\mathcal{F}, I] \text{ SAT } \mathcal{F}$ for all interpretations $I$; here $[\mathcal{F}, I]$ is the interpretation defined by

$[\mathcal{F}, I] = \bigsqcup \{ J \mid J \text{ SAT } \mathcal{F} \land J \upharpoonright_{\mathcal{R}_B} = I \}$

We then have:

**Theorem 4.4.** For a well-defined Flow Logic $\mathcal{F}$ the condition $I \text{ sat } \vec{R} \vdash \gamma^\Gamma P : \vec{T}$ is equivalent to $[\mathcal{F}, I] \left( \vec{R} \vdash \gamma^\Gamma P : \vec{T} \right)$.

**Two Approaches to Well-Definedness.** To show that the Flow Logic $\mathcal{F}$ is well-defined it is sufficient to ensure that the set $\{ J \mid J \text{ SAT } \mathcal{F} \land J \upharpoonright_{\mathcal{R}_B} = I \}$ is a singleton set. This means that there is only one way to extend $I$ to an interpretation $J$ for the Flow Logic $\mathcal{F}$. This is the case whenever the following holds:

for all judgements $\vec{R} \vdash \gamma^\Gamma P : \vec{T}$ of $\mathcal{F}$ and for all concrete programs $P'$ of the corresponding syntactic category: the formula $\vec{R} \vdash_{\vec{p}} \Gamma P' : \vec{T}$ can be finitely unfolded (using the clauses of $\mathcal{F}$) to a formula $\phi$ that is based on $\mathcal{R}_B$ only.

The notion of unfolding is as in Table IV and the above procedure succeeds whenever $\mathcal{F}$ is a compositional Flow Logic. It also succeeds when the syntactic parts of judgements on the right hand sides of clauses are proper subformulae of the one on the left hand side — and even more generally, whenever they are strictly smaller in some well-founded order.

Another approach to ensuring well-definedness is to rely on fixed point theory. The Flow Logic $\mathcal{F}$ and an interpretation $I$ can be seen as defining an operator $\langle \mathcal{F}, I \rangle$ over the complete lattice of interpretations; it is given by

$\langle \mathcal{F}, I \rangle(J) = J'$
where

$$J'(R) = \begin{cases} \mathcal{I}(R) & \text{if } R \in \mathcal{R}_B \\ \mathcal{J} = \phi[P'_1/P_1 \cdots P'_n/P_n] & \text{if } R \text{ is } \bar{R} \vdash_{\bar{p}}^\gamma, \bar{T} \sigma(P'_1, \ldots, P'_n) : \bar{T} \text{ and } \mathcal{F} \text{ contains } \bar{R} \vdash_{\bar{p}}^\gamma, \bar{T} \sigma(P_1, \ldots, P_n) : \bar{T} \text{ iff } \phi \end{cases}$$

This definition is unambiguous because $\mathcal{F}$ contains exactly one clause with a left hand side involving the syntactic operator $\sigma$. It follows that $J' \text{ SAT } \mathcal{F} \land J |_{\mathcal{R}_B} = \mathcal{I}$ is equivalent to $J = \langle \langle \mathcal{F}, \mathcal{I} \rangle \rangle(J)$ and hence that $[\mathcal{F}, \mathcal{I}](J) = \bigcup \{ J \mid J = \langle \langle \mathcal{F}, \mathcal{I} \rangle \rangle(J) \}$ is the least upper bound of all fixed points of $\langle \langle \mathcal{F}, \mathcal{I} \rangle \rangle(J)$. To ensure that this is itself a fixed point it suffices to ensure that $\langle \langle \mathcal{F}, \mathcal{I} \rangle \rangle$ is monotonic because then Tarski’s fixed point theorem [Tarski 1955] ensures the existence of a complete lattice of fixed points and in particular a greatest fixed point. Monotonicity of $\langle \langle \mathcal{F}, \mathcal{I} \rangle \rangle$ is ensured when all judgements (i.e. predicates of $\mathcal{R}_J$) occurring on the right hand side of the clauses of the Flow Logic occur in positive positions only. This is known as a co-inductive definition and intuitively says that interpretations are only rejected if they explicitly violate the Flow Logic (as opposed to being admitted if they explicitly satisfy the Flow Logic). As an example, if the specification is extremely underspecified to the extent of saying that $R \vdash_{\bar{p}}^\gamma, \bar{T} \sigma(P_1, \ldots, P_n) : \bar{T}$ then the co-inductive interpretation admits all analysis estimates, whereas the inductive interpretation (corresponding to the least fixed point) would admit no analysis estimate.

The two approaches actually coincide in the sense that they both designate the greatest fixed point of a functional and hence both can be viewed as co-inductive definitions. In the first approach this is obvious because there only is one fixed point, and in the second approach it is due to Tarski’s fixed point theorem [Tarski 1955]. This is one place where our approach differs from that of Type Systems that generally favour the inductive rather than the co-inductive interpretation of clauses. For process calculi, co-inductive definitions may be needed when there are higher-order features, such as processes that can be communicated over channels, or named process constants as in let $A(x) = P$ in $Q$, whereas for simpler process calculi using replication instead of recursion (like the $\pi$-calculus) one can usually dispense with co-induction.

### 4.6 The Subject Reduction Result

So far in this section we have ignored the semantics of our programming notation. Clearly the acceptability of analysis estimates, i.e. interpretations $\mathcal{I}$, must be related to the semantics in a suitable way. For process calculi it is customary to use a semantics based on (possibly labelled) transition systems, perhaps in the form of a structural operational semantics, perhaps in the form of a reaction semantics, and possibly the semantics is based on a structural congruence.

In case there is a structural congruence, $P_1 \equiv P_2$, we usually want to ensure that structurally congruent processes cannot be differentiated from the point of view of the static analysis. This would suggest that we should establish the following property:
If $P_1 \equiv P_2$ then $\mathcal{I} \text{ sat } \bar{R} \vdash_{\nu_p}^{\gamma, \bar{T}} \ P_1 : \bar{T} \iff \mathcal{I} \text{ sat } \bar{R} \vdash_{\nu_p}^{\gamma, \bar{T}} \ P_2 : \bar{T}$.

However, this may be problematic because the structural congruence usually makes use of $\alpha$-renaming of names while the Flow Logic incorporates names into the predicates constituting the analysis estimate; hence the property does not hold as it stands. Our proposed solution is to view the countably infinite set of names as being a countable union of pairwise disjoint sets of countably infinite names each having a canonical representative; the canonical representative for the set of names in which $n$ is a member is denoted $\lfloor n \rfloor$. Without loss of generality the structural congruence is modified so as to used disciplined $\alpha$-renaming (e.g. [Bodei et al. 2001b]) where $n$ can only be replaced by $m$ if $\lfloor n \rfloor = \lfloor m \rfloor$. It is then possible to ignore the issue in the Flow Logic specification itself by simply ensuring that it is only invoked on syntactic elements where all names have been replaced by their canonical representative. We have written $\lfloor P \rfloor$ for the result of renaming $P$ in this way and the property then reads:

Property 4.5. Structural Congruence. Assume that $P_1 \equiv P_2$. Then $\mathcal{I} \text{ sat } \bar{R} \vdash_{\nu_p}^{\gamma, \bar{T}} \lfloor P_1 \rfloor : \bar{T}$ if and only if $\mathcal{I} \text{ sat } \bar{R} \vdash_{\nu_p}^{\gamma, \bar{T}} \lfloor P_2 \rfloor : \bar{T}$.

Lemma 2.4 is an instance of this property whereas Lemma 3.15 is a variation of it. The proof is usually by induction on the proof tree establishing $P_1 \equiv P_2$.

For an unlabelled transition system, $P_1 \rightarrow P_2$, the semantic correctness statement often takes the form of a subject reduction result as borrowed from Type Systems. This merely says that the analysis information is preserved under the semantics:

Property 4.6. Subject Reduction. If $P_1 \rightarrow P_2$ and $\mathcal{I} \text{ sat } \bar{R} \vdash_{\nu_p}^{\gamma, \bar{T}} \lfloor P_1 \rfloor : \bar{T}$ then $\mathcal{I} \text{ sat } \bar{R} \vdash_{\nu_p}^{\gamma, \bar{T}} \lfloor P_2 \rfloor : \bar{T}$.

Theorems 2.6, 3.2 and 3.9 are all instances of this property whereas Theorems 3.4 and 3.16 are variations of it. The proof is usually by induction on the proof tree establishing $P_1 \rightarrow P_2$.

For a labelled transition system we need to add information about the relationship of the labels to the analysis information. In the case of output labels this usually gives rise to extra conclusions in the subject reduction result. In the case of input labels (in particular input from the environment as in the case of open systems) this usually gives rise to extra assumptions in the subject reduction result.

4.7 The Adequacy Result

Semantic soundness of an analysis is often seen as composed of two components. One is the subject reduction result saying that the information is preserved during evaluation as discussed above. The other is an adequacy result saying that the analysis provides a decidable approximation (the static property) to a usually undecidable property expressed directly in terms of the semantics (the dynamic property). Clearly the details depend greatly on the applications that one has in mind ranging from identifying dead code to ensuring the correct delivery of messages.

In this paper we have given several examples of adequacy results namely Theorems 2.7, 2.8 and 2.9. Often they take the form of using an “error component”
among the $\overline{T}$ in $\overline{R} \vdash_{P}^{\gamma, \Gamma} P : \overline{T}$ (e.g. $\psi$). The error component is used to report any offending behaviour as regards the static property. The adequacy result then states than an empty error component ensures the dynamic property.

We have experimented with a different way of working with “error components” [Nielson and Nielson 2007a] (see also [Bodei et al. 2007]). This has taken the form of working with a variation of Alternation-free Least Fixed Point Logic where predicates can be annotated with exclamation marks in some of their positive occurrences. For each predicate $R$ we then have an error predicate $E_R$ and whenever the Flow Logic contains a literal $R!(\overline{u})$ it is treated as standing for $\neg R(\overline{u}) \Rightarrow E_R(\overline{u})$ (possibly incorporating additional entities from the context). This results in a more prescriptive flavour of Flow Logics than the usual descriptive one — from a technical point of view it is essential to make use of stratification as to be discussed in Subsection 4.8 when dealing with the Moore Family property.

4.8 The Moore Family Result

Just as Flow Logic borrows a lot from Type Systems in the distinction between acceptability of analysis estimates and the computation of best analysis estimates, it also borrows a lot from Abstract Interpretation as regards how to characterise best analysis estimates. Analysis estimates $I(\overline{R}, \overline{T})$ are elements of a complete lattice and we want to ensure that there is a least (and hence unique) analysis estimate that is admissible for a process $P$ with respect to the Flow Logic $F$.

We formulate this as the property that the set of admissible analysis estimates $A_{\gamma, \Gamma, P} = \{I(\overline{R}, \overline{T}) \mid I \text{ sat } \overline{R} \vdash_{P}^{\gamma, \Gamma} P : \overline{T}\}$ constitutes a Moore Family for all choices of $P$, $\gamma$ and $\Gamma$. Recall that a Moore Family (e.g. [Nielson et al. 1999]) is a set $A$ such that $\cap Y \in A$ whenever $Y \subseteq A$; consequently $\cap A$ will be the desired analysis estimate. This is a mere proof of existence — the situation in which it is computable is dealt with below in Subsection 4.9. In the world of type systems this corresponds to the existence of principal typings.

Property 4.7 Moore Family. $\{I(\overline{R}, \overline{T}) \mid I \text{ sat } \overline{R} \vdash_{P}^{\gamma, \Gamma} P : \overline{T}\}$ is a Moore Family for all choices of $P$, $\gamma$ and $\Gamma$.

Proposition 2.10 is an instance of this property.

The Moore Family property does not hold for arbitrary first order flow logics. However, in the same way that a semantics-directed approach to the formulation of the Flow Logic greatly facilitates the ease with which a correct Flow Logic can be developed, the use of appropriate fragments of First Order Logic greatly facilitates the ease with which a Flow Logic satisfying the Moore Family property can be developed. It is here we have found the use of Alternation-free Least Fixed Point Logic, ALFP (see Subsection 4.3) very useful: when the clauses are expressed in ALFP the Moore Family property is usually straightforward to prove.

The proof of the Moore Family property generally follows the same proof strategy showing that $\overline{R} \vdash_{P}^{\gamma, \Gamma} P : \overline{T}$ was well-defined in the first place. In the case of compositional definitions this is a straightforward inductive proof; in other cases some kind of well-founded induction might suffice, whereas, in the case of co-inductive definitions, a proof by co-induction is called for.

It is possible to formulate large parts of Abstract Interpretation, in particular the considerations of Galois connections in this setting, but we shall refrain from
embarking upon this here (e.g. [Hansen et al. 1999]).

4.9 Implementation

Whenever possible, perhaps after several transformations on the Flow Logic specification, we would like to obtain a specification such that:

1. it stays within the fragment of First Order Logic known as ALFP (as motivated above),
2. it is compositional (so that no co-induction is needed in order to ensure well-definedness),
3. it is verbose (so that there are no existentially quantified second-order predicates – or they arise in a very benign manner only), and
4. it uses a flat universe of discourse whose size is proportional to the program being analysed.

In this case the analysis of a program gives rise to a formula in ALFP over a flat universe of discourse. The formula is linear in the size of the program analysed (viewing the size of the Flow Logic itself as being of constant size since it is fixed for each analysis performed).

This approach gives us a number of benefits. One is that there is a very direct algorithm for computing the least solutions. Our own Succinct Solver [Nielson et al. 2002b] can deal with all of ALFP, while packages such as Datalog and XSB Prolog can deal with fragments of ALFP that still extend Horn clauses.

Another main benefit is that there is no complexity gap between verifying whether or not \( I(\vec{R}, \vec{T}) \) is an admissible analysis estimate and computing the least, admissible analysis estimate [McAllester 1999]. This is unlike the situation in Type Systems where often type checking is doable in polynomial time whereas type inference may take exponential time (or be hard for non-deterministic polynomial time).

