AALBORG UNIVERSITET

Department of Computer Science

Network failure detection using type systems in DπF

Software System Engineering Master Thesis

Supervised by Hans Hüttel
Spring Semester, 2009

AALBORG UNIVERSITET

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Oest, Denmark
Telephone +45 9940 9940
Fax +45 9940 9798
http://www.cs.aau.dk

Title:

'Network failure detection using type systems in DπF'

Project period:

4^{th} Semester (SSE4)
1^{st} February 2009 - 10^{th} June 2009

Project group:

d619a

Participants:

Jorge Martín García
José Luis López Sánchez
Raúl Martínez Pérez

Supervisor:

Hans Hüttel

Number of copies: 5

Number of pages: 106

Number of appendixes: 3

Front page: Picture is from the website of the UVic Grid Research.

Finished: 10^{th} of June 2009

Abstract:

The main objective of our project is to prevent unwanted deaths of the locations that conform a network. In order to do that, we build Type Systems to control if a network defined with DπF has any premature killing (kill-safe) or unwanted murder (migration-safe). We use the typing rules of the type systems to analyze the DπF code of a network to be able to say if it will have the expected behaviour.

Firstly the main characteristics of DπF are explained. Later, the different options of kill-safety are analyzed and an unimportance predicate is presented as a generalization of the previous kill-safety proposals. A type system for kill-safety is also presented. Afterwards, an improvement to the previous system is introduced in order to control the murders from a location to another one. Finally the problem of indirect murders is analyzed and a new type system is developed as its solution. This report ends with the main project conclusions and the proposal of some further work.

Preface

This project report was written by 3 students of the 4^{th} Semester of the master program Software System Engineering (SSE4) at the Department of Computer Science at Aalborg University. It was conducted during the period from 1^{st} February to 10^{th} June 2009.

Literature references are marked with numbers in square brackets. The appendices containing supplementary materials are assigned with letters and arranged in alphabetical order. Tables are numbered in format *Table x.y*, where *x* is the chapter number and *y* is the number of the item. Finally, definitions are numbered in order of appearance.

The group expresses special thanks to its supervisor Hans Hüttel for his support and ideas provided during the project work.

The report is developed by:

Jorge Martín García

José Luis López Sánchez

Raúl Martínez Pérez

Contents

Chapter 1

Introduction

1.1 Problem context

In a distributed network with several locations, various problematic situations can take place. Detecting some of these problems is the main goal of our report. If we do that, we can rely on the expected behaviour of the network. The chosen networks to carry out our research are grid networks, due to the current importance of their work and their possibilities in the future.

A grid network is a network where the processes are distributed among several nodes that could be dedicated or not. In the case of *dedicated grids*, the nodes are only meant to perform highly distributed computations. For example, Aalborg University is involved in the NORDUGRID network [1] whose aim is to deliver a robust, scalable, portable and fully featured solution for a global computational and data Grid system. On the other hand, in the case of *ad hoc grids*, the nodes belong to people who donate idle computation time of their computers when they are turned on, to big processing projects which use a lot of small computers instead of a supercomputer to perform a large-source computation. For example, the grid projects controlled by NASA focussed on discovering new stars or the World Community grid projects for humanitarian intentions like finding a cure for cancer or the project called Discovering Dengue Drugs for Hepatitis C [3].

This kind of computer network can experience some problems because any node, i.e. any location, could be inoperative or disconnected unexpectedly. We cannot prevent that a user turns off his computer while it is performing an important computation. But detecting this situation

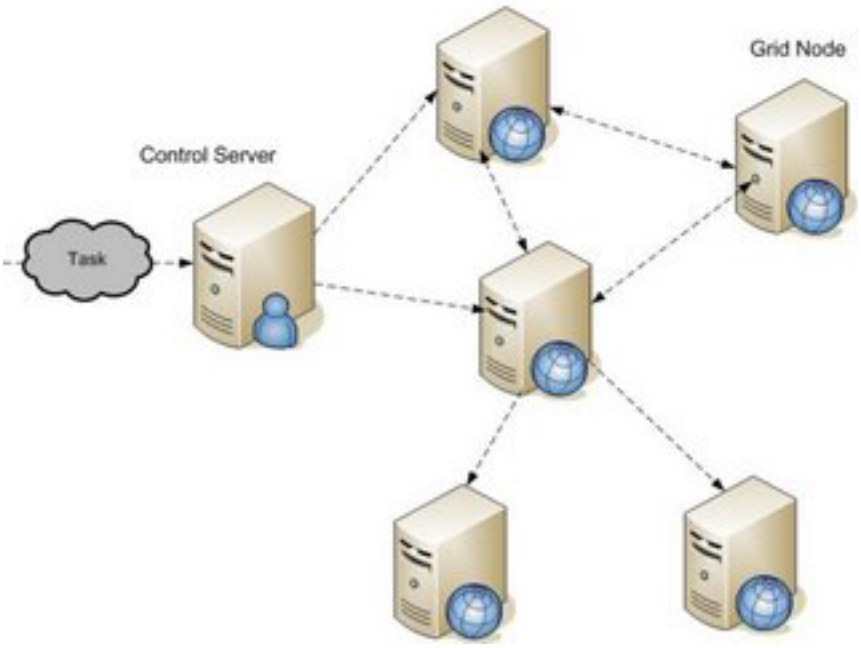

Figure 1.1: *grid structure*

and trying to avoid it would be desirable, making the network tolerant to these failures.

DπF could be used to represent this kind of problem. It is a process calculus that can help us with verification tasks in a network. DπF is an extension of Dπ [7], which is in turn based on the π-calculus [9] and [11]. DπF can be used to give a formal description of the behaviour of a distributed computation. Also, it can be used to provide the basis of a formal method for analyzing distributed systems. Once we have defined the network, we can study its behaviour using the operational semantics of DπF.

In DπF a system is composed of different locations and we can transfer processes between them, kill locations and break the different communication channels. If we kill a location, all the results from the tasks located on it will be lost. Although DπF gives us the control of our network, we could obtain an unexpected behaviour. For example, starvation cases can occur when a node waits for a result of a process from a killed node and, in other cases, a node which sends a message to a killed node. It happens when a killing order is run in some location, before another location receives the results of the processes located on it. Having control of the network in all situations is very important.

We wish to take care of these situations, think about the problem and propose several

solutions. We think that DπF could help in the purpose of designing and verifying a network because it is a simple way to represent these kinds of situations and it gives us a very solid base. However, we have to avoid to kill locations in a network before obtaining its useful results, and this is one of the main goals of our work.

Furthermore, one possible situation of DπF systems can be studied to continue with our main purpose of analyzing grid networks: the killing migrations. If this problem is analyzed, a problematic behaviour which has not been studied can be observed: the possibility of migrating a kill order from a location to another one which should not be killed since it is important. It is an unwanted situation which can endanger the correct behaviour of the grid, since an unimportant location can kill a location which is managing the processes in the different personal computers and waiting for the results of them.

1.2 Methodology and related work

The chosen methodology for the study was to construct a type system. As Hennessy states in his work [6], types provide a simple method to avoid runtime errors during the execution of high-level programs. Types add semantic information to the code to indicate the intended use of the resources, e.g. variables. Type checking detects statically if a source code could have an unwanted running. To perform this checking, the type annotated code is syntactically analyzed in order to ensure the correct use of these resources in the network. This way, we can conclude the correction of the network depending on the properties defined in the type system. An example of the application of this methodology is presented in [8] where it is developed a type system to control when it is allowed a migration using authentication. Another example of type system is shown in [4], used to guarantee that any migration message will find an appropriate receiver at its destination locality.

Another related work, as proposed in [5], uses a different methodology to detect the correct working of networks: behavioural equivalence. In this methodology, DπF specification of a network is compared with another one whose safety or whose problematic situation are well known, trying to find if the two specifications are behavioural equivalent. This way, the running of both of them will have the same results. However, it is not easy to have a method that shows always the equivalence between the behaviour of two systems and it is easier to work statically with a type system.

At this point, the advantages and disadvantages of type systems versus behavioural equivalences should be analyzed. Having our DπF network implementation N_{imp} and the correctness specification in the form of a network which is known to be correct N_{spec}, the correctness checking of behavioural equivalences is the same as checking if $N_{imp} \sim N_{spec}$. This approach is *sound*, since if we get a yes the network N_{imp} is correct, and *complete* in the sense that if we get a no, then N_{imp} is incorrect.

However, if our process calculus is Turing-complete and $\sim$ is an 'interesting' notion of equivalence (such as bisimulation), then the problem

$$\text{Given } N_1, N_2, \text{ is } N_1 \sim N_2?$$

is undecidable. Therefore, there can be no general algorithm for checking correctness of implementations.

For the type system based approach, we design a set of typing rules which prevents some kind of unwanted situation defined using a safety property. This approach continues being *sound*, since if N_{imp} is well-typed, then N_{imp} is always safe. Nevertheless, it is not *complete*: if the network N_{imp} is not well-typed, then N_{imp} may still be safe because of the *slack* of the type system. Even so, we chose type-base approach, since it is a good implementable option, and it does not have any undecidability problem.

We refer to *Nomadic Pict* [12] to see an implementation of Dπ (not DπF) using types. Here, type checking is used, but a partial type inference algorithm is also included. In our report we will work only with type checking, since every type has to be defined.

1.3 Overview of report

The main objective of our project is to build the Type Systems that control if a network defined with DπF has kill-safe and migration-safe behaviour in special situations. We use the proposed rules of the type systems to analyze the DπF code of the network in order to say if a network will work correctly in a specific situation with the commented problems (premature killing and unwanted murders) that will be explained later.

In this report we consider different properties to define when a network is kill-safe. These properties correspond to different conditions under which a location can safely be killed. They

vary between the number of concurrent components and different priority levels considered on the locations and the processes. In the case of migration-safety, we will analyze the concept of grid hierarchy to represent the relations between the nodes. In this way we can control that a node does not kill another one with a higher position in the hierarchy.

This report is divided in different chapters. In Chapter 2 the main characteristics of DπF are explained, using the definition in [5]. In Chapter 3 the different options of kill-safety are analyzed and an important predicate is presented as generalization of the previous kill-safety proposals. A type system for kill-safety is also presented. In Chapter 4 we introduce an improvement to the previous system to control the murder process constructs. In chapter 5 some derived problems, the indirect murders, are analyzed and a new type system is developed as their solution. This report ends with the main project conclusions and the proposal of some further work.

Chapter 2

DπF calculus

In this chapter we present some basic notions about DπF modeling language. We show its syntax and its semantic. Finally, we explain an usage example.

2.1 Syntax

Firstly, the syntax of Dπ introduced by Hennessy and Riely [7] is presented.

Names	$::= u, v, x, s, t, l, n$
Processes	$P, Q ::= u!\langle v\rangle.P \mid u?(x).P \mid *u?(x).P \mid \texttt{if}\ [s = t]\ \texttt{then}\ P\ \texttt{else}\ Q \mid 0 \mid (P \mid Q) \mid (\nu n)P \mid \texttt{go}\ u.P$
Systems	$M, N, O ::= \emptyset \mid (\nu n)N \mid l[P] \mid (N \mid M)$

Table 2.1: *Syntax of Dπ*

The main Dπ feature, and π-calculus feature as well, is that it uses *names*. Different locations, variables or channels can be specified, but all of them are names.

There are two kind of names: *free* and *bound* names (*fn* and *bn*). Referencing Parrow's explanation [10], *bn(P)* is a set of names that are bound occurrences in *P*. They are local variables

or inputs. On the other hand, *fn(P)* is a set of names that are not bound occurrences in P. Let us define these sets. *fn* and *bn* are defined recursively in the structure of P and N:

Definition 1 ***Bound names of networks***

$$\begin{aligned} bn(0) &= \emptyset \\ bn((\nu n)N) &= \{n\} \cup bn(N) \\ bn(l[P]) &= bn(P) \\ bn((N \mid M)) &= bn(N) \cup bn(M) \end{aligned}$$

The *bound names* of the processes will be:

Definition 2 ***Bound names of processes***

$$\begin{aligned} bn(0) &= \emptyset \\ bn(x?(y).P) &= \{y\} \cup bn(P) \\ bn(x!\langle y\rangle.P) &= bn(P) \\ bn(*P) &= bn(P) \\ bn(\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q) &= bn(P) \cup bn(Q) \\ bn((P \mid Q)) &= bn(P) \cup bn(Q) \\ bn((\nu n)P) &= \{n\} \cup bn(P) \\ bn(\texttt{go } u.P) &= bn(P) \end{aligned}$$

On the other hand, the *free names* will be:

Definition 3 ***Free names of networks***

$$\begin{aligned} fn(0) &= \emptyset \\ fn((\nu n)N) &= fn(N) \\ fn(l[P]) &= fn(P) \\ fn((N \mid M)) &= fn(N) \cup fn(M) \end{aligned}$$

Definition 4 ***Free names of processes***

$$
\begin{aligned}
fn(0) &= \emptyset \\
fn(x?(y).P) &= \{x\} \cup fn(P) \\
fn(x!\langle y\rangle.P) &= \{x, y\} \cup fn(P) \\
fn(*P) &= fn(P) \\
fn(\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q) &= \{v, u\} \cup fn(P) \cup fn(Q) \\
fn((P \mid Q)) &= fn(P) \cup fn(Q) \\
fn((\nu n)P) &= fn(P) \\
fn(\texttt{go } u.P) &= \{u\} \cup fn(P)
\end{aligned}
$$

The main objective of this syntax is to represent a system with different locations, with scoped names (νn) and different processes located on the different locations. It also represents the different ways of communication between processes (using inputs -?- and outputs -!-), parallelism(|), replication (*), logical comparison tests (`if` $[s = t]$ `then` P `else` Q) and migration of processes from one location to another one (*go*).

It is important to say that the syntax just has replicated inputs, since it is enough. As it is mentioned in [10], replicated inputs is all that it is needed because it allows us to code up recursion.

Hennessy and Francalanza [5] proposed some syntactic constructs to be added to DπF and they are used to kill a specific location (*kill*), to break communication channels (*break*) and to do an action depending on the accessibility of a location (*ping*).

Processes	$P', Q' ::= \texttt{kill} \mid \texttt{break } u \mid \texttt{ping } u.P' \texttt{ else } Q'$

Table 2.2: *Added syntactic constructs in DπF*

2.2 Semantics

Using this definition of the syntax of DπF , we can specify a system which represents a particular situation. Moreover, DπF needs some operational semantics to show how a system evolves.

These rules, proposed by Hennessy and Francalanza [5], govern the reductions used to study the evolution of a network with DπF and they are given in the following tables.

Firstly, Table 2.3 gives the standard rules for (local) communication, and the management of replication, matching and parallelism. But before, we are going to introduce the symbol Δ which is a representation of the state of the network. Intuitively, this must records the set of existing locations, whether they are alive or dead, and any live links between them.

Definition 5 ***(Network Representation.)*** *We first introduce some notation to represent the links in a network [5]. A binary relation $\mathcal{L}$ over locations is called a linkset if it is:*

- symmetric, that is, $< l, k > \epsilon \mathcal{L}$ implies $< k, l >$ is also in $\mathcal{L}$

- reflexive, that is, $< l, k > \epsilon \mathcal{L}$ implies $< l, l >$ and $< k, k >$ are also in $\mathcal{L}$.

A network representation, Δ is any triple $< \mathcal{N}, \mathcal{A}, \mathcal{L} >$ where:

- $\mathcal{N}$ is a set of names, divided into $\mathbf{loc}(\mathcal{N})$ (location names) and $\mathbf{chan}(\mathcal{N})$(channel names)

- $\mathcal{A} \subseteq \mathbf{loc}(\mathcal{N})$ represents the set of live locations

- $\mathcal{L} \subseteq \mathbf{loc}(\mathcal{N}) \times \mathbf{loc}(\mathcal{N})$ is a linkset representing the set of live connections between locations.

In the sequel, we use the abbreviation $l \leftrightarrow k$ in linksets to denote the pairs $< l, l >, < k, k >, < l, k >, < k, l >$; we also denote the components of Δ as $\Delta_{\mathcal{N}}$, $\Delta_{\mathcal{A}}$ and $\Delta_{\mathcal{L}}$.

