Università di Pisa
Dipartimento di Informatica

Technical Report: TR-04-02

A Logical Framework for the `spi-calculus`

Ugo Montanari   Emilio Tuosto

January 19, 2004

Address: Via F. Buonarroti 2, 56127 Pisa, Italy.
Tel: +39 050 2212700 — Fax: +39 050 2212726

A Logical Framework for the `spi-calculus`

Ugo Montanari   Emilio Tuosto

January 19, 2004

Abstract

`spi-calculus` [AG98] has been presented as a (formal) setting used to describe and analyze protocols. Here we will present a short description of the calculus, introduce a new LTS, proving its equivalence to the LTS defined by Abadi and Gordon, and give a representation of the `spi-calculus` in the Edimburg Logical Framework introduced by Haper, Honsell and Plotkin. This Logical Framework provides a setting used to define formal systems based on a general treatment of syntax, rules and proofs by means of a typed λ-calculus with dependent types. Finally, we will prove that the LF-semantics and the LTS-semantics are equivalent.

Contents

Chapter 1

The spi-calculus

In this section, syntax and semantics of the `spi-calculus`, a process algebra with cryptographic primitives, are reported.

1.1 Syntax of the spi-calculus

The `spi-calculus` [AG98] is an extension of the π-calculus with cryptographic constructs for the description and the analysis of secure communication protocols. Although the π-calculus is sufficient for programming some abstract protocols, some cryptographic issues can be studied in more details, extending the π-calculus with some *ad hoc* primitives.

The `spi-calculus` syntax has been thought to write communication protocols in a formal fashion avoiding the complications that arise using the informal notations usually adopted in the literature.

As usual, we assume an infinite set of names $\mathcal{N}$ and let k, m, n, p, q, r or x, y, z range over $\mathcal{N}$. We differ from Abadi and Gordon since here there is no distinction between *names*, i.e. communication channels, and *variables*.

The `spi-calculus` syntax is defined with the grammar given in table 1.1. The keys used to encrypt the messages are modeled as names (in [AG98], keys

			p, q, r	$::=$	processes
				$\overline{n}\langle M\rangle.p$	output
				$n(x).p$	input
M, N	$::=$	terms		$p \mid q$	composition
	n	channels		$(\nu\, n)p$	restriction
	k	keys (are names)		$!p$	bang
	$\{M\}_k$	encryption		$[M = N]p$	match
				$\mathtt{0}$	nil
				$\mathbf{case}\ M\ \mathbf{of}\ \{x\}_k\ \mathbf{in}\ p$	decryption

Table 1.1: `spi-calculus` syntax

are modeled by natural numbers). The terms of the calculus are channels/keys or encrypted terms. The grammar of the processes is the similar to the grammar π-calculus extended with a decryption process that tries to decrypt M using the key k. However, two main differences are that not only simple names are exchanged in communications or compared in matchings: the `spi-calculus` can send and compare complex terms (i.e. encrypted messages). Note that in table 1.1, general terms appear in subject position of the input/output prefixes and in encryption key position. In fact, this is an over specification of the `spi-calculus`: as we will see later on, phrases like $\{m\}_n(x).p$ are meaningless, but may be derived in the (untyped) version of the calculus. Furthermore, differently from Abadi and Gordon, we assume that only names (and not encryptions) are used as keys to encrypt/decrypt messages.

A process may be prefixed by output or input actions that allows process communications or may be the parallel component of processes that can concurrently evolve or interact, the ν operator creates and binds a private name in its continuation. The process $!p$ may be thought as the parallel composition of infinite copies of p. The void process `0` is the process that cannot make actions. The decryption process tries to decrypt the message M with the key k and, in case of success, the continuation is the process p in which the decrypted message is substituted for x.

Given a process p, the sets $fn(p)$ and $bn(p)$ of *free* and *bound* names, respectively, are defined as usual for all π-calculus-like processes, furthermore, in **case** M **of** $\{x\}_N$ **in** p, the occurrences of x in p are bound. As usual, given a process p, we consider equivalent any process obtained by an α-renaming of bound names in p.

The `spi-calculus` processes are used to model protocols and their security properties are stated in terms of equivalence between suitable processes. For instance, a protocol respects the *secrecy* of a parameter, say x, if the following property holds for the corresponding process p:

$$p[x \mapsto M] \simeq p[x \mapsto M'] \qquad \text{for all messages } M \text{ and } M'$$

(the equivalence relation adopted by Abadi and Gordon is the *may-testing* equivalence [DH84].

Note. Here, the **let** and the integer **case** constructs introduced in [AG98], are not adopted because they may be considered only syntactic sugar.

1.1.1 A `spi-calculus` protocol example

Here we show how it is possible to program a simple protocol using the `spi-calculus`.

Suppose that two principals A and B share a private key k and that A wants to communicate a message M to B on a public (i.e. non-secure) channel n. The protocol, that simply is "A sends M encrypted with k to B on the channel c, may be informally described as:

$$\text{Message 1} \qquad A \rightarrow B\text{: } \{M\}_k \text{ on } c$$

In the `spi-calculus`, we write:

$$\begin{array}{rcl} A(M) & \triangleq & \overline{c}\langle\{M\}_k\rangle \\ B & \triangleq & c(x).\mathbf{case}\ x\ \mathbf{of}\ \{y\}_k\ \mathbf{in}\ F[y] \\ \mathit{Inst}(M) & \triangleq & (\nu\, k)(A(M) \mid B) \end{array}$$

We can see that A sends the encrypted message $\{M\}_k$ and B waits a message on c; after the interaction, B tries to decrypt the received message with the key k. If such a decryption succeed, the decrypted message (M) is substituted for y in the continuation of B, $F[y]$. Otherwise B is stuck. Finally, it is useful to observe that in the instantiation of the protocol $\mathit{Inst}(M)$, the k is known only to A and B because it is restricted by the ν operator.

1.2 Semantics of the `spi-calculus`: two LTS

In this section we give two LTS for the `spi-calculus`. The first transition system is the one defined by Abadi and Gordon in [AG98]. The second LTS is an equivalent set of rules more suitable for the LF encoding.

1.2.1 The "original" LTS

In [AG98] two semantics for `spi-calculus` are defined. The first is inspired on the *Chemical Abstract Machine* of Berry and Boudol [BB92] and on a *testing equivalence* relation (De Nicola and Hennessy [DH84]); the second semantics is based on the notion of a *Labelled Transition System* [Plo81] and on a *bisimulation equivalence* [Par80]. We will consider the latter semantics.

The `spi-calculus` semantics is defined via some new syntactic constructs reported in table 1.2.

Differently from the usual π-calculus like LTS, the labels of the transitions system have only the *subject* part and may be the silent action, (the subject of) an input action or an output action. In this manner the labels only give visibility of the names used in process interactions and no information about the exchanged messages.

A `spi-calculus` agent is a process, a process abstraction, i.e. the continuation of an input transition, or a concretion, that is the continuation of an output action.

a	$::=$	agents	α, β	$::=$	labels
	p	process		τ	silent
	$\lambda x.p$	abstraction		n	name
	$(\nu\, \vec{m})\langle M\rangle p$	concretion		$\overline{n}$	co-name

Table 1.2: `spi-calculus` syntactic extension

Informally, an agent is the continuation of an action. For instance, if p emits an input label n, the continuation will be an agent of the form $\lambda x.p$. The intended meaning is that p "declares" (to the environment) that it is ready to read something on the channel n and when a synchronization happens, an *interaction* (see below) will perform the communication. The term $(\nu\, \vec{m})\langle M\rangle p$ is a shorthand for $(\nu\, m_1)\ldots(\nu\, m_h)\langle M\rangle p$, $(h \geq 0)$ and its intended meaning is that, on synchronization, the message M, in which the private names $\vec{m}$ appear, will be sent to the receiver (if $h = 0$, we interpret $(\nu\, m_1)\ldots(\nu\, m_h)\langle M\rangle p$ as $\langle M\rangle p$.

It is necessary to extend the restriction and composition operators over an arbitrary agent:

$$(\nu\, m)\lambda x.p \quad \triangleq \quad \lambda x.(\nu\, m)p \tag{1.1}$$

$$r \mid \lambda x.p \quad \triangleq \quad \lambda x.(r \mid p) \tag{1.2}$$

provided that m is different from x in 1.1 and that $x \notin fn(r)$ in 1.2. For concretions, we set:

$$(\nu\, m)(\nu\, \vec{n})\langle M\rangle p \quad \triangleq \quad \begin{cases} (\nu\, m, \vec{n})\langle M\rangle p & \text{if } m \in n(M) \setminus \{\vec{n}\} \\ (\nu\, \vec{n})\langle M\rangle (\nu\, m)p & \text{otherwise} \end{cases} \tag{1.3}$$

$$r \mid (\nu\, \vec{n})\langle M\rangle p \quad \triangleq \quad (\nu\, \vec{n})\langle M\rangle (r \mid p) \tag{1.4}$$

in 1.3 we assume that $m \notin \vec{n}$ and in 1.4 we assume $\vec{n} \cap fn(r) = \emptyset$. In the same way, we define the symmetric case $a \mid q$.