A final benefit is that it is very easy to give a useful upper bound on the worst-case time complexity. If the program \( P \) is of size \( n \) the logical formula will often be of size \( O(n) \) as will the universe of discourse. In this case the computational complexity of either of the two problems discussed above is bounded by \( O(n^{r+1}) \) where \( r \) is the maximal nesting depth of quantifiers in the logical formula generated for the judgement \( \vec{R} \vdash _p ^\gamma \vec{T} : \vec{T} \) for the program \( P \). While \( r \) may in the worst-case be proportional to \( n \) it is often a small constant. This is the case whenever the Flow Logic specification contains no definition whose right hand side contains any judgements within the scope of a quantifier; in this case \( r \) can be determined as the maximum nesting depth of quantifiers on the right hand sides of clauses in the Flow Logic. This is the case for all the Flow Logics of this paper as can easily be seen by inspecting Tables III, V, VII, VIII and X. The reason is that each occurrence of \( \forall i \in I \) in the clauses named \([\text{sum}]\) merely is a shorthand for the clause to be generated for \( \Sigma _{i \in I} \pi _i . P _i \).

**Property 4.8 Polynomial Time Analysis.** Given a Flow Logic \( \mathcal{F} \) fulfilling conditions (1)–(4) from above and a program \( P \) and universe \( \mathcal{U} \) both of size \( n \). Assume that the maximum nesting depth of quantifiers in any clause of \( \mathcal{F} \) is \( r \) and that no clause contains any judgements within the scope of a quantifier. Then the solution guaranteed by Theorem 4.7 can be found in time \( O(n^{r+1}) \).
For the simple class of context independent analyses illustrated in Section 2, \( r \) takes the value 2, giving an overall cubic time bound on the complexity. We refer to [Nielson et al. 2002b] for an example proof of this property.

### 4.10 Pragmatics

As is clear from the development in the previous sections it is helpful to allow a few abbreviations that can all be expanded into first order logic as part of the algorithm in Table IV. We often write \( R \) we write \( R \) in Table IV. We often write abbreviations that can all be expanded into first order logic as part of the algorithm.

As is clear from the development in the previous sections it is helpful to allow a few abbreviations that can all be expanded into first order logic as part of the algorithm.

It is worth pointing out that this allows their use in inclusions and that they can usually be expanded away as follows: \( R_1[R_2] \subseteq R_3 \) is equivalent to \( \forall \vec{v} \in R_2: R_1(\vec{v}, \vec{w}) \subseteq R_3 \), and \( R_1 \subseteq R_2[R_3] \) is equivalent to \( \forall \vec{v} \in R_3: R_1(\vec{v}, \vec{w}) \subseteq R_2 \). (Clearly the inclusions can be expanded away as well.)

Clearly the above account of First Order Flow Logic can be generalised in many ways. The first order logic can have a multi-sorted universe of discourse. The first order logic itself could be multi-sorted, e.g. to distinguish between positive and negative positions (as is in fact the case in ALFP); the first order logic could be replaced by a second order logic (in extension of our implicit use of second order existential quantification for succinct Flow Logics) as might be needed to deal with so-called pathway analyses [Nielson and Nielson 2009; Pilegaard et al. 2008]; we might allow more general complete lattices than the powersets in which the relations take their values. Finally, the first order logic can be replaced by a modal logic [Bolander and Hansen 2007]. The list is endless. However, it is important that we retain the desirable properties of Flow Logic: well-definedness, subject reduction, adequacy, Moore Family result, and a notion of implementability.

### 5. CONCLUSION

Flow Logic started out as an approach to the static analysis of programming languages that would be able to integrate key elements of the approaches of Data Flow Analysis, Constraint Based Analysis (in particular the propagation of constraints as expressible using inclusions or implications) and Abstract Interpretation (in particular the Moore Family property) while taking an approach sharing the philosophy of Type Systems (in particular the distinction between type checking and type inference). As illustrated in a number of papers (surveyed in [Nielson and Nielson 2002]) this has allowed the development of a number of static analyses for a variety of language paradigms (including imperative, functional, object oriented, concurrent, and distributed and mobile features) in a rather succinct way
while at the same time opening up for transferring analysis ideas between language paradigms – the need for which is illustrated by the many places where [Nielson et al. 1999] reports on independent discovery of the same basic analysis idea among different programming language communities. The logical format used for presenting specifications focuses on ensuring the implementability of the analyses — often in low polynomial (cubic) time — while ensuring semantic correctness which often takes the form of subject reduction and adequacy results. The relationship between Flow Logic and Type Systems is studied in [De Nicola et al. 2008; 2009] and the relationship between Flow Logic and Model Checking of logics in the CTL family is studied in [Nielson and Nielson 2010].

Looking beyond programming languages quite a number of papers have developed Flow Logic for the $\pi$-calculus; the aim was to establish various security properties using the information obtainable by static analysis. In Section 2 we covered the basic context independent approach while illustrating the key points of Flow Logic. Various ways of dealing with context were then considered in Section 3: reachability; which actions can interact with each other; separating different defining occurrences of the same name; and taking the relation between the values of variables into account. Subsequently we have been able to transfer our insights from analysing the $\pi$-calculus to a variety of other process calculi embodying other computational paradigms. We shall survey some of these developments below.

Distributed and mobile processes have been considered in the context of Mobile Ambients [Cardelli and Gordon 2000] where all computational processes are encapsulated in so-called ambients that may nest and move. In [Hansen et al. 1999] best described in [Nielson et al. 2002], it is demonstrated how a Flow Logic can be used to validate the security properties of a firewall modelled in this setting. Extensions of this work considered Discretionary Ambients [Nielson et al. 2004] and showed how to deal with mandatory access control in a setting that incorporates both a Bell-LaPadula model for confidentiality as well as a Biba model for integrity. Finally, a number of Flow Logics have been formulated in the context of the BioAmbients calculus [Regev et al. 2004] in order to analyse reachability properties of biological systems [Nielson et al. 2004; Pilegaard et al. 2006b].

Flow Logics for other aspects of code mobility have been developed in the context of the Kernel Language for Agents Interaction and Mobility, KLAIM [De Nicola et al. 1998], where both data and processes have mobility over a network of localised tuple-spaces. In [Tolstrup et al. 2007] it was shown how to ensure that systems conform to a notion of locality based security policies and in [Probst et al. 2007] it was shown how to assess the security threat imposed by an organisation insider. A notion of sandboxing has been considered in [Hansen et al. 2006] that addresses a version of KLAIM where remote process invocation is governed by a security policy to be enforced on the remote invocation, and the paper shows how a Flow Logic can ensure secure sandboxing. This was further extended in [Hansen et al. 2008] that developed a Flow Logic for validating the conformance of client software with respect to a license conformance policy. A comparison between Flow Logic and Type Systems for a dialect of Klaim can be found in [De Nicola et al. 2009].

Flow Logics for the analysis of cryptographic protocols and regular term languages were first considered in the context of the Spi calculus [Abadi 1999a] that
enriches the \( \pi \)-calculus with a cryptographic term language. In this context it was shown how to use Flow Logic for analysing cryptographic protocols in cubic time [Nielson et al. 2002a]. Later work addressed the \( \nu \)-Spi calculus, an extension of Spi where each instance of the same encryption is guaranteed to yield a different ciphertext corresponding to the practical notion of confounders [Abadi 1999b], and [Bodei et al. 2002] showed how to adapt the Flow Logic to this setting while incorporating an attacker model of Dolev-Yao strength. This was followed by a number of contributions that addressed the LySa calculus [Bodei et al. 2003], a Spi-inspired construction that abandons the notion of named channels in favour of a single global ether and allows annotation of points of origin and destination for messages. The Flow Logics developed in [Bodei et al. 2003; 2005] addressed both cryptographically protected secrecy and authentication.

ACKNOWLEDGMENT

We should like to thank the members of the Language Based Technology Section at DTU Informatics for their collaboration on Flow Logic; in particular, Jörg Kreiker, René Rydho Hansen, Christoffer Rosenkilde Nielsen, Christian Probst and Terkel Tolstrup. A special thanks goes to our students Piotr Filipiuk, Alejandro Hernandez, Lei Song and Fuyuan Zhang for working with us on Appendix A and the corresponding proofs in Appendix B.

REFERENCES


Submitted November 2009; Revised March 2010.

Flow Logic for Process Calculi: Electronic Supplements

Hanne Riis Nielson, Flemming Nielson and Henrik Pilegaard
The Technical University of Denmark
with contributions from
Piotr Filipiuk, Alejandro Hernandez, Lei Song and Fuyuan Zhang.

A. EXTENDING THE LINGUISTIC PRIMITIVES

In Section 2 of the main paper we have shown how to develop a Flow Logic for the π-calculus and in Section 3 we have covered a number of techniques that can increase the precision of the Flow Logic. In this section we are going to illustrate how the development may be extended to address language features that go beyond those of the standard π-calculus.

We shall focus on the notions of distribution and code mobility that have been extensively investigated in the context of calculi as the Dπ-calculus [Hennessy 2007], KLAIM [Bettini et al. 2003] and Mobile Ambients [Cardelli and Gordon 2000] to name just a few. We have decided to follow the approach of the Dπ-calculus [Hennessy 2007] as it most directly extends the concepts from the π-calculus. We conclude with a few pointers to the literature. Proofs may be found in Appendix B.4.

The material reported here is collaboration between the authors of the main paper and our students Piotr Filipiuk, Alejandro Hernandez, Lei Song and Fuyuan Zhang as part of an intensive PhD course on Flow Logic.

A.1 The Process Calculus

The fundamental idea of the Dπ-calculus [Hennessy 2007] is that processes are spatially distributed over a collection of named localities. The syntax of networks, processes and actions are given by:

\[
N ::= \emptyset \mid k\{P\} \mid (\text{new}_k n)N \mid N_1 \mid N_2
\]

\[
P ::= (\text{new} n)P \mid P_1 \mid P_2 \mid !P \mid \Sigma_{i \in I} \pi_i.P_i
\]

\[
\pi ::= u(\bar{v}) \mid u(\bar{x}) \mid \text{go } u \mid \tau
\]

The networks, \( N \in \text{Net} \), can be constructed as the empty network, \( \emptyset \), a named locality, \( k\{P\} \), a localised name restriction, \( (\text{new}_k n)N \), or a distributed composition, \( N_1 \mid N_2 \), of sub-networks \( N_1 \) and \( N_2 \). The localities, \( k \), are constants, that is, \( k \in \text{Const} \). In this context code mobility is achieved by an action for process translocation, \( \text{go } u.P \), that will move the continuation \( P \) to the locality denoted by the name \( u \). The remaining constructs are much as in the π-calculus. In particular, we retain the distinction between names, \( u \), that can be either constants, \( n \), or variables, \( x \), and we continue to use \( \text{fc}(P) \) and \( \text{fv}(P) \) for the free constants, resp. free variables, of \( P \).

It is convenient to specify the behaviour of spatially distributed processes at the level of networks [Hennessy 2007; Bettini et al. 2003]. The structural congruence is the least congruence relation that satisfies the axiom schemes of Table XIII. The
Abelian monoid laws for parallel:  
\[
(N | M) \cdot K \equiv (N | M) \cdot K \\
N | M \equiv M | N \\
N | \circ \equiv N \\
\]

Scope laws:
\[
(new_k \circ n) \circ \equiv \circ \\
N | (new_k \circ n)M \equiv (new_k \circ n)(N | M) \quad \text{if } n \neq fc(N) \\
(new_k \circ n)(new_m)N \equiv (new_m)(new_k \circ n)N \\
\]

Unfolding of replication:
\[
k[!]P \equiv k[P] | k[!]P \\
\]

Process lifting:
\[
k[P] | Q \equiv k[P] | k[Q] \\
k[0] \equiv \circ \\
k[\text{new}(n)]P \equiv (\text{new}_k \circ n)k[P] \quad \text{if } k \neq n \\
\]

Summands can be freely permuted in \( \Sigma_{i \in 1, \pi_i, P} \).

### Table XIII. Structural congruence for the D\(\pi\)-calculus: \( N \equiv M \).

| \text{com} | [k[n(m)].P + P'] | [k[n(z)].Q + Q'] \rightarrow k[P] | k[Q[m/z]] | \text{if } |m| = |z| |
| \text{tau} | k[r.P + Q] \rightarrow k[P] & \text{mov} | l[go \cdot k].P \rightarrow k[P] |
| \text{equiv} | N \equiv N' | N' \rightarrow M' \rightarrow M' \equiv M |
| \text{par} | N \rightarrow N' | N | M \equiv N' | M |
| \text{nre} | (\text{new}_k \circ n)N \rightarrow (\text{new}_k \circ n)M |

\[
\text{Table XIV. Reduction relation for the D\(\pi\)-calculus: } N \rightarrow M. \\
\]

Congruence relies on the concept of \( \alpha \)-renaming defined as follows for processes
\[
(\text{new} \circ n)P \equiv_\alpha (\text{new} m)(P[m/n]) \quad \text{if } m \neq fc(P) \\
\]

and lifted to networks using the other axioms. As in the main paper, we shall later replace the notion of \( \alpha \)-renaming by the notion of disciplined \( \alpha \)-renaming.

Similarly, the reduction relation, \( N \rightarrow M \), is the least relation satisfying the rules and axioms of Table XIV. Besides of the movement rule, \([\text{mov}]\), the definition constitutes a straightforward lifting of the standard \( \pi \)-calculus reduction relation to the level of localities. Note that processes may only communicate if situated at the same locality, as expressed by \([\text{com}]\). In this way channels are local to the individual localities.

\textit{Example} A.1. Consider the following network, \( \text{N}_i \), consisting of three localities \( k, l_1 \) and \( l_2 \), and where \( Q = ((\text{new} n)c(n).c(n)) | c(x).((\text{new} m)x(m)), Q_1 = c(y).y(x), \) and \( Q_2 = c(x,y).x(y) \):
\[
k[\quad (c(l_1) + c(l_2)) | c(z).go \cdot z.Q ] \quad \mid \quad l_1[Q_1] \quad \mid \quad l_2[Q_2] \\
\]

Let us assume that the first step is a communication at locality \( k \) where \( l_1 \) is bound to \( z \); this results in the process:
\[
k[\quad go \cdot l_1.Q] \quad \mid \quad l_1[Q_1] \quad \mid \quad l_2[Q_2] \\
\]

Now the process at locality \( k \) is ready to move to locality \( l_1 \) and the process becomes
\[
l_1[P] \quad \mid \quad l_2[Q_2] \\
\]

where \( P = ((\text{new} n)c(n).c(n)) | c(x).((\text{new} m)x(m)) | c(y).y(x). \) Note that \( P \) is similar to the process \( P_\ast \) considered in Example 2.1.