Assuming $\Delta \vdash l$: **alive**

(r-comm) $$\frac{}{\Delta \triangleright l[\![a!\langle V\rangle.P]\!] \mid l[\![a?(X).Q]\!] \rightarrow \Delta \triangleright l[\![P]\!] \mid l[\![Q\{V/_X\}]\!]}$$

(r-rep) $$\frac{}{\Delta \triangleright l[\![*a?(X).P]\!] \rightarrow \Delta \triangleright l[\![a?(X).(P \mid *a?(X).P)]\!]}$$

(r-fork) $$\frac{}{\Delta \triangleright l[\![P|Q]\!] \rightarrow \Delta \triangleright l[P] \mid l[\![Q]\!]}$$

(r-eq) $$\frac{}{\Delta \triangleright l[\![\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q]\!] \rightarrow \Delta \triangleright l[\![P]\!]}$$

(r-neq) $$\frac{}{\Delta \triangleright l[\![\texttt{if } [u = v] \texttt{ then } P \texttt{ else } Q]\!] \rightarrow \Delta \triangleright l[\![Q]\!]}\, u \neq v$$

Table 2.3: *Local Reduction Rules for DπF*

In Table 2.4 appear the rules that belong to the DπF extension that is being introduced. There are some rules such as (r-go) and (r-ngo) focused on migration of code which depend on the accessibility of the destination. In the same way, there are two rules (r-ping and r-nping) that can only determine the accessibility of a location. The rules (r-kill), (r-brk) make changes to the current network killing locations or breaking links and the last rules, (r-newc) and (r-newl), regulate the generation of new local names.

Assuming $\Delta \vdash l$: **alive**

(r-go)	$\dfrac{}{\Delta \triangleright l[\![\texttt{go}\ k.P]\!] \to \Delta \triangleright k[\![P]\!]}\Delta \vdash k \leftarrow l$
(r-ngo)	$\dfrac{}{\Delta \triangleright l[\![\texttt{go}\ k.P]\!] \to \Delta \triangleright k[\![0]\!]}\Delta \nvdash k \leftarrow l$
(r-ping)	$\dfrac{}{\Delta \triangleright l[\![\texttt{ping}\ k.P\ \texttt{else}\ Q]\!] \to \Delta \triangleright l[\![P]\!]}\Delta \vdash k \leftarrow l$
(r-nping)	$\dfrac{}{\Delta \triangleright l[\![\texttt{ping}\ k.P\ \texttt{else}\ Q]\!] \to \Delta \triangleright l[\![Q]\!]}\Delta \nvdash k \leftarrow l$
(r-kill)	$\dfrac{}{\Delta \triangleright l[\![\texttt{kill}]\!] \to (\Delta - l) \triangleright l[\![0]\!]}$
(r-brk)	$\dfrac{}{\Delta \triangleright l[\![\texttt{break}\ k]\!] \to (\Delta - l \leftrightarrow k) \triangleright l[\![0]\!]}\Delta \vdash l \leftrightarrow k$
(r-new)	$\dfrac{}{\Delta \triangleright l[\![(\nu c)P]\!] \to \Delta \triangleright (\nu c)l[\![P]\!]}$

Table 2.4: *Network Reduction Rules for DπF*

Finally, in Table 2.5 there is an adaptation of the standard contextual rules, which allows the basic reductions to occur in evaluation contexts, and Table 2.6 shows the DπF structural rules.

(r-str)	$\dfrac{\Delta \triangleright N' \equiv \Delta \triangleright N \quad \Delta \triangleright N \to \Delta' \triangleright M \quad \Delta' \triangleright M \equiv \Delta' \triangleright M'}{\Delta \triangleright N' \to \Delta' \triangleright M'}$
(r-ctxt-rest)	$\dfrac{\Delta + n : U \triangleright N \to \Delta' + n : W \triangleright M}{\Delta \triangleright (\nu n : U)N \to \Delta' \triangleright (\nu n : W)M}$
(r-ctxt-par)	$\dfrac{\Delta \triangleright N \to \Delta' \triangleright N'}{\Delta \triangleright N \mid M \to \Delta' \triangleright N' \mid M}\Delta \vdash M$

Table 2.5: *Contextual reduction rules for DπF*

In the next section we present an example to show how to use DπF.

(s-comm)	$N \mid M \equiv M \mid N$	
(s-assoc)	$(N \mid M) \mid M' \equiv N \mid (M \mid M')$	
(s-unit)	$N \mid l[0] \equiv N$	
(s-extr)	$(\nu n : U)(N \mid M) \equiv N \mid (\nu n : U)M$	$n \notin \mathbf{fn}(\mathbf{N})$
(s-flip-1)	$(\nu n : U)(\nu m : W)N \equiv (\nu m : W)(\nu n : U)N$	$n \notin \mathbf{fn}(\mathbf{W})$
(s-flip-2)	$(\nu n : U)(\nu m : W)N \equiv (\nu m : W - n)(\nu n : U + m)N$	$n \in \mathbf{fn}(\mathbf{W})$
(s-inact)	$(\nu n : U)N \equiv N$	$n \notin \mathbf{fn}(\mathbf{N})$

Table 2.6: *Structural rules for DπF*

2.3 An example using DπF calculus

In this example, we specify a system with three different locations. In one location (l_3) there is a generic process P. In the location l_1, the process located ($u!\langle l_3\rangle.0$) is transferred to the location l_2. Here, the identifier l_3 is transferred through the channel u, and it is received in the parallel process ($u?(x).\texttt{go}\ x.\texttt{kill}$). Later, `kill` is transferred to the received identifier, in this case l_3. Then the location l_3 is killed using `kill`.

$$\Delta \triangleright l_1[\texttt{go}\ l_2.u!\langle l_3\rangle.0] \mid l_2[u?(x).\texttt{go}\ x.\texttt{kill}] \mid l_3[P] \tag{2.1}$$

$$\rightarrow \Delta \triangleright l_1[0] \mid l_2[u!\langle l_3\rangle.0 \mid u?(x).\texttt{go}\ x.\texttt{kill}] \mid l_3[P] \tag{2.2}$$

$$\rightarrow \Delta \triangleright l_1[0] \mid l_2[u!\langle l_3\rangle.0] \mid l_2[u?(x).\texttt{go}\ x.\texttt{kill}] \mid l_3[P] \tag{2.3}$$

$$\rightarrow \Delta \triangleright l_1[0] \mid l_2[0] \mid l_2[\texttt{go}\ x.\texttt{kill}\{l_3/x\}] \mid l_3[P] \tag{2.4}$$

Using the structural rule *s-unit*:

$$l_1[0] \mid l_2[0] \mid l_2[\texttt{go}\ x.\texttt{kill}\{l_3/x\}] \mid l_3[P] \equiv l_2[\texttt{go}\ x.\texttt{kill}\{l_3/x\}] \mid l_3[P] \tag{2.5}$$

$$\Delta \triangleright l_2[\texttt{go}\ x.\texttt{kill}\{l_3/x\}] \mid l_3[P] \tag{2.6}$$

$$\rightarrow \Delta \triangleright l_2[\texttt{go}\ l_3.\texttt{kill}] \mid l_3[P] \tag{2.7}$$

$$\rightarrow \Delta \triangleright l_2[0] \mid l_3[\texttt{kill} \mid P] \tag{2.8}$$

Now the reduction sequence will be described meticulously.

At the first step 2.1 it is necessary to take into consideration the contextual reduction rule (r-ctxt-par) to apply the (r-go) rule on location l_1. The consequences are that $u!\langle l_3\rangle.0$ will be migrated to l_2. l_3 does not change at this step.

In reduction 2.2, in order to apply the necessary rule (r-comm), we have to use the rule (r-fork). So, $l_2[u!\langle l_3\rangle.0 \mid u?(x).\texttt{go}\ x.\texttt{kill}]$ is now $l_2[u!\langle l_3\rangle.0] \mid l_2[u?(x).\texttt{go}\ x.\texttt{kill}]$. After that, we can use (r-comm) at l_2 with this result: $l_2[0] \mid l_2[\texttt{go}\ X.\texttt{kill}\{l_3/X\}]$.

Having two locations with empty processes we can apply the (s-unit) structural rule in the step 2.5 to eliminate them. The next step is to use again the (r-go) rule in the reduction 2.7.

Finally the last reduction 2.8 could be (r-kill) statment or executing P. In the first case, the state of the location l_3 is set to ”dead”. The (s-unit) structual rule is also applied in this step to eliminate l_2.

Chapter 3

Controlling premature `kill`

In this chapter some properties of kill-safety are analyzed, and a possible type system which controls some of these properties is defined.

3.1 Kill-safety properties

Different possible conditions of kill-safety can be considered, depending on the network properties which you decide to study. The kill-safety will vary depending on the chosen property.

3.1.1 One process location

One of the possible options could be to consider the possibility of killing a location where just one component process is being run, i.e. just a concurrent DπF component is located in the killed location. This way, the location that can be killed will only have the `kill` component.

Using this property we permit that only locations with a low number of processes can be killed. In this way, the locations with several concurrent components running on them must not be killed. If that happens, the network would be kill-unsafe.

Definition 6 *A network N is* kill-safe *if whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$ then if $l[P] \equiv l[\texttt{kill} \mid P']$ then $l[P'] \equiv l[0]$ where P' does not contain parallel components.*

3.1.2 Important channels

Different channels priority in a network could be considered. So we could define different network sections, and each section will have its own interest level (important or not important) depending on the used channels. We have to set a specific list of important channels to test if a network is kill-safe.

Definition 7 *Given a set S which contains names considered important, a process P is S-*important *if* $\mathcal{N}(P) \cap S \neq \emptyset$.

Definition 8 *Let S be a set of channel names considered important. A network N is* kill-safe *if whenever* $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ *then if* $N' \equiv l[P] \mid N''$ *then if* $l[P] \equiv l[\texttt{kill} \mid P']$ *then* $\mathcal{N}(P') \cap S = \emptyset$

3.1.3 Locations and processes priorities

Another interesting definition of kill-safe networks could depend on level priorities. These level priorities could be established on locations or processes. In this way, we would be considering a location as important with a specific importance level, or unimportant with a concrete unimportance value. Moreover, in these locations could be running important or normal processes.

Having:

$\mathcal{P}$ *is a totally ordered set of priority levels of locations with ordering* $\sqsubseteq_P$

$\mathcal{Q}$ *is a totally ordered set of priority levels of processes with ordering* $\sqsubseteq_Q$

where $\text{limit}_{\mathcal{P}}$ *is the border value between not-important location priority values and the priority values considered as important, and* $\text{limit}_{\mathcal{Q}}$ *is the border value between not-impotant priority values and important priority values of processes*

Definition 9 *Let* $l_{\mathcal{P}}$ *and* $P'_{\mathcal{Q}}$ *be the priority values of the location l and the process P' respectively, where* $l_{\mathcal{P}} \in \mathcal{P}$ *and* $P'_{\mathcal{Q}} \in \mathcal{Q}$*. A network N is* kill-safe *if whenever* $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ *then if* $N' \equiv l[P] \mid N''$ *then if* $l[P] \equiv l[\texttt{kill} \mid P']$ *then* $l_{\mathcal{P}} \sqsubseteq_P \text{limit}_{\mathcal{P}} \vee P'_{\mathcal{Q}} \sqsubseteq_Q \text{limit}_{\mathcal{Q}}$.

This way, a network will be kill-safe if a `kill` is placed in an unimportant location or, if it is important, the rest of the processes located on it are unimportant. The network designer has

to assign the explained priority levels and the limit values manually. If we analyze the previous definition, we can see how it works using two levels of priority: *low* or *high*. It could have been defined using just this two priority levels, but it was defined using different numeric values to be more generic. Consequently, the definition can vary setting a different number of *limit values* in order to represent our wanted priority levels.

3.1.4 Proposal: UnImp predicate

If the different explained kill-safety properties are analyzed, a general *unimportance* predicate can be detected. On the one hand, it can be represented using important and unimportant channels. On the other hand, it is defined as different priority levels.

Firstly, let us define the set of the truth values:

Definition 10 $\mathbb{T}$ *denotes the finite set of truth values.* $\mathbb{T} = \{true, false\}$.

A general definition of *Unimportance* can be defined in order to detect if the presence of a *kill* in a location converts the network kill-unsafe or not.

Definition 11 *An* unimportance *predicate is a map* $UnImp : \mathcal{A} \rightarrow \mathbb{T}$ *from alive location identifiers (*$\mathcal{A}$*) to a set of truth values* $\mathbb{T}$ *which specifies if a name is unimportant (*true*) or important (*false*). So, if UnImp(l)=true then l is unimportant.*

Definition 12 *Having an UnImp predicate, a network N is* kill-safe *if whenever* $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ *then if* $N' \equiv l[P] \mid N''$ *then if* $l[P] \equiv l[\texttt{kill} \mid P']$ *then* $UnImp(l)$

Now, we generalize the different kill-safety properties presented in this chapter using the new $UnImp$ predicate.

One process location using $UnImp$ predicate

The $UnImp$ predicate will be defined in the following way:

Definition 13 *The predicate* UnImp *of a location l will be $UnImp(l) = true$ whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$ then if $l[P] \equiv l[\texttt{kill} \mid P']$ then $l[P'] \equiv l[0]$ where $l[P']$ does not contain parallel components. Otherwise, $UnImp(l) = false$.*

This way, a location will be unimportant when it does not have more than one component, and this component is a `kill`.

Important channels using *UnImp* predicate

Definition 14 *Given a set S which contains channel names considered important, the predicate* UnImp *of a location l will be $UnImp(l) = true$ whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$ then if $l[P] \equiv l[\texttt{kill} \mid P']$ then $\mathcal{N}(P') \cap S = \emptyset$. Otherwise, $UnImp(l) = false$.*

Then, a location will be considered unimportant when it does not use any important channel.

Locations and processes priorities using *UnImp* predicate

The *UnImp* predicate will be defined in the following way:

Having:

$\mathcal{P}$ is a totally ordered set of priority levels of locations with ordering $\sqsubseteq_P$

$\mathcal{Q}$ is a totally ordered set of priority levels of processes with ordering $\sqsubseteq_Q$

where $\text{limit}_{\mathcal{P}}$ *is the border value between not-important location priority values and the priority values considered as important, and* $\text{limit}_{\mathcal{Q}}$ *is the border value between not-impotant priority values and important priority values of processes*

Definition 15 *Let $l_{\mathcal{P}}$ and $P'_{\mathcal{Q}}$ be the priority values of the location l and the process P' respectively, where $l_{\mathcal{P}} \in \mathcal{P}$ and $P'_{\mathcal{Q}} \in \mathcal{Q}$. The predicate* UnImp *of a location l will be $UnImp(l) = true$ whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$ then if $l[P] \equiv l[\texttt{kill} \mid P']$ then $l_{\mathcal{P}} \sqsubseteq_P \text{limit}_{\mathcal{P}} \vee P'_{\mathcal{Q}} \sqsubseteq_Q \text{limit}_{\mathcal{Q}}$. Otherwise, $UnImp(l) = false$.*

Therefore, a location will be considered unimportant when its priority is low or, if it is high, then the processes located on it have a low priority.

This way, the problem of concluding if a location is important or not using our *UnImp* predicate is a decidable property, whenever the network administrator set the correct values in the different parameters that define the predicate.

3.1.5 Usage example

One of the main fields where DπF network modeling with kill-safety properties can be used is the grid network design. In a grid network, a big process is distributed as smaller processes into the different nodes which form the network. These nodes are separated physically, so the network and the nodes by itself can have a different behaviour, i.e. a node can be killed in any moment, e.g. when the owner of the computer which is running the process shuts down the computer. This way, detecting the kill-safety of a grid network design is very important in order to notice if some information is lost. If this happens, the entire process cannot end correctly, because one of the small processes has not ended its tasks.