In order to model communications, the last notion we need, before introducing the LTS for `spi-calculus`, is the *interaction*. Let F be the abstraction $\lambda x.p$, let C be the concretion $(\nu\, \vec{m})\langle M\rangle q$, and suppose $\vec{m} \cap fn(p) = \emptyset$, then $F@C$ and $C@F$ are called *interactions* and are defined by the following equation:

$$F@C \quad \triangleq \quad (\nu\, \vec{m})(p[M/x] \mid q)$$

$$C@F \quad \triangleq \quad (\nu\, \vec{m})(q \mid p[M/x])$$

Finally, the LTS is given in the table 1.3 below. The transition system relies

$$\text{(IN)} \quad \frac{}{m(x).p \xrightarrow{m} \lambda x.p} \qquad\qquad \frac{}{\overline{m}\langle M\rangle.p \xrightarrow{\overline{m}} \langle M\rangle p} \quad \text{(OUT)}$$

$$\text{(INTER1)} \quad \frac{p \xrightarrow{m} F \qquad q \xrightarrow{\overline{m}} C}{p \mid q \xrightarrow{\tau} F@C} \qquad\qquad \frac{p \xrightarrow{\overline{m}} C \qquad q \xrightarrow{m} F}{p \mid q \xrightarrow{\tau} C@F} \quad \text{(INTER2)}$$

$$\text{(PAR1)} \quad \frac{p \xrightarrow{\alpha} a}{p \mid q \xrightarrow{\alpha} a \mid q} \qquad\qquad \frac{q \xrightarrow{\alpha} a}{p \mid q \xrightarrow{\alpha} p \mid a} \quad \text{(PAR2)}$$

$$\text{(RES)} \quad \frac{p \xrightarrow{\alpha} a \qquad \alpha \notin \{m, \overline{m}\}}{(\nu\, m)p \xrightarrow{\alpha} (\nu\, m)a} \qquad\qquad \frac{p \succ q \qquad q \xrightarrow{\alpha} a}{p \xrightarrow{\alpha} a} \quad \text{(RED)}$$

Table 1.3: `spi-calculus` semantics

on the reduction relation specified below:

$$!p \quad \succ \quad p \mid !p \tag{1.5}$$

$$[M = M]p \quad \succ \quad p \tag{1.6}$$

$$\textbf{case } \{M\}_k \textbf{ of } \{x\}_k \textbf{ in } p \quad \succ \quad p[M/x] \tag{1.7}$$

These rules give a way for simplifying the process.

An important thing to note is that if a process p tries to decrypt a message with a key different from the key used for the encryption, then p cannot perform any action.

1.2.2 Another LTS

In this section we define another LTS for the `spi-calculus` semantics. In the table are not reported the symmetric cases of the rules COM^l, $\mathrm{PAR}^l_{1\div 3}$ and CLOSE^l.

This system is obtained by removing the structural rules of the LTS given by Abadi and Gordon. In particular, the reduction rules for replication, matching and decryption are substituted by the rules (BANG), (MATCH) and (DECR) of the table 1.4. The interaction operator @ is explicited by the rules COM^l,

$$\text{IN}\quad \frac{}{m(x).p \xrightarrow{m} \lambda x.p} \qquad\qquad \frac{}{\overline{m}\langle M\rangle.p \xrightarrow{\overline{m}} \langle M\rangle p}\quad \text{OUT}$$

$$\text{COM}^l\quad \frac{p \xrightarrow{m} \lambda x.p' \qquad q \xrightarrow{\overline{m}} \langle M\rangle p}{p \mid q \xrightarrow{\tau} p'[M/x] \mid q'} \qquad\qquad \frac{p \xrightarrow{\overline{m}} \langle M\rangle q \qquad m \notin \vec{n} \qquad \vec{n} \cap fn(M) \neq \emptyset}{(\nu\,\vec{n})p \xrightarrow{\overline{m}} (\nu\,\vec{n})(\langle M\rangle q)}\quad \text{OPEN}$$

$$\text{RES}_1\quad \frac{p \xrightarrow{m} \lambda x.r \qquad m \notin \vec{m}}{(\nu\,\vec{m})p \xrightarrow{m} \lambda x.(\nu\,\vec{m})r} \qquad\qquad \frac{p \xrightarrow{m} \lambda x.p' \qquad q \xrightarrow{\overline{m}} (\nu\,\vec{n})(\langle M\rangle q')}{p \mid q \xrightarrow{\tau} (\nu\,\vec{n})(p'[M/x] \mid q)}\quad \text{CLOSE}^l$$

$$\text{RES}_2\quad \frac{p \xrightarrow{\tau} a}{(\nu\,\vec{m})p \xrightarrow{\tau} (\nu\,\vec{m})a} \qquad\qquad \frac{p \xrightarrow{\overline{m}} \langle M\rangle q \qquad \vec{m} \cap (fn(M) \cup \{m\}) = \emptyset}{(\nu\,\vec{m})p \xrightarrow{\overline{m}} \langle M\rangle(\nu\,\vec{m})q}\quad \text{RES}_3$$

$$\text{MATCH}\quad \frac{p \xrightarrow{\alpha} a}{[M = M]p \xrightarrow{\alpha} a} \qquad\qquad \frac{p \mid !p \xrightarrow{\alpha} a}{!p \xrightarrow{\alpha} a}\quad \text{BANG}$$

$$\text{PAR}^l_1\quad \frac{p \xrightarrow{\overline{m}} \langle M\rangle r}{p \mid q \xrightarrow{\overline{m}} \langle M\rangle(r \mid q)} \qquad\qquad \frac{p \xrightarrow{\overline{m}} (\nu\,\vec{n})(\langle M\rangle r) \quad \vec{n} \cap fn(q) = \emptyset \quad \vec{n} \cap fn(M) \neq \emptyset}{p \mid q \xrightarrow{\overline{m}} (\nu\,\vec{n})(\langle M\rangle(r \mid q))}\quad \text{PAR}^l_2$$

$$\text{PAR}^l_3\quad \frac{p \xrightarrow{m} \lambda x.r \qquad x \notin fn(q)}{p \mid q \xrightarrow{m} \lambda x.(r \mid q)} \qquad\qquad \frac{p \xrightarrow{\tau} r}{p \mid q \xrightarrow{\tau} r \mid q}\quad \text{PAR}^l_4$$

$$\text{DECR}\quad \frac{p[M/x] \xrightarrow{\alpha} a}{\textbf{case } \{M\}_k \textbf{ of } \{x\}_k \textbf{ in } p \xrightarrow{\alpha} a}$$

Table 1.4: Another `spi-calculus` LTS

OPEN and CLOSE^l, the equivalence 1.1 is modeled by the rule RES_1, the agent equivalence 1.2 is replaced by the rule PAR^l_3 and the rules $\mathrm{PAR}^l_{1\div 2}$ are introduced to eliminate the equivalence 1.4. We point out that the equivalence 1.3 is simply discharged because we avoid to move the restrictions "near" the parallel component that really uses a restricted name. Intuitively, the two LTS are equivalent and, in fact, if we use $\underset{ag}{\rightarrow}$ for the relation defined by the LTS of Abadi and Gordon, it is possible to prove the following result:

Proposition 1.1 *Let p a* `spi-calculus` *process. Then the transition $p \underset{ag}{\xrightarrow{\alpha}} a$ is derivable if, and only if, $p \xrightarrow{\alpha} a'$ (in the new LTS) for an $a' \stackrel{\triangle}{=} a$.*

In the proof of the proposition 1.1 we use the following two lemmas:

Lemma 1.1.1 *If $p \overset{m}{\underset{ag}{\rightarrow}} a$, then there exists a process r and a name x such that $a \overset{\triangle}{=} \lambda x.r$.*

Proof: We proof the lemma by induction on the length of the proof of the transition $p \overset{m}{\underset{\text{ag}}{\rightarrow}} a$.

Suppose that $p \overset{m}{\underset{\text{ag}}{\rightarrow}} a$ in one step, then the rule (IN) is used and p has the form $m(x).r$; in this case we have immediately the thesis.
Now suppose that the statement holds for derivation of length n and that $p \overset{m}{\underset{\text{ag}}{\rightarrow}} a$ is derivable in $n+1$ step. We proceed by case analysis on the last rule applied: if

- (RED) rule is used, then there exist two processes, q and r, such that $p \succ q \overset{m}{\underset{\text{ag}}{\rightarrow}} \lambda x.r$. Then, by inductive hypothesis, $q \overset{m}{\longrightarrow} \overset{\triangle}{=} \lambda x.r$ and this is the searched result.
- (PAR1) is the last rule applied, then $p \equiv p_1 \mid p_2$ and by inductive hypothesis $p_1 \overset{m}{\underset{\text{ag}}{\rightarrow}} \overset{\triangle}{=} \lambda x.r$ for some r, furthermore, without lost of generality, we can assume that $x \notin fn(p_2)$, so we can apply the equivalence 1.2 and obtain the proof.
- (PAR2) is used, the proof proceeds in an analogous manner of the proof in the case (PAR1).
- Otherwise, if the last rule used is (RES) then $p \equiv (\nu\, n)p'$ and by inductive hypothesis we have that $p' \overset{m}{\underset{\text{ag}}{\rightarrow}} \lambda x.r$; furthermore $n \neq m$; with these assumptions we have $p \overset{m}{\underset{\text{ag}}{\rightarrow}} (\nu\, n)\lambda x.r$ and, by the equivalence 1.1, we have the thesis.

$\diamond$

Lemma 1.1.2 *If $p \overset{\overline{m}}{\underset{ag}{\rightarrow}} a$, then there exist $\vec{n}, M$ and q such that $a \overset{\triangle}{=} (\nu\, \vec{n})\langle M \rangle q$. (Note that $\vec{n}$ may also be a void tuple of names).*

Proof: As for lemma 1.1.1, the thesis easily follows by induction on the length of the proof the only non trivial case being the (RES), (PAR1) and (PAR2) rules application.

Let (PAR1) the last rule applied. The process p has the form $p_1 \mid p_2$ and, by inductive hypothesis, there exist a (possibly void) tuple of names $\vec{n}$, a term M and a process q such that $p_1 \overset{\overline{m}}{\underset{\text{ag}}{\rightarrow}} (\nu\, \vec{n})\langle M \rangle q$. We can assume that $\vec{n} \cap fn(p_2) = \emptyset$, otherwise it is possible to α-rename the free names in p_2 that occur in q and M. By applying the equivalence 1.4 we proof the lemma.