As an alternative the first step of the communication at locality \( k \) could also result in \( l_2 \) being bound to \( z \):

\[
\text{\begin{align*}
\kappa[k[\text{go } l_2, Q] & | l_1][Q_1] & | l_2][Q_2]\end{align*}}
\]

In this case the process at locality \( k \) is ready to move to locality \( l_2 \) and becomes

\[
\text{\begin{align*}
l_1][Q_1] & | l_2][P']\end{align*}}
\]

where \( P' = (\text{new } n)(c(n), c(n)) | c(x), ((\text{new } m)x(m)) | c(x,y), x(y) \). Note that \( P' \) is similar to the process \( P'_e \) of Example 2.1.

A.2 The Flow Logic Specification

Abstract domains. To analyse the calculus it is convenient to use localised environments as was already explored in Subsection 3.3 of the main paper – but here the environments will be local to each locality rather than to each program point. In order to achieve this we design the analysis domains as follows:

\( -\) \( \rho : \text{Const} \times \text{Var} \rightarrow \wp(\text{Const}) \) is the abstract environment that given a locality maps a variable to the set of constants that it might be bound to.

\( -\) \( \kappa : \text{Const} \times \text{Const} \rightarrow \wp(\text{Const}^*) \) is the abstract channel environment that given a locality maps a (channel) constant to the set of sequences of constants that may be communicated over it.

\( -\) \( \psi : \wp(\text{Const} \times \text{Const}) \) is the error component that for each locality records the set of (channel) constants where there may be an arity mismatch in a communication.

We shall use notation similar to that of the main paper; in particular we let \( \rho(k,n) \) denote \( \{n\} \) whenever \( n \) is a constant.

Judgements. When developing the analysis we also have to consider networks and therefore we define three analysis judgements:

\( -\) \( \rho, \kappa \vdash_n N : \psi \) expresses that \( \rho, \kappa, \) and \( \psi \) constitute an acceptable analysis estimate for the network \( N \).

\( -\) \( \rho, \kappa \vdash^P P : \psi \) expresses that \( \rho, \kappa, \) and \( \psi \) constitute an acceptable analysis estimate for the process \( P \) when located at locality \( k \).

\( -\) \( \rho, \kappa \vdash^A \pi : \psi \& \Lambda \) expresses that \( \rho, \kappa, \psi, \) and \( \Lambda \) constitute an acceptable estimate of the effect of the action \( \pi \) if performed at locality \( k \); here \( \Lambda \subseteq \text{Const} \) is a local analysis component capturing the set of possible localities where the continuation of \( \pi \) might be executed.

Clauses. The clauses defining these judgements are shown in Table XV and will be explained below.

Firstly, a new set of clauses defines the analysis of networks. The simplest of these, \([\text{EMP}]\), states that each analysis estimate, \( \rho, \kappa, \) and \( \psi \), is valid for the empty network. The clause, \([\text{LOC}]\), for localities ensures that an estimate \( \rho, \kappa, \) and \( \psi \) is valid for \( k[ P ] \) exactly when it constitutes a valid analysis estimate for the process \( P \) in the context of locality \( k \). This ensures that each process is analysed in a locality-aware manner. As for processes we shall ignore the scoping of names; this leads
to the clause [NRE] and also the clause [DIS] for parallel composition of networks is very similar to the clause [PAR] for parallel composition of processes.

Secondly, the clauses for processes are parameterised on the name, \( k \), of the current locality. In the cases of restriction, parallel composition, and replication ([RES], [PAR], and [REP]) this is of little consequence and the clauses are practically identical to the corresponding clauses of Table III. However, in the case of guarded sum, [SUM], we have to consider that a given continuation, \( P_i \), may be moved to a new locality by the corresponding action prefix, \( \pi_i \). Thus, when analysing the action \( \pi_i \), we use the local component \( \Lambda_i \) to collect the set of localities where \( P_i \) may show up. For each locality, \( k' \), in this set we ensure that \( P_i \) is analysed in the context of \( k' \). We recall that the name binding information relevant to \( P_i \) is local to \( k \). Therefore the analysis of \( P_i \) at \( k' \) is only correct if this information is also propagated from context \( k \) to \( k' \) and we ensure this by copying the binding information relevant to the free names of \( P_i \). This selective propagation suffices because the analysis mimics a substitution semantics, meaning that the bindings of names not explicitly represented in the continuation are irrelevant. Indeed the same phenomenon is also present in the clauses of Table VIII with the main difference being that now the propagation happens in the clause for guarded sums rather than that for actions.

Table Xv. Flow Logic for the Dπ-calculus: \( \rho, \kappa \vdash_N N : \psi, \rho, \kappa \vdash_P P : \psi \), and \( \rho, \kappa \vdash_{\pi} \pi.P : \psi \& \Lambda \).

| EMP | \( \rho, \kappa \vdash_N N \otimes \psi \) \iff \text{true} |
| LOC | \( \rho, \kappa \vdash_N k \ldots P \vdash \psi \|
| NRE | \( \rho, \kappa \vdash_N (\text{new}_k n)N : \psi \) \iff \( \rho, \kappa \vdash_N N : \psi \|
| DIS | \( \rho, \kappa \vdash_N N \mid M : \psi \) \iff \( \rho, \kappa \vdash_N N : \psi_N \land \rho, \kappa \vdash_M M : \psi_M \land \psi_N \cup \psi_M \subseteq \psi \|
| RES | \( \rho, \kappa \vdash_P (\text{new}_k n)P : \psi \) \iff \( \rho, \kappa \vdash_P P \|
| PAR | \( \rho, \kappa \vdash_P P_1 \mid P_2 : \psi \) \iff \( \rho, \kappa \vdash_P P_1 : \psi_1 \land \rho, \kappa \vdash_P P_2 : \psi_2 \land \psi_1 \cup \psi_2 \subseteq \psi \|
| REP | \( \rho, \kappa \vdash_P !P : \psi \) \iff \( \rho, \kappa \vdash_P P \|
| SUM | \( \rho, \kappa \vdash_P \sum_i \pi_i.P_i : \psi \) \iff \( \forall i \in I \colon (\rho, \kappa \vdash_{\Lambda_i} \pi_i : \psi_i \& \Lambda_i \land \psi_i \subseteq \psi \land \forall k' \in \Lambda_i : (\rho, \kappa \vdash_{\Lambda_i}^{k'} P_i : \psi_{i,k'} \land \psi_{i,k'} \subseteq \psi \land \forall z \in \psi(P_i) : \rho(k,z) \subseteq \rho(k',z)) \)
| OUT | \( \rho, \kappa \vdash_{\Lambda} u(\bar{v}) : \psi \& \Lambda \) \iff \( (\forall n \in \rho(k,u) : \rho(k,v) \subseteq \kappa(k,n)) \land k \in \Lambda \|
| IN | \( \rho, \kappa \vdash_{\Lambda} u(\bar{x}) : \psi \& \Lambda \) \iff \( (\forall n \in \rho(k,u) : \kappa(k,n) \cap \text{Const}(\bar{v}) \subseteq \rho(k,v) \land \kappa(k,n) \setminus \text{Const}(\bar{v}) \neq \emptyset \Rightarrow (k,n) \in \psi) \land k \in \Lambda \|
| TAU | \( \rho, \kappa \vdash_{\Lambda} \tau : \psi \& \Lambda \) \iff \( k \in \Lambda \|
| GO | \( \rho, \kappa \vdash_{\Lambda} \text{go} u : \psi \& \Lambda \) \iff \( \rho(k,u) \subseteq \Lambda \|

the set of localities where the continuation might be executed. Much as in Section 2 we use the notation \( \rho(k, (v_1, \ldots, v_m)) \) as a shorthand for \( (\rho(k, v_1), \ldots, \rho(k, v_m)) \) and we write \( \kappa(k, n) \cap \text{Const}^{[\rho]} \subseteq \rho(k, (x_1, \ldots, x_m)) \) as a shorthand for \( \forall (o_1, \ldots, o_m) \in \kappa(k, n) \cap \text{Const}^{[\rho]} : \bigwedge_{i=1}^{m} o_i \in \rho(k, x_i) \). In the case of the silent action or the communication actions we simply require that \( k \in \Lambda \) corresponding to the process not moving. In the case of a \text{go u} action we ensure that all potential destinations are captured by \( \Lambda \) by requiring that \( \rho(k, u) \subseteq \Lambda \).

Example A.2. Let us return to the network, \( N_* \), of Example A.1. Taking \( \rho, \kappa \) and \( \psi \) as follows

\[
\begin{array}{c|c|c|c}
  l & \rho(l, x) & \rho(l, y) & \rho(l, z) \\
  k & 0 & 0 & \{l_1, l_2\} \\
  l_1 & \{m, n\} & \{n\} & 0 \\
  l_2 & \{n\} & \{m\} & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
  l & \kappa(l, c) & \kappa(l, m) \\
  k & \{l_1, l_2\} & 0 \\
  l_1 & \{n\} & \{m\} \\
  l_2 & \{n\} & \{m\} \\
\end{array}
\]

\[
\psi = \begin{bmatrix} k & l_1 & l_2 \\
  0 & 0 & \{c\} \end{bmatrix}
\]

we obtain an acceptable analysis result as can easily be verified from the clauses of Table XV. This is the most precise analysis result for \( N_* \), and it correctly captures that \( z \) may be bound to \( l_1 \) as well as \( l_2 \). The former causes the analysis result for locality \( l_1 \) to correspond to that for the process \( P_* \) of Example 2.2. Similarly, the latter causes the analysis result for locality \( l_2 \) to correspond to that for the process \( P'_* \) of Example 2.2.

Note that the analysis correctly determines that the bindings of \( z \) are no longer relevant after the move to \( l_1 \) or \( l_2 \). Furthermore, the \( \psi \) component correctly reflects that the process may indeed become stuck when trying to communicate over \( c \) at locality \( l_2 \). \( \square \)

Well-definedness. As the analysis specification is syntax-directed it is straightforward to establish that it is well-defined.

A.3 The Subject Reduction and Adequacy Results

Subject Reduction. Semantic correctness means that the judgment \( \rho, \kappa \vdash_N N : \psi \) correctly captures the behaviour of the network \( N \). We express this as a subject reduction result meaning that, if we have an analysis result for \( N \) and if \( N \) evolves into some network \( M \), then the very same analysis result is also valid for \( M \) and all networks that are obtained from \( M \) by disciplined \( \alpha \)-renaming.

The notion of disciplined \( \alpha \)-renaming for processes takes the same form as in the main paper:

\[
\begin{array}{c}
\text{(new } n) P \equiv_\alpha (\text{new } m) P[m/n] \quad \text{iff } m \notin \text{fc}(P) \land |n| = |m| \\
\end{array}
\]

For networks it amounts to:

\[
\begin{array}{c}
\text{(new } k \ n) N \equiv_\alpha (\text{new } k \ m) N[m/n] \quad \text{iff } m \notin \text{fc}(N) \land m \neq k \land |n| = |m| \\
\end{array}
\]

Here the \( |n| \) is the canonical name of \( n \). It is assumed that there is infinite supply of names having the same canonical name. In the following we shall replace the law for \( \alpha \)-renaming in Table XIII with the above law for disciplined \( \alpha \)-renaming.

The clauses specifying the analysis judgments refer to the constants of the initial network \( N_* \). Hence, for all constants \( n \) occurring in \( N_* \), we have \( n = |n| \). Similarly,
for all variables $x$, we take $x = |x|$. Generally, the operation $|\cdot|$ is lifted in the point-wise manner for all syntactic categories.

We rely on the following fact, which expresses that it is possible to enlarge the error component when we have a valid analysis either for a process or for a network:

**Fact A.3. Monotonicity.** The two following statements hold:

1. If $\rho, \kappa \vdash_P [P] : \psi$ and $\psi \subseteq \psi'$ then $\rho, \kappa \vdash_P [P] : \psi'$;
2. If $\rho, \kappa \vdash_N [N] : \psi$ and $\psi \subseteq \psi'$ then $\rho, \kappa \vdash_N [N] : \psi'$.

For proving the subject reduction result, we shall first establish the following results (proved in Appendix B.4):

**Lemma A.4. Structural Congruence.** If $M = N$ then $\rho, \kappa \vdash_N [N] : \psi$ if and only if $\rho, \kappa \vdash_M [M] : \psi$.

**Lemma A.5. Substitution.** If $\rho, \kappa \vdash^k_P [P] : \psi$ and $[m] \in \rho(k, y)$ then $\rho, \kappa \vdash^k_P [P[m/y]] : \psi$.

We are now ready to state the subject reduction result:

**Theorem A.6 Subject Reduction.** If $N \rightarrow_M$ and $\rho, \kappa \vdash_N [N] : \psi$ then $\rho, \kappa \vdash_M [M] : \psi$.

**Adequacy.** The adequacy results considered in Section 2.4 generalise to the present development. The notions of dynamic and static well-behavedness are easily generalised by considering the communications taking place in all localities. Similarly, the notions of dynamic and static well-sortedness can be extended to checking the sorting information for communications in all localities. The generalisation of the notion of non-leaking processes is slightly more interesting as we may associate security levels with the localities and thereby impose conditions on the permitted communications.

**Non-leaking networks.** As in Subsection 2.4 of the main paper we shall, for the sake of simplicity, assume that we have just two security levels, and take $\text{LEVEL} = \{\text{low, high}\}$ where the ordering $\subseteq$ satisfies $\text{low} \subseteq \text{high}$. The mapping $\Xi : \text{Const} \rightarrow \text{LEVEL}$ associates a security level with each constant and we shall require that if $[n] = [m]$ then also $\Xi(n) = \Xi(m)$ meaning that the security levels are preserved by disciplined $\alpha$-renaming.