An example of grid design could be:

$\Delta \triangleright l_1[\texttt{go}\ l_2.(a!\langle Y\rangle.0 \mid \texttt{kill})] \mid l_2[a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0] \mid l_3[b?(Z).Q]$

The network of the example is not kill-safe and now we are going to see the reason. It will also be seen how the proposed methods could detect that situation. We have three nodes that are a small part of a large network. These nodes could be personal computers of people that yield their computer time to the grid network when their computers are not doing any other tasks. In this kind of grid network, controlling the nodes that are part of the global process is specially complicated. Leaving an important process to a computer whose owner could turn off it whenever he wants is unwise.

l_1 is a node which can do a migration to location l_2. Then, it can send a message through a channel a, or it can execute the kill order. We also appreciate that l_2 is a location which needs necessarily pick up the message from l_1 to continue its tasks. l_3 also needs that the communication between l_1 and l_2 works well because l_3 is waiting for a message from l_2 to run a generic process Q. l_1 could stop the running of the other two nodes.

Application of different kill-safety properties

For example, let us analyze the problem considering the two first commented safety properties to determine if the network is kill-safe (sections 3.1.1 and 3.1.2).

If definition 6 is taken into account, we can prove that the network is not *kill-safe* in this way:

A network N is *kill-safe* if whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$ then if $l[P] \equiv l[\texttt{kill} \mid P']$ then $l[P'] \equiv l[0]$

Just matching our network with the terms of the definition ($\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$), after the first migration from l_1 to l_2 takes place, we can reach a conclusion:

$N \equiv \Delta \triangleright l_1[0] \mid l_2[a!\langle Y\rangle.0 \mid \texttt{kill} \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0] \mid l_3[b?(Z).Q]$

$N' \equiv N$ to make true $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ without changing the network's state.

$N'' \equiv l_1[0] \mid l_3[b?(Z).Q]$

$l[P] \equiv l_2[a!\langle Y\rangle.0 \mid \texttt{kill} \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0]$

$l[P'] \equiv l_2[a!\langle y\rangle.0 \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0] \rightarrow l[P'] \not\equiv l[0]$!! Not *Kill-safe*

Another possible kill-safety approach is to take care about the important channels that must be set before the begin of any process. A specific list of important channels have to be set to test if a network is kill-safe.

Now we rely on section 3.1.2, "Important Channels", to reach the same conclusion than before.

a and b are important channels since critical messages can be sent through them. This way, we include them into the set of important channels S. As defined in definition 8:

Let S be a set of names. A network N is kill-safe *if whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$ then if $P \equiv \texttt{kill} \mid P'$ then $\mathcal{N}(P') \cap S = \emptyset$.*

The existence of a kill order in a location which uses important channels will put the network at risk of a not kill-safe situation. We can detect the risk matching the terms in definition 8 like in the previous case.

$N \equiv \Delta \triangleright l_1[0] \mid l_2[a!\langle Y\rangle.0 \mid \texttt{kill} \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0] \mid l_3[b?(Z).Q]$

$N' \equiv N$ to make true $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ without changing the network's state.

$N'' \equiv l_1[0] \mid l_3[b?(Z).Q]$

$l[P] \equiv l_2[a!\langle Y\rangle.0 \mid \texttt{kill} \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0]$

$l[P'] \equiv l_2[a!\langle y\rangle.0 \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0]$

$(a \in \mathcal{N}(P') \wedge a \in S)$ so $\mathcal{N}(P') \cap S = a \rightarrow a \neq \emptyset$!! Not *Kill-safe*

Using the other kill-safety property, the result will be similar.

3.2 A type system for DπF

In this section we define a type system for DπF which controls the kill-safety of a network.

In this first approach, our type system is able to detect the presence of a *kill* in a kill-unsafe situation, in order to conclude if the network is correctly specified. The priority levels of location importance is the choosen property for our type system.

3.2.1 Types

In our case, the importance type is the priority type. This way, we assign a priority value to a location.

In the next definition, $ch(S)$ is a type of a channel which can send channel types or location types, but just those whose type is S. In this way, the priority of the channel is defined by S. So we control that sending important location identifiers can only be sent through important channels. Typing the channels in our type system we can control the information that is being sent in our network. So, we can control statically the type of the identifiers that are sent through a concrete channel in order to not receive identifiers of incorrect type.

Definition 16 *Let S and B types for the type system:*

$S ::= ch(S) \mid B$ *(composite types)*

$B ::= M$ *(base types)*

where $M \epsilon \mathbb{I}$ *is a concrete priority value, depending on the definition 18.*

Having the set of type identifiers:

Definition 17 *Let* $\mathcal{T}$ *be the set of type identifiers.*

The type identifier of a name identifier can be obtained using a *type environment*:

Definition 18 *A type environment is a map* $\Gamma : \mathcal{N} \rightarrow \mathcal{T}$*, from the name identifiers to their type identifiers.*

Moreover, the specific type (a priority value) of a type identifier will be obtained by the following security policy:

Definition 19 *A security policy* $\mathcal{S}$ *is a function* $\mathcal{S} : \mathcal{T} \rightarrow S$ *from the type identifiers to their definition, and it is* ***closed*** *(every type that occurs in a definition is already defined).*

Then, we define a connectivity context whose objective is to represent the connectivity between the type identifiers of the locations:

Definition 20 *A connectivity context is a map* $\mathcal{D} : \mathcal{T} \times \mathcal{T} \rightarrow \mathbb{T}$*. It shows what types of our type system are connected (*true*).*

Thus, we distinguish between two kinds of contexts: the context of the semantics and the context of the types. However, they are related:

Definition 21 *The connectivity context* Δ *conforms to* $\mathcal{D}$ *with* Γ *if* $\Delta \vdash l_1 \leftrightarrow l_2$ *whenever* $\Gamma(l_1) = T_1 \wedge \Gamma(l_2) = T_2$ *then* $\mathcal{D} \vdash T_1 \leftrightarrow T_2$*.*

Referencing the *Unimportance predicate* introduced in section 3.1.4, we define a $\mathcal{U}$ *Unimportance predicate of the location types (priority values)* as a type oriented definition of this semantic predicate:

Definition 22 *An unimportance predicate for types is a map $\mathcal{U} : S \to \mathbb{T}$, where S is a priority value. It shows if the priority type value is unimportant (*true*) or important(*false*).*

In this case, the semantics and the types are also related:

Definition 23 *The unimportance predicate $\mathcal{U}$ conforms to UnImp with $\mathcal{S}$ if whenever UnImp(l) and $\mathcal{S}(T) = M$ where $\Gamma(l) = T$ then $\mathcal{U}(M)$.*

This way, if a location is considered not inmportant, both *UnImp* of its identifier and $\mathcal{U}$ of its priority type value will be true.

3.2.2 Typing rules

Let us specify the syntax of typed DπF, in order to apply the typing rules to it. All the typing rules are relative to the previous connectivity context description.

Networks	$N ::= 0 \mid (\nu l : S)N \mid l[P] \mid (N_1 \mid N_2)$
Processes	$P, Q ::= u!\langle V\rangle.P \mid u?(X).P \mid *u?(X).P \mid \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q \mid 0$
	$\mid (P \mid Q) \mid \texttt{go}\ u.P \mid \texttt{kill} \mid \texttt{break}\ u \mid \texttt{ping}\ u.P\ \texttt{else}\ Q$

Table 3.1: *Syntax of typed DπF*

The rules of our type system have the following format. They have a name to reference it that represents the typed process construct and an antecedent or hypothesis that has to be respected to infer the consequent. In the formula, the antecedent is placed above the consequent. Moreover, in some rules there are extra conditions that have to be aslo accomplished. This way, the general format of the typing rules is:

$$(\textit{Name of the rule})\ \frac{\textit{Antecedent}}{\textit{Consequent}}(\textit{Extra conditions}).$$

In the rules for typing networks we have *(S-NIL-NET)* which is very simple, since it just represents a network without locations and ensures that it will be well-typed in this case. The second rule is *(S-PAR-NET)* that says that two distinct networks can be checked just separating the two networks. The next network rule is *(S-SITE)* that shows a location with only a process P. If we verify that P is well-typed, then the location will be also well-typed. And the last one *(S-NEW)* assumes that a network with a declaration of a new name l is well-typed if the network without this declaration is well-typed.

(S-NIL-NET)	$\dfrac{}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} 0 : ok}$
(S-PAR-NET)	$\dfrac{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} N_2 : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 \mid N_2 : ok}$
(S-SITE)	$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$
(S-NEW)	$\dfrac{\Gamma, l : U \vdash^{\mathcal{D},\mathcal{S}} N : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu l : U)N : ok}$

Table 3.2: *Kill-safety network rules*

Then, we have the rules for typing processes. *(S-NIL-PROC)* is a rule to ensure that a nil process is safe. *(S-PAR)* separates a pair of processes to check them individually. The rule *(S-BANG)* says that if we have a replicated process we can check it just checking the code inside the replication. In *(S-NEW2)* we assume that if a process is well-typed then the declaration of a new name n will be also well-typed. In *(S-OUT)* the process construction after the output has to be well-typed and it has to take care about the types of the channel and the output. The case of *(S-IN)* is quite similar to the previous one but having inputs. *(S-GO)* and *(S-NGO)* work taking into consideration the type identifiers of both locations to determine if they are connected. The type system checks also the conditional process construct *(S-IF-THEN)*, where the process after the ”then” and the process after the ”else” have to be *ok* in order to be *ok* the conditional process construct *ok*. *(S-KILL)* checks the importance level of the location to consider if a kill could endanger the network behaviour. The last typing rules *(S-BREAK),(S-PING-OK),(S-PING-OK)* check the types of both involved locations to ensure that are connected types, and the ping or the break order can be succesfully done or not.

Assuming $\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{S}(T_l) = M_l$

(S-NIL-PROC)	$\dfrac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0 : ok}$
(S-PAR)	$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$
(S-BANG)	$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P : ok}$
(S-NEW2)	$\dfrac{\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P : ok}$
(S-OUT)	$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle v : U\rangle.P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = U)$
(S-IN)	$\dfrac{\Gamma, v_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = U)$
(S-GO)	$\dfrac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\mathcal{D}(T_l, T_k))$
(S-NGO)	$\dfrac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\neg\mathcal{D}(T_l, T_k))$
(S-IF-THEN)	$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$
(S-KILL)	$\dfrac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok}(\mathcal{U}(M_l))$
(S-BREAK)	$\dfrac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k : ok}(\mathcal{D}(T_l, T_k))$
(S-PING-OK)	$\dfrac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\mathcal{D}(T_l, T_k))$
(S-PING-NOT-OK)	$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\neg\mathcal{D}(T_l, T_k))$

Table 3.3: *Kill-safety process rules*

Correctness of the type system

At this moment, we now state two important theorems. Using them we can ensure that the well-typing of a network will be maintained and that if a network is well-typed then it will be *safe*.

Theorem 1 ***(Subject Reduction)*** *If* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$, Δ *conforms to* $\mathcal{D}$ *and* $\Delta \triangleright N \rightarrow \Delta' \triangleright N'$ *then also* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ *and* Δ*' conforms to* $\mathcal{D}$.

Theorem 2 ***(Safety)*** *If* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *and* Δ *conforms to* $\mathcal{D}$ *then* $\Delta \triangleright N$ *is safe.*

We can ensure with the theorem 1, subject reduction, that if a network is kill-safe (*ok*), it will be kill-safe with any reduction rule you apply to it. It is useful because if you can prove that a network is kill-safe, you will not have to prove it again with the applying of a new reduction rule. This way, if a network is ensured as "*ok*", the network will be safe (kill-safe), and it will continue being safe in any semantic reduction. If the typing rules are applied to a network which is considered well-typed then, by any semantic reduction, it will be always safe.

The lemma that shows the conforming between the predicate *Unimp* and the presence of a `kill` is the following:

Lemma 1 ***(Unimportance)*** *If* $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *and the unimportance predicate for locations (Unimp) conforms to the unimportance predicate for types (*$\mathcal{U}$*) then if* $l[P] \equiv l[\texttt{kill} \mid P']$ *then* $Unimp(l)$.

In order to ensure the previous theorems, the following lemma of subject equivalence has to be also also proved. If this lemma is proved, the *ok* typing of a network will be kept in any equivalent network:

Lemma 2 ***(Subject equivalence)*** *If* $N \equiv N'$ *and* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *then* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$.

The next step is to test if a network specified in DπF, that is typed as "*ok*", will conform with safety property. If this fact happens, this network will be kill-safe using the theorem 2. We will do this mathematical proof using induction, testing the different typing rules in order to ensure this conforming.

The final conclusion will be that our type system is capable to test and prevent any possible kill-unsafe situation. For additional information, go to appendix A.

3.2.3 A tiny example of application of the type system

Let us to consider this example:

$\Delta \triangleright l_1[\texttt{go}\ l_2.a!\langle Y\rangle.0] \mid l_2[\texttt{go}\ l_1.\texttt{kill} \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0] \mid l_3[b?(Z).Q]$

In this case, different possibilities can occur. On the one hand, let us $\Gamma(l_1) = T_1$, $\Gamma(l_2) = T_2$, $\mathcal{S}(T_1) = M_1$, $\mathcal{S}(T_2) = M_2$, $\mathcal{U}(M_1) = false$, $\mathcal{U}(M_2) = true$, i.e. l_1 is important and l_2 is unimportant. It is being assumed that $\mathcal{D}(T_1, T_2)$ is true, i.e. the type of the two locations are connected (so the locations are also connected). When the *kill* was transfered to the location l_1, the type system will detect the kill-unsafe situation in the same way it was commented before, using the rule *(S-KILL)*. On the other hand, if the important location is l_2, the network will be safe, because the kill will be migrated to an unimportant location. Finally, if both locations (l_1 and l_2) are important, the network will be kill-unsafe as in the first situation, using the same typing rule.

If the specified type system is used to detect whether the example is kill-safe, the result will be that the network is not kill-safe. But, first of all, the types have to be specified. If, for example, we assume $\Gamma(l_1) = T_1$, $\mathcal{S}(T_1) = M_1$ and $\mathcal{U}(M_1) = false$, i.e. the location l_1 is important, the type system will detect with *(S-KILL)* of table 3.3 the kill-unsafe situation of the network. This way, the network will not be kill-safe.

Otherwise, if the location l_1 is not important, the network will be kill-safe, because there is not situation where the network state was kill-unsafe. This is because the defined kill-safe property was based on importance level, and our location is not important, so a *kill* can be run in l_1 without compromise the safety of the network.

Let us show the type derivation tree, location by location, in order to conclude that the network is well-typed and, this way, it is kill-safe.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_2} 0 : ok}\,\text{(Step 3)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_2} a!\langle Y\rangle.0 : ok}\,\text{(Step 3)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_1} \texttt{go}\ l_2.a!\langle Y\rangle.0 : ok}\,\text{(Step 2)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l_1[\texttt{go}\ l_2.a!\langle Y\rangle.0] : ok}\,\text{(Step 1)}$$

If we analyze the tree, it can be observed how the different typing rules were applied. In the **step 1** the rule *(S-SITE)* of table 3.2 was used. This way, the kill-safety of the content of the

location l_1 is analyzed, so the table of process rules has to be used (table 3.3). In the **step 2** the rule *(S-GO)* was applied and in the **step 3** the rule *(S-OUT)* was applied. In the next step the only thing that rest to do is to apply the *(S-NIL-PROC)* to verify that finally, l_1 is well-typed.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_1} \texttt{kill} : ok}\,(\Gamma(l_1) = T_1; \mathcal{S}(T_1) = M_1; \mathcal{U}(M_1))}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_2} \texttt{go}\ l_1.\texttt{kill} : ok}\,(\Gamma(l_1) = T_1; \Gamma(l_2) = T_2; \mathcal{D}(T_1, T_2)) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_3} 0 : ok}\,\text{(Step 6)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_3} b!\langle X\rangle.0 : ok}\,\text{(Step 5)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_1} \texttt{go}\ l_3.b!\langle X\rangle.0 : ok}\,\text{(Step 4)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_1} a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0 : ok}\,\text{(Step 3)}}{\dfrac{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_2} \texttt{go}\ l_1.\texttt{kill} \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0 : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l_2[\texttt{go}\ l_1.\texttt{kill} \mid a?(X).\texttt{go}\ l_3.b!\langle X\rangle.0] : ok}\,\text{(Step 1)}}\,\text{(Step 2)}$$

The rule *(S-SITE)* is applied in the derivation **Step 1**. Applying the rule *(S-PAR)* on the derivation **Step 2**, two ramifications will be obtained. At the right, **Step 3**, *(S-IN)* is applied. Now, in **Step 4**, *(S-GO)* is applied for the migration order and immediatly later *(S-OUT)*, **Step 5**. Ending finally with *(S-NIL-PROC)*. The left ramification is more interesting because the migration from l_2 to l_1 is necessary to take into consideration the types of both locations in order to know if these types are conected by $\mathcal{D}(T_1, T_2)$. If T_1 and T_2 are types that cannot be connected to ensure the safety of the network then the unsafety of the network will be detected at this step. One of the main purpose of *(S-GO)* is to avoid that an important process ends in a location considered not important. Finally *(S-KILL)* is used to verify if the location that runs a kill is unimportant or not. If it is an important location then the network is not well-typed and therefore is not kill safe.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_3} Q : ok}\,\text{(Step 3)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_3} b?(Z).Q : ok}\,\text{(Step 2)}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l_3[b?(Z).Q] : ok}\,\text{(Step 1)}$$

The location l_3 will be well-typed too, using the rules *(S-SITE)* and *(S-IN)*. Finally, Q will be also *ok*, because it is an undefined process which is not supposed to have any kill-safety dangerous task.