The case where (PAR2) is the last rule applied in the derivation of the transition is similar to the case above.

In the (RES) case, by inductive hypothesis we have that $p \xrightarrow[\text{ag}]{\overline{m}} (\nu\,\vec{n})\langle M\rangle p'$ for some $\vec{n}$, M and p'. The proof is completed by using the equivalences 1.3.

$\diamond$

Proof of proposition 1.1.

($\Rightarrow$) The proof is given by induction on the length of the derivation $p \xrightarrow[\text{ag}]{\alpha} a$.

Base Case. If the derivation has length 1, then the (IN) rule or the (OUT) rule of the Abadi-Gordon LTS have been used, then, in the new LTS, the rules (IN) and (OUT) give us the same transition.

Inductive Case. Let $n+1$ the length of the derivation of $p \xrightarrow[\text{ag}]{\alpha} a$ and assume that

$$p \xrightarrow[\text{ag}]{\alpha} a \Rightarrow p \xrightarrow{\alpha} a' \qquad \text{where } a \stackrel{\triangle}{=} a'$$

for all the derivation with length less than $n+1$. We reason by case analysis on the last rule of such derivation:

PAR1. In this case, $p \equiv p_1 \mid p_2$, $p_1 \xrightarrow[\text{ag}]{\alpha} a$ and $p_1 \xrightarrow{\alpha} \stackrel{\triangle}{=} a$ by inductive hypothesis.
If $\alpha = \overline{m}$, for a name m, then, by Lemma 1.1.2, a has the form $(\nu\,\vec{n})\langle M\rangle r$ for suitable $\vec{n}$, M and r. The thesis follows by applying (PAR$^l_1$) or (PAR$^l_2$) depending on $\vec{n}$ is void or not.
In the case $\alpha = m$, by Lemma 1.1.1 $a \stackrel{\triangle}{=} \lambda x.r$ and then, given $y \notin fn(r, p_2)$, we have $p \xrightarrow{m} \lambda y.(r[y/x] \mid p_2)$ by rule (PAR$^l_3$).
If α is the silent action, the thesis follows by applying the rule (PAR$^l_4$) of the new LTS.

PAR2. The proof is similar to the previous case.

RES. The thesis follows easily by the inductive hypothesis.

RED. If the last rule applied is the RED rule, then $p \succ q \xrightarrow[\text{ag}]{\alpha} a$. By definition, $p \succ q$ if, and only if, reduction 1.5, 1.6 or 1.7 hold; by applying rule BANG, rule MATCH or rule DECR respectively we have the thesis.

INTER1. In this case, $p \equiv p_1 \mid p_2$, $p_1 \xrightarrow[\text{ag}]{m} F$, $p_2 \xrightarrow[\text{ag}]{\overline{m}} C$ and $a \equiv F@C$. By Lemmas 1.1.1 and 1.1.2, $F \equiv \lambda x.r$ and C has the form $(\nu\,\vec{n})\langle M\rangle q$. If $\vec{n}$ is a void tuple of names, we can apply the rule COM^l of the new LTS and obtain, by inductive hypothesis, $p \xrightarrow{\tau} \langle M\rangle q$; otherwise, the CLOSE^l rule gives the thesis.

INTER2. We proceed analogously to the case INTER1.

($\Leftarrow$) The proof proceed by induction on the length of the derivation $p \xrightarrow{\alpha} a$ in this case also.

Base Case. If the derivation is obtained by IN or OUT rule of the new LTS, then an equivalent transition can be performed in the LTS of Abadi and Gordon with their IN or OUT rule, respectively.

Inductive Case. In the inductive step, the proof proceeds by case analysis on the last rule applied. We assume that the statement holds for derivations with length less than $n+1$ and prove the thesis for $n+1$-steps derivations.

OPEN. If the last rule is OPEN then in the new LTS there exists a derivation of the transition $p \xrightarrow{\overline{m}} \langle M \rangle q$ with length n from which it is possible to derive $(\nu\, \vec{n})p \xrightarrow{\overline{m}} (\nu\, \vec{n})\langle M \rangle q$ where $m \notin \vec{n}$. Then, by inductive hypothesis, $p \underset{\text{ag}}{\xrightarrow{\overline{m}}} \triangleq \langle M \rangle q$ and by applying the RES rule k times, where k is the length of the tuple $\vec{n}$, we have the thesis.

COM^l. We have that $p \equiv p_1 \mid p_2$ and, there are two derivations of length less than $n+1$ for $p_1 \xrightarrow{m} \lambda x.r$ and $p_2 \xrightarrow{\overline{m}} \langle M \rangle q$, respectively. Then, by inductive hypothesis, $p_1 \underset{\text{ag}}{\xrightarrow{m}} \lambda x.r$ and $p_2 \underset{\text{ag}}{\xrightarrow{\overline{m}}} \langle M \rangle q$. The rule INTER1 of the Abadi-Gordon LTS is applicable to obtain $p \underset{\text{ag}}{\xrightarrow{\tau}} p_1[M/x] \mid p_2$.

CLOSE^l. This case is analogous to the previous one.

RES$_{1,2}$. The thesis follows trivially by the inductive hypothesis.

BANG. If the BANG rule is applied, it is sufficient to note that $!p \succ p \mid !p$ and that, by induction hypothesis, $p \mid !p \underset{\text{ag}}{\xrightarrow{\alpha}} \triangleq a$; then it is possible to apply the RED rule of the Abadi-Gordon LTS.

MATCH. The proof is similar to the BANG case.

PAR$^l_1$. Let $p \equiv p_1 \mid p_2$. A derivation in less than $n+1$-steps of $p_1 \xrightarrow{\overline{m}} \langle M \rangle r$ there exists in the new LTS. Then $p_1 \underset{\text{ag}}{\xrightarrow{\overline{m}}} \langle M \rangle r$ and, by PAR1 rule, $p \underset{\text{ag}}{\xrightarrow{\overline{m}}} \langle M \rangle r \mid p_2$. The equivalence 1.4 give the thesis.

PAR$^l_2$. Proceeding as in PAR$^l_1$, we have $p_1 \underset{\text{ag}}{\xrightarrow{\overline{m}}} (\nu\, \vec{n})\langle M \rangle r$ with $\vec{n} \cap fn(p_2) = \emptyset$. Again, by equivalence 1.4 we obtain the thesis.

PAR$^l_3$. Suppose $p \equiv p_1 \mid p_2$ and $p_1 \xrightarrow{m} \lambda x.r$ is derivable in the new LTS in less than $n+1$ steps and $x \notin fn(p_2)$. Then $p_1 \underset{\text{ag}}{\xrightarrow{m}} \lambda x.r$; the rule PAR1 and the equivalence 1.2 give the thesis.

PAR$^l_4$. This case trivially follows from inductive hypothesis.

DECR. We have that $p \equiv$ **case** $\{M\}_k$ **of** $\{x\}_k$ **in** p. Then, by reduction rule 1.7 $p \succ p[M/x]$ and by hypothesis, $p[M/x] \xrightarrow{\alpha} a$ with a derivation shorter than $n+1$ steps. We can apply the inductive hypothesis and the RED rule to obtain the thesis.

$\diamond$

Chapter 2

Logical Frameworks

The Edimburg Logical Framework [HHP93] is a general "environment" in which it is possible to specify an arbitrary formal system. Logical frameworks (from now on, LF for short) adopt the Curry-Howard isomorphism ("formulas as types", "proofs as λ-terms" principle) in order to provide a metalanguage in which it is possible to present an arbitrary logical system. In this regard, a system is thought as a "calculus" in which express a collection of judgments about some entities.

LF relies on typed λ-calculus with first-order dependent types. The type system has three level of terms: *objects*, *types* and *kinds*. Objects (denoted by M, N) represent syntactic entities, proofs or inference rules of the logical system. Types (meta-variables A, B, C) are used to classify objects; families of type may be thought as mapping from objects to types. Syntactic classes, judgment of assertions are represented via types and families of types. Kinds (denoted by K, L) classify (families of) types.

The abstract syntax of LF is:

$$\begin{array}{lrcl} \textit{Objects} & M & ::= & x \mid MN \mid \lambda x : A.M \\ \textit{Types} & A & ::= & X \mid AM \mid \prod_{x:A}.B \mid \lambda x : A.B \\ \textit{Kinds} & K & ::= & \mathsf{Type} \mid \prod_{x:A}.K \end{array}$$

An object is a term of simply typed λ-calculus (with dependent types), i.e. an object may be a variable, the application of two objects or an object abstraction.

(Families of) types are type variables, the application of a type to an object. $\prod$ is the dependent-type constructor. The type $\prod_{x:A}.B_x$ denotes the type of the functions that have domain A and whose codomain depends on the input, i.e. $f(a) \in B_a$. When x does not appear in B, we abbreviate $\prod_{x:A}.B$ with $A \longrightarrow B$; in fact, in this case we simply have the functions from A to B. As usual, we assume that $\longrightarrow$ is right-associative. We remark here, that in the "propositions as type" principle, $\prod_{x:A}.B$ corresponds to $\forall x \in A.B$. Note that both $\prod$ and λ are binding operators: they bind x in the second argument.

Kinds are introduced only for technical reason and are used to classify types. A dependent type has kind $\prod_{x:A}.K$. A generic LF-term will be denoted by U, V and W.