We want to impose the security policy that a process that has been running in a high locality cannot move into a low locality and perform communications. The rationale is that the process might leak information at the low locality that it obtained while it was at the high locality. Indeed, we shall allow a process to perform the move as long as it does not engage in any communications. As an example, if $\Xi(k) = \text{high}$ and $\Xi(l) = \text{low}$ then the network $k[\text{go}].0$ will be accepted whereas the network $k[\text{go}].n(m)$ will be rejected unless both $\Xi(n) = \text{low}$ and $\Xi(m) = \text{low}$. More precisely, we shall only allow information to be sent on a channel if either the channel or the locality of the communication has at least the security level of the information being sent.

We shall now define that $N_n$ is **dynamically non-leaking** if whenever $N_n$ evolves into a network that is structurally congruent to one of the form $C[k[n(m)]].P + P' \equiv [k[n(\bar{x})].Q + Q']$ then $|\bar{m}| = |\bar{x}|$ as well as $\Xi(\bar{m}) \subseteq (\Xi(n) \cup \Xi(k))^{\bar{m}}$. In the
case where \( \Xi(n) = \Xi(k) = \text{low} \) this ensures that all the constants of \( \vec{m} \) will be of low security level and when either \( \Xi(n) = \text{high} \) or \( \Xi(k) = \text{high} \) it does not impose any restrictions.

We shall say the network \( N_* \) is \textit{statically non-leaking} if there exists \( \rho \) and \( \kappa \) such that \( \rho,\kappa \vdash_N N_* : \emptyset \) and furthermore for any \( k, n \) and \( \vec{m} \in \kappa(k,n) \) we have \( \Xi(\vec{m}) \subseteq (\Xi(n) \sqcup \Xi(k))|[^{\vec{m}}] \).

We then have:

\textbf{Theorem A.7. Adequacy for non-leaking networks.} If the network \( N_* \) is \textit{statically non-leaking} then it is also \textit{dynamically non-leaking}.

\textit{Example A.8.} The network of Example A.1 is not statically non-leaking since it reports a potential arity mismatch. We shall therefore consider a slight variation of it

\[ k[[(c(l_1) + c(l_2)) \mid c(z).go \in Q] \mid l_1[\in Q_1] \mid l_2[\in Q_2] \]

where \( Q = ((\text{new} n)c(n).c(n)) \mid c(x).(\text{new} m)x(m)), Q_1 = c(y).y(x), \text{and } Q_2 = c(x).x(y). \) Let us assume that \( k,l_1 \) and \( n \) have security levels high and that \( c,l_2 \) and \( m \) have security level low.

The following is an acceptable analysis result for the network:

\[
\begin{array}{c|c|c|c|c|}
\rho & \rho & \rho & \kappa & \psi \\
\hline
\kappa & \emptyset & \emptyset & \{l_1,l_2\} & \emptyset \\
\kappa & \emptyset & \{n\} & \emptyset & \emptyset \\
\kappa & \emptyset & \emptyset & \{m\} & \emptyset \\
\end{array}
\]

It is now easy to see that the network is \textit{not} statically non-leaking since \( n \in \kappa(l_2,c) \) and \( \Xi(n) \not\subseteq (\Xi(l_1) \cup \Xi(c)) \) reflecting that the low channel \( c \) is used to communicate the high information \( n \) in the low locality \( l_2 \). If we modify the security mapping \( \Xi \) to have \( \Xi(c) = \text{low} \), then the network is \textit{statically non-leaking}, and therefore also dynamically non-leaking. \( \square \)

\textbf{A.4 The Moore Family Result and Implementation}

\textit{Moore Family.} The theorems presented so far have been concerned with interpreting the meaning of a valid analysis result for a network \( N \) in relation to its semantics and some property of interest. However, can we be sure that \( N \) has any valid analysis results at all? And if it has more than one valid analysis result, is there a “best” one? These questions will be answered in the affirmative by the Moore Family result, also known as the \textit{model intersection} property.

Formally, a \textit{Moore Family} is a subset \( \mathcal{Y} \) of a complete lattice that is closed under greatest lower bounds, that is, it satisfies \( \forall Y \subseteq \emptyset : \cap Y \in \mathcal{Y} \). We have:

\textbf{Proposition A.9. Moore Family.} The set \( \{(\rho,\kappa,\psi) \mid \rho,\kappa \vdash_N N : \psi\} \) is a Moore Family for all networks \( N \).

This result has the interesting corollary that all networks have a \textit{least}, or best, analysis result. This follows by observing that

\[ \{((\rho,\kappa,\psi) \mid \rho,\kappa \vdash_N N : \psi)\} \subseteq \{(\rho,\kappa,\psi) \mid \rho,\kappa \vdash_N N : \psi\} \]

and therefore \( \cap \{(\rho,\kappa,\psi) \mid \rho,\kappa \vdash_N N : \psi\} \) will be a valid analysis result for \( N \) and clearly it is the least such analysis result for \( N \).
Let's consider the following network, $N$:

$$\rho, \kappa \vdash_N N : \psi \iff \phi$$

The algorithm has two steps that we explain in detail below: (1) unfold the Flow Logic for a network $N$ into a formula $\phi$, and (2) compute the most precise among the acceptable models for $\phi$ using an appropriate solver.

**Unfolding the Flow Logic specification.** Algorithmically the unfolding is performed by the algorithm presented in Table IV. Based on the finiteness of the syntactic representation of network $N$ and the fact that the developed Flow Logic specification is syntax-directed, it follows that the algorithm terminates.

**Example A.10.** Let's consider the following network, $N_*$,

$$k[ (c(l_1) + c(l_2)) | c(z) \text{go} z : Q ]$$

where $Q = ((\text{new} n) c(n)) | c(x)$. Applying the algorithm from Table IV to the judgment $\rho, \kappa \vdash_N N_* : \psi$ we get the following formula:

$$\begin{align*}
\rho, \kappa \vdash_{A} c(l_1) & : \psi_{11} & \land \psi_{11} & \subseteq \psi_1 \land \\
\rho, \kappa \vdash_{A} c(l_2) & : \psi_{12} & \land \psi_{12} & \subseteq \psi_1 \land \\
\rho, \kappa \vdash_{A} c(z) & : \psi_{21} & \land \psi_{21} & \subseteq \psi_2 \\
\forall k' \in \Lambda_{21} : (\rho, \kappa \vdash_{A} \text{go} z : Q : \psi_{21,k'} & \land \psi_{21,k'} & \subseteq \psi_2 & \land \\
\forall z \in \text{fv}(Q) : \rho(k, z) \subseteq \rho(k', z)) & \land \\
\psi_1 & \cup \psi_2 & \subseteq \psi
\end{align*}$$

Unfolding the judgment $\rho, \kappa \vdash_{A} \text{go} z : Q : \psi_{21,k'}$ results in:

$$\begin{align*}
\rho, \kappa \vdash_{A} \text{go} z : Q & : \psi_{22} & \land \psi_{22} & \subseteq \psi_{21,k'} \\
\forall k'' \in \Lambda_{22} : (\rho, \kappa \vdash_{A} Q : \psi_{22,k''} & \land \psi_{22,k''} & \subseteq \psi_{21,k'} & \land \\
\forall z \in \text{fv}(Q) : \rho(k', z) & \subseteq \rho(k'', z))
\end{align*}$$

Finally consider the judgment $\rho, \kappa \vdash_{A} Q : \psi_{22,k''}$, which is unfolded as follows:

$$\rho, \kappa \vdash_{A} c(n) : \psi_{231} & \land \rho, \kappa \vdash_{A} c(x) : \psi_{232} & \land \psi_{231} \cup \psi_{232} \subseteq \psi_{22,k''}$$

Gathering all auxiliary results we get the following judgments for actions and set inclusions:

$$\begin{align*}
\rho, \kappa \vdash_{A} c(l_1) & : \psi_{11} & \land \Lambda_{11} \\
\rho, \kappa \vdash_{A} c(l_2) & : \psi_{12} & \land \Lambda_{12} \\
\forall k' \in \Lambda_{21} : (\rho, \kappa \vdash_{A} \text{go} z : \psi_{22} & \land \Lambda_{22} \\
\forall k'' \in \Lambda_{22} : (\rho, \kappa \vdash_{A} c(n) : \psi_{231} & \land \Lambda_{231} \\
\forall k''' \in \Lambda_{22} : (\rho, \kappa \vdash_{A} c(x) : \psi_{232} & \land \Lambda_{232} \\
\psi_1 & \cup \psi_2 & \subseteq \psi \\
\psi_{12} & \subseteq \psi_1 \\
\psi_{21,k'} & \subseteq \psi_2 \\
\psi_{22,k''} & \subseteq \psi_{21,k'} \\
\psi_{231} \cup \psi_{232} & \subseteq \psi_{22,k''}
\end{align*}$$

The judgments for actions can be further expanded using appropriate clauses from.

the Flow Logic specification in Table XV:

$$\forall k' \in A_{21} : \rho(k', z) \subseteq A_{22}$$

$$(\forall n \in \rho(k, c) : \rho(k, l_1) \subseteq \kappa(k, n)) \land k \in A_{11}$$

$$(\forall n \in \rho(k, c) : \rho(k, l_2) \subseteq \kappa(k, n)) \land k \in A_{12}$$

$$\forall k'' \in A_{22} : ((\forall n' \in \rho(k'', c) : \rho(k'', n') \subseteq \kappa(k'', n')) \land k'' \in A_{231})$$

$$(\forall n \in \rho(k, c) : \kappa(k, n) \cap \text{Const} \subseteq \rho(k, z) \land$$

$$\kappa(k, n) \setminus \text{Const} \neq \emptyset \Rightarrow (k, n) \in \psi_{21}) \land k \in A_{21}$$

$$(\forall k''' \in A_{22} : (\forall n \in \rho(k''', c) :$$

$$\kappa(k''', n) \cap \text{Const} \subseteq \rho(k''', x) \land$$

$$\kappa(k''', n) \setminus \text{Const} \neq \emptyset \Rightarrow (k''', n) \in \psi_{232}) \land k''' \in A_{232})$$

This completes the unfolding of the clauses.

**Solving the constraints.** Having obtained the formula $\phi$, the next step is to calculate the most precise model $(\rho, \kappa, \psi)$ that is acceptable for $\phi$. In order to do so, the formula will be transformed to an appropriate subset of first order logic, namely $ALFP$ and then it will be solved using an appropriate solver. We shall illustrate this for the above example.

**Example A.11.** The idea is to transform each component from formula $\phi$ into predicate and to rewrite the constraints into subset of first order logic. Let’s first focus on subset inclusion in form of $\psi_{11} \subseteq \psi_1$ and $\psi_1 \cup \psi_2 \subseteq \psi$ and $\rho(k, z) \subseteq A_{22}$. They will be rewritten as follows:

$$\forall k : \forall n' : \psi_{11}(k, n') \Rightarrow \psi_1(k, n')$$

$$\forall k : \forall n' : \psi_1(k, n') \lor \psi_2(k, n') \Rightarrow \psi(k, n')$$

$$\forall k : \forall n' : \rho(k, z, n') \Rightarrow \Lambda_{22}(k, n')$$

Now consider the output action

$$(\forall n \in \rho(k, c) : \rho(k, l_1) \subseteq \kappa(k, n)) \land k \in A_{11}$$

which will be rewritten as follows:

$$(\forall n : \rho(k, c, n) \Rightarrow (\forall m : \rho(k, l_1, m) \Rightarrow \kappa(k, n, m)) \land \Lambda_{11}(k))$$

Turning our attention to the clause for copying values of constants and variables into a new locality

$$\forall k' \in A_{21} : (\forall x \in \text{fv}((\text{go z}).Q) : \rho(k, x) \subseteq \rho(k', x))$$

we get the following rewritten version:

$$\forall k' : A_{21}(k') \Rightarrow (\forall x : \text{fv}((\text{go z}).Q), x) \Rightarrow (\forall n : \rho(k, x, n) \Rightarrow \rho(k', x, n)))$$

Finally consider the constraints generated for the input actions:

$$(\forall n \in \rho(k, c) :$$

$$\kappa(k, n) \cap \text{Const} \subseteq \rho(k, z) \land$$

$$\kappa(k, n) \setminus \text{Const} \neq \emptyset \Rightarrow (k, n) \in \psi_{21}) \land k \in A_{21}$$
Here we use Const as a unary predicate and rewrite the constraints as follows:

\[
\forall n : \rho(k, c, n) \Rightarrow \\
\forall m : \kappa(k, n, m) \land Const(m) \Rightarrow \rho(k, z, m) \land \\
(\exists m : \kappa(k, n, m) \land \neg Const(m)) \Rightarrow \psi_{21}(k, n) \land \Lambda_{21}(k)
\]

The conjunction of the formula obtained in this way is in ALFP and the most precise among the acceptable analysis results is as follows:

\[
\begin{array}{c|c|c}
 l & \rho(l, x) & \rho(l, z) \\
\hline
 k & \emptyset & \emptyset \\
 l_1 & \{n\} & \emptyset \\
 l_2 & \{n\} & \emptyset \\
\end{array}
\quad
\begin{array}{c|c|c}
 l & \kappa(l, c) \\
\hline
 k & \{l_1, l_2\} \\
 l_1 & \{n\} \\
 l_2 & \{n\} \\
\end{array}
\quad
\begin{array}{c|c|c}
 l & \psi \\
\hline
 k & \emptyset & \emptyset & \emptyset \\
 l_1 & \emptyset & \emptyset & \emptyset \\
 l_2 & \emptyset & \emptyset & \emptyset \\
\end{array}
\]

This completes the treatment of the example. \qed

An implementation of the analysis following the above description has a worst-case time complexity considerably higher than the one estimated in Section 4.9 because the specification in Table XV contains analysis judgments inside the scope of quantifiers and therefore quantifiers can be nested arbitrarily deep once the clauses are unfolded. Using the techniques to estimate the worst-case time complexity may result in an exponential rather than polynomial bound.