Chapter 4

Controlling unwanted `kill` migrations

In this chapter a property of migration safety is analyzed, and we define an extension of the previous type system to control this property.

4.1 `kill` migration problems

4.1.1 Problem description

In chapter 3.1, different kill-safety properties were analyzed in order to detect if a network specified in DπF was correctly defined, taking into account the property of unimportance *(UnImp)* of the locations. This way, we analyzed that a location could be killed when its level of importance was not high, i.e. the location was unimportant.

However, if the killing migration problem is analyzed, a problematic situation which has not been studied can be observed: the possibility of migrating a `kill` from a location to another one which should not be killed since it is important. For example, considering a grid network, an important location as a core node of the network can migrate a killing process construct to a not important location, i.e. the different personal computers which run the processes. Nevertheless, migrating a `kill` from these personal computers to an important location (a core node of the network) can be possible if the network is defined using DπF. It is an unwanted situation which can endanger the correct behaviour of the grid since an unimportant location can kill a location which is managing the processes in the different personal computers and waiting their results.

4.1.2 DπF extension: `kill` k

Before analyzing the choosen property to study the migration safety, we decide to add a new kind of process construct to DπF syntax: `kill` k. The main objective of this addition is to represent the migration of a `kill` from a location to another one. One might think of representing this new operator as: `go` k.[...]`kill`. Although its result is the expected one, i.e. migrating a `kill` process to a different location k (using a `go` to migrate it), it is not the correct way to analyze the situation.

In order to detect a migration of a `kill` in the way we want to do it, i.e. sending of a killing order to a location whose unique objective is to kill the location, it should be made in an atomic process. This way, even though the previous migration of a `kill` using a `go` will kill the location, it will not do it in an atomic way. Between the `go` and the `kill` could exist several different process constructs, which may even migrate the `kill` again.

For example, let us define a tiny example using the new process construct `kill` k:

$$\Delta \triangleright l_1[\texttt{kill}\ l_2] \mid l_2[0] \tag{4.1}$$

$$\rightarrow (\Delta - l_1) \triangleright l_1[0] \mid l_2[0] \tag{4.2}$$

The location l_1 wants to kill l_2 immediately. Assuming that the locations are connected and both of them are alive, the result of the reduction 4.1 will be to set l_2 as *dead*. If the same setting is defined using `go` to migrate the `kill` instead using `kill` k, the following situation could happen:

$$\Delta \triangleright l_1[\texttt{go}\ l_2.\texttt{go}\ l_1.\texttt{kill}] \mid l_2[0] \tag{4.3}$$

$$\rightarrow \Delta \triangleright l_1[0] \mid l_2[\texttt{go}\ l_1.\texttt{kill}] \tag{4.4}$$

$$\rightarrow \Delta \triangleright l_1[\texttt{kill}] \mid l_2[0] \tag{4.5}$$

$$\rightarrow (\Delta - l_1) \triangleright l_1[0] \mid l_2[0] \tag{4.6}$$

In this case, there are two migrations: firstly from the location l_1 to the location l_2 and next to the location l_1 again, using in both of them (4.3 and 4.4) the reduction rule (R-GO). Finally, the killed location is l_1 what we do not want. The objective of the example was to kill l_2. It happens because using `go` for killing is a not atomic action, so other process constructs could appear between the `go` and the `kill`. For this reason, and atomic `kill` to kill a different location than the actual one is necessary.

In order to complete the explanation of the differences between the two process constructs, let us consider a security policy that prevents the killing of a node by itself but sometimes it allows the murder of a different location. Consequently, if the use of `kill` is forbidden, murdering a location without the order `kill` l_1 will be impossible:

$$\Delta \triangleright l_1[\texttt{go}\ l_2.\texttt{kill}] \mid l_2[Q] \tag{4.7}$$

$$\rightarrow \Delta \triangleright l_1[0] \mid l_2[Q \mid \texttt{kill}] \tag{4.8}$$

The previous situations will be detected as incorrect, since the `kill` is forbidden. The correct way will be:

$$\Delta \triangleright l_1[\texttt{kill}\ l_2] \mid l_2[Q] \tag{4.9}$$

$$\rightarrow (\Delta - l_2) \triangleright l_1[0] \mid l_2[0] \tag{4.10}$$

These examples show that the `go` k.`kill` has a different behaviour than `kill` k. This way, a new basic process construct `kill` k should be added to the DπF syntax to represent this atomic killing of a different location k, which will run as an atomic migration of a `kill` process construct to the location k and the running of this `kill` in k.

The bound and free names of the new construct will be:

Definition 24 ***Bound and free names of the new kill process construct***

$bn(\texttt{kill}\ k) = \emptyset$

$fn(\texttt{kill}\ k) = k$

On the other hand, the reduction semantics of the new process construct will be, considering the notation and network representation explained in section 2.2:

The result of the reduction will be killing the location k whenever it is accessible. Otherwise, the location k will not be removed but `kill` k will also disappear from l.

Assuming $\Delta \vdash l$: **alive**	
(r-kill-loc)	$\dfrac{}{\Delta \triangleright l[\![\texttt{kill}\ k]\!] \rightarrow (\Delta - k) \triangleright l[\![0]\!]} \Delta \vdash k \leftarrow l$
(r-nkill-loc)	$\dfrac{}{\Delta \triangleright l[\![\texttt{kill}\ k]\!] \rightarrow \Delta \triangleright l[\![0]\!]} \Delta \nvdash k \leftarrow l$

Table 4.1: *Murder Reduction Rule*

4.1.3 Migration property

In order to represent the safety property that has to be considered, a location hierarchy is analyzed. In this hierarchy the locations which compose the core of the network will have higher priority than the child locations. This way, a migration safety definition using priority levels of locations has to be defined.

For example, in Unix there is a similar hierarchy of possible users of the system [2]. The superuser (administrator) is in the top of the hierarchy with the highest priority in the system. Then, there are regular users which have a unique username and they belong to groups: a user is a member of at least one group and it can also be a member of one or more other groups. On the other hand, every directory and file on the system has an owner, and also an associated group. It also has a set of permission flags which specify separate read, write and execute permissions for the 'user' (owner), 'group', and 'other'. This way, in this security policy there are actions that only can be done by the superuser (e.g. only he can create new groups or add/delete group members) and there are different ones that can be done by everyone with the appropiate permision (e.g. only one user can modify a file if he is its owner whether its permission flags are set in this way).

This scene is similar to the hierarchy that we want to suggest, where only some more prioritary locations can do actions as killing a different location.

At this time, we can define the new migration-safe property:

Definition 25 *Let l_γ and k_γ be the integer priority value of the location l and k respectively. A network N is* migration-safe *if whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then if $N' \equiv l[P] \mid N''$ then if $l[P] \equiv l[\texttt{kill}\ k \mid P']$ then $l_\mathcal{P} = x$ and $k_\mathcal{P} = y$, where $x \epsilon \mathcal{P}, y \epsilon \mathcal{P} \wedge y \sqsubseteq x$, where $\sqsubseteq$ is the usual ordering of the integers.*

This way, just a location which is a high prioritary location can send a killing order to a less prioritary location, i.e. the core nodes can control the situation of their different children location and kill them, but this last ones can not kill their parents.

4.2 A type system to control `kill` migration

Once the problem about inappropriate `kill` migration has been studied, this section will try to design a type system which controls the correct behaviour of the network depending on the property defined in definition 25. This way, the type system will detect the migration of a `kill` and it will analyze if it is *migration-safe* or not.

Note that the type system is an extension of the proposed one in the previous chapter, so it will control the *kill-safety* and the *migration-safety* at the same time.

4.2.1 Types

This type system is also based on location priorities as the previous one, from their identifier to their type identifiers (with Γ), and later from their type identifiers to the specific integer value of priority (with $\mathcal{S}$). Remember that the set of priority values is a totally ordered set of integers with the usual ordering of these numbers. This way, the priorities of the locations will be defined in the following way:

Definition 26 *The set of priorities $\mathcal{P}$ is given by $\mathcal{P} = (\mathbb{Z}, \leq)$.*

A type will be defined as a priority value (when it is a location type) or as a channel type value. These types of our type system are similar to the defined in definition 16 of chapter 3. We adapt this definition using definition 26 for having the ordering of integers:

Definition 27 *Let S and B types for the type system:*

$S ::= ch(S) \mid B$ *(composite types)*

$B ::= M$ *(base types)*

$where M \epsilon \mathcal{P}$.

This way, as in the previous type system of chapter 3, the type identifiers will be obtained:

Definition 28 *A type environment is a map* $\Gamma : \mathcal{N} \rightarrow \mathcal{T}$*, from the name identifiers to their type identifiers.*

And later, we can obtain the type definitions using the security policy that works with type identifiers:

Definition 29 *The set of type identifiers is assigned to specific type definitions by a security policy* $\mathcal{S}$ *that is a function* $\mathcal{S} : \mathcal{T} \rightarrow S$*, and it is* ***closed*** *(every type that occurs in a definition is already defined).*

In the previous security policy, S will be $\mathcal{P}$ when the type is identifying a location. Otherwise, S will be a channel type.

In the new property to be detected (*migration-safety*), these priority values of locations will be used directly. The priority levels will be a ordered set of integer numbers which represents the priority level of a specific location, and these values will be compared between the different locations through which the migration is being done. This way, depending on this comparison the network will be well-typed or not.

The conectivity context (definition 20) and the conforming between Δ and $\mathcal{D}$ (definition 21) will be the same. Moreover, the predicates used to control the *kill-safety* of a network will not vary (definitions 22 and 23).

This way, the main changes in the type system will be in the typing rules.

4.2.2 Typing rules

Referencing the syntax of typed DπF in 3.1, the priority levels of the locations have to be compared in order to conclude if a network is *migration-safe* or not. It will be *migration-unsafe* when the migration is done from a location with a priority level of *W* to a location whose priority is *W'*, being *W'* a higher value than *W* with the ordering of the integer numbers.

The following process rules have to be added to the rules of section 3.2.2, setting a process construct as ”ok” when it is correctly typed using the type system, both kill-safe and migration-

safe property. The format of the rules will be the same as the explained one in section 3.2.2 of the previous chapter:

Assuming $\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{S}(T_l) = M_l; \mathcal{S}(T_k) = M_k$	
(S-MURDER-OK)	$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\mathcal{D}(T_l, T_k); M_k \sqsubseteq M_l)$
(S-MURDER-NOT-OK)	$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\neg\mathcal{D}(T_l, T_k))$

Table 4.2: *Added process rules*

In the rule (S-MURDER-NOT-OK) of table 4.2 the relation between the type identifiers of the locations was not considered because the locations (the type identifiers of the locations) were not related, i.e. they are not connected on $\mathcal{D}$.

Analyzing the type system, two kind of `kill` orders are being allowed: the `kill` of a location using a `kill` order placed in this location, or the kill of a location from a different location using `kill` k (being k the killed location). This way, the addition in the type system controls the migration of *murder* actions from a location, i.e. `kill` k located in this location. The behaviour of `kill` order is not supposed to be used to kill a different location, but only to kill a location by itself.

This way, the new type system detects if a network is correctly typed using two properties (kill-safety and migration-safety). If the network does not respect one of the properties, the network will be incorrectly typed, even though the other property is being respected.

Correctness of the type system

In the following,we present the theorems and lemmas interesting for our type system. On the one hand, the explained in the previous chapter subject reduction theorem continues being the same in this chapter:

(Subject Reduction theorem) *If* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$, Δ *conforms to* $\mathcal{D}$ *and* $\Delta \triangleright N \rightarrow \Delta' \triangleright N'$ *then also* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ *and* Δ*' conforms to* $\mathcal{D}$*.*

The proof of the lemma only has to be completed with the proof of the new added local reduction rules, since the rest of the typing rules continue being the same as the presented ones in chapter 3 and they were proved in appendix A. We refer to appendix B to see the new proofs.

On the other hand, the safety theorem continues being also the same:

(Safety theorem) *If* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *and* Δ *conforms to* $\mathcal{D}$ *then* $\Delta \triangleright N$ *is safe.*

In order to prove the theorem, the lemma 1 of the previous chapter has to be proved with the two new typing rules to see if it continues being right (since the rest of the typing rules are the same and they were proved in appendix A:

(Unimportance lemma) *If* $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *and the unimportance predicate for locations (Unimp) conforms to the unimportance predicate for types (*$\mathcal{U}$*) then if* $l[P] \equiv l[\texttt{kill} \mid P']$ *then* $Unimp(l)$*.*

Moreover, we present a new lemma using the murder-safe property presented in this chapter, and it must also be proved using every typing rule:

Lemma 3 ***(Murder-safe)*** *If* $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *then if* $l[P] \equiv l[\texttt{kill } k \mid P']$ *then* $\Gamma(l) = N_l$,$\Gamma(k) = N_k$ *and* $N_k \sqsubseteq N_l$*.*

Finally, the subject equivalence lemma continues being the same and the networks typing rules are also the same, so we refer to Appendix A to see the proof of it:

(Subject equivalence lemma) *If* $N \equiv N'$ *and* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *then* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$

For additional information about the proofs of the theorems, go to appendix B.

Chapter 5

The problem of indirect *kills*

In chapter 4, a type system to control the unwanted migration of a `kill` was defined. However, this type system deals only with direct `kill` migrations, from a location to another one trying to kill this destination location. Indirect murder process constructions are also possible.

5.1 Indirect murders

If the problem of the killing migrations is analyzed, some problems can be found in order to conclude that there are behaviours that the previous type system does not prevent.

5.1.1 Murder by migrated proxy

It is well-known that the type system prevents that a location kills another one of higher priority. However, indirect murder actions could be reached through the migration of a `kill` k from the location that wants to kill k to a different location. This way, a location with low priority can kill a high priority location (k), using a different location of high priority with the same or higher priority than the priority of the location k.

Let us define an example that shows this:

$\Delta \triangleright l_1[\texttt{go}\ l_2.\texttt{kill}\ l_3] \mid l_2[0] \mid l_3[0]$

We assume that the priorities of the locations are: $\Gamma(l_1) = N_1$, $\Gamma(l_2) = N_2$ $\Gamma(l_3) = N_3$ where

$N_1 < N_2$ and $N_2 > N_3$. This way, l_1 could not kill directly location l_3 because its priority is lower than the priority of l_3. In the example we can see how l_1 places the murderer at location l_2. Then, the murder can be done because the priority of l_2 is higher than priority of l_3.

The proposed type system does not prevent this situation. It should check the migration of `kill` k process constructs, since this migration could be trying to kill someone through a different location, taking advantage of the higher priority of this location. The aim of this kind of migrations has to be analyzed.

5.1.2 Murder by listening murderer

There is another way to perform an indirect kill. Let us define an example in order to explain it:

$\Delta \triangleright l_1[\texttt{go}\ l_2.x!\langle l_3\rangle.0] \mid l_2[x?(z).(\texttt{kill}\ z \mid 0)] \mid l_3[0]$

Having the same priority levels than the defined in the previous example, if we use the behaviour explained in section 5.1.1, this situation will not be detected because no `kill` k process constructs are being migrated. Instead of that, l_1 asks l_2 to kill l_3, using a migrated input process to do it.