An *environment* is a finite list of pairs consisting of a variable and its type or kind:

$$\textit{Environments} \quad \Gamma \quad ::= \quad \langle\rangle \mid \Gamma, x : A \mid \Gamma, X : K$$

The Edimburg Logical Framework is equipped with a notion of *definitional equality* that is extremely simple; in fact it is the β-equivalence of the LF entities of all the three levels. The definitional equality is the transitive and symmetric closure of the *parallel nested reduction* relation, defined in table 2.1.

$$\begin{array}{ccc}
\dfrac{}{M \to M} & \dfrac{M \to M' \qquad N \to N'}{(\lambda x : A.M)N \to (\lambda x : A.M')N'} & \dfrac{B \to B' \qquad N \to N'}{(\lambda x : A.B)N \to B'[N'/x]} \\[2ex]
\dfrac{M \to M' \qquad N \to N'}{MN \to M'N'} & \dfrac{A \to A' \qquad M \to M'}{AM \to A'M'} & \dfrac{A \to A' \qquad M \to M'}{\lambda x : A.M \to \lambda x : A'.M'} \\[2ex]
\dfrac{A \to A' \qquad B \to B'}{\lambda x : A.B \to \lambda x : A'.B'} & \dfrac{A \to A' \qquad B \to B'}{\prod_{x:A}.B \to \prod_{x:A'}.B'} & \dfrac{A \to A' \qquad K \to K'}{\prod_{x:A}.K \to \prod_{x:A'}.K'}
\end{array}$$

Table 2.1: The LF reduction relation

The LF type theory is a context for deriving assertion of the following form:

$$\begin{array}{lcl}
\vdash \Gamma & \equiv & \text{“}\Gamma \text{ is a valid environment”} \\
\Gamma \vdash K & \equiv & \text{“}K \text{ is a kind in } \Gamma\text{”} \\
\Gamma \vdash A : K & \equiv & \text{“}A \text{ has kind } K \text{ in } \Gamma\text{”} \\
\Gamma \vdash M : A & \equiv & \text{“}M \text{ has type } A \text{ in } \Gamma\text{”}
\end{array}$$

The rules for deriving those judgments are given in table 2.2. The environment rules state that a *valid environment* is an eventually void list of pairs object/type or type/kind variables. Any variable appears in this list at most one time. A *valid kind* is Type or a dependent kind whose “body” is a valid kind. *Valid type families* or *valid objects* are obtained by typical rules for deriving well-formed terms in a typed λ-calculus.

2.1 A Logical Framework for the `spi-calculus`

Our encoding of `spi-calculus` in LF will follow the scheme adopted in [HLMP98]. In particular, we will map the syntactic categories of the source language in the following type variables:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\mathcal{L} & : & \mathsf{Type} & \mathcal{P} & : & \mathsf{Type} \\
\mathcal{N}_n & : & \mathsf{Type} & \mathcal{A} & : & \mathsf{Type} \\
\mathcal{N}_e & : & \mathsf{Type} & \mathcal{E} & : & \mathsf{Type}
\end{array}$$

$\mathcal{L}$ is the label type, $\mathcal{N}_n$ and $\mathcal{N}_e$ are a “partition” of the names. An object variable $m : \mathcal{N}_n$ is a variable that may denote only channels that may communicate names; $m : \mathcal{N}_e$ is a channel that may carry encrypted messages. As stated before, we assume that keys are names, so we can choose between $\mathcal{N}_n$ and $\mathcal{N}_e$:

Environments	$\dfrac{}{\vdash \langle\rangle}$ $\qquad$ $\dfrac{\vdash \Gamma \qquad \Gamma \vdash K \qquad X \notin dom(\Gamma)}{\vdash \Gamma, X : K}$ $\dfrac{\vdash \Gamma \qquad \Gamma \vdash A : \mathsf{Type} \qquad x \notin dom(\Gamma)}{\vdash \Gamma, x : A}$
Kind	$\dfrac{\vdash \Gamma}{\Gamma \vdash \mathsf{Type}}$ $\qquad$ $\dfrac{\Gamma, x : A \vdash K}{\Gamma \vdash \prod_{x:A}.K}$
Families	$\dfrac{\vdash \Gamma \qquad X : K \in \Gamma}{\Gamma \vdash X : K}$ $\qquad$ $\dfrac{\Gamma, x : A \vdash B : \mathsf{Type}}{\Gamma \vdash \prod_{x:A}.B : \mathsf{Type}}$ $\dfrac{\Gamma, x : A \vdash B : K}{\Gamma \vdash \lambda x : A.B : \prod_{x:A}.K}$ $\qquad$ $\dfrac{\Gamma \vdash A : \prod_{x:B} K \qquad \Gamma \vdash M : B}{\Gamma \vdash AM : K[M/x]}$ $\dfrac{\Gamma \vdash A : K \qquad \Gamma \vdash K' \qquad \Gamma \vdash K \equiv K'}{\Gamma \vdash A : K'}$
Objects	$\dfrac{\vdash \Gamma \qquad x : A \in \Gamma}{\Gamma \vdash x : A}$ $\qquad$ $\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : \prod_{x:A}.B}$ $\dfrac{\Gamma \vdash M : \prod_{x:A}.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$ $\qquad$ $\dfrac{\Gamma \vdash M : \prod_{x:A}.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$

Table 2.2: LF judgment rules

say we use $\mathcal{N}_n$. The variables $\mathcal{P}$ and $\mathcal{A}$ are the process and agent type variables respectively. $\mathcal{E}$ is the type for the encryptions.

Two auxiliary types variable, introduced for technical reasons, are

$$\begin{array}{rcl} \mathbf{D} & : & \mathsf{Type} \\ \aleph & : & \mathsf{Type} \end{array}$$

The constructors for $\mathbf{D}$ are three "constants": $\delta_n : \mathbf{D}$, $\delta_e : \mathbf{D}$ and $\delta_l : \mathbf{D}$. These constants are used to model types that depends on the (type of) a channel. The constructors for $\aleph$ are $\mathbf{0} : \aleph$ and $\mathbf{succ} : \aleph \to \aleph$ (in the rest of the paper, we will write $z + 1$ in place of $\mathbf{succ}(z)$); then it is easy to see that support of $\aleph$ is isomorph to the set of natural numbers and, in fact, we will use $\aleph$ to model list of names with a certain length, i.e. $\prod_{z:\aleph}.\mathcal{N}_l{}^{(z)}$, where the constructors $\perp$, $\hookrightarrow_\psi$

for ${\mathcal{N}_l}^{(z)}$ are $(\psi = e, n)$:

$$\bot : {\mathcal{N}_l}^{(\mathbf{0})} \qquad\qquad \hookrightarrow_\psi : \mathcal{N}_\psi \longrightarrow \prod_{z:\aleph} .{\mathcal{N}_l}^{(z)} \longrightarrow {\mathcal{N}_l}^{(z+1)}$$

and for ${\mathcal{N}_l}^{(z)}$ are

$$\mathrm{hd} : \prod_{z:\aleph} .\prod_{d:\mathbf{D}} .{\mathcal{N}_l}^{(z+1)} \longrightarrow \mathcal{N}(d) \qquad\qquad \mathrm{tl} : \prod_{z:\aleph} .{\mathcal{N}_l}^{(z+1)} \longrightarrow {\mathcal{N}_l}^{(z)}$$

From now on, we will write $\mathrm{hd}^z_d(l)$ and $\mathrm{tl}^z(l)$ in place of $\mathrm{hd}(z, d, l)$ and $\mathrm{tl}(z, l)$, respectively.

We introduce the type expression below:

$$\mathcal{N}(d) = \begin{cases} \mathcal{N}_n, & \text{if } d = \delta_n \\ \mathcal{N}_e, & \text{if } d = \delta_e \\ \prod_{z:\aleph} .{\mathcal{N}_l}^{(z)}, & \text{if } d = \delta_l \end{cases} \qquad \mathcal{T}(d, d') = \begin{cases} \mathcal{E}, & \text{if } d = \delta_e \\ \mathcal{N}(d'), & \text{if } d = \delta_n \text{ and } d' \neq \delta_l \\ \prod_{z:\aleph} .{\mathcal{N}_l}^{(z)}, & \text{if } d = \delta_l \end{cases}$$

The list of names are introduced because the restriction operator ν of the `spi-calculus` acts on (finite) tuple of names.

The syntactic constructors are translated in term variables of the appropriate type and are reported in table 2.3. Some comments are in order.

tau	:	$\mathcal{L}$	
get^d	:	$\mathcal{N}(d) \longrightarrow \mathcal{L}$	*Labels*
put^d	:	$\mathcal{N}(d) \longrightarrow \mathcal{L}$	
$\mathrm{Tch}^{d,d'}$	:	$(\mathcal{N}(d) \longrightarrow \mathcal{T}(d, d'))$	*Terms*
$\mathrm{Tenc}^{d,d'}$	:	$\mathcal{E} \longrightarrow \mathcal{T}(d, d')$	
$\{\_\}\_^{d,d'}$	:	$\mathcal{T}(d, d') \longrightarrow \mathcal{N}_e \longrightarrow \mathcal{E}$	*Encryptions*
$\mathrm{out}^{d,d'}$	:	$\mathcal{N}(d) \longrightarrow \mathcal{T}(d, d') \longrightarrow \mathcal{P} \longrightarrow \mathcal{P}$	
$\mathrm{in}^{d,d'}$	:	$\mathcal{N}(d) \longrightarrow (\mathcal{T}(d, d') \longrightarrow \mathcal{P}) \longrightarrow \mathcal{P}$	
$\mid$	:	$\mathcal{P} \longrightarrow \mathcal{P} \longrightarrow \mathcal{P}$	
ν	:	$(\mathcal{N}_l \longrightarrow \mathcal{A}) \longrightarrow \mathcal{P}$	
!	:	$\mathcal{P} \longrightarrow \mathcal{P}$	*Processes*
$[\_ = \_]^{d,d'}$	:	$\mathcal{T}(d, d') \longrightarrow \mathcal{T}(d, d') \longrightarrow \mathcal{P} \longrightarrow \mathcal{P}$	
nil	:	$\mathcal{P}$	
decr^d	:	$\mathcal{E} \longrightarrow \mathcal{N}_n \longrightarrow (\mathcal{T}(d, \delta_n) \longrightarrow \mathcal{P}) \longrightarrow \mathcal{P}$	
proc	:	$\mathcal{P} \longrightarrow \mathcal{A}$	*Agents*
$\mathrm{conc}^{d,d'}$	:	$\mathcal{T}(d, d') \longrightarrow \mathcal{P} \longrightarrow \mathcal{A}$	

Table 2.3: Syntactic constructors

The labels constructors get^d and put^d take a name in $\mathcal{N}_n$ or in $\mathcal{N}_e$ (depending on d) and return a label for the input or the output, respectively. The terms are built with Tch and Tenc for channels and encryption. The encryption constructors is $\{\_\}_\_^{d,d'}$; they take a term (of the correct type $\mathcal{T}(d,d')$), a key and return an encryption. The process constructors are quite intuitive; we only want to underline that, for the input process, the $\text{in}^{d,d'}$ constructors take a name (the input channel) and a process abstraction (the abstraction is on term variables and represents the continuation of the input action). This is because the input channel might be a channel for encryptions. Also note that the operator ν acts on an abstraction from lists of names to processes.