One way to reduce the worst-case time complexity is to change the format of judgments for processes, \(\rho, \kappa \vdash^P_k P : \psi\), and actions, \(\rho, \kappa \vdash^A_\pi \pi : \psi \land \Lambda\), such that they take the form

\[
\rho, \kappa \vdash^{A'}_P P : \psi \quad \rho, \kappa \vdash^{A'}_A \pi : \psi \land \Lambda
\]

where \(A'\) is a set of localities (including \(k\)) where the process or action may be located. This necessitates some changes to definitions of the clauses for \(\rho, \kappa \vdash^{A'}_P P : \psi\) and \(\rho, \kappa \vdash^{A'}_A \pi : \psi \land \Lambda\) but results in a somewhat coarser analysis where there is no longer arbitrarily deep nesting of quantifiers and where the worst-case time complexity can be guaranteed to be polynomial.

A.5 Bibliographical Notes

We now slightly elaborate on the bibliographic notes already contained in Conclusion of the main paper as far as mobile calculi are concerned.

Flow Logics for the analysis of spatially mobile processes were first developed in the context of Mobile Ambients [Cardelli and Gordon 2000] where all computational processes are encapsulated in ambients that may nest and move. The earliest work [Hansen et al. 1999], best described in [Nielson et al. 2002], demonstrated how a Flow Logic can be used to validate the security properties of a firewall modelled in this setting. Later work addressed the calculus of Discretionary Ambients [Nielson et al. 2004], derived from Safe Ambients [Levi and Sangiorgi 2003], and showed how to deal with mandatory access control in a setting that incorporates both a Bell-LaPadula model for confidentiality as well as a Biba model for integrity. This paper also showed how to analyse cryptographic key exchange protocols in the context of Boxed Ambients [Bugliesi et al. 2001]. Finally, a number of Flow Logics have been formulated in the context of the BioAmbients calculus [Regev et al. 2004] in order to analyse reachability properties of biological systems [Nielson et al. 2004; Pilegaard et al. 2006b].
Flow Logics for other aspects of code mobility have been developed in the context of the Kernel Language for Agents Interaction and Mobility (KLAIM) [De Nicola et al. 1998]. In contrast to the various ambient calculi, KLAIM defines a setting where both data and processes have mobility over a network of localised tuple-spaces. The earliest work in this area considered µKLAIM [Gorla and Pugliese 2003], a minimal variant, in conjunction with a notion of locality based security policies and showed how a Flow Logic can be used to ensure that systems conform to such policies [Tolstrup et al. 2007]. Another branch of work addressed acKlaim, a variant with support for access control, and showed how a Flow Logic can be used to assess the security threat imposed by an organisation insider [Probst et al. 2007]. A notion of sandboxing has been considered in [Hansen et al. 2006] that addresses myKLAIM, a variant where remote process invocation is governed by a security policy to be enforced on the remote invocation, and shows how a Flow Logic can ensure secure sandboxing. The myKLAIM theme is further investigated in [De Nicola et al. 2008] where a Flow Logic serves as inspiration for the development of a fully static type system for the calculus. Finally, a variant of KLAIM is considered in [Hansen et al. 2008], in order to develop a Flow Logic for validating the conformance of client software with respect to a licence conformance policy. A comparison between Flow Logic and Type Systems for a dialect of Klaim can be found in [De Nicola et al. 2009].

Finally, we should mention the Flow Logic of calculi where the scoping of names (in particular variables) is independent of the place (typically an input) where the names get their value [Bauer et al. 2008]. This is a phenomenon that occasionally arises in the analysis of calculi for services.

B. PROOFS

B.1 Proofs of Main Results of Section 2

Lemma 2.4 Structural Congruence. If $P \equiv Q$ then $\rho, \kappa \vdash [P] : \psi$ if and only if $\rho, \kappa \vdash [Q] : \psi$.

Proof. The proof is by induction on how $P \equiv Q$ is obtained from Table I. Only the case of disciplined $\alpha$-renaming is non-trivial. Here we assume that $(\textit{new } n)P \equiv_m (\textit{new } m)P[m/n]$ because $\textit{fc}(P)$ and $[n] = [m]$ and we shall prove that $\rho, \kappa \vdash [\textit{new } m]P[m/n] : \psi$ if and only if $\rho, \kappa \vdash [\textit{new } n]P : \psi$. However, this is straightforward since $[(\textit{new } n)P] = [(\textit{new } m)P[m/n]]$.

Lemma 2.5 Substitution. If $\rho, \kappa \vdash [P] : \psi$ then $\rho, \kappa \vdash [P[y/m]] : \psi$ provided that $[m] \in \rho(y)$.

Proof. The proof is by structural induction on $P$ and only the case $\Sigma_{i \in I} \pi_i, P_i$ is non-trivial. So assume that

$$\rho, \kappa \vdash \pi_{i'} [\psi_i'] \quad \text{and} \quad \rho, \kappa \vdash [P_i] : \psi_i$$

for some $i \in I$. It is now sufficient to show that:

1. if $y \notin \text{bv}(\pi_i)$ then $\rho, \kappa \vdash [\pi_i[y/m]] : \psi_i'$ and $\rho, \kappa \vdash [P_i[m/y]] : \psi_i$
2. if $y \in \text{bv}(\pi_i)$ then $\rho, \kappa \vdash [\pi_i[m/y]] : \psi_i'$ and $\rho, \kappa \vdash [P_i] : \psi_i$
We first observe that the second conjunct of (1) follows from the induction hypothesis and the second conjunct of (2) holds trivially. Next we observe that in both cases we have \( \rho([z[m/y]]) \subseteq \rho([z]) \) for all variables \( z \): this is immediate if \( z \neq y \), and if \( z = y \) it follows from the assumption that \( [m] \in \rho(y) \) and the convention that \( \rho([m]) = \{[m]\} \).

Let us then consider output and assume that \( \rho, \kappa \vdash \lambda u \langle \tilde{v} \rangle \vdash \psi \), that is, \( \forall n \in \rho([u]) : \rho([\tilde{v}]) \subseteq \kappa(n) \). We shall prove that

\[
\forall n \in \rho([u[m/y]]) : \rho([\tilde{v}[m/y]]) \subseteq \kappa(n)
\]

since then \( \rho, \kappa \vdash \lambda u \langle \tilde{v} \rangle[m/y] \vdash \psi \). From the above observation we get \( \rho([u[m/y]]) \subseteq \rho([u]) \) and \( \rho([\tilde{v}[m/y]]) \subseteq \rho([\tilde{v}]) \) and the result follows. Turning to inputs we assume that \( \rho, \kappa \vdash \lambda u \langle \tilde{x} \rangle \vdash \psi \), that is, \( \forall n \in \rho([u]) : (\kappa(n) \cap \mathit{Const}^{[\tilde{x}]} \subseteq \rho(\tilde{x})) \land (\kappa(n) \backslash \mathit{Const}^{[\tilde{x}]} \neq \emptyset \Rightarrow n \in \psi) \). We shall prove that

\[
\forall n \in \rho([u[m/y]]) : (\kappa(n) \cap \mathit{Const}^{[\tilde{x}]} \subseteq \rho(\tilde{x})) \land (\kappa(n) \backslash \mathit{Const}^{[\tilde{x}]} \neq \emptyset \Rightarrow n \in \psi)
\]

since then \( \rho, \kappa \vdash \lambda u \langle \tilde{x} \rangle[m/y] \vdash \psi \). As above we have \( \rho([u[m/y]]) \subseteq \rho([u]) \) and the result follows.

\begin{proposition}
THEOREM 2.6 Subject Reduction. If \( P \rightarrow Q \) and \( \rho, \kappa \vdash P : \psi \) then \( \rho, \kappa \vdash Q : \psi \).
\end{proposition}

\begin{proof}
The proof is by induction on how \( P \rightarrow Q \) is obtained using the axioms and rules of Table II and there are two non-trivial cases. First consider the communication axiom \( \mathsf{com} \) where \( \langle \tilde{m}, \sum P + P' \rangle \mid (\langle \tilde{x} \rangle Q + Q') \rightarrow P \mid Q[\tilde{m}/\tilde{x}] \) because \( |\tilde{m}| = |\tilde{x}| \). Since we assumed that

\[
\rho, \kappa \vdash \lambda \langle \tilde{m} \rangle \mid (\langle \tilde{m} \rangle . \sum P + P') \mid (\langle \tilde{x} \rangle, Q + Q') : \psi
\]

it follows from the clauses of Table III that there exist \( \psi_1, \psi_2 \) and \( \psi_2' \) such that

\[
\begin{align*}
&\neg |\tilde{m}| \in \kappa(|n|) \text{ and } \rho, \kappa \vdash P : \psi_1 \\
&\neg |\langle n \rangle | \cap \mathit{Const}^{[\tilde{x}]} \subseteq \rho(\tilde{x}), \kappa(|n|) \backslash \mathit{Const}^{[\tilde{x}]} \neq \emptyset \Rightarrow |n| \in \psi_2' \text{ and } \rho, \kappa \vdash Q : \psi_2 \\
&\neg \psi_1 \cup \psi_2' \cup \psi_2 \subseteq \psi
\end{align*}
\]

Since \( |\tilde{m}| = |\tilde{x}| \) we have \( |\tilde{m}| \in \kappa(|n|) \backslash \mathit{Const}^{[\tilde{x}]} \subseteq \rho(\tilde{x}) \) so Lemma 2.5 gives \( \rho, \kappa \vdash Q[\tilde{m}/\tilde{x}] : \psi_2 \). Thus we have \( \rho, \kappa \vdash Q[\tilde{m}/\tilde{x}] : \psi_1 \cup \psi_2 \) and Fact 2.3 gives the required \( \rho, \kappa \vdash P \mid Q[\tilde{m}/\tilde{x}] : \psi \).

The second non-trivial case \( \mathsf{eq} \) concludes \( P \rightarrow P' \) because \( P \equiv Q \), \( Q \rightarrow Q' \) and \( Q' \equiv P' \). Here the result follows directly from Lemma 2.4.

\begin{proposition}
THEOREM 2.7 Adequacy for Well-Behaved Processes. If the process \( \pi \) is statically well-behaved then it is also dynamically well-behaved.
\end{proposition}

\begin{proof}
The proof is by contradiction so assume that \( \rho, \kappa \vdash_{\pi} P_0 : \emptyset \) and that \( P_0 \rightarrow \cdots \rightarrow C[(\langle n \rangle m).R + R'] \mid (\langle n \rangle, Q + Q')] \) but \( |\tilde{m}| \neq |\tilde{x}| \). From Theorem 2.6 we get \( \rho, \kappa \vdash C[(\langle n \rangle m).R + R'] \mid (\langle n \rangle, Q + Q')] : \emptyset \) so in particular it must be the case that \( \rho, \kappa \vdash_A \emptyset \langle n \rangle m \vdash : \emptyset \) and \( \rho, \kappa \vdash_A \langle n \rangle m : \emptyset \). Thus it follows that \( |\tilde{m}| \not\in \kappa(|n|) \) and since \( |\tilde{m}| \neq |\tilde{x}| \) it must be the case that \( |n| \) is included in the error component but this is a contradiction since it is empty.

THEOREM 2.8 Adequacy for well-sorted processes. If the process \( P \) is
statically well-sorted then it is also dynamically well-sorted.

PROOF. The proof follows the same pattern as that of Theorem 2.7 and we omit
the details. \( \square \)

THEOREM 2.9 Adequacy for non-leaking processes. If the process \( P \) is
statically non-leaking then it is also dynamically non-leaking.

PROOF. The proof follows the same pattern as that of Theorem 2.7 and we omit
the details. \( \square \)

PROPOSITION 2.10 Moore Family. The set \( \{ (\rho, \kappa, \psi) \mid \rho, \kappa \vdash P : \psi \} \) is a
Moore Family for all processes \( P \).

PROOF. Let us consider an index set \( I \) and assume that
\[
\forall i \in I : \rho^i, \kappa^i \vdash P : \psi^i
\]
so that we shall prove that
\[
\bigcap_{i \in I} \rho^i, \bigcap_{i \in I} \kappa^i \vdash P : \bigcap_{i \in I} \psi^i
\]
We shall proceed by induction on \( P \). The cases of \((\text{new } n)P\) and \(!P\) are straight-
forward so let us consider \( P_1 \mid P_2 \). From \( \rho^i, \kappa^i \vdash P_1 \mid P_2 : \psi^i \) we get \( \rho^i, \kappa^i \vdash P_1 : \psi^i \)
and \( \rho^i, \kappa^i \vdash P_2 : \psi^i \) as well as \( \psi^1 \cup \psi^2 \subset \psi^i \). The induction hypothe-
sis gives \( \bigcap_{i \in I} \rho^i, \bigcap_{i \in I} \kappa^i \vdash P_1 : \bigcap_{i \in I} \psi^1 \) and \( \bigcap_{i \in I} \rho^i, \bigcap_{i \in I} \kappa^i \vdash P_2 : \bigcap_{i \in I} \psi^2 \). From
\( \psi^1 \cup \psi^2 \subset \psi^i \) (for all \( i \in I \)) we get \((\bigcap_{i \in I} \psi^1) \cup (\bigcap_{i \in I} \psi^2) \subset \psi^1 \cup \psi^2 \subset \psi^i \) for all
\( i \in I \) and thereby \((\bigcap_{i \in I} \psi^1) \cup (\bigcap_{i \in I} \psi^2) \subset (\bigcap_{i \in I} \psi^i)\) as required. This shows that
\( \bigcap_{i \in I} \rho^i, \bigcap_{i \in I} \kappa^i \vdash P_1 \mid P_2 : \bigcap_{i \in I} \psi^i \).

For the guarded sums \( \Sigma_{i \in I} \pi_i . P_i \) the result follows in a similar way once we have
established a Moore Family result for the analysis of actions. Thus we shall prove that
\( \rho^i, \kappa^i \vdash \pi : \psi^i \) holds for all \( i \in I \) then also \( \bigcap_{i \in I} \rho^i, \bigcap_{i \in I} \kappa^i \vdash \pi : \bigcap_{i \in I} \psi^i \).
The proof is by cases on \( \pi \) and for output \( u(\ell) \) we observe that if we have \( \forall n \in \rho^i(u) : \rho^i(\ell) \subseteq \kappa^i(n) \) for all \( i \in I \) then we also have \( \forall n \in \bigcap_{i \in I} \rho^i(u) : \bigcap_{i \in I} \rho^i(\ell) \subseteq \bigcap_{i \in I} \kappa^i(n) \) for all \( i \in I \) and the result follows. For input the line of reasoning is similar and we omit the details. \( \square \)

B.2 Proofs of Main Results of Subsection 3.3

LEMMA 3.7 Substitution. If \( \rho, \kappa \vdash P : \psi \) then \( \rho, \kappa \vdash P[m/y] : \psi \) provided that \( [m] \in \rho(\ell, y) \).