This way, the location l_1 is also using the high priority of location l_2 to kill l_3. It should not be done, because l_1 is a low priority location that cannot kill locations with higher priority than its priority.

5.2 A type system that prevents indirect murders

In order to detect these indirect situations, a different type system should be defined. The previous type systems contained the notation *ok* to conclude if a specific network configuration was correctly defined in accordance with the chosen safety property: priority levels of locations. The new type system will use a different property to set a network as *ok*, and it will be a set of different locations that can be killed from each location of the network.

5.2.1 Safety property

The safety property that the type system will detect is the following one. It will not work comparing priority levels between locations as the previous type system, but it will work using specific sets of location types.

$l_{\mathcal{K}}$ is the set of location types that can be killed by location l

T_k is the type of location k

$k_{\mathcal{K}}$ is the set of location types that can be killed by location k

Definition 30 *A network N is* now-safe *if whenever $N \equiv l[P] \mid N'$ then*
if $l[P] \equiv l[\texttt{kill}\ k \mid P']$ then $T_k \in l_{\mathcal{K}}$
else if $l[P] \equiv l[\texttt{go}\ k.Q \mid P'']$ then $k_{\mathcal{K}} \subseteq l_{\mathcal{K}}$.

The previous definition prevents direct and indirect murders. This way, a network will be *safe* when it is always *now-safe*:

Definition 31 *A network N is* safe *if whenever $\Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ then N' is* now-safe.

We can understand the notion of safety presented in section 4.1.3 of Chapter 4 (a hierarchical network where only a location with a higher position in the hierarchy could migrate something to a lower position location, and only the first one can kill a location placed in a lower position in the hierarchy) as a special case of our new notion of safety as follows.

5.2.2 Types

In this type system, we assign a type to each location like in the previous type system. However, this type will have a different meaning, namely a set of types that can be killed from a location of this type.

Using the proposed ideas in [8], the types of our type system will be:

Definition 32 *Let R range over the set of types. This set has its syntax given by:*

$$R ::= \quad loc(\widetilde{K}) \quad \textit{(location type)}$$

$$\mid ch(R) \quad \textit{(channel type)}$$

where $\widetilde{K}$ is the set of location types that can be killed by the location.

This way, if previously the types of the type system were the priority levels of the locations, now the types will be location labels with a set of location types: those than can be killed by the location. So the property considered in the type system is quite different.

Having the set of type identifiers:

Definition 33 *Let $\mathcal{T}$ be the set of type identifiers.*

The type identifiers will be obtained:

Definition 34 *The type environment $\Gamma : \mathcal{N} \rightarrow \mathcal{T}$ is a map from the name identifiers to their type identifiers.*

Using these definitions of types and type identifiers, the security policy that contains the types which conform our type system will be:

Definition 35 *A security policy $\mathcal{S}$ is a function $\mathcal{S} : \mathcal{T} \rightarrow R$ from the type identifiers to their definition, and it is **closed** (every type that occurs in a definition is already defined).*

So, the security policy specifies what types could be killed by every location, i.e. it specifies the content of the set $\widetilde{K}$ for the type of each location.

This way, the new type system can be considered as an evolution of the previous one in the following way. Where we had priority levels to be the type of a location, now we have labels which have a set of location types that can be killed from the location represented by the label.

Let us explain the equivalence between the previous types system and the new one using an example of the definition of three locations l_1, l_2 and l_3, where l_1 is the most prioritary, i.e. it can kill any location, and l_3 is the less prioritary, so it cannot kill any location. Using the older type system we had:

$$\Gamma(l_1) = M, \Gamma(l_2) = N, \Gamma(l_3) = S \text{ where } M > N, N > S \text{ and } M, N, S \in \mathbb{N}.$$

With the new type system we have:

$$\Gamma(l_1) = T_1, \Gamma(l_2) = T_2, \Gamma(l_3) = T_3$$

$$\mathcal{S}(T_1) = loc(\widetilde{K_1}) = loc(\{T_2, T_3\})$$

$$\mathcal{S}(T_2) = loc(\widetilde{K_2}) = loc(\{T_3\})$$

$$\mathcal{S}(T_3) = loc(\widetilde{K_3}) = loc(\emptyset)$$

If previously the killing of locations was controlled comparing location priority levels (M, N, S), now the sets $\widetilde{K}$ are used to represent the locations that can be killed by the location where $\widetilde{K}$ is placed. This way, using two different but equivalent notations l_1 can kill l_2 and l_3, and location l_2 can just kill l_3.

5.3 How *Kill-safety* is also prevented in the new type system

The new type system prevents murder problems but it will also control the *kill-safety* of a network. However, the type system will work with a different property since it do not use unimportance predicates. This way, the list of possible locations that a location can kill has to contain this location when it is considered not important.

In the type system of chapter 3.1, a location can be killed (by a `kill`) when this location is considered not important using the predicate *UnImp*. In this adapted type system, a not important location will be represented by the inclusion of this location in its own list $\widetilde{K}$ of its type $loc(\widetilde{K})$. It will be the new characteristic to analyze in order to conclude if the presence of a `kill` is correct or not depending on the importance of the location, i.e. it will be the characteristic that will be used to conclude the *kill-safety* of a `kill` process construct in a DπF network definition.

Let us show an example of three locations (l_1, l_2, l_3), where just one of them (l_2) is unimportant and the other two locations are important, so they should not be killed. This setting has to be defined as follows:

$$\Gamma(l_1) = T_1, \Gamma(l_2) = T_2, \Gamma(l_3) = T_3$$

$$\mathcal{S}(T_1) = loc(\emptyset), \mathcal{S}(T_2) = loc(\{T_2\}), \mathcal{S}(T_3) = loc(\emptyset).$$

This way, only l_2 can be killed since it the only unimportant location. In this example we have not defined the location types that can be murder by each location.

Therefore, the new type system will control the *kill-safety* and the *murder-safety* of a DπF network.

5.4 Typing rules

The rules that controls the new type system are presented in tables 5.1 and 5.2.

In the case of the rules for typing networks, they continue being the same than the presented in the two commented before type systems, since they do not add any special property. They just do that the analysis of they well-typing of a network continues with the analysis of the processes contained on each location (M-SITE), with the analysis of the networks that forms two concurrent networks (M-PAR-NET), or with the analysis of the network after the declaration of a new variable of type (M-NEW). It also controls the typing of a nil network (M-NIL-NET), that is always *ok*.

(M-NIL-NET)	$\dfrac{}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} 0 : ok}$
(M-PAR-NET)	$\dfrac{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} N_2 : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 \mid N_2 : ok}$
(M-SITE)	$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$
(M-NEW)	$\dfrac{\Gamma, l : U \vdash^{\mathcal{D},\mathcal{S}} N : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu l : U)N : ok}$

Table 5.1: *Migration-safety network rules*

On the other hand, the rules for typing processes are quite similar than the previous ones, but they have some differences that we are going to comment. First, the declaration of the types will use the new types that conforms our type system instead of using priority levels. This way,

some typing rules change. For example, (M-KILL) will test if the location that wants to die with a `kill` has its location identifier inside of its set $\widetilde{K}$. If it happens, the location will be considered unimportant. On the other hand, we also control the migrations in our type system (M-GO), in order to prevent unwanted situation. The sets $\widetilde{K}$ of the origin location and of the location where we are migrating are tested. If the origin one is smaller than the destination one, the type system will detect that as not well-typed. Doing that, we prevent indirect murders. Unwanted direct murders are also prevented with (M-MURDER-OK) typing rule, testing if the location that we want to kill is inside of our set $\widetilde{K}$. The rest of typing rules are quite similar than the rules that conforms the previous type systems.

Correctness of the type system

In section 5.2.1 we explained the property that the security policy has to control in order to prevent migration by proxy problems. Using that, we can define a lemma to detect if a network is *now-safe* and, then, try to conclude if a network is safe.

Lemma 4 ***(Now-safe lemma)***

$$\begin{aligned} \textit{If } \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \quad & \textit{where } \Gamma(l) = T_l,\ \mathcal{S}(T_l) = loc(\widetilde{K}) \textit{ then} \\ & \textit{if } l[P] \equiv l[\texttt{kill}\ k \mid P'] \textit{ where } \Gamma(k) = T_k \textit{ then } T_k \in \widetilde{K} \\ & \textit{else if } l[P] \equiv l[\texttt{go}\ k.Q \mid P''] \textit{ where } \Gamma(k) = T_k,\ \mathcal{S}(T_k) = loc(\widetilde{K'}) \textit{ then } \widetilde{K'} \subseteq \widetilde{K}. \end{aligned}$$

On the other hand, we will also need two more theorems, one to show that a *ok* network will continue being *ok* after any semantic reduction, and another one that shows the equivalence between a well-typed network and a safe network, i.e. if a well-typed network will be always safe. We will use the previous lemma 4 in the proof of this last theorem.

The theorems, that are similar to the commented ones in chapter 3, will be the following ones:

Theorem 3 ***(Subject Reduction)*** *If* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$, Δ *conforms to* $\mathcal{D}$ *and* $\Delta \triangleright N \rightarrow \Delta' \triangleright N'$ *then also* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ *and* Δ*' conforms to* $\mathcal{D}$*.*

Theorem 4 ***(Safety)*** *If* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *and* Δ *conforms to* $\mathcal{D}$ *then* $\Delta \triangleright N$ *is safe.*

Assuming $\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{S}(T_l) = loc(\widetilde{K}); \mathcal{S}(T_k) = loc(\widetilde{K'}); \Gamma(u) = T_u; \Gamma(v) = T_v$

(M-NIL-PROC) $$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0 : ok}$$

(M-PAR) $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$$

(M-BANG) $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P : ok}$$

(M-NEW2) $$\frac{\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P : ok}$$

(M-OUT) $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle v : U\rangle.P : ok}(\mathcal{S}(T_u) = ch(U),\ \mathcal{S}(T_v) = U)$$

(M-IN) $$\frac{\Gamma, v_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}(\mathcal{S}(T_u) = ch(U),\ \mathcal{S}(T_v) = U)$$

(M-GO) $$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\mathcal{D}(T_l, T_k); \widetilde{K'} \subseteq \widetilde{K})$$

(M-NGO) $$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\neg\mathcal{D}(T_l, T_k))$$

(M-IF-THEN) $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$

(M-KILL) $$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok}(T_l \in \widetilde{K})$$

(M-BREAK) $$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k : ok}(\mathcal{D}(T_l, T_k))$$

(M-PING-OK) $$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\mathcal{D}(T_l, T_k))$$

(M-PING-NOT-OK) $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\neg\mathcal{D}(T_l, T_k))$$

(M-MURDER-OK) $$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\mathcal{D}(T_l, T_k); T_k \in \widetilde{K})$$

(M-MURDER-NOT-OK) $$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\neg\mathcal{D}(T_l, T_k))$$

Table 5.2: *Migration-safety process rules*

As in chapter 3, in order to ensure the previous theorems, the following lemma of structural equivalence has to be also proved. If this lemma is proved, the *ok* typing of a network will be kept in any equivalent network:

Lemma 5 ***(Subject equivalence)*** *If* $N \equiv N'$ *and* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *then* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$

These theorems will show that our type system is correct, and they are proved in appendix C.

5.4.1 Examples of application of the type system

We can analyze again two of the previous examples that showed indirect murder situations, with the typing rules that we have defined in this chapter. As we saw, a low priority node could kill a high priority location using a different high priority location to kill it indirectly.

In both examples, location l_2 was the most prioritary location and l_1 and l_3 the less prioritary locations, so the types will be: $\Gamma(l_1) = T_1, \Gamma(l_2) = T_2, \Gamma(l_3) = T_3$ and $\mathcal{S}(T_1) = loc(\emptyset), \mathcal{S}(T_2) = loc(\{T_1, T_3\}), \mathcal{S}(T_3) = loc(\emptyset)$.

First, let us remember the *murder by migrated proxy* example:

$\Delta \triangleright l_1[\texttt{go}\ l_2.\texttt{kill}\ l_3] \mid l_2[0] \mid l_3[0]$

We are going to see how our new type system is able to detect this unwanted situation.

$$\dfrac{\dfrac{\text{ERROR}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}}_{l_1} \texttt{go}\ l_2.\texttt{kill}\ l_3 : ok}\ (\text{Step 2})\ (\mathcal{D}(loc(\widetilde{K}), loc(\widetilde{K'})); l_{2\mathcal{K}} \subsetneq l_{1\mathcal{K}})}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l_1[\texttt{go}\ l_2.\texttt{kill}\ l_3] : ok}\ (\text{Step 1})$$

The first step is to apply the rule (M-SITE). In the **step 2** we apply the rule (M-GO) which detects the risk of migrate from a low priority node to other of higher priority. The set of nodes that can be killed by location l_2 is not include inside of the set of nodes that can be killed by l_1 ($l_{2\mathcal{K}} \subsetneq l_{1\mathcal{K}}$), then (M-GO) confirms that the code is not well-typed. Finally, l_2 and l_3 checking is trivial by (M-NIL-PROC).

In the case of the example of *Murder by listening murderer* we work in a similar way.

$\Delta \triangleright l_1[\texttt{go}\ l_2.x!\langle l_3\rangle.0] \mid l_2[x?(z).(\texttt{kill}\ z \mid 0)] \mid l_3[0]$

$$\dfrac{\dfrac{\dfrac{\text{ERROR}}{\Gamma \vdash_{l_1}^{\mathcal{D},\mathcal{S}} \texttt{go}\ l_2.x!\langle l_3\rangle.0 : ok}\ (\mathcal{D}(loc(\widetilde{K}), loc(\widetilde{K'})); l_{2\mathcal{K}} \subsetneq l_{1\mathcal{K}})}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l_1[\texttt{go}\ l_2.x!\langle l_3\rangle.0] : ok}\ \text{(Step 1)}}{}\ \text{(Step 2)}$$

The **step 1** is to apply the rule (M-SITE). In **step 2** we apply like before the rule (M-GO) to finally conclude that l_1 is trying to migrate to a node of higher prioriry and our new type system does not allow it. Then, the network will not be well-typed. But let us see how the rest of locations are well-typed:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_{l_2}^{\mathcal{D},\mathcal{S}} \texttt{kill}\ z : ok}\ (\mathcal{D}(loc(\widetilde{K}), U); U \in \widetilde{K}) \quad \dfrac{}{\Gamma \vdash_{l_2}^{\mathcal{D},\mathcal{S}} 0 : ok}\ \text{(Step 4)}}{\Gamma, z_{l_2} \vdash_{l_2}^{\mathcal{D},\mathcal{S}} \texttt{kill}\ z \mid 0 : ok}\ \text{(Step 3)}}{\Gamma \vdash_{l_2}^{\mathcal{D},\mathcal{S}} x?(z).(\texttt{kill}\ z \mid 0) : ok}\ (\Gamma(x) = ch(U),\ \Gamma(z) = U)}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l_2[x?(z).(\texttt{kill}\ z \mid 0)] : ok}\ \text{(Step 1)}}{}\ \text{(Step 2)}$$

The first step is to apply the rule (M-SITE). After that, it is possible to apply the rule (M-IN) that does not detect any problem. Now, in **step 3**, we have to resort to (M-PAR) to divide the two location processes. The branch at **step 4** is well-typed taking into account (M-NIL-PROC) and on the other branch (M-MURDER-OK) is applied. At this point, we already now the type of the name z and it is possible to conclude that the location source is well-typed.

To finish, it is easy to see that $l_3[0]$ is well-typed by using (M-NIL-PROC). However, since the content of location l_1 is not well-typed, the network will not be well-typed.

5.4.2 How our type system works using a restrictive property

The proposed type system will prevent direct murders, but it also prevents indirect murders. Every location that can be reached by a migration from a different location has to avoid to kill a location whose type is not in the set $\widetilde{K}$ of the origin location. This way, we are detecting if the location that recieves the migration of a `kill` k is able to kill more locations than the sender (because its position in the hierarchy is higher) and then, this location could be used to kill a different location from a location whose priority was lower.