Observe that there is no constructor analogous to the `spi-calculus` abstractions $\lambda x.p$: it is possible to represent such agents with a suitable LF λ-abstraction. In the rest of the paper, the following conventions are adopted in order to facilitate readability: the constructor tau will be represented as τ; m and $\overline{m}$ are used if place of $\text{get}^d(m)$ and $\text{put}^d(m)$ respectively (provided that $d = \hat{m}$, see below);

2.1.1 LF syntax of the `spi-calculus`

In the previous section, we have defined a LF signature suitable to encode the `spi-calculus`. Some care must be taken in the translation of names. The problem is that in the LF context, every name has a particular type ($\mathcal{N}_n$ or $\mathcal{N}_e$); this avoids to derive terms in which one can send objects of different type on the same channel. On the other hand, such terms are derivable in the (untyped) `spi-calculus`, delegating to the LTS the discovery of such "errors". For this reason, we use the function $\hat{\bullet} : \mathcal{N} \longrightarrow \{\delta_n, \delta_e\}$ that assigns "types" to names, ($\hat{n}$ denotes the application of $\hat{\bullet}$ to n) and the intended meaning is that $\hat{n} = \delta_n$ if n is a name carrying channel, otherwise, if n is used to exchange encryptions, $\hat{n} = \delta_e$; in other words, $n : \mathcal{N}(\hat{n})$. A name carrying channel compels us to define the function $\breve{\bullet} : \mathcal{N}_n \longrightarrow \{\delta_n, \delta_e\}$ that, given a name $n : \mathcal{N}_n$, returns the type of the object names that may be communicated along n, if $\hat{n} = \delta_n$, otherwise returns δ_e. The function $\widehat{\bullet} : \mathit{term} \longrightarrow \{\delta_n, \delta_e\}$ given a `spi`-term M, returns δ_n or δ_e depending on M is a channel/key term or an encryption term (the application of $\widehat{\bullet}$ to M is written as $\widehat{M}$).

With the above assumption it is possible to translate the terms with the following equations:

$$[\![n]\!] = \begin{cases} \text{Tch}^{\delta_n,\breve{n}}(n), & \text{if } \hat{n} = \delta_n \\ \text{Tch}^{\delta_e,\delta_e}(n), & \text{if } \hat{n} = \delta_e \end{cases}$$

$$[\![\{M\}_k]\!] = \begin{cases} \text{Tenc}^{\hat{M},\breve{M}}(\{[\![M]\!]\}_{[\![k]\!]}^{\delta_n,\hat{M}}) & \text{if } \widehat{M} = \delta_n \\ \text{Tenc}^{\widehat{M},\widehat{M}}(\{[\![M]\!]\}_{[\![k]\!]}^{\delta_e,\delta_e}) & \text{if } \widehat{M} = \delta_e \end{cases}$$

The encoding of the `spi`-processes is given in table 2.4 The translation is quite natural. We remark that, in order to encode $\overline{n}\langle M\rangle.p$, the "types" of the channel and of the message are required to be "compatible". The abstraction in the translation of the input prefix, $n(x).p$, allows to use the variable x as a "placeholder" in p; similarly for the **case** construct.

Some comments about the translation of the restriction operator are required. The translation of a process p appearing under the restriction of the names $n_1, \ldots n_k$ is obtained by applying the ν constructor (of the logical framework) to a particular agent abstraction. In fact, the translation is obtained by

$$
\begin{array}{rcl}
[\![n(x).p]\!] & = & \mathrm{in}^{\hat{n},\hat{x}}([\![n]\!],\ \lambda x:\mathcal{T}(\hat{n},\hat{x}).[\![p]\!][x/[\![x]\!]]) \\[2ex]
[\![\overline{n}\langle M\rangle.p]\!] & = & \mathrm{out}^{\hat{n},\widehat{M}}([\![n]\!],\ [\![M]\!],\ [\![p]\!]) \qquad \text{if } \hat{n}=\widehat{M} \\[2ex]
[\![0]\!] & = & \mathrm{nil} \\[2ex]
[\![p \mid q]\!] & = & [\![p]\!] \ \mid [\![q]\!] \\[2ex]
[\![!p]\!] & = & ![\![p]\!] \\[2ex]
[\![(\nu\, n_1)\ldots(\nu\, n_k)p]\!] & = & \nu\,(\lambda l:\mathcal{N}_l{}^{(k)}.\mathrm{proc}([\![p]\!]\sigma)) \\
& & \text{where } \sigma=[\mathrm{hd}_{\hat{n}_i}^{k-i+1}(\mathrm{tl}^{k-i+1}(l_i))/[\![n_i]\!]]_{i=1,\ldots,k} \\
& & \qquad l_1=l \quad l_{i+1}=\mathrm{tl}^{i+1}(l_i) \quad i=1,\ldots,k-1 \\[2ex]
[\![[M=N]p]\!] & = & \begin{cases} [\,[\![M]\!]=[\![N]\!]\,]^{\widehat{M},\widehat{M}}[\![p]\!] & \text{if } \widehat{M}=\delta_n \\ [\,[\![M]\!]=[\![N]\!]\,]^{\delta_e,\delta_e}[\![p]\!] & \text{if } \widehat{M}=\delta_e \end{cases} \\[2ex]
[\![\textbf{case } M \textbf{ of } \{x\}_k \textbf{ in } p]\!] & = & \mathrm{decr}^{\hat{x}}([\![M]\!],[\![k]\!],\lambda x:\mathcal{T}(\hat{x},\delta_n).[\![p]\!][x/[\![x]\!]])
\end{array}
$$

Table 2.4: LF Encoding

abstracting away from (the translation) of p a list of k names whose i-th name has the same type of n_i.

2.1.2 LF semantics of the `spi-calculus`

Judgments are encoded as variables whose kind is a typed-valued function and rules are term-variable of the appropriate type. In this encoding an assertion becomes a term of a particular type and, therefore, checking if a derivation is correct corresponds to type-check the term well-typedness.

As in [HLMP98], two judgments are introduced:

$$
\begin{array}{rcl}
\longmapsto & : & \mathcal{P} \longrightarrow \mathcal{L} \longrightarrow \mathcal{A} \longrightarrow \mathsf{Type} \\
\longmapsto\!\!\!\!\rightarrow & : & \prod_{d,d':\mathbf{D}}.\mathcal{P} \longrightarrow \mathcal{L} \longrightarrow (\mathcal{T}(d,d') \longrightarrow \mathcal{A}) \longrightarrow \mathsf{Type}
\end{array}
$$

The first is used to model the unbound transitions, the latter for the bound actions. Observe that, roughly speaking, $\longmapsto\!\!\!\!\rightarrow$ relates a process, a label (that will be an input label or an bound output label) with an abstraction of agent. We write $p \overset{\alpha}{\longmapsto} a$ and $p \underset{d,\,d'}{\overset{\alpha}{\longmapsto\!\!\!\!\rightarrow}} a$ instead of $\longmapsto p\alpha a$ and $\longmapsto\!\!\!\!\rightarrow dd'p\alpha a$ respectively.

The LF-rules give in table 2.5.

Table 2.5: LF rules

IN	:	$\prod_{d,d':\mathbf{D}}.\ \prod_{m:\mathcal{N}(d)}.\ \prod_{p:\mathcal{T}(d,d')} \longrightarrow \mathcal{P}.$ $\mathrm{in}^{d,d'}(m,p) \underset{d,\,d'}{\overset{m}{\longmapsto\!\!\!\!\rightarrow}} \lambda t:\mathcal{T}(d,d').\mathrm{proc}(pt)$