PROOF. The proof is by structural induction on \( P \) and only the case \( \Sigma_{i \in I} \pi_i . P_i \)
is non-trivial. As in the proof of Lemma 2.5 we assume that
\[
\rho, \kappa \vdash_{A, \ell}^{\pi_i}[\pi_i] : \psi_i \text{ and } \rho, \kappa \vdash_{P_i[m/y]}^{\lab(\pi_i, \ell)} [P_i] : \psi_i
\]
for some \( i \in I \) and we shall prove that
\begin{align*}
(1) & \quad \text{if } y \notin \text{bv}(\pi_i) \text{ then } \\
& \quad \rho, \kappa \vdash_{A, \ell}^{\pi_i[m/y]} [\pi_i[m/y]] : \psi_i \text{ and } \rho, \kappa \vdash_{P_i[m/y]}^{\lab(\pi_i, \ell)} [P_i[m/y]] : \psi_i \\
(2) & \quad \text{if } y \in \text{bv}(\pi_i) \text{ then } \\
& \quad \rho, \kappa \vdash_{A, \ell}^{\pi_i} [\pi_i[m/y]] : \psi_i \text{ and } \rho, \kappa \vdash_{P_i}^{\lab(\pi_i, \ell)} [P_i] : \psi_i
\end{align*}
This is sufficient to establish the required result.

We shall first observe that \( \rho(\ell, \lfloor z/m/y \rfloor) \subseteq \rho(\ell, z) \) for all variables \( z \); this follows from the assumption that \( \lfloor m \rfloor \in \rho(\ell, y) \) and the convention that \( \rho(\ell, \lfloor m \rfloor) = \{ \lfloor m \rfloor \} \).

To prove (1) we note that \( \text{fv}(P_i[m/y]) = \text{fv}(P_i) \setminus \{ y \} \). This suffices for showing the first conjunct (by case analysis on the form of \( \pi_i \) as in the proof of Lemma 2.5). The second conjunct holds trivially unless \( y \in \text{fv}(P_i) \). In this case we note that \( \lfloor m \rfloor \in \rho(\text{lab}(\pi_i, \ell), y) \); this is immediate except when \( \pi_i \) takes the form of an input in which case we know that \( \forall z \in \text{fv}(P_i) \setminus \text{bv}(\pi_i) : \rho(\ell, z) \subseteq \rho(\text{lab}(\pi_i, \ell), z) \) and hence that \( \lfloor m \rfloor \in \rho(\ell, y) \subseteq \rho(\text{lab}(\pi_i, \ell), y) \). The second conjunct then follows from the induction hypothesis.

To prove (2) we note that the first conjunct follows from the observation that \( \rho(\ell, \lfloor z/m/y \rfloor) \subseteq \rho(\ell, \lfloor z \rfloor) \) for all variables \( z \) (by case analysis on the form of \( \pi_i \)). The second conjunct holds trivially.

**Lemma 3.8 Relocation.** If \( \rho, \kappa \vdash^p P : \psi \) then \( \rho, \kappa \vdash^p [P] : \psi \) provided that \( \forall z \in \text{fv}(P) : \rho(\ell, z) \subseteq \rho(\ell', z) \).

**Proof.** The proof is by structural induction on \( P \) and only the case \( \Sigma_{i \in I} \pi_i, P_i \) is non-trivial. So assume that

\[
\rho, \kappa \vdash^A_{\ell'} \text{fv}(P_i) \psi_i \quad \text{and} \quad \rho, \kappa \vdash^P \text{lab}(\pi_i, \ell') [P_i] : \psi_i
\]

for some \( i \in I \) and let us prove that

\[
\rho, \kappa \vdash^A_{\ell'} \text{fv}(P_i) \psi_i \quad \text{and} \quad \rho, \kappa \vdash^P \text{lab}(\pi_i, \ell') [P_i] : \psi_i
\]

The proof of the first conjunct is straightforward except in the case where \( \pi_i \) is an input. For an input \( \pi_i = u(\bar{x})^\ell_i \) we assume that for all \( n \in \rho(\ell', [u]) \) we have \( \kappa(n) \cap \text{Const}[^\ell_i] \subseteq \rho(\ell_i, \bar{x}), \forall z \in \text{fv}(P_i) \setminus \{ \bar{x} \} : \rho(\ell', z) \subseteq \rho(\ell_i, z) \), and \( \kappa(n) \setminus \text{Const}[^\ell_i] \neq \emptyset \Rightarrow (n, \ell_i) \in \psi_i' \). Next consider some \( n \in \rho(\ell, [u]) \) which by our assumptions satisfies \( n \in \rho(\ell', [u]) \). We must prove that \( \kappa(n) \cap \text{Const}[^\ell_i] \subseteq \rho(\ell_i, \bar{x}) \) which is immediate, that \( \forall z \in \text{fv}(P_i) \setminus \{ \bar{x} \} : \rho(\ell, z) \subseteq \rho(\ell_i, z) \) which follows from \( \rho(\ell, z) \subseteq \rho(\ell', z) \), and that \( \kappa(n) \setminus \text{Const}[^\ell_i] \neq \emptyset \Rightarrow (n, \ell_i) \in \psi_i' \) which also is immediate.

To prove the second conjunct above we first observe that when \( \pi_i \) is a silent action or an output we have \( \text{lab}(\pi_i, \ell') = \ell' \) and \( \text{lab}(\pi_i, \ell) = \ell \) and the result follows from the induction hypothesis. When \( \pi_i \) is an input action we have \( \text{lab}(\pi_i, \ell') = \text{lab}(\pi_i, \ell) \) and the result is immediate.

**Theorem 3.9 Subject Reduction.** If \( P \rightarrow Q \) and \( \rho, \kappa \vdash^p P : \psi \) then \( \rho, \kappa \vdash^p Q : \psi \).

**Proof.** The proof is a variation of that of Theorem 2.6 and only the case of communication is non-trivial. So assume that

\[
(n(\bar{m}), P + P') | (n(\bar{x})^\ell_i.Q + Q') \rightarrow_P Q[\bar{m}/\bar{x}]
\]

because \( |\bar{m}| = |\bar{x}| \) and that

\[
\rho, \kappa \vdash^A (n(\bar{m}), P + P') | (n(\bar{x})^\ell_i.Q + Q') : \psi
\]

From the clauses of Table VIII we thus have that there exist \( \psi_1, \psi_2 \) and \( \psi_2' \) such that

The proof is by induction on the inference


Consider some \( \rho, \kappa \) and \( \psi_1 \) and \( \psi_2 \).

Since \( |\bar{m}| = |\bar{x}| \) we have \( |\bar{m}| \in \kappa(|\bar{n}|) \cap \text{Const}[\bar{m}] \supseteq \rho(\bar{\ell}, \bar{x}) \) so Lemma 3.7 gives \( \rho, \kappa \vdash \rho'(Q[\bar{m}/\bar{x}]) : \psi_2 \). We also have \( \forall y \in \text{fv}(Q[\bar{m}/\bar{x}]) : \rho(\bar{\ell}, y) \subseteq \rho(\bar{\ell}, y) \) and Lemma 3.8 then gives \( \rho, \kappa \vdash \rho'(Q[\bar{m}/\bar{x}]) : \psi_2 \). This shows the desired result that \( \rho, \kappa \vdash \rho'(P[\bar{Q}[\bar{m}/\bar{x}]] : \psi_2 \). □

B.3 Proofs of Main Results of Subsection 3.4

**Lemma 3.14 Initial Process.** Let \( P_*, B, L \) and \( \ell_* \) as above. If \( \rho, \kappa \vdash P_* : \psi \) and \( \epsilon \in \rho(\ell_*) \) then \( \rho, \kappa \vdash \rho' : [P_*] : \psi \).

**Proof.** Consider some \( \beta^\ell, Q' \) that is exposed in \( [P_*] \) and note that \( B.\ell = \ell_* \) and \( [P_*] = P_* \). Next choose \( \alpha^\ell, P' = \beta^\ell, Q' \) and \( \bar{m} = \epsilon \). It is then immediate that \( \beta^\ell, Q' = (\alpha^\ell, P')[\bar{m}/L.(B.\ell)] \). Finally, \( \rho, \kappa \vdash \alpha^\ell, P' : \psi \) follows from Lemma 3.13 and we have established that \( \rho, \kappa \vdash \rho' : [P_*] : \psi \). □

**Lemma 3.15 Structural Congruence.** Let \( P \) and \( Q \) be two processes such that \( P \equiv Q \). If \( \rho, \kappa \vdash \rho' \vdash [P] : \psi \) then \( \rho, \kappa \vdash \rho' \vdash [Q] : \psi \).

**Proof.** Let \( P \equiv Q \) and \( \rho, \kappa \vdash \rho' \vdash [P] : \psi \) and consider some exposed subprocess \( \gamma^\ell, Q' \) of \( [Q] \). Using Lemma 3.12 we find an exposed subprocess \( \beta^\ell, P' \) of \( [P] \) such that \( \gamma^\ell, Q' \equiv \beta^\ell, P' \). From the assumption \( \rho, \kappa \vdash \rho' \vdash [P] : \psi \) we find a subprocess \( \alpha^\ell, R' \) of \( P_* \) and some \( \bar{m} \in \rho(B.\ell) \) such that \( \beta^\ell, P' \equiv (\alpha^\ell, R')[\bar{m}/L.(B.\ell)] \) and \( \rho, \kappa \vdash \alpha^\ell, R' : \psi \). It now follows that \( \gamma^\ell, Q' \equiv (\alpha^\ell, R')[\bar{m}/L.(B.\ell)] \) and \( \rho, \kappa \vdash \alpha^\ell, R' : \psi \) so we have established \( \rho, \kappa \vdash \rho' \vdash [Q] : \psi \). □

**Theorem 3.16 Subject Reduction.** If \( P \rightarrow Q \) and \( \rho, \kappa \vdash P \vdash [P] : \psi \) then \( \rho, \kappa \vdash P \vdash [Q] : \psi \).

**Proof.** The proof is by induction on the inference \( P \rightarrow Q \) as specified in Table II. The axiom [TAU] is straightforward as are the rules [NEW] and [PAR]. The non-trivial rule [EQ] is taken care of by Lemma 3.15.

This leaves us with the axiom [COM] where we consider a transition

\[
(n(\bar{\sigma})^f_0.R_0 + \cdots) | (n(\bar{x})^f_1.R_1 + \cdots) \rightarrow R_0 | R_1[\bar{\sigma}/\bar{x}]
\]

where \( |\bar{\sigma}| = |\bar{x}| \). We also assume \( \rho, \kappa \vdash \rho' \vdash [(n(\bar{\sigma})^f_0.R_0 + \cdots) | (n(\bar{x})^f_1.R_1 + \cdots)] : \psi \).

From the latter it follows that there are subprocesses \( u_0(\bar{\sigma})^f_0.P_0 \) and \( u_1(\bar{x})^f_1.P_1 \) of \( P_* \) and “local environments” \( \bar{m}_0 \in \rho(B.\ell_0) \) and \( \bar{m}_1 \in \rho(B.\ell_1) \) such that \( n(\bar{\sigma})^f_0.R_0 \equiv (u_0(\bar{\sigma})^f_0.P_0)[\bar{m}_0/L.(B.\ell_0)] \) and \( n(\bar{x})^f_1.R_1 \equiv (u_1(\bar{x})^f_1.P_1)[\bar{m}_1/L.(B.\ell_1)] \) and furthermore \( \rho, \kappa \vdash u_0(\bar{\sigma})^f_0.P_0 : \psi \) and \( \rho, \kappa \vdash u_1(\bar{x})^f_1.P_1 : \psi \).

We need to show that \( \rho, \kappa \vdash \rho' \vdash [R_0 | R_1[\bar{\sigma}/\bar{x}]] : \psi \) and for this we consider some exposed subprocess \( \beta^\ell, Q \) of \( [R_0 | R_1[\bar{\sigma}/\bar{x}]] \). We need to find some subprocess \( \alpha^\ell, P \) of \( P_* \) and some \( \bar{m} \in \rho(B.\ell) \) such that (i) \( \beta^\ell, Q \equiv (\alpha^\ell, P)[\bar{m}/L.(B.\ell)] \) and (ii) \( \rho, \kappa \vdash \alpha^\ell, P : \psi \). We shall do so by cases of whether \( \beta^\ell, Q \) is exposed in \( [R_0] \) or in \( [R_1[\bar{\sigma}/\bar{x}]] \).
Consider first the case where $\beta^\ell.Q$ is exposed in $[R_0]$: here we have $B.\ell = \ell_0$. We already know that $[R_0] = P_0[\vec{m}_0/L.(B.\ell_0)]$ and it follows from Lemma 3.12 that there exists an exposed subprocess $\beta^\ell.Q'$ of $P_0[\vec{m}_0/L.(B.\ell_0)]$ such that $\beta^\ell.Q \equiv \beta^\ell.Q'$. Clearly there exists some subprocess $\alpha^\ell.P$ in $P_0$ such that $\beta^\ell.Q' \equiv (\alpha^\ell.P)[\vec{m}_0/L.(B.\ell_0)]$.

We now have to find some $\vec{m}$ such that (i) holds and here we first notice that from Table IX we get $L.(B.\ell) = L.\ell_0 = L.(B.\ell_0)$ (since $\ell_0$ is the label of an output action) and from Table X and $\rho, \kappa \vdash_P u_0(\vec{v})^{\ell_0}.P_0 : \psi$ it follows that $\rho(B.\ell_0) \subseteq \rho(B.\ell)$. To prove (i) it therefore suffices to take $\vec{m} = \vec{m}_0$ and observe that $\vec{m} \in \rho(B.\ell)$ and $\beta^\ell.Q \equiv (\alpha^\ell.P)[\vec{m}_0/L.(B.\ell_0)]$. Finally we have to establish (ii), that is $\rho, \kappa \vdash_P \alpha^\ell.P : \psi$. From $\rho, \kappa \vdash_P u_0(\vec{v})^{\ell_0}.P_0 : \psi$ we also get $\rho, \kappa \vdash_P P_0 : \psi$ from which $\rho, \kappa \vdash_P \alpha^\ell.P : \psi$ follows using Lemma 3.13.