But, at the same time, the indirect murders are prevented in the next step, i.e. the location which recieves correctly a `kill` k will detect if there is some migration attempt to a location with a larger set $\widetilde{K}$ than its set. This way, the transitivity of the indirect murders is respected in

each step of the migration from a location to another one that is not directly connected to the first one.

Analyzing the proposed property that defines our type system, it prevents this kind of situations and complies with the explained policy. It achieves that because it is a restrictive property: it does not let migrate any process construct to a location whose $\widetilde{K}$ is not a subset of the $\widetilde{K}$ of the origin location. This way, although it prevents murder by proxy problems, different situations which are not dangerous are forbidden, because the migration of not murder process constructs is also forbidden.

Let us show this situation with an example:

$\Delta \triangleright l_1[a!\langle Y\rangle.0] \mid l_2[\mathsf{go}\ l_1.a?(X).P] \mid l_3[0]$

Having:

$$\Gamma(l_1) = T_1 \text{ and } \Gamma(l_2) = T_2$$

$$\text{where } \mathcal{S}(T_1) = loc(\widetilde{K_1}) = loc(\{T_2, T_3\}) \text{ and } \mathcal{S}(T_2) = loc(\widetilde{K_2}) = loc(\{T_1\})$$

The defined security policy of the example will not prevent murders by migrated proxy, because a migration is being made from the location l_2 to l_1 and the set $\widetilde{K}$ of l_1 is not contained in the set $\widetilde{K}$ of l_2, since it is larger. This way, the type system which controls these migrated murders safety will conclude that the network of the example is not well-typed. However, if the example is analyzed, it does not have any *murder* process construct on it, so the type system is detecting a not dangerous situation as dangerous. It will be one of the limitations of the type system: it will be very restrictive.

Nevertheless, the type system will represent the behaviour of the previously commented hierarchical network that we wanted to represent. On it, only nodes placed at a higher position can migrate information to a lower position nodes, and these last ones can only execute processes, do inputs or outputs, or have any other DπF process construct different of a go. This way, the situation that is shown in this last example is the expected one according to our objectives.

Chapter 6

Conclusions

6.1 Results

Firstly, we presented simple properties to analyze the kill-safety condition in any network modeled with DπF . Maybe, more kill-safety options could be defined but the explained rules are enough to understand the behaviour of any network, e.g. a grid network. Their main objective were to detect killing important nodes. This concept of ”importance” is ambiguous and we defined it in each proposal in different ways:

- locations with only one `kill` running on it.
- locations with processes that use important channels
- processes/locations with high priority levels (established by the network administrator)

So we have to choose the most appropriate property in each situation. In the first type system proposed, the importance of the locations was chosen. With this type system we have demonstrated that the application of it in a process calculus as DπF to analyze the status of a network is a good option. Through the application of our rules we have proved that we can find a deterministic way to certificate that exists or not risks with the existence of a killing order in a specific location.

In the second part of the report a different problem was analyzed: the unwanted murders or the migration of a `kill` to an important location, using priority levels as importance values. This

way, the hierarchical relation of a grid network was respected. In order to develop the new type system, a new process construct for DπF was introduced (`kill` k), whose objective was to kill a location from a different one. We think it is an interesting addition, since it lets kill a different location in an atomic way. It did not exist in DπF and it brings us a better control of murders.

Finally, we found the problem of indirect murders and we solved it designing a new type system. This type system reformulates the notion of importance of a location. It works using sets of killable locations from a specific location. After the definition of the type system we concluded that it prevents both direct murders and indirect murders (and also killing orders), but it is not a complete solution to prevent them since it is very restrictive. The type system prevents the problem but there are networks typed as not well-typed but they are safe.

There are several applications for that investigation. Nowadays, distributed computation is more and more important for enterprise and investigation uses. If we do not trust completely in the well behaviour of the nodes, we should trust in an algorithm which guarantees the detection of an unsafe network status. If we cannot be sure of the correct behaviour of the nodes taken into consideration, we have to ensure the way to detect the possible unexpected situations. The proposed type systems are a deterministic and affordable choice.

We would like to note that we consider DπF a very powerful and useful language to model network systems. Its syntax is well-defined and easy to understand, and it has also very precise semantic rules. This way, we can design a type system easily. A type system is also a good methodology to prevent the problematic situations that we wanted to avoid since it works statically. Despite of the restrictions of the type systems that we commented before, the correctiveness of one of them depends on the safety property that the type system prevents. The more correct property, the more accurate type system.

6.2 Further work

As we commented at the end of the previous chapter, the last type system that controls killing orders, direct murders and indirect murders should be improved, since it is very restrictive. Doing that, it will be more accurate typing a network. A possible approach is to detect if after a migration order there is any murder process construct or not. If it is affirmative, this system will not be well-typed. Therefore, we could define a predicate of admittance of a `kill` process construct in a process P to analyze this situation:

Definition 36 *A predicate* $l[P] \downarrow_{\texttt{kill}}$ *holds whenever* $l[P] \equiv l[\texttt{kill} \mid P']$ *for some* P'

And we could add a new rule to our type system to control that whatever process P placed after a `go` would not contain any `kill`.

$$\text{(NEW-MIGRATION-RULE): } \frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok \wedge \neg k[P] \downarrow_{\texttt{kill}}}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

This way, if after a migration there is no killing or murder, then the system will be well-typed since it is migrating process constructs that do not endanger the behaviour of the network

On the other hand, we prevented different problems of `kill` process construct, and we tried to detect this kind of problematic situation in order to expect a correct behaviour of a network defined in DπF. However, there are different situations that should be also detected in order to control any problematic situation of a network. Unwanted `break` k is one of them. A *break* could be done in an important channel, e.g. in a central channel which recieves the data from all the locations in order to be analyzed in a core location. If this channel is broken before receiving this kind of important information, the behaviour of the network will not be the expected one. This situation should be controlled analyzing the problem and trying to define a type system to prevent it.

Finally, the type system that we have developed could be integrated in a software tool to design networks. It would help to the network administrators to design secure networks under certain properties. This properties will be identified as potentially hazardous for the proper functioning of the network. It could be developed as a graphic software to show the state of a grid network in order to analyze its behaviour after applying changes on it, e.g. breaking channels, killing locations, etc.

This way, a DπF language processor that includes types should be developed improving, for example, the Dπ language processor Nomadic Pict [12]. After that, we should add the necessary language checkers to detect the unwanted behaviours represented by our proposed typing rules. So, the final user can design graphically its network or he could define the network directly in DπF language with its specific types. In the first case, the network will be translated to DπF language. Later, this system in DπF language will be analyzed in order to conclude if the network is safe. The program will also let do the previously commented actions (kills, breaks, etc.) to observe the evolution of the system behaviour. This way, this software would be a very useful tool in the field of designing computer networks.

Appendix A

Proof of typing rules: Kill safety

The aim of this part of the report is to prove the correctness of the type system presented. The two theorems presented in 1 and 2, and now presented again, have to be proved in order to ensure the kill-safety of a DπF network.

A.1 Subject equivalence lemma

Analyzing the structural rules of DπF syntax, the lemma 2 have to be proved:

Lemma 2:

If $N \equiv N' ok$ *and* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *then* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$

We will prove it by induction in the rules defining structural equivalence. If it is made, we can ensure that an equivalent network of a specific network will be also *ok*.

A.1.1 Structural rules

(S-COMM)

$N \mid M \equiv M \mid N$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

But then, by the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M \mid N : ok$

(S-ASSOC)

$(N \mid M) \mid M' \equiv N \mid (M \mid M')$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (N \mid M) : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M' : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (N \mid M) \mid M' : ok$

But then, by the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M' : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M \mid M' : ok$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} (M \mid M') : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid (M \mid M') : ok$

(S-UNIT)

$N \mid l[0] \equiv N$

By the rule (S-NIL-NET): $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid l[0] : ok$

But then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$

(S-EXTR)

$(\nu n : U)(N \mid M) \equiv N \mid (\nu n : U)M \; ; \; n \notin \mathbf{fn}(\mathbf{N})$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

By the rule (S-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (N \mid M) : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)(N \mid M) : ok$

But then, by the rule (S-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)M : ok$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid (\nu n : U)M : ok$ whenever $n \notin \mathbf{fn}(\mathbf{N})$

(S-FLIP-1)

$(\nu n : U)(\nu m : W)N \equiv (\nu m : W)(\nu n : U)N \; ; \; n \notin \mathbf{fn}(\mathbf{W})$

By the rule (S-NEW): if $\Gamma, m : W \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$

By the rule (S-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)(\nu m : W)N : ok$

But then, by the rule (S-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$

By the rule (S-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)(\nu n : U)N : ok$ whenever $n \notin \mathbf{fn}(\mathbf{W})$

(S-FLIP-2)

$(\nu n : U)(\nu m : W)N \equiv (\nu m : W - n)(\nu n : U + m)N \; ; \; n \in \mathbf{fn}(\mathbf{W})$

By the rule (S-NEW): if $\Gamma, m : W \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$

By the rule (S-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)(\nu m : W)N : ok$

But then, by the rule (S-NEW): if $\Gamma, n : U + m \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U + m)N : ok$

By the rule (S-NEW): if $\Gamma, m : W - n \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U + m)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W - n)(\nu n : U + m)N : ok$ whenever $n \in \mathbf{fn}(\mathbf{W})$

(S-INACT)

$(\nu n : U)N \equiv N \; ; \; n \notin \mathbf{fn}(\mathbf{N})$

By the rule (S-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$

Otherwise, we prove it using Induction Hypothesis in the structure of the network:

If $N \equiv \emptyset$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ because $n \notin fn(N) = \emptyset$

If $N \equiv (\nu m)M$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ because if $n \notin fn(N)$ then $n \notin fn(M)$ because $fn(N) = fn(M)$

If $N \equiv l[P]$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ because if $n \notin fn(N)$ then $n \notin fn(l[P])$ because $fn(N) = fn(l[P])$

If $N \equiv (M \mid M')$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ because if $n \notin fn(N)$ then $n \notin (fn(M) \cup fn(M'))$ because $fn(N) = fn((M) \cup fn(M'))$

A.2 Subject Reduction theorem

Theorem 1:

$\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$, Δ *conforms to* $\mathcal{D}$ *and* $\Delta \triangleright N \rightarrow \Delta' \triangleright N'$ *then also* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ *and* Δ*' conforms to* $\mathcal{D}$

PROOF: This theorem will be proved by induction, analyzing one by one the different semantic rules of DπF. If we ensure that the different reduction rules do not vary the *ok* typing of a network, we can affirm that a network will be *ok* without needing to analyze the situation of the network after a specific number of reductions.

A.2.1 Substitution lemma

Firstly, we have to prove the *substitution lemma* which shows that any replacement in a process P will keep it well-typed:

Lemma 6 ***(Substitution)*** *If* $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *then if* $\Gamma(v) = \Gamma(x)$ *then* $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

We need to prove it by induction in the typing rules.

(S-NIL-PROC)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0 : ok}$$

Any subsitution in a nil process is *ok*: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0\{v/_x\} : ok$.

(S-PAR)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ and $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok$.

But by (S-PAR): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} \mid Q\{v/_x\} : ok$.

Finally: $P\{v/_x\} \mid Q\{v/_x\} = (P \mid Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (P \mid Q)\{v/_x\} : ok$

(S-BANG)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P : ok}$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P\{v/_x\} : ok$.

But by (S-BANG): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P\{v/_x\} : ok$

Finally: $*u?(v : U).P\{v/_x\} = (*u?(v : U).P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (*u?(v : U).P)\{v/_x\} : ok$

(S-NEW2)

$$\frac{\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P : ok}$$

We suppose that $\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$.

Then, by Induction Hypothesis: $\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (S-NEW2): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P\{v/_x\} : ok$

Finally: $(\nu n : U)P\{v/_x\} = ((\nu n : U)P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} ((\nu n : U)P)\{v/_x\} : ok$

(S-OUT)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle y : U\rangle.P : ok}(\Gamma(u) = T_u, \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = T_y,\ \mathcal{S}(T_y) = U)$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ and all bound names in P are different from x.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (S-OUT): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle y : U\rangle.P\{v/_x\} : ok$

Finally: $u!\langle y : U\rangle.P\{v/_x\} = (u!\langle y : U\rangle.P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (u!\langle y : U\rangle.P)\{v/_x\} : ok$

(S-IN)

$$\frac{\Gamma, y : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(y : U).P : ok}(\Gamma(u) = T_u, \mathcal{S}(T_u) = ch(U),\ \Gamma(y) = T_y,\ \mathcal{S}(T_y) = U)$$

We suppose that $\Gamma, y : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ and all bound names in P are different from x.

Then, by Induction Hypothesis: $\Gamma, y : U \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (S-IN): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(y : U).P\{v/_x\} : ok$

Finally: $u?(y : U).P\{v/_x\} = (u?(y : U).P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (u?(y : U).P)\{v/_x\} : ok$

(S-GO)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \mathsf{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

We suppose that $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (S-GO): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \mathsf{go}\ k.P\{v/_x\} : ok$

Finally: $\mathsf{go}\ k.P\{v/_x\} = (\mathsf{go}\ k.P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\mathsf{go}\ k.P)\{v/_x\} : ok$

(S-NGO)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \mathsf{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \neg\mathcal{D}(T_l, T_k))$$

Any $\mathsf{go}\ k.P$ is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \mathsf{go}\ k.P\{v/_x\} : ok$ since $\neg\mathcal{D}(T_l, T_k)$ and $x \notin fn(\mathsf{go}\ k.)$.

(S-IF-THEN)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok$

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok$.

But by (S-IF-THEN): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\{v/_x\}\ \texttt{else}\ Q\{v/_x\} : ok$

Finally: $\texttt{if}\ [v = u]\ \texttt{then}\ P\{v/_x\}\ \texttt{else}\ Q\{v/_x\} = (\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q)\{v/_x\} : ok$

(S-KILL)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok}(\Gamma(l) = T_l; \mathcal{S}(T_l) = M_l; \mathcal{U}(M_l))$$

Any `kill` is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\{v/_x\} : ok$

(S-BREAK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

Any `break` k is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k\{v/_x\} : ok$

(S-PING-OK)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

We suppose that $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (S-PING-OK): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\{v/_x\}\ \texttt{else}\ Q : ok$

Finally: $\texttt{ping}\ k.P\{v/_x\}\ \texttt{else}\ Q = (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\} : ok$

(S-PING-NOT-OK)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ u.P\ \texttt{else}\ Q : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \neg\mathcal{D}(T_l, T_k))$$

We suppose that $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} Q : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok$.

But by (S-PING-OK): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q\{v/_x\} : ok$

Finally: $\texttt{ping}\ k.P\ \texttt{else}\ Q\{v/_x\} = (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\} : ok$

A.2.2 Local reduction rules

Now, we are going to prove the theorem by induction in the local reduction rules.