Table 2.5: LF rules

OUT	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{m:\mathcal{N}(d)}\cdot \prod_{t:\mathcal{T}(d,d')}\cdot \prod_{p:\mathcal{P}}\cdot$ $\mathrm{out}^{d,d'}(m,t,p) \overset{\overline{m}}{\longmapsto} \mathrm{conc}^{d,d'}(t,p)$
COM^{l}	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{m:\mathcal{N}(d)}\cdot \prod_{p,q,q':\mathcal{P}}\cdot \prod_{p':\mathcal{T}(d,d')} \longrightarrow \mathcal{P}\cdot$ $\prod_{t:\mathcal{T}(d,d')}\cdot\ p \overset{m}{\underset{d,\,d'}{\twoheadrightarrow}} \lambda t' : \mathcal{T}(d,d').\mathrm{proc}(p't') \longrightarrow$ $q \overset{\overline{m}}{\longmapsto} \mathrm{conc}^{d,d'}(t,q') \longrightarrow (p \mid q) \overset{\tau}{\longmapsto} \mathrm{proc}(p't \mid q')$
OPEN	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{z:\aleph}\cdot \prod_{p,p':\mathcal{N}_l^{(z)}} \longrightarrow \mathcal{P}\cdot \prod_{m:\mathcal{N}(d)}\cdot \prod_{t:\mathcal{N}_l^{(z)}} \longrightarrow \mathcal{T}(d,d')\cdot$ $(\prod_{l':\mathcal{N}_l^{(z)}}\cdot\ pl' \overset{\overline{m}}{\longmapsto} \mathrm{conc}^{d,d'}(tl',p'l')) \longrightarrow$ $\nu\,(\lambda l'' : \mathcal{N}_l^{(z)}.proc(pl'')) \overset{\overline{m}}{\underset{\delta_l,\,d'}{\twoheadrightarrow}} \lambda l : \mathcal{N}_l^{(z)}.\mathrm{conc}^{d,d'}(tl,p'l)$
CLOSE^{l}	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{z:\aleph}\cdot \prod_{p,q:\mathcal{P}}\cdot \prod_{m:\mathcal{N}(d)}\cdot \prod_{p':\mathcal{T}(d,d')} \longrightarrow \mathcal{P}\cdot$ $\prod_{q':\mathcal{N}_l^{(z)}} \longrightarrow \mathcal{P}\cdot \prod_{t:\mathcal{N}_l^{(z)}} \longrightarrow \mathcal{T}(d,d')\cdot$ $p \overset{m}{\underset{d,\,d'}{\twoheadrightarrow}} \lambda t' : \mathcal{T}(d,d').\mathrm{proc}(p't') \longrightarrow$ $q \overset{\overline{m}}{\underset{d,\,d'}{\twoheadrightarrow}} \lambda l'' : \mathcal{N}_l^{(z)}.\mathrm{conc}^{d,d'}(tl'',q'l'') \longrightarrow$ $(p \mid q) \overset{\tau}{\longmapsto} \mathrm{proc}(\nu\,(\lambda l : \mathcal{N}_l^{(z)}.\mathrm{proc}(p'(tl) \mid q'l)))$
PAR$_1^{l}$	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{p,q,r:\mathcal{P}}\cdot \prod_{m:\mathcal{N}(d)}\cdot \prod_{t:\mathcal{T}(d,d')}\cdot$ $p \overset{\overline{m}}{\longmapsto} \mathrm{conc}^{d,d'}(t,r) \longrightarrow (p \mid q) \overset{\overline{m}}{\longmapsto} \mathrm{conc}^{d,d'}(t,r \mid q)$
PAR$_2^{l}$	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{z:\aleph}\cdot \prod_{p,q:\mathcal{P}}\cdot \prod_{r:\mathcal{N}_l^{(z)}} \longrightarrow \mathcal{P}\cdot \prod_{m:\mathcal{N}(d)}\cdot$ $\prod_{t:\mathcal{N}_l^{(z)}} \longrightarrow \mathcal{T}(d,d')\cdot$ $p \overset{\overline{m}}{\underset{\delta_l,\,d'}{\twoheadrightarrow}} \lambda l : \mathcal{N}_l^{(z)}.\mathrm{conc}^{d,d'}(tl,rl) \longrightarrow$ $(p \mid q) \overset{\overline{m}}{\underset{\delta_l,\,d'}{\twoheadrightarrow}} \lambda l' : \mathcal{N}_l^{(z)}.\mathrm{conc}^{d,d'}(tl',rl' \mid q))$
PAR$_3^{l}$	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{p,q:\mathcal{P}}\cdot \prod_{r:\mathcal{T}(d,d')} \longrightarrow \mathcal{P}\cdot \prod_{m:\mathcal{N}(d)}\cdot$ $p \overset{m}{\underset{d,\,d'}{\twoheadrightarrow}} \lambda t : \mathcal{T}(d,d').\mathrm{proc}(rt) \longrightarrow$ $(p \mid q) \overset{m}{\underset{d,\,d'}{\twoheadrightarrow}} \lambda t : \mathcal{T}(d,d').\mathrm{proc}(rt \mid q)$
PAR$_4^{l}$	:	$\prod_{p,q,r:\mathcal{P}}\cdot\ p \overset{\tau}{\longmapsto} \mathrm{proc}(r) \longrightarrow (p \mid q) \overset{\tau}{\longmapsto} \mathrm{proc}(r \mid q)$
BANG$_f$	:	$\prod_{p:\mathcal{P}}\cdot \prod_{a:\mathcal{A}}\cdot \prod_{\alpha:\mathcal{L}}\cdot\ (p \mid !p) \overset{\alpha}{\longmapsto} a \longrightarrow !p \overset{\alpha}{\longmapsto} a$
BANG$_b$	:	$\prod_{d,d':\mathbf{D}}\cdot \prod_{p:\mathcal{P}}\cdot \prod_{\alpha:\mathcal{L}}\cdot \prod_{a:\mathcal{T}(d,d')} \longrightarrow \mathcal{A}\cdot$ $(p \mid !p) \overset{\alpha}{\underset{d,\,d'}{\twoheadrightarrow}} a \longrightarrow !p \overset{\alpha}{\underset{d,\,d'}{\twoheadrightarrow}} a$

Table 2.5: LF rules

RES^1_f	:	$\prod_{z:\aleph}\cdot\ \prod_{p:\mathcal{N}_l{}^{(z)}} \longrightarrow \mathcal{P}\cdot\ \prod_{q:\mathcal{N}_l{}^{(z)}} \longrightarrow \mathcal{P}\cdot$ $(\prod_{l':\mathcal{N}_l{}^{(z)}}.pl' \xmapsto{\tau} ql') \longrightarrow$ $\nu\,(\lambda l : \mathcal{N}_l{}^{(z)}.\mathrm{proc}(pl)) \xmapsto{\tau} \mathrm{proc}(\nu\,(q))$
RES^2_f	:	$\prod_{d,d':\mathbf{D}}\cdot\ \prod_{z:\aleph}\cdot\ \prod_{p,q:\mathcal{N}_l{}^{(z)}} \longrightarrow \mathcal{P}\cdot\ \prod_{m:\mathcal{N}(d)}\cdot\ \prod_{t:\mathcal{T}(d,d')}$ $(\prod_{l':\mathcal{N}_l{}^{(z)}}\cdot\ pl' \xmapsto{\overline{m}} \mathrm{conc}^{d,d'}(t, ql')) \longrightarrow$ $\nu\,(\lambda l : \mathcal{N}_l{}^{(z)}.\mathrm{proc}(pl)) \xmapsto{\overline{m}} \mathrm{conc}^{d,d'}(t, \nu\,(\lambda l : \mathcal{N}_l{}^{(z)}.\mathrm{proc}(ql)))$
RES_b	:	$\prod_{d,d':\mathbf{D}}\cdot\ \prod_{z:\aleph}\cdot\ \prod_{p:\mathcal{N}_l{}^{(z)}} \longrightarrow \mathcal{P}\cdot\ \prod_{q:\mathcal{N}_l{}^{(z)}} \longrightarrow \mathcal{T}(d,d') \longrightarrow \mathcal{A}\cdot\ \prod_{\alpha:\mathcal{L}}\cdot$ $(\prod_{l':\mathcal{N}_l{}^{(z)}}\cdot\ pl' \overset{\alpha}{\underset{d,\,d'}{\longmapsto\!\!\!\!\twoheadrightarrow}} ql') \longrightarrow$ $\nu\,(\lambda l : \mathcal{N}_l{}^{(z)}.\mathrm{proc}(pl)) \overset{\alpha}{\underset{d,\,d'}{\longmapsto\!\!\!\!\twoheadrightarrow}} \lambda t : \mathcal{T}(d,d').\mathrm{proc}(\nu\,(\lambda l'' : \mathcal{N}_l{}^{(z)}.ql''t))$
COND_f	:	$\prod_{d,d':\mathbf{D}}\cdot\ \prod_{p:\mathcal{P}}\cdot\ \prod_{a:\mathcal{T}(d,d')} \longrightarrow \mathcal{A}\cdot\ \prod_{\alpha:\mathcal{L}}\cdot$ $\prod_{t:\mathcal{T}(d,d')}\cdot\ p \xmapsto{\alpha} a \longrightarrow [t=t]^{d,d'}p \xmapsto{\alpha} a$
COND_b	:	$\prod_{d,d':\mathbf{D}}\cdot\ \prod_{p:\mathcal{P}}\cdot\ \prod_{\alpha:\mathcal{L}}\cdot\ \prod_{a:\mathcal{T}(d,d')} \longrightarrow \mathcal{A}\cdot\ \prod_{d_1,d_2:\mathbf{D}}\cdot$ $\prod_{t:\mathcal{T}(d_1,d_2)}\cdot\ p \overset{\alpha}{\underset{d,\,d'}{\longmapsto\!\!\!\!\twoheadrightarrow}} a \longrightarrow [t=t]^{d_1,d_2}p \overset{\alpha}{\underset{d,\,d'}{\longmapsto\!\!\!\!\twoheadrightarrow}} a$
DECR_f	:	$\prod_{d:\mathbf{D}}\cdot\ \prod_{p:\mathcal{T}(d,\delta_n)} \longrightarrow \mathcal{P}\cdot\ \prod_{t:\mathcal{T}(d,\delta_n)}\cdot\ \prod_{a:\mathcal{A}}\cdot\ \prod_{\alpha:\mathcal{L}}\cdot\ \prod_{k:\mathcal{N}_n}\cdot$ $pt \xmapsto{\alpha} a \longrightarrow \mathrm{decr}^d(\{t\}_k{}^{d,\delta_n}, k, p) \xmapsto{\alpha} a$
DECR_b	:	$\prod_{d:\mathbf{D}}\cdot\ \prod_{p:\mathcal{T}(d,\delta_n)} \longrightarrow \mathcal{P}\cdot\ \prod_{t:\mathcal{T}(d,\delta_n)}\cdot\ \prod_{\alpha:\mathcal{L}}\cdot$ $\prod_{d_1,d_2:\mathbf{D}}\cdot\ \prod_{a:\mathcal{T}(d_1,d_2)} \longrightarrow \mathcal{A}\cdot\ \prod_{k:\mathcal{N}_n}\cdot$ $pt \overset{\alpha}{\underset{d_1,\,d_2}{\longmapsto\!\!\!\!\twoheadrightarrow}} a \longrightarrow \mathrm{decr}^d(\{t\}_k{}^{d,\delta_n}, k, p) \overset{\alpha}{\underset{d_1,\,d_2}{\longmapsto\!\!\!\!\twoheadrightarrow}} a$

The rules COM^r, CLOSE^r and $\mathrm{PAR}^r_{1\div 4}$ are analogous to their left version and have been omitted.