For later reference we observe that Table X and $\rho, \kappa \vdash_P u_0(\vec{v})^{\ell_0}.P_0 : \psi$ also ensure that $n\vec{o} = \Pi_{u_0 \in \ell_0(L.\ell_0)}(\vec{m}_0) \in \kappa$.

Consider next the case where $\beta^\ell.Q$ is an exposed subprocess in $[R_1[\vec{a}/\vec{x}]]$: here we have $B.\ell = \ell_1$. Since $R_1 = P_1[\vec{m}_1/L.(B.\ell_1)]$ it follows that $R_1[\vec{a}/\vec{x}] = P_1[\vec{m}_1\vec{a}/L.(B.\ell_1)[\vec{x}]]$. It hence follows from Lemma 3.12 that there is some exposed subprocess $\beta^\ell.Q'$ of $P_1[\vec{m}_1\vec{a}/L.(B.\ell_1)[\vec{x}]]$ such that $\beta^\ell.Q \equiv \beta^\ell.Q'$. Clearly there exists some subprocess $\alpha^\ell.P$ in $P_1$ such that $\beta^\ell.Q' \equiv (\alpha^\ell.P)[\vec{m}_1\vec{a}/L.(B.\ell_1)[\vec{x}]]$ and hence $\beta^\ell.Q \equiv (\alpha^\ell.P)[\vec{m}_1\vec{a}/L.(B.\ell_1)[\vec{x}]]$.

Also here we have to find some $\vec{m}$ such that (i) holds. To do so we observe that from Table IX it follows that $L.(B.\ell) = L.\ell_1 = L.(B.\ell_1)[\vec{x}]$. From Table X and $\rho, \kappa \vdash_P u_1(\vec{a})^{\ell_1}.P_1 : \psi$, and using the facts $\vec{m}_1 \in \rho(B.\ell_1)$ and $n\vec{o} \in \kappa$ established above, it follows that $\vec{m}_1\vec{a} \in \rho(\ell_1)$ since $n = \Pi_{u_1 \in \ell_1(L.\ell_1)}(\vec{m}_1)$. To prove (i) it therefore suffices to take $\vec{m} = \vec{m}_1\vec{a}$ and observe that $\vec{m} \in \rho(B.\ell)$ and $\beta^\ell.Q \equiv (\alpha^\ell.P)[\vec{m}_1\vec{a}/L.(B.\ell_1)[\vec{x}]]$. To prove (ii) we note that the assumption $\rho, \kappa \vdash_P P_1 : \psi$ gives $\rho, \kappa \vdash_P \alpha^\ell.P : \psi$ using Lemma 3.13.

B.4 Proofs of Main Results of Appendix A

The material reported here is collaboration between the authors of the main paper and our students Piotr Filipiuk, Alejandro Hernandez, Lei Song and Fuyuan Zhang as part of an intensive PhD course on Flow Logic.

**Lemma A.4 Structural Congruence.** If $M \equiv N$ then $\rho, \kappa \vdash_N [N] : \psi$ if and only if $\rho, \kappa \vdash_N [M] : \psi$.

**Proof.** The proof is done by induction on congruence relation defined in Table XIII. We consider two cases:

Consider the axiom $N \mid \circ \equiv N$, and let us assume that $\rho, \kappa \vdash_N [N \mid \circ] : \psi$. Based on Table XV we get that $\rho, \kappa \vdash_N [N] : \psi$ and $\rho, \kappa \vdash_N [\circ] : \psi \land \psi_N \cup \psi_\circ \land \psi_N \cup \psi_\circ \subseteq \psi$. From above as well as monotonicity of $\vdash_N$, we get required $\rho, \kappa \vdash_N [N] : \psi$.

Next consider the axiom for disciplined $\alpha$-renaming and assume that $\langle \text{new}_k n \rangle N \equiv \langle \text{new}_k n \rangle N[m/n]$ where $m \notin \text{fc}(N)$, $m \neq k$ and $[n] = [m]$. We have to prove that $\rho, \kappa \vdash_N \langle \text{new}_k n \rangle N : \psi$ if and only if $\rho, \kappa \vdash_N \langle \text{new}_k n \rangle N[m/n] : \psi$. It follows from definition of disciplined $\alpha$-renaming that $\langle \text{new}_k n \rangle N = \langle \text{new}_k n \rangle N[m/n]$ and we get the result.

**Lemma A.5 Substitution.** If $\rho, \kappa \vdash^\ell [P] : \psi$ and $[m] \in \rho(k, y)$ then $\rho, \kappa \vdash^\ell [P[m/y]] : \psi$.
Proof. The proof is by structural induction on the form of the process $P$, and then we have to analyse four different cases.

First consider the case where $P = !P$. Assuming that $\rho, \kappa \vdash_{\pi} [!P] : \psi$, we apply clause [REP] of Table XV obtaining $\rho, \kappa \vdash_{\pi} [P] : \psi$. The Induction Hypothesis gives us $\rho, \kappa \vdash_{\pi} [P[m/y]] : \psi$, and now if we apply [REP] again we get $\rho, \kappa \vdash_{\pi} ![\{P[m/y]\}] : \psi$, which due to the point-wise definition of substitution gives our result that $\rho, \kappa \vdash_{\pi} ![\{!P\}[m/y]] : \psi$.

Next consider the case where $P = P_1 | P_2$. Assuming $\rho, \kappa \vdash_{\pi} [P_1 | P_2] : \psi$, we apply clause [PAR] of Table XV obtaining

$$\rho, \kappa \vdash_{\pi} [P_1] : \psi_1 \land \rho, \kappa \vdash_{\pi} [P_2] : \psi_2 \land \psi_1 \cup \psi_2 \subseteq \psi$$

Now we can apply the Induction Hypothesis to the first two conjuncts, then we apply [PAR] again and, due to the point-wise definition of substitution, we get $\rho, \kappa \vdash_{\pi} [(P_1 | P_2)[m/y]] : \psi$ which is our result.

Then we consider the case $P = (\text{new} n)P$: Before starting this case, let us remember the definition of substitution for the case where the process is a restriction:

$$((\text{new} n)P)[u/v] = \begin{cases} (\text{new} n)P & \text{if } n = v \\ (\text{new} n)(P[u/v]) & \text{if } n \neq v \land n \neq u \\ (\text{new} n')(P[n'/n][u/v]) & \text{if } n \neq v \land n = u \\ & \text{and where } n' \neq u \land n' \notin \text{fc}(P) \end{cases}$$

Assuming $\rho, \kappa \vdash_{\pi} [(\text{new} n)P] : \psi$, we have to prove $\rho, \kappa \vdash_{\pi} [(\text{new} n)P[m/y]] : \psi$. The case $n = y$ is trivial by definition of substitution. For the case $n \neq y$ we apply clause [RES] of Table XV obtaining $\rho, \kappa \vdash_{\pi} [P] : \psi$. Applying the Induction Hypothesis and [RES] again we get $\rho, \kappa \vdash_{\pi} [(\text{new} n')(P[m/y])] : \psi$. But this differs from our desired result, so now we shall prove that whenever $n \neq y$, then $(\text{new} n)(P[m/y]) \equiv ((\text{new} n)P)[m/y]$, and then our result will follow from Lemma A.4. The case $n \neq m$ is trivial by definition of substitution, so let's focus in the case $n = m$. If we take some $n'$ such that $n' \neq m$, $[n'] = [n]$ and $n' \notin \text{fc}(P)$ then we can apply the following 3 steps:

$$(\text{new} n)(P[m/y]) \equiv (\text{new} n')(P[m/y][n'/n]) \quad \text{by disciplined } \alpha\text{-renaming}$$

$$= (\text{new} n')(P[n'/n][m/y]) \quad \text{by property of substitution}$$

$$= ((\text{new} n)P)[m/y] \quad \text{by definition of substitution}$$

and our result follows.

Finally let us consider the case $P = \Sigma_{i \in I} \pi_i P_i$: Assuming $\rho, \kappa \vdash_{\pi} [\Sigma_{i \in I} \pi_i P_i] : \psi$, we apply clause [SUM] of Table XV obtaining

$$\forall i \in I : (\rho, \kappa \vdash_{\pi} [\pi_i] : \psi_i \land \Lambda_i \land \psi_i \subseteq \psi \land$$

$$\forall k' \in \Lambda_i : (\rho, \kappa \vdash_{\pi\pi} [P_i] : \psi_{i,k'} \land \psi_{i,k'} \subseteq \psi \land$$

$$\forall z \in \text{fv}(P_i) : \rho(k, z) \subseteq \rho(k', z)))$$

This is a conjunction of $|I|$ different conjuncts, so we shall consider each of them separately. Thus it is sufficient to prove that if for an arbitrary conjunct $\rho, \kappa \vdash_{\pi\pi} [P_i] : \psi_{i,k'}$ we have $\rho, \kappa \vdash_{\pi\pi} [(\pi_i P_i)[m/y]] : \psi_i$, since then our final result $\rho, \kappa \vdash_{\pi\pi} [(\Sigma_{i \in I} \pi_i P_i)[m/y]] : \psi$ holds as well. For this purpose, it is enough to prove:
The proof is by induction on how
First from above and Fact A.3 we have that
We also know that
among others) the following:
We assume that:
again the above observations we get
Finally consider the sub-case where
Next consider the sub-case where
Next consider the sub-case where
Finally consider the sub-case where
Next consider the sub-case where
\[ z \]
This completes the proof. \qed

Theorem A.6 Subject Reduction. If \( N \rightarrow M \) and \( \rho,\kappa \vdash_N \left[ N \right] : \psi \) then \( \rho,\kappa \vdash_N \left[ M \right] : \psi \).

Proof. The proof is by induction on how \( M \) is obtained from \( N \), using the axioms and rules from Table XIV. We have to analyse six different cases.

First consider the axiom [tau] expressing that

\[ k \cdot \left[ \tau.P + Q \right] \rightarrow k \cdot \left[ P \right] \]

We assume that: \( \rho,\kappa \vdash_N \left[ k \cdot \left[ \tau.P + Q \right] \right] : \psi \). From the clauses in Table XV we have (among others) the following:

\[ \rho,\kappa \vdash_\Lambda \left[ \tau \right] : \psi_1 \land \psi_1 \subseteq \psi \land \forall k' \in \Lambda_1 : (\rho,\kappa \vdash_\beta \left[ P \right] : \psi_{1,k'} \land \psi_{1,k'} \subseteq \psi \land \forall z \in \text{fv}(P) : (\rho(k, z) \subseteq \rho(k', z)) \]

We also know that \( k \in \Lambda_1 \), which implies that: \( \rho,\kappa \vdash_\beta \left[ P \right] : \psi_{1,k} \land \psi_{1,k} \subseteq \psi \). From above and Fact A.3 we have that \( \rho,\kappa \vdash_\beta \left[ P \right] : \psi \). Using the rule [Loc] from Table XV we get the required \( \rho,\kappa \vdash_N \left[ k \left[ \beta \right] P \right] : \psi \).

Next consider the case [nre] expressing that

if \( N \rightarrow M \) then \( (\text{new}_k n)N \rightarrow (\text{new}_k n)M \)
From the induction hypothesis we have that if \( N \rightarrow M \) and \( \rho, \kappa \vdash_N [N] : \psi \) then \( \rho, \kappa \vdash_N [M] : \psi \). Let us assume that \( \rho, \kappa \vdash_N [(\text{new}_k n)N] : \psi \). From the clauses in Table XV we have that \( \rho, \kappa \vdash_N [N] : \psi \). Using the induction hypothesis we therefore get \( \rho, \kappa \vdash_N [M] : \psi \). From the rule [NRE] of Table XV we now get the required \( \rho, \kappa \vdash_N [(\text{new}_k n)M] : \psi \).

Next consider the case [PAR] that expresses that

\[
\text{if } N \rightarrow N' \text{ then } N \mid M \rightarrow N' \mid M
\]

From the induction hypothesis we know that if \( N \rightarrow N' \) and \( \rho, \kappa \vdash_N [N] : \psi_N \) then \( \rho, \kappa \vdash_N [N'] : \psi_N \). Moreover we assume \( \rho, \kappa \vdash_N [N \mid M] : \psi \). From the clauses in Table XV we have the following:

\[
\rho, \kappa \vdash_N [N] : \psi_N \land \rho, \kappa \vdash_N [M] : \psi_M \land \psi_N \cup \psi_M \subseteq \psi
\]

Using the induction hypothesis we obtain \( \rho, \kappa \vdash_N [N'] : \psi_N \land \rho, \kappa \vdash_N [M] : \psi_M \land \psi_N \cup \psi_M \subseteq \psi \). Using the rule [DIS] from Table XV we now get the required \( \rho, \kappa \vdash_N [N' \mid M] : \psi_N \).

Let us now consider the rule [EQ] that expresses

\[
\text{if } N \equiv N' \text{ and } N' \rightarrow M' \text{ and } M' \equiv M \text{ then } N \rightarrow M
\]

Let us assume that \( \rho, \kappa \vdash_N [N] : \psi \). Using Lemma A.4 and the fact that \( N \equiv N' \), we get that \( \rho, \kappa \vdash_N [N'] : \psi \). Using the induction hypothesis we therefore get \( \rho, \kappa \vdash_N [M'] : \psi \). From Lemma A.4 and the fact that \( M' \equiv M \) we get required \( \rho, \kappa \vdash_N [M] : \psi \).