(R-COMM)

$$\overline{\Delta \triangleright l[a!\langle v\rangle.P] \mid l[a?(x).Q] \rightarrow \Delta \triangleright l[P] \mid l[Q\{v/x\}]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[a!\langle v\rangle.P] \mid l[a?(x).Q] : ok$. Then, by the rule (S-SITE):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle V\rangle.P \mid a?(X).Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[a!\langle V\rangle.P] \mid l[a?(X).Q] : ok}$$ by the rule (S-PAR):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle v\rangle.P : ok\ ;\ \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(x).Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle v\rangle.P \mid a?(x).Q : ok}$$

We need to prove it separately:

A) We assume $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle v\rangle.P : ok$. By the rule (S-OUT):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle V\rangle.P : ok}$$ whenever $\Gamma(a) = T_a \wedge \mathcal{S}(T_a) = ch(M_v) \wedge \Gamma(v) = T_v \wedge \mathcal{S}(T_v) = M_v$

but then: $\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$

B) We assume $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(x).Q : ok$. By the rule (S-IN):

$$\frac{\Gamma, x : U \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(x).Q : ok} \text{ whenever } \Gamma(a) = T_a \wedge \mathcal{S}(T_a) = ch(U) \wedge \Gamma(x) = T_x \wedge \mathcal{S}(T_x) = U$$

Using the previous proof of the lemma 6 (*substitution lemma*) in A.2.1:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok}{\Gamma, x : U \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok} \text{ whenever } \Gamma(v) = T_v \wedge \mathcal{S}(T_v) = U$$

Moreover: $\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q\{v/_x\}] : ok}$

We obtain the final conclusion joining both previous results :

$$\frac{\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok} \quad \dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q\{v/_x\}] : ok}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] \mid l[Q\{v/_x\}] : ok}$$

(R-REP)

$$\overline{\Delta \triangleright l[*a?(X).P] \rightarrow \Delta \triangleright l[a?(X).(P \mid *a?(X).P)]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[*a?(X).P] : ok$. Then, by the rule (S-SITE):

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *a?(X).P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[*a?(X).P] : ok}$ by the rule (S-BANG):

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(X).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *a?(X).P : ok}$ by the rule (S-IN):

$$\frac{\Gamma, x \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(X).P : ok} \text{ whenever } \Gamma(a) = T_a \wedge \mathcal{S}(T_a) = ch(U) \wedge \Gamma(x) = T_x \wedge \mathcal{S}(T_x) = U)$$

but then: $\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$

(R-FORK)

$$\overline{\Delta \triangleright l[P \mid Q] \rightarrow \Delta \triangleright l[P] \mid l[Q]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P \mid Q] : ok$. Then, by the rule (S-SITE):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} l[P \mid Q] : ok}$$ by the rule (S-PAR):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \;;\; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$$

but then:

$$\frac{\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok} \qquad \dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q] : ok}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] \mid l[Q] : ok}$$

(R-EQ)

$$\frac{}{\Delta \triangleright l[\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q] \to \Delta \triangleright l[P]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q] : ok$. Then, by the rule (S-SITE):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q] : ok}$$ by the rule (S-IF-THEN):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q : ok}$$ but then, in our case: $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$$

(R-NEQ)

$$\frac{}{\Delta \triangleright l[\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q] \to \Delta \triangleright l[P]}\, u \neq v$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q] : ok$. Then, by the rule (S-SITE):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q] : ok}$$ by the rule (S-IF-THEN):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if } [v = u] \texttt{ then } P \texttt{ else } Q : ok}$$ but then, in our case: $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q] : ok}$$

A.2.3 Network reduction rules

We need to prove all the network reduction rules by induction in order to prove the theorem 1.

(R-GO)

$$\frac{}{\Delta \triangleright l[\texttt{go}\ k.P] \rightarrow \Delta \triangleright k[P]} \Delta \vdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P] : ok$. Then, by the rule *(S-SITE)*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P] : ok}$$ by the rule *(S-GO)*:

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}$$ but then: $$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} k[P] : ok}$$ whenever $\Gamma(l) = T_1 \wedge \Gamma(k) = T_k \wedge \mathcal{D}(T_l, T_k)$

(R-NGO)

$$\frac{}{\Delta \triangleright l[\texttt{go}\ k.P] \rightarrow \Delta \triangleright k[0]} \Delta \nvdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P]] : ok$. Then, by the rule *(S-SITE)*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P] : ok}$$, but then, by the rule *(S-NGO)*:

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}$$ whenever $\Gamma(l) = T_l \wedge \Gamma(k) = T_k \wedge \neg\mathcal{D}(T_l, T_k)$

(R-PING)

$$\frac{}{\Delta \triangleright l[\texttt{ping}\ k.P\ \texttt{else}\ Q] \rightarrow \Delta \triangleright l[P]} \Delta \vdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok$. Then, by the rule *(S-SITE)*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok}$$ by the rule *(S-PING-OK)*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}$$ whenever $\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k)$

but then: $\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$

(R-NPING)

$$\frac{}{\Delta \triangleright l[\texttt{ping}\ k.P\ \texttt{else}\ Q] \rightarrow \Delta \triangleright l[Q]} \Delta \not\vdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok$. Then, by the rule *(S-SITE)*:

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok}$ by the rule *(S-PING-NOT-OK)*:

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}$ whenever $\Gamma(l) = T_l; \Gamma(k) = T_k; \neg\mathcal{D}(T_l, T_k)$

but then: $\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q] : ok}$

(R-KILL)

$$\frac{}{\Delta \triangleright l[\texttt{kill}] \rightarrow (\Delta - l) \triangleright l[0]}$$

As we know, any nil process is *ok* in any location: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

Also, $\Delta - l$ conforms to $\mathcal{D}$ because it is a smaller set.

(R-BRK)

$$\frac{}{\Delta \triangleright l[\texttt{break}\ k] \rightarrow (\Delta - l \leftrightarrow k) \triangleright l[0]} \Delta \vdash l \leftrightarrow k$$

As we know, any nil process is *ok* in any location: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

Also, $\Delta - l \leftrightarrow k$ conforms to $\mathcal{D}$ because it is a smaller set.

(R-NEW)

$$\overline{\Delta \triangleright l[(\nu c)P] \rightarrow \Delta \triangleright (\nu c)l[P]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[(\nu c)P]] : ok$. Then, by the rule *(S-SITE)*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu c)P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[(\nu c)P] : ok}$$ by the rule *(S-NEW2)*:

$$\frac{\Gamma, c \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu c)P : ok}$$ but then: $$\frac{\Gamma, c \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu c)l[P] : ok}$$

A.2.4 Contextual reduction rules

We need to prove all the contextual reduction rules by induction in order to prove the theorem 1.

(R-STR)

$$\frac{\Delta \triangleright N' \equiv \Delta \triangleright N \quad \Delta \triangleright N \rightarrow \Delta' \triangleright M \quad \Delta' \triangleright M \equiv \Delta' \triangleright M'}{\Delta \triangleright N' \rightarrow \Delta' \triangleright M'}$$

Using the proof of the structural rules in A.1.1, any structural congruence is safe *(ok)*, so:

$\Gamma \vdash^{\mathcal{D},\mathcal{S}} (N' \equiv N) : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} (M \equiv M') : ok$

Furthermore, it was proof that any reduction rule keep the *safety* property, so:

if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$

Then, it can be deduced the second part of the (R-STR) rule:

if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M' : ok$

(R-CTXT-REST)

$$\frac{\Delta + n : U \triangleright N \rightarrow \Delta' + n : W \triangleright M}{\Delta \triangleright (\nu n : U)N \rightarrow \Delta' \triangleright (\nu n : W)M}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ and Δ conforms to $\mathcal{D}$

We need to show that $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)M : ok$

By Induction Hypothesis, and using rule *(S-NEW)*:

if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} M : ok$

(R-CTXT-PAR)

$$\frac{\Delta \triangleright N \rightarrow \Delta' \triangleright N'}{\Delta \triangleright N \mid M \rightarrow \Delta' \triangleright N' \mid M} \Delta \vdash M$$

It was proof that any reduction rule keep the *ok* typing, so:

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$

By the rule (S-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

So: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' \mid M : ok$

A.3 Safety theorem

Theorem 2:

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *and* Δ *conforms to* $\mathcal{D}$ *then* $\Delta \triangleright N$ *is safe.*

PROOF:

At this moment, we have ensured that a well-typed network will be also well-typed after any semantic reduction. With the previous theorem, it will be proved that if a network is well-typed then the network will be kill-safe.

If we have the definition of the safety property taken:

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then N is safe and Δ conforms to $\mathcal{D}$

$\forall N' : \Delta \triangleright N \rightarrow^* \Delta' \triangleright N'$ and $N' = l[P] \mid N''$ and $l[P] \equiv l[\texttt{kill} \mid P']$ then $UnImp(l)$

Using the previous proved theorem 1, if we prove that the network is ***now-safe*** in a concrete situation, then the network will be *safe* in any situation.

In order to prove it, the different typing rules of the proposed type system will be analyzed

by induction, to analyze if the *ok* typing represents the kill-safety of the network. If it is made, finally we can ensure that if a network is well-typed, it will be *safe*.

A.3.1 Network typing rules

(S-NIL-NET)

$$\frac{}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} 0 : ok}$$

Any nil network will be *safe*.

(S-PAR-NET)

$$\frac{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 : ok; \Gamma \vdash^{\mathcal{D},\mathcal{S}} N_2 : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 \mid N_2 : ok}$$

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 : ok$ then N_1 does not have any `kill` in an important location, and it is safe. On the other hand, $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_2 : ok$ then N_2 does not have a `kill` in an important location either.

So, the parallel composition of them $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 \mid N_2 : ok$ will be also *safe*.

(S-SITE)

$$\frac{\Gamma \vdash_{l}^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$$

If $\Gamma \vdash_{l}^{\mathcal{D},\mathcal{S}} P : ok$, it is because, using the lemma 1 that is proved in section A.3.2, P does not have any `kill` inside or, if it has it, then l is not important.

So the location of P in l will be also *safe*.

(S-NEW)

$$\frac{\Gamma, l : U \vdash^{\mathcal{D},\mathcal{S}} N : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu l : U)N : ok}$$

If a network N is *ok*, with a location l included in the network configuration, it is because the network does not have any `kill` in an important location. Then, the network is safe.

So, the declaration of a new location l will be also *safe*.

A.3.2 Process typing rules

Analyzing the process typing rules of the type system, the lemma shown in lemma 1 have to be proved:

Lemma 1

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *and the unimportance predicate Unimp conforms to* $\mathcal{U}$ *then if* $l[P] \equiv l[\texttt{kill} \mid P']$ *then* $Unimp(l)$

In order to prove it, the different process typing rules will be proved by induction. If it is made, we can ensure that if any process is well-typed (*ok*) then it will be *safe*.

(S-NIL-PROC)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0 : ok}$$

Any nil process is *ok* and then, it will be *safe*.

(S-PAR)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ then if P includes any `kill` then $Unimp(l)$. So it is *safe*.

On the other hand, $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok$ then if Q has `kill` then $Unimp(l)$. So it is also *safe*.

Then, the parallel composition of them $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok$ will be also *safe*.

(S-BANG)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P : ok}$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok$, it is because it does not have any `kill`. This way, it will be *safe*.

So the replication of multiple $u?(v : U).P$ in l will be also *safe*.

(S-NEW2)

$$\frac{\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P : ok}$$

If a process P is ok, with a location l included in the network configuration, it is because if the process P has any `kill` then $Unimp(l)$. Then, the process P in l is *safe*.

So, the declaration of a new variable l will be also *safe*.

(S-OUT)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle v : U\rangle.P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = T_v,\ \mathcal{S}(T_v) = U)$$

If a process P is ok, with a location l included in the network configuration, it is because if the process P has any `kill` then $Unimp(l)$. Then, the process is *safe*.

This way, if an output is included, it will continue being *safe* because it does not contain any `kill`.

(S-IN)

$$\frac{\Gamma, v_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = T_v,\ \mathcal{S}(T_v) = U)$$

If a process P is ok, with a location l included in the network configuration, it is because if the process P has any `kill` then $Unimp(l)$. Then, the process is *safe*.

If an input is included, it will continue being *safe* because it does not contain any `kill`.

(S-GO)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

If a process P is ok, with a location k included in the network configuration, it is because if the process P has any `kill` then $Unimp(k)$. Then, the process is *safe*.

If a `go` $k.P$ is included, it will continue being *safe*, being the types of the locations l and k related under $\mathcal{D}$, because it does not contain any `kill`.

(S-NGO)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \neg\mathcal{D}(T_l, T_k))$$

It is *safe* since the types of the locations l and k are not related under $\mathcal{D}$.

(S-IF-THEN)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` then $Unimp(l)$. Then, the process is *safe*.

Moreover, if a process Q is *ok*, with a location l included in the network configuration, it is because if the process Q has any `kill` then $Unimp(l)$. Then, the process is *safe*.

So if a matching is ”true”, it will be also *safe*, because the process will continue being the same P. Otherwise, if a matching is ”false”, it will be also *safe*, because the process will continue being the same Q.

(S-KILL)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok}(\Gamma(l) = T_l; \mathcal{S}(T_l) = M_l; \mathcal{U}(M_l))$$

If a $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}}$ `kill` : *ok*, it is because the `kill` is placed in a location l and $Unimp(l)$, i.e. the type of the location is not important ($\mathcal{U}(\mathcal{S}(\Gamma(l)))$).

Then, it is *safe*.

(S-BREAK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

If a `break` k is *safe* it is because it does not have any `kill`, whenever the types of the locations l and k were related under $\mathcal{D}$.

(S-PING-OK)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{D}(T_l, T_k))$$

We know that process P is ok with a location k included in the network configuration, and it is because if the process P has any `kill` then $Unimp(k)$. Then, the process is *safe*.

If a `ping` $k.P$ `else` Q is included, it will continue being *safe* whether the type of the locations l and k were related under $\mathcal{D}$, because the process will be the same than the commented previously, i.e. P located in k.

(S-PING-NOT-OK)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \neg\mathcal{D}(T_l, T_k))$$

We know that a process Q is ok with a location l included in the network configuration, and it is because if the process Q has any `kill` then $Unimp(l)$. Then, the process is *safe*.

If a `ping` $k.P$ `else` Q is included, it will continue being *safe* whether the type of the locations l and k were not related under $\mathcal{D}$, because the process will be the same than the commented previously, i.e. Q located in l.

Appendix B

Proof of typing rules: Migration safety

We need to prove in this appendix that the type system with the new modification presented in chapter 4 continues being correctly defined. So, the two theorems and the lemmas of the previous appendix are proved in this one with the new type system modification.

The main difference between the previous type system and the new one is the presence of two new semantic rules with their typing rules. They are used to represent the murder actions from a location to another one.

In the previous appendix we proved that the type system was correctly defined, since the theorems and the lemmas were respected. Then, in this appendix we have to analyze if the new characteristics of our modification of DπF and the new type system mantain this proving. *Subject equivalence lemma* was proved in appendix A and the new type system does not do any modification in structural equivalence rules, so the proving will be the same that the explained in appendix A.

B.1 Subject Reduction theorem

Theorem 1:

$\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$, Δ *conforms to* $\mathcal{D}$ *and and* $\Delta \triangleright N \rightarrow \Delta' \triangleright N'$ *then also* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ *and* Δ' *conforms to* $\mathcal{D}$

PROOF: This theorem will be proved by induction. However, we will only prove the new semantic rules, since the other ones were proved in appendix A. If we prove that the new two local reduction rules mantain the ”well-typing” of a network, the type system will continue satisfying the *Subject Reduction theorem*.

B.1.1 Substitution lemma

Firstly, we have to prove the *substitution lemma* presented in A.2.1 for the new typing rules:

Lemma 6: *If* $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *then if* $\Gamma(v) = \Gamma(x)$ *then* $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

(S-MURDER-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{S}(T_k) = M_k; \mathcal{S}(T_l) = M_l; \mathcal{D}(T_l, T_k); M_k \sqsubseteq M_l)$$

Any `kill` k is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k\{v/_x\} : ok$ since $x \notin fn(\texttt{kill}\ k)$.

(S-MURDER-NOT-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{D}(T_l, T_k)$$

Any `kill` k is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k\{v/_x\} : ok$ since $x \notin fn(\texttt{kill}\ k)$.

B.1.2 Added local reduction rules

(R-KILL-LOC)

$$\frac{}{\Delta \triangleright l[\![\texttt{kill}\ k]\!] \rightarrow (\Delta - k) \triangleright l[\![0]\!]}\Delta \vdash k \leftarrow l$$

As we know, any nil process is ok in any location: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

Also, $\Delta - k$ conforms to $\mathcal{D}$ because it is a smaller set.

(R-NKILL-LOC)

$$\frac{}{\Delta \triangleright l[\![\texttt{kill}\ k]\!] \rightarrow \Delta \triangleright l[\![0]\!]} \Delta \nvdash k \leftarrow l$$

As we know, any nil process is *ok* in any location: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

Δ does not vary, so the network will continue being *ok*.

B.2 Safety theorem

Theorem 2:

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *and* Δ *conforms to* $\mathcal{D}$ *then* $\Delta \triangleright N$ *is safe.*

PROOF: This theorem was proved with the previously proposed typing rules of the type system, and they do not vary, so we have to prove that the two new typing rules will continue setting a network as *"ok"* and this situation represents the safety of the network. This way, *Safety theorem* will be also satisfied by our type system. The proving method will be also induction.