It is important to point out that, in all the variables whose the type depends on the parameter d, δ_l cannot be assigned to d because of the constructor definitions (table 2.3); in fact, this is necessary in order to have (well-typed) process in our specified environment. The δ_l parameter acts only on the bound output transitions, where $\longmapsto\!\!\!\!\twoheadrightarrow$ relates a process and an agent abstraction of the form $\lambda l : \mathcal{N}_l{}^{(z)}.a$.

The first variable typed by the rule IN regards the input process transition; its type is a family that, taken a suitable agent abstraction p (depending on d and d') prefixed by an input operator, returns p. The OUT type is similar but in this case we have a free transition regarding a (suitable) concretion. The COM^l variable relates an input and an output processes that communicate an a channel m of the appropriate type. We can observe that the inter-process communication corresponds to the application of a λ-abstraction to a term. The bound output is obtained via the OPEN and CLOSE variable. The type of OPEN is a family of types that, give two abstractions $p, p' : \prod_{p,p':\mathcal{N}_l^{(z)}} \longrightarrow \mathcal{P}$ such that, for all the l' name lists of length z, pl' performs a free output along the channel m, becoming p', relates the closure of p (via ν) to the concretion associated to p'. The CLOSE^l type permits a communication with scope extrusion; note that t is an abstraction of a term that may contain some of the restricted names. The variable $\mathrm{PAR}^l_{1\div 4}$ manage the transition of the components of the parallel composition of processes. It is necessary to explicitly specify the rule for input, bound or free output and silent action in order to transform process object into agents. To the LTS rules relative to the restriction, RES1 $\div$ 3, are associated three variables, RES^1_f and RES^2_f for the free transitions derivable from processes with (top-level) restricted names and the rule RES_b for the input transition of a process with (top-level) restricted names. We omit the complementary rule for the (bound) output transition because we simply use the open rule for all output under the restriction operator. The type assigned to the rules BANG, COND and DECR are quit natural, we only point out that for any rule we need two version: one for the free transitions and the other one for the bound transitions.

The usual advantages of using a LF to express a formal system are that no α-conversion is needed, side-conditions are avoided, all binder are translated using a suitable λ-abstraction and name (term) substitution is not required because of the use of β-reduction [Pfe96].

Let Σ denote the set of variables declarations introduced in this section; Σ is enriched with two sets $\{m_i : \mathcal{N}_n\}_{i\in N}$ and $\{\tilde{m}_i : \mathcal{N}_e\}_{i\in N}$. In this manner we have a countable set of names typed by $\mathcal{N}_n$ and a disjoint countable set of names whose type is $\mathcal{N}_l{}^{(z)}$. Let $\Gamma = \Sigma \cup \{m_i : \mathcal{N}_n\}_{i\in N} \cup \{\tilde{m}_i : \mathcal{N}_e\}_{i\in N}$.

Chapter 3

Adequacy

In this chapter we state an *Adequacy* result, i.e. a result that proves a correspondence between the LF judgments derivable in the environment Γ and the same concepts phrased in terms of the LTS we have provided in section 1.2.2. What we want to emphasize here is the*judgments-as-types* principle adopted in the Edimburg Logical Framework: a judgment is represented with a particular type and proofs are represented as terms whose type is the representation of the judgment they prove.

Proposition 3.1 *Let p and q two* `spi`*-processes.*

a) if $\hat{m} = \delta_n$ and $\breve{m} = \hat{x}$ or $\hat{m} = \hat{x} = \delta_e$) then $p \xrightarrow{m} \lambda x.q$ iff $[\![p]\!] \underset{\hat{n},\,\hat{x}}{\overset{m}{\longmapsto\!\!\!\!\rightarrow}} \lambda x : \mathcal{T}(\hat{n}, \hat{x}).\mathrm{proc}([\![q]\!][x/\mathrm{Tch}(\hat{x}, \breve{x}, x)])$ is inhabited;

b) if $\hat{m} = \widehat{M}$ then $p \xrightarrow{\overline{m}} \langle M\rangle q$ iff $[\![p]\!] \overset{\overline{m}}{\longmapsto} \mathrm{conc}(\hat{m}, \widehat{M}, [\![M]\!], [\![q]\!])$ is inhabited;

c) $p \xrightarrow{\overline{m}} (\nu\, n_1) \dots (\nu\, n_k)\langle M\rangle q$ iff $[\![p]\!] \underset{\delta_l,\,\delta_l}{\overset{\overline{m}}{\longmapsto\!\!\!\!\rightarrow}} \lambda l : \mathcal{N}_l{}^{(k)}.\mathrm{conc}(\delta_l, \delta_l, [\![M]\!], [\![q]\!])\sigma$ is inhabited, where, assuming $l_1 = l$ and $l_{i+1} = \mathrm{tl}^{k-i+1}(l_i)$ for $i = 1, \dots, k-1$ we set

$$\sigma = [\mathrm{hd}_{\hat{n_i}}^{k-i+1}(\mathrm{tl}^{k-i+1}(l_i))/\mathrm{Tch}(\hat{n_i}, \breve{n_i}, n_i)]_{i=1,\dots,k}$$

d) if $p \xrightarrow{\tau} q$ iff $[\![p]\!] \overset{\tau}{\longmapsto} [\![q]\!]$ is inhabited.

Proof (sketch): The adequacy proof is given by induction on the on the length of the derivation $p \xrightarrow{\alpha} a$. Here we consider only some cases for each action α.

First consider the case a). The base case is the application of the IN rule, then $p \equiv m(x).q$. Let assume that $\hat{m} = \delta_n$ and $\breve{m} = \hat{x}$. In the sequel, we set $\rho = [x/\mathrm{tch}(\hat{x}, \breve{x}, x)]$. We have that $[\![p]\!] = \mathrm{in}(\hat{m}, \hat{x}, \lambda x : \mathcal{T}(\hat{m}, \hat{x}).[\![q]\!]\rho)$ Thanks to the fact that $m : \mathcal{N}(\hat{m})$, the object $\mathrm{IN}(\hat{m}, \hat{x}, m, \lambda x : \mathcal{T}(\hat{m}, \hat{x}).\mathrm{proc}([\![q]\!]\rho))$ has type $p \underset{\hat{m},\,\hat{x}}{\overset{m}{\longmapsto\!\!\!\!\rightarrow}} \lambda x : \mathcal{T}(\hat{m}, \hat{x}).\mathrm{proc}([\![q]\!]\rho)$ and this complete the proof in this case.

An input transition may be obtained in $n+1$-step by a derivation whose last rule is one among RES_1, PAR_3^l, BANG, MATCH or DECR. With respect to our proposition, the non-trivial case are the application of RES_1 and PAR_3^l.

RES$_1$: In this case we have $p \equiv (\nu\, n_1)\dots(\nu\, n_k)p_1$, $p_1 \xrightarrow{m} \lambda x.r$ and $m \notin \{n_1,\dots,n_k\}$. Observe that the hypothesis on m implies that, for any substitution μ whose support is a subset of $\{n_1,\dots,n_k\}$, we have $p_1\mu \xrightarrow{m} \lambda x.r\mu$. By inductive hypothesis, the exist a term object ι whose type is $[\![p_1]\!] \underset{\hat{m},\,\hat{x}}{\overset{m}{\longmapsto\!\!\!\!\rightarrow}} \lambda x : \mathcal{T}(\hat{m},\hat{x}).\mathrm{proc}([\![r]\!]\rho)$.

By the definition of the encoding, $[\![p]\!] = \nu\,(\lambda l : \mathcal{N}_l{}^{(k)}.\mathrm{proc}([\![p_1]\!]\sigma))$ where σ is defined as in table 2.4 and this is equivalent to say that

$$\mathrm{RES}_b(\hat{m},\hat{x},k,\lambda l : \mathcal{N}_l{}^{(k)}.\mathrm{proc}([\![p_1]\!]\sigma), \lambda l : \mathcal{N}_l{}^{(k)}.\lambda x : \mathcal{T}(\hat{m},\hat{x}).\mathrm{proc}([\![r]\!]\rho),\iota)$$

has type

$$[\![p]\!] \underset{\hat{m},\,\hat{x}}{\overset{m}{\longmapsto\!\!\!\!\rightarrow}} \lambda x : \mathcal{T}(\hat{m},\hat{x}).\mathrm{proc}(\nu\,(\lambda l : \mathcal{N}_l{}^{(k)}.\lambda x : \mathcal{T}(\hat{m},\hat{x}).\mathrm{proc}([\![r]\!]\rho)))$$

in fact, it is sufficient to note that $[\![(\nu\, n_1)\dots(\nu\, n_k)r]\!]$ is exactly the term under the proc constructor in the previous abstraction.