The next case is [MOV] where we have

\[
k[\text{go} l.P] \rightarrow l[P]
\]

We assume that \( \rho, \kappa \vdash_N [k[\text{go} l.P]] : \psi \) and we have to prove that \( \rho, \kappa \vdash_N [l[P]] : \psi \). From the clauses [LOC] and [SUM] of Table XV we get

\[
\rho, \kappa \vdash_k^k [\text{go} l] : \psi_1 \land \& \psi_1 \subseteq \psi \land \\
\forall k' \in \Lambda_1 : (\rho, \kappa \vdash_P^P [P] : \psi_{1l'} \land \psi_{1l'} \subseteq \psi \land \\
\forall z \in \text{fv}([P]) : \rho(k, z) \subseteq \rho(k', z))
\]

and applying [GO] to the first conjunct we get that \( \rho(k, l) \subseteq \Lambda_1 \), which means that from the second line we can get the following

\[
\forall k' \in \rho(k, l) : \rho, \kappa \vdash_k^P [P] : \psi_{1l'} \land \psi_{1l'} \subseteq \psi
\]

which is enough to prove \( \rho, \kappa \vdash_k^l [P] : \psi \) by Fact A.3. Now, applying clause [LOC] from Table XV we get our result.

Finally consider the case [COM] where we have

\[
k[n(\bar{m}).P + P'] \mid k[n(\bar{x}).Q + Q'] \rightarrow k[P] \mid k[Q[\bar{m}/\bar{x}]]
\]

and \( |\bar{m}| = |\bar{x}| \). We assume that \( \rho, \kappa \vdash_N [k[n(\bar{m}).P + P'] \mid k[n(\bar{x}).Q + Q']] : \psi \) and we have to prove that \( \rho, \kappa \vdash_N [k[P] \mid k[Q[\bar{m}/\bar{x}]]] : \psi \). Using the axioms for process lifting in the structural congruence and Lemma A.4 we get

\[
\rho, \kappa \vdash_N [k[n(\bar{m}).P + P' \mid n(\bar{x}).Q + Q']] : \psi
\]
and then we can apply the clauses $[\text{LOC}]$ and $[\text{PAR}]$ from Table XV and we get $\rho, \kappa \vdash^k \rho [n(m).P + P'] : \psi_1 \land \rho, \kappa \vdash^k [n(x).Q + Q'] : \psi_2 \land \psi_1 \cup \psi_2 \subseteq \psi$

Now, applying the clause $[\text{SUM}]$ to the first and second conjunct we get, for the first conjunct (ignoring the second summand):

$$\rho, \kappa \vdash^k \rho [n(m)] : \psi_{11} \land \Lambda_{11} \land \psi_{11} \subseteq \psi_1 \land \forall k' \in \Lambda_{11} : (\rho, \kappa \vdash^k [P] : \psi_{11,k'} \land \psi_{11,k'} \subseteq \psi_1 \land \forall z \in \text{fv}(P) : \rho(k,z) \subseteq \rho(k',z))$$

and for the second conjunct (ignoring the second summand):

$$\rho, \kappa \vdash^k \rho [n(x)] : \psi_{21} \land \Lambda_{21} \land \psi_{21} \subseteq \psi_2 \land \forall k' \in \Lambda_{21} : (\rho, \kappa \vdash^k [Q] : \psi_{21,k'} \land \psi_{21,k'} \subseteq \psi_2 \land \forall z \in \text{fv}(Q) : \rho(k,z) \subseteq \rho(k',z))$$

Applying $[\text{OUT}]$ and $[\text{IN}]$ respectively to the first conjuncts of each of the previous two conjunctions we get the following three conditions:

$$k \in \Lambda_{11}$$

$$k \in \Lambda_{21}$$

$$\rho(k,m) \subseteq \kappa(k,n) \subseteq \rho(k,x)$$

Now we are able to get $\rho, \kappa \vdash^k [P] : \psi_1$ by Fact A.3 and $\rho, \kappa \vdash^k [Q[m/x]] : \psi_2$ by Fact A.3 and Lemma A.5. We also had $\psi_1 \cup \psi_2 \subseteq \psi$ so applying first the clause $[\text{PAR}]$, then the clause $[\text{LOC}]$, and finally the axiom for process lifting in the structural congruence our result follows from Lemma A.4. \hfill $\square$

**Theorem A.7 Adequacy for non-leaking networks.** If the network $N_*$ is statically non-leaking then it is also dynamically non-leaking.

**Proof.** The proof is by contradiction so assume that $\rho, \kappa \vdash_N N_* : \emptyset$ and $\forall k,n \in \text{Const} : (\forall m \in \kappa(k,n) : (\exists(m) \subseteq (\exists(n) \cup \exists(k))^{[m]}))$ and that

$$N_* \rightarrow \ldots \rightarrow C[k \parallel n(m).P + P'] [k \parallel n(x).Q + Q']$$

but $\neg([m] = [x] \land \exists(m) \subseteq (\exists(n) \cup \exists(k))^{[m]}))$.

From the Subject Reduction Theorem we get

$$\rho, \kappa \vdash_N C[k \parallel n(m).P + P'] [k \parallel n(x).Q + Q'] : \emptyset$$

so in particular it must be the case that $\rho, \kappa \vdash^k [n(m)] : \emptyset \land \Lambda_1$ and $\rho, \kappa \vdash^k [n(x)] : \emptyset \land \Lambda_2$ for some $\Lambda_1$ and $\Lambda_2$. Thus it follows that $[m] \in \kappa([k],[n])$.

From our assumptions we get $\exists(m) \subseteq (\exists(n) \cup \exists(k))^{[m]}$, and therefore it must be the case that $\neg([m] = [x])$. But then it must be the case that $([k],[n]) \in \psi$ and this is a contradiction since $\psi = \emptyset$. \hfill $\square$

**Proposition A.9 Moore Family.** The set $\{ (\rho, \kappa, \psi) \mid \rho, \kappa \vdash_N N : \psi \}$ is a Moore Family for all networks $N$.

**Proof.** Let us consider an index set $I$ and assume that

$$\forall i \in I : \rho^i, \kappa^i \vdash_N N : \psi^i$$

so that we shall prove that

$$\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_N N : \cap_{i \in I} \psi^i$$

We shall proceed by induction on $N$.

The cases of $\cap$ and $\cap_k n N$ are straightforward.

Let us consider the case $N \mid M$. From $\rho^i, \kappa^i \vdash_N N \mid M : \psi^i$, we get $\rho^i, \kappa^i \vdash_N N : \psi^i_N$ and $\rho^i, \kappa^i \vdash_N M : \psi^i_M$ as well as $\psi^i_N \cup \psi^i_M \subseteq \psi^i$. The induction hypothesis gives $\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_N N : \cap_{i \in I} \psi^i_N$ and $\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_N M : \cap_{i \in I} \psi^i_M$. From $\psi^i_N \cup \psi^i_M \subseteq \psi^i$ (for all $i \in I$) we get $(\cap_{i \in I} \psi^i_N) \cup (\cap_{i \in I} \psi^i_M) \subseteq \psi^i_N \cup \psi^i_M \subseteq \psi^i$ for all $i \in I$ and thereby $(\cap_{i \in I} \psi^i_N) \cup (\cap_{i \in I} \psi^i_M) \subseteq (\cap_{i \in I} \psi^i)$ as required. This shows that $\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_N N \mid M : \cap_{i \in I} \psi^i$.

For the case $k \parallel P$ the results follows in a similar way once we have established a Moore Family result for the analysis of processes. Thus we shall prove that if $\rho^i, \kappa^i \vdash_{p^i} P : \psi^i$

holds for all $i \in I$, then also

$$\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_{p^i} P : \cap_{i \in I} \psi^i$$

The proof of this result is by induction on $P$.

The cases of $(\text{new } n) P$ and $! P$ are straightforward.

Let us consider the case $P_1 \parallel P_2$. From $\rho^i, \kappa^i \vdash_{p^i} P_1 \parallel P_2 : \psi^i$, we get $\rho^i, \kappa^i \vdash_{p^i} P_1 : \psi^i$ and $\rho^i, \kappa^i \vdash_{p^i} P_2 : \psi^i$ as well as $\psi^i_1 \cup \psi^i_2 \subseteq \psi^i$. The induction hypothesis gives $\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_{p^i} P_1 : \cap_{i \in I} \psi^i_1$ and $\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_{p^i} P_2 : \cap_{i \in I} \psi^i_2$. From $\psi^i_1 \cup \psi^i_2 \subseteq \psi^i$ (for all $i \in I$) we get $(\cap_{i \in I} \psi^i_1) \cup (\cap_{i \in I} \psi^i_2) \subseteq \psi^i_1 \cup \psi^i_2 \subseteq \psi^i$ for all $i \in I$ and thereby $(\cap_{i \in I} \psi^i_1) \cup (\cap_{i \in I} \psi^i_2) \subseteq (\cap_{i \in I} \psi^i)$ as required. This shows that $\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_{p^i} P_1 \parallel P_2 : \cap_{i \in I} \psi^i$.

Finally let us consider the case of guarded sums $\sum_{j \in J} \pi_j, P_j$. From $\rho^i, \kappa^i \vdash_{p^i} \sum_{j \in J} \pi_j, P_j : \psi^i$, we know that the following holds

$$\forall j \in J : (\rho^i, \kappa^i \vdash_{p^i} \pi_j : \psi^i_j \wedge \psi^i_j \subseteq \psi^i) \wedge$$

$$\forall k' \in \Lambda^i_j : (\rho^i, \kappa^i \vdash_{p^i} P_j : \psi^i_{j,k'} \wedge \psi^i_{j,k'} \subseteq \psi^i) \wedge$$

$$\forall z \in \text{fv}(P_j) : \rho^i(k, z) \subseteq \rho^i(k', z))$$

According to induction hypothesis, we have $\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_{p^i} P_j : \cap_{i \in I} \psi^i_{j,k'}$. From $\psi^i_{j,k'} \subseteq \psi^i$ we get $\cap_{i \in I} \psi^i_{j,k'} \subseteq \psi^i$ (for all $i \in I$) and thereby $\cap_{i \in I} \psi^i_{j,k'} \subseteq \cap_{i \in I} \psi^i$ as required. From $\rho^i(k, z) \subseteq \rho^i(k', z)$, we have $\cap_{i \in I} \rho^i(k, z) \subseteq \rho^i(k', z)$ for all $i \in I$ and thereby we have $\cap_{i \in I} \rho^i(k, z) \subseteq \cap_{i \in I} \rho^i(k', z)$. From above, we have for all $j \in J$ the following holds

$$\forall k' \in \Lambda^i_j : (\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_{p^i} P_j : \cap_{i \in I} \psi^i_{j,k'} \wedge \cap_{i \in I} \psi^i_{j,k'} \subseteq \cap_{i \in I} \psi^i \wedge$$

$$\forall z \in \text{fv}(P_j) : \cap_{i \in I} \rho^i(k, z) \subseteq \cap_{i \in I} \rho^i(k', z))$$

Therefore, we get

$$\forall k' \in \cap_{i \in I} \Lambda^i_j : (\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_{p^i} P_j : \cap_{i \in I} \psi^i_{j,k'} \wedge \cap_{i \in I} \psi^i_{j,k'} \subseteq \cap_{i \in I} \psi^i \wedge$$

∀z ∈ fv(Pj) : \cap_{i \in I} \rho^i(k, z) \subseteq \cap_{i \in I} \rho^i(k', z))

It is straightforward that \cap_{i \in I} \psi_j^i \subseteq \cap_{i \in I} \psi^i holds for all j \in J. Therefore, for guarded sums \sum_{j \in J} \pi_j, P_j the results follows once we have established a Moore Family result for the analysis of actions. Thus we shall prove that if

\rho^i, \kappa^i \vdash_A \pi : \psi^i & \Lambda^i

holds for all i \in I, then also

\cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_A \pi : \cap_{i \in I} \psi^i & \cap_{i \in I} \Lambda^i

The proof is by cases on \pi.

The case of the \tau action is straightforward.

Next consider the action go u. From \rho^i, \kappa^i \vdash_A \go u : \psi^i & \Lambda^i we get \rho^i(k, u) \subseteq \Lambda^i. Therefore we get \cap_{i \in I} \rho^i(k, u) \subseteq \Lambda^i and \cap_{i \in I} \rho^i(k, u) \subseteq \cap_{i \in I} \Lambda^i. Therefore we have \cap_{i \in I} \rho^i, \cap_{i \in I} \kappa^i \vdash_A \go u : \cap_{i \in I} \psi^i & \cap_{i \in I} \Lambda^i.

For the output action u(\vec{x}) we note that from \rho^i, \kappa^i \vdash_A \u(\vec{x}) : \psi^i & \Lambda^i (for all i \in I), we have \forall n \in \rho^i(k, u) : \rho^i(k, \vec{x}) \subseteq \kappa^i(k, n) \land k \in \Lambda^i holds. We observe that if we have k \in \Lambda^i, we get k \in \cap_{i \in I} \Lambda^i. We also observe that if we have \forall n \in \rho^i(k, u) : \rho^i(k, \vec{x}) \subseteq \kappa^i(k, n) and thereby \forall n \in \cap_{i \in I} \rho^i(k, u) \cap \kappa^i(k, n) holds. Therefore, the results follows.

Finally for the input action u(\vec{x}) we note that from \rho^i, \kappa^i \vdash_A \u(\vec{x}) : \psi^i & \Lambda^i (for all i \in I), we have k \in \Lambda^i and \forall n \in \rho^i(k, u) : \rho^i(k, \vec{x}) \subseteq \kappa^i(k, n) \land k \in \Lambda^i \land \kappa^i(k, n) \neq \emptyset \Rightarrow (k, n) \in \psi^i. If we have k \in \Lambda^i, we will get k \in \cap_{i \in I} \Lambda^i. If we have \forall n \in \rho^i(k, u) : \kappa^i(k, n) \cap \kappa^i(k, n) \neq \emptyset, we will get \forall n \in \cap_{i \in I} \rho^i(k, u) : \cap_{i \in I} \kappa^i(k, n) \cap \kappa^i(k, n) \neq \emptyset \Rightarrow (k, n) \in \psi^i. We can also get \forall n \in \cap_{i \in I} \rho^i(k, u) : \cap_{i \in I} \kappa^i(k, n) \cap \kappa^i(k, n) \neq \emptyset \Rightarrow (k, n) \in \psi^i and thereby \forall n \in \cap_{i \in I} \rho^i(k, u) : \cap_{i \in I} \kappa^i(k, n) \cap \kappa^i(k, n) \neq \emptyset \Rightarrow (k, n) \in \cap_{i \in I} \psi^i. Therefore, the results follows. This completes the proof. □