B.2.1 Added process typing rules

Analyzing the added process typing rules of the type system, we need to prove the lemma 1 shown in appendix A, that continues working by using the unimportance predicate, with these new rules:

Lemma 1

If $\Gamma \vdash_{l}^{\mathcal{D},\mathcal{S}} P : ok$ *and the unimportance predicate Unimp conforms to* $\mathcal{U}$ *then if* $l[P] \equiv l[\texttt{kill} \mid P']$ *then* $Unimp(l)$

In order to prove it, the new process typing rules will be analyzed by induction. If it is made, we can ensure that if any process is well-typed (*ok*) then it will be *safe*.

(S-MURDER-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{S}(T_k) = M_k; \mathcal{S}(T_l) = M_l; \mathcal{D}(T_l, T_k); M_k \sqsubseteq M_l)$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok$, it is because it does not have any `kill`. This way, it will be *safe*.

(S-MURDER-NOT-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \neg\mathcal{D}(T_l, T_k))$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok$, it is because it does not have any `kill` and the locations are not connected. This way, it will be *safe*.

B.2.2 Process typing rules

As we added a new lemma 3 for the two new process typing rules, we have to prove it with all the typing rules of our type system:

Lemma 3

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *then if* $l[P] \equiv l[\texttt{kill}\ k \mid P']$ *then* $\Gamma(l) = T_l$,$\Gamma(k) = T_k$,$\mathcal{S}(T_l) = M_l$,$\mathcal{S}(T_k) = M_k$ *and* $M_k \sqsubseteq M_l$

In order to prove it, the different process typing rules will be analyzed by induction. If we make that, we can ensure that if any process is well-typed (*ok*) then it will be *safe*.

(S-NIL-PROC)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0 : ok}$$

Any nil process is *ok* and then, it will be *safe*.

(S-PAR)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ then if P includes any `kill` k then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) = M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. So it is *safe*.

On the other hand, $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok$ then if Q has `kill` k then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) = M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. So it is also *safe*.

Then, the parallel composition of them $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok$ will be also *safe*.

(S-BANG)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P : ok}$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok$, it is because it does not have any `kill` k. This way, it will be *safe*.

So the replication of multiple $u?(v : U).P$ in l will be also *safe*.

(S-NEW2)

$$\frac{\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P : ok}$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(l) = T_l$,$\Gamma(k) = T_k$,$\mathcal{S}(T_l) = M_l$,$\mathcal{S}(T_k) = M_k$ and $M_k \sqsubseteq M_l$. Then, the process P in l is *safe*.

So, the declaration of a new variable l will be also *safe*.

(S-OUT)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle v : U\rangle.P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = T_v,\ \mathcal{S}(T_v) = U)$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(l) = T_l$,$\Gamma(k) = T_k$,$\mathcal{S}(T_l) = M_l$,$\mathcal{S}(T_k) = M_k$ and $M_k \sqsubseteq M_l$. Then, the process is *safe*.

This way, if an output is included, it will continue being *safe* because it does not contain any `kill`.

(S-IN)

$$\frac{\Gamma, v_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = T_v,\ \mathcal{S}(T_v) = U)$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) = M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. Then, the process is *safe*.

If an input is included, it will continue being *safe* because it does not contain any `kill`.

(S-GO)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

If a process P is *ok*, with a location k included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) = M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. Then, the process is *safe*.

If a `go` $k.P$ is included, it will continue being *safe* whether the type of the locations l and k were related under $\mathcal{D}$ because it does not contain any `kill`.

(S-NGO)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \neg\mathcal{D}(T_l, T_k))$$

It is always *safe* since the types of the locations l and k are not related under $\mathcal{D}$.

(S-IF-THEN)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) = M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. Then, the process is *safe*.

Moreover, if a process Q is *ok*, with a location l included in the network configuration, it is because if the process Q has any `kill` k then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) =$

$M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. Then, the process is *safe*.

So if a matching is "true", it will be also *safe*, because the process will continue being the same P. Otherwise, if a matching is "false", it will be also *safe*, because the process will continue being the same Q.

(S-KILL)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok}(\Gamma(l) = T_l; \mathcal{S}(T_l) = M_l \mathcal{U}(M_l))$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok$, it is because it does not have any `kill` k . Then, it is *safe*.

(S-BREAK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

If a `break` k is *safe* it is because it does not have any `kill` k, whenever the types of the locations l and k were related under $\mathcal{D}$.

(S-PING-OK)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

We know that process P is *ok* with a location k included in the network configuration, and it is because if the process P has any `kill` m then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) = M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. Then, the process is *safe*.

If a `ping` $k.P$ `else` Q is included, it will continue being *safe* whether the type of the locations l and k were related under $\mathcal{D}$, because the process will be the same than the commented previously, i.e. P located in k.

(S-PING-NOT-OK)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \neg\mathcal{D}(T_l, T_k))$$

We know that a process Q is *ok* with a location l included in the network configuration, and

it is because if the process Q has any `kill` k then $\Gamma(k) = T_k, \Gamma(l) = T_l, \mathcal{S}(T_k) = M_k, \mathcal{S}(T_l) = M_l, \mathcal{D}(T_l, T_k)$ and $M_k \sqsubseteq M_l$. Then, the process is *safe*.

If a `ping` $k.P$ `else` Q is included, it will continue being *safe* whether the type of the locations l and k were not related under $\mathcal{D}$, because the process will be the same than the commented previously, i.e. Q located in l.

(S-MURDER-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{S}(T_k) = M_k; \mathcal{S}(T_l) = M_l; \mathcal{D}(T_l, T_k); M_k \sqsubseteq M_l)$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}}$ `kill` $k : ok$, it is because the priority of location k (M_k) is lower than the priority of l (M_l) whenever the locations are connected ($\mathcal{D}(T_l, T_k)$). So `kill` k can be sent from l to k. This way, it will be *safe* according to the safety definition 25.

(S-MURDER-NOT-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \neg\mathcal{D}(T_l, T_k))$$

It will be ”ok” because the locations are not connected, and the killing of location k is not finally done. Then it will be always *safe*.

Appendix C

Proof of typing rules: Indirect murders safety

The aim of this part of the report is to prove the theorems presented in chapter 5. In order to do that, we have to analyze the semantic rules of our modified DπF and the typing rules of the new type system.

C.1 Subject equivalence lemma

Analyzing the structural rules of our modified DπF syntax, the lemma shown in lemma 5 have to be proved:

Lemma 5:

If $N \equiv N' ok$ *and* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *then* $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$

In order to prove it, the different structural rules of DπF will be proved by induction. If it is made, we can ensure that an equivalent network of a specific network will be also *ok*.

C.1.1 Structural rules

(S-COMM)

$N \mid M \equiv M \mid N$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

But then, by the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M \mid N : ok$

(S-ASSOC)

$(N \mid M) \mid M' \equiv N \mid (M \mid M')$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (N \mid M) : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M' : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (N \mid M) \mid M' : ok$

But then, by the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M' : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M \mid M' : ok$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} (M \mid M') : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid (M \mid M') : ok$

(S-UNIT)

$N \mid l[0] \equiv N$

By the rule (M-NIL-NET): $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid l[0] : ok$

But then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$

(S-EXTR)

$(\nu n : U)(N \mid M) \equiv N \mid (\nu n : U)M \ ; \ n \notin \mathbf{fn}(\mathbf{N})$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

By the rule (M-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (N \mid M) : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)(N \mid M) : ok$

But then, by the rule (M-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)M : ok$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid (\nu n : U)M : ok$ whenever $n \notin \mathbf{fn}(\mathbf{N})$

(S-FLIP-1)

$(\nu n : U)(\nu m : W)N \equiv (\nu m : W)(\nu n : U)N \;;\; n \notin \mathbf{fn}(\mathbf{W})$

By the rule (M-NEW): if $\Gamma, m : W \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$

By the rule (M-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)(\nu m : W)N : ok$

But then, by the rule (M-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$

By the rule (M-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)(\nu n : U)N : ok$ whenever $n \notin \mathbf{fn}(\mathbf{W})$

(S-FLIP-2)

$(\nu n : U)(\nu m : W)N \equiv (\nu m : W - n)(\nu n : U + m)N \;;\; n \in \mathbf{fn}(\mathbf{W})$

By the rule (M-NEW): if $\Gamma, m : W \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$

By the rule (M-NEW): if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)(\nu m : W)N : ok$

But then, by the rule (M-NEW): if $\Gamma, n : U + m \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U + m)N : ok$

By the rule (M-NEW): if $\Gamma, m : W - n \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U + m)N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu m : W - n)(\nu n : U + m)N : ok$ whenever $n \in \mathbf{fn}(\mathbf{W})$

(S-INACT)

$(\nu n : U)N \equiv N \;;\; n \notin \mathbf{fn}(\mathbf{N})$

By the rule (M-NEW): if $\Gamma, n : U \vdash^{\mathcal{D}} N : ok$ then $\Gamma \vdash^{\mathcal{D}} (\nu n : U)N : ok$

Otherwise, we prove it using Induction Hypothesis in the structure of the network:

If $N \equiv \emptyset$ then $\Gamma \vdash^{\mathcal{D}} (\nu n : U)N : ok$ because $n \notin fn(N) = \emptyset$

If $N \equiv (\nu m)M$ then $\Gamma \vdash^{\mathcal{D}} (\nu n : U)N : ok$ because if $n \notin fn(N)$ then $n \notin fn(M)$ because $fn(N) = fn(M)$

If $N \equiv l[P]$ then $\Gamma \vdash^{\mathcal{D}} (\nu n : U)N : ok$ because if $n \notin fn(N)$ then $n \notin fn(l[P])$ because $fn(N) = fn(l[P])$

If $N \equiv (M \mid M')$ then $\Gamma \vdash^{\mathcal{D}} (\nu n : U)N : ok$ because if $n \notin fn(N)$ then $n \notin (fn(M) \cup fn(M'))$ because $fn(N) = fn((M) \cup fn(M'))$

C.2 Subject Reduction theorem

Theorem 3:

$\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$, Δ conforms to $\mathcal{D}$ and $\Delta \triangleright N \rightarrow \Delta' \triangleright N'$ then also $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ and Δ' conforms to $\mathcal{D}$

PROOF: This theorem will be also proved by induction, analyzing one by one the different semantic rules of DπF with the new murder process construct commented in chapter 5. If we ensure that the different reduction rules do not vary the *ok* typing of a network, we can affirm that a network will be *ok* without needing to analyze the situation of the network after a specific number of reductions.

C.2.1 Substitution lemma

Firstly, we have to prove the *substitution lemma* presented in A.2.1:

Lemma 6:

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ then if $\Gamma(v) = \Gamma(x)$ then $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

Although this lemma was already proved for the other type system, we need to prove it now for the new type system because the typing rules are different.

(M-NIL-PROC)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0 : ok}$$

Any subsitution in a nil process is *ok*: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0\{v/_x\} : ok$.

(M-PAR)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ and $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok$.

But by (M-PAR): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} \mid Q\{v/_x\} : ok$.

Finally: $P\{v/_x\} \mid Q\{v/_x\} = (P \mid Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (P \mid Q)\{v/_x\} : ok$

(M-BANG)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P : ok}$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P\{v/_x\} : ok$.

But by (M-BANG): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P\{v/_x\} : ok$

Finally: $*u?(v : U).P\{v/_x\} = (*u?(v : U).P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (*u?(v : U).P)\{v/_x\} : ok$

(M-NEW2)

$$\frac{\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P : ok}$$

We suppose that $\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$.

Then, by Induction Hypothesis: $\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (M-NEW2): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P\{v/_x\} : ok$

Finally: $(\nu n : U)P\{v/_x\} = ((\nu n : U)P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} ((\nu n : U)P)\{v/_x\} : ok$

(M-OUT)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle y : U\rangle.P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(y) = T_y,\ \mathcal{S}(T_y) = U)$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ and all bound names in P are different from x.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (M-OUT): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle y : U\rangle.P\{v/_x\} : ok$

Finally: $u!\langle y : U\rangle.P\{v/_x\} = (u!\langle y : U\rangle.P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (u!\langle y : U\rangle.P)\{v/_x\} : ok$

(M-IN)

$$\frac{\Gamma, v_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(y : U).P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(y) = T_y,\ \mathcal{S}(T_y) = U)$$

We suppose that $\Gamma, y : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ and all bound names in P are different from x.

Then, by Induction Hypothesis: $\Gamma, y : U \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (M-IN): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(y : U).P\{v/_x\} : ok$

Finally: $u?(y : U).P\{v/_x\} = (u?(y : U).P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (u?(y : U).P)\{v/_x\} : ok$

(M-GO)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \mathsf{go}\ k.P : ok}(\Gamma(l) = T_l; \mathcal{S}(T_l) = loc(\widetilde{K}); \Gamma(k) = T_k; \mathcal{S}(T_k) = loc(\widetilde{K'}); \mathcal{D}(T_l, T_k); \widetilde{K'} \subseteq \widetilde{K})$$

We suppose that $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (M-GO): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \mathsf{go}\ k.P\{v/_x\} : ok$

Finally: $\mathsf{go}\ k.P\{v/_x\} = (\mathsf{go}\ k.P)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\mathsf{go}\ k.P)\{v/_x\} : ok$

(M-NGO)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{S}(T_k) = loc(\widetilde{K'}); \neg\mathcal{D}(T_l, T_k))$$

Any `go` $k.P$ is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P\{v/_x\} : ok$ since $\neg\mathcal{D}(T_l, T_k)$ and $x \notin fn(\texttt{go}\ k.)$.

(M-IF-THEN)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$

We suppose that $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok$

Then, by Induction Hypothesis: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok$.

But by (M-IF-THEN): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\{v/_x\}\ \texttt{else}\ Q\{v/_x\} : ok$

Finally: $\texttt{if}\ [v = u]\ \texttt{then}\ P\{v/_x\}\ \texttt{else}\ Q\{v/_x\} = (\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q)\{v/_x\} : ok$

(M-KILL)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok}(\Gamma(l) = T; \mathcal{S}(T_l) = loc(\widetilde{K}); T \in \widetilde{K})$$

Any `kill` is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\{v/_x\} : ok$

(M-BREAK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

Any `break` k is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k\{v/_x\} : ok$

(M-PING-OK)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{D}(T_l, T_k))$$

We suppose that $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P\{v/_x\} : ok$.

But by (M-PING-OK): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\{v/_x\}\ \texttt{else}\ Q : ok$

Finally: $\texttt{ping}\ k.P\{v/_x\}\ \texttt{else}\ Q = (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\} : ok$

(M-PING-NOT-OK)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ u.P\ \texttt{else}\ Q : ok}(\Gamma(u) = T_u; \Gamma(l) = T_l; \neg\mathcal{D}(T_l, T_u))$$

We suppose that $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} Q : ok$.

Then, by Induction Hypothesis: $\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok$.

But by (M-PING-NOT-OK): $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q\{v/_x\} : ok$

Finally: $\texttt{ping}\ k.P\ \texttt{else}\ Q\{v/_x\} = (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\}$, so: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\texttt{ping}\ k.P\ \texttt{else}\ Q)\{v/_x\} : ok$

(M-MURDER-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{S}(T_l) = loc(\widetilde{K}); \mathcal{D}(T_l, T_k); T_k \in \widetilde{K})$$

Any $\texttt{kill}\ k$ is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k\{v/_x\} : ok$

(M-MURDER-NOT-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \neg\mathcal{D}(T_k, T_l))$$

Any $\texttt{kill}\ k$ is ok with any substitution: $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k\{v/_x\} : ok$

C.2.2 Local reductions rules

Now, we are going to prove the theorem by induction in the local reduction rules.

(R-COMM)

$$\frac{}{\Delta \triangleright l[a!\langle v\rangle.P] \mid l[a?(x).Q] \rightarrow \Delta \triangleright l[P] \mid l[Q\{v/x\}]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[a!\langle v\rangle.P] \mid l[a?(x).Q] : ok$. Then, by the rule (M-SITE):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle V\rangle.P \mid a?(X).Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[a!\langle V\rangle.P] \mid l[a?(X).Q] : ok}$$ by the rule (M-PAR):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle v\rangle.P : ok \;;\; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(x).Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle v\rangle.P \mid a?(x).Q : ok}$$

We need to prove it separately:

A) We assume $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle V\