PAR$^l_3$: p has the form $p_1 \mid p_2$ and by hypothesis, $p_1 \xrightarrow{m} \lambda x.(r \mid p_2)$ and $x \notin fn(p_2)$. By inductive hypothesis, there exist a term ι typed by $p_1 \underset{\hat{m},\,\hat{x}}{\overset{m}{\longmapsto\!\!\!\!\rightarrow}} \lambda x : \mathcal{T}(\hat{m},\hat{x}).\mathrm{proc}([\![r]\!]\rho)$. The last statement holds if, and only if, the term

$$\mathrm{PAR}^l_3(\hat{m},\hat{x},[\![p_1]\!],[\![p_2]\!],\lambda x : \mathcal{T}(\hat{m},\hat{x}).[\![r]\!]\rho, m,\iota)$$

is typable by $[\![p]\!] \underset{\hat{m},\,\hat{x}}{\overset{m}{\longmapsto\!\!\!\!\rightarrow}} \lambda x : \mathcal{T}(\hat{m},\hat{x}).\mathrm{proc}([\![r]\!]\rho \mid p_2)$

The other cases follows trivially by induction.

Now we consider the b) part of the proposition. Let $\hat{m} = \widehat{M}$ the base case is given by the application of the OUT rule and, therefore, we have $p \equiv \overline{m}\langle M\rangle.q$ and $p \xrightarrow{\overline{m}} \langle M\rangle q$. We can observe that the term $\mathrm{OUT}(\hat{m},\hat{M},m,[\![M]\!],[\![q]\!])$ has type $\mathrm{out}(\hat{m},\hat{M},m,[\![M]\!],[\![q]\!]) \xrightarrow{\overline{m}}\!\!\!\!\!\!\longmapsto \mathrm{conc}(\hat{m},\hat{M},[\![M]\!],[\![q]\!])$ in fact, it is easy to see that $[\![p]\!] = \mathrm{out}(\hat{m},\hat{M},m,[\![M]\!],[\![q]\!]$.

The derivation of an output transition may also be performed by the parallel component of a process. In this case $p \equiv p_1\ mmparq$ and $p_1 \xrightarrow{\overline{m}} \langle M\rangle r$ and, using the rule PAR$^l_1$, we have $p \xrightarrow{\overline{m}} \langle M\rangle(r \mid q)$. Then it is possible to type a term ι with $[\![p_1]\!] \mid [\![q]\!] \overset{\overline{m}}{\longmapsto} \mathrm{conc}(\hat{m},\widehat{M},[\![M]\!],[\![r]\!] \mid [\![q]\!])$ and, observing that $[\![p]\!] = [\![p_1]\!] \mid [\![q]\!]$ and that $[\![r \mid q]\!] = [\![r]\!] \mid [\![q]\!]$ we obtain the thesis.

An interesting case to consider is the application of the RES$_3$ rule. Assuming $m \notin \{n_1,\dots,n_k\}$, we have

$$p \equiv (\nu\, n_1)\dots(\nu\, n_k)q \qquad q \xrightarrow{\overline{m}} \langle M\rangle r$$

and , by inductive hypothesis, there exist a term ι whose type is $q \overset{\overline{m}}{\longmapsto} \mathrm{conc}(\hat{m},\widehat{M},[\![M]\!],[\![r]\!])$.

For a suitable substitution σ (see table 2.4), we have

$$[\![p]\!] = \nu\,(\lambda l : \mathcal{N}_l{}^{(k)}.\mathrm{proc}([\![q]\!]\sigma)) \tag{3.1}$$
$$[\![(\nu\, n_1)\dots(\nu\, n_k)r]\!] = \nu\,(\lambda l : \mathcal{N}_l{}^{(k)}.[\![r]\!]\sigma) \tag{3.2}$$

then $\mathrm{RES}^2_f(\hat{m},\widehat{M},k,\lambda l : \mathcal{N}_l{}^{(k)}.\mathrm{proc}([\![q]\!]\sigma),\lambda l : \mathcal{N}_l{}^{(k)}.[\![r]\!]\sigma, m,[\![M]\!],\iota)$, by 3.1 and 3.2, has type $p \overset{\overline{m}}{\longmapsto} \mathrm{conc}(\hat{m},\widehat{M},[\![M]\!],\nu\,(\lambda l : \mathcal{N}_l{}^{(k)}.[\![r]\!]\sigma))$; q.e.d.

As for the input transitions, we omit the other cases that follow trivially by induction.

Now we consider the bound output transitions, i.e. the case c). A bound output transition may be derived by applying the rules OPEN, PAR_2^l, BANG, MATCH and DECR.

In the first case, $p \equiv (\nu\, n_1)\ldots(\nu\, n_k)q$ and, by hypothesis, $p \xrightarrow{\overline{m}} \langle M \rangle r$ and $m \notin \{n_1, \ldots, n_k\}$. Then, for all substitution σ' with support in $\{n_1, \ldots, n_k\}$, we can derive $p\sigma' \xrightarrow{\overline{m}} \langle M\sigma' \rangle r\sigma'$; furthermore, by the part b) of this proposition, there exist a term ι typed by $p\sigma' \overset{\overline{m}}{\longmapsto} \mathrm{conc}(\hat{m}, \widehat{M}, [\![M\sigma']\!], [\![r\sigma']\!])$. This means that the term $\mathrm{OPEN}(\hat{m}, \widehat{M}, k, \lambda l : \mathcal{N}_l{}^{(k)}.[\![p]\!]\sigma, \lambda l : \mathcal{N}_l{}^{(k)}.[\![r]\!]\sigma, m, \lambda l : \mathcal{N}_l{}^{(k)}.[\![M]\!]\sigma, \lambda l : \mathcal{N}_l{}^{(k)}.[\![\iota]\!]\sigma,)$ is typed by $\nu\,(\lambda l : \mathcal{N}_l{}^{(k)}.[\![p]\!]\sigma) \underset{\delta_l\,,\,\delta_l}{\overset{\overline{m}}{\longmapsto\!\!\!\!\rightarrow}} \lambda l : \mathcal{N}_l{}^{(k)}.\mathrm{conc}(\delta_l, \delta_l, [\![M]\!]\sigma, [\![q]\!]\sigma)$ and this completes the proof.

If the PAR_2^l rule is used, we proceed in a manner similar to the previous case: the only difference is that we use the inductive hypothesis in the place where the b) part of this proposition.

The other cases are omitted.

The part d) of the proposition follows easily from the parts a), b) and c) and by induction.

$\diamond$

Chapter 4

Conclusions

In general, a logical framework is a meta-language useful for the specification of deductive systems and, in this respect, we can encode a language (viewed as a logical system) in a particular LF. As usual, we have specified the *source* language by defining its syntax via the grammar of table 1.1 and an operational semantics that is a formed by a set of "axioms" and "inference rules".

A first advantage that we can obtain by mapping a language in a LF is the uniformity in the treatment of bind operators and α-renaming. All the object variables are represented as variables in the LF and all binder operators reduce to λ-abstraction. In this way, a bound variable of the language is bound with corresponding scope in the meta-language: this is the main idea of *higher-order abstract syntax* [HHP93]. Two expressions differing only for the name assigned to their bound variables are identified through an α-conversion in the higher-order abstract syntax. Furthermore, in the particular context of concurrent languages, we can observe that the process communications may be modeled by the β-reduction [HLMP98].

A practical benefit of the logical frameworks is the possibility of define tools (such as proof editors or proof checkers) that are independent from the formal systems that one wants to represent in the LF. In fact, the judgment-as-type principle permits to reduce the proof checking to the type checking of the proof-representing terms.

A limitation of the ELF lays in that it does not permit the *subsorting*; for instance, often the syntactic categories of a grammar are not completely disjoint, for example in the grammar we gave for the `spi-calculus`, a process is also an agent. This aspect of the languages is not well managed in the Edimburg Logical Framework, in fact, we have be compelled to explicitly "cast" the type of processes and agents when necessary. In general, a logical framework is a meta-language useful for the specification of deductive systems and, in this respect, we can encode a language (viewed as a logical system) in a particular LF. As usual, we have specified the *source* language by defining its syntax via the grammar of table 1.1 and an operational semantics that is a formed by a set of "axioms" and "inference rules".

A first advantage that we can obtain by mapping a language in a LF is the uniformity in the treatment of bind operators and α-renaming. All the object variables are represented as variables in the LF and all binder operators reduce to λ-abstraction. In this way, a bound variable of the language is bound with

corresponding scope in the meta-language: this is the main idea of *higher-order abstract syntax* [HHP93]. Two expressions differing only for the name assigned to their bound variables are identified through an α-conversion in the higher-order abstract syntax. Furthermore, in the particular context of concurrent languages, we can observe that the process communications may be modeled by the β-reduction [HLMP98].

A practical benefit of the logical frameworks is the possibility of define tools (such as proof editors or proof checkers) that are independent from the formal systems that one wants to represent in the LF. In fact, the judgment-as-type principle permits to reduce the proof checking to the type checking of the proof-representing terms.

A limitation of the ELF lays in that it does not permit the *subsorting*; for instance, often the syntactic categories of a grammar are not completely disjoint, for example in the grammar we gave for the `spi-calculus`, a process is also an agent. This aspect of the languages is not well managed in the Edimburg Logical Framework, in fact, we have be compelled to explicitly "cast" the type of processes and agents when necessary.

Bibliography

[AG98] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 1998. To Appear. Available as Compaq SRC Research Report 149 (1998).

[BB92] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[DH84] Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[HLMP98] Furio Honsell, Marina Lenisa, Ugo Montanari, and Marco Pistore. Final semantics for the pi-calculus. In D. Gries and W-P. de Roever, editors, *PROCOMET'98*, volume 40(1), pages 226–243. Chapman & Hall, 1998.

[Par80] D. Park. Concurrency and automata on infinite sequences. *LNCS*, 104, 1980. Springer Verlag.

[Pfe96] Frank Pfenning. The practice of logical frameworks. In Hélène Kirchner, editor, *Proceedings of the Colloquium on Trees in Algebra and Programming*, pages 119–134. LNCS 1059 Springer-Verlag, 1996.

[Plo81] Gordon G. Plotkin. A structural approach to operational semantics. Technical report, Computer Science Department, Aarhus University, 1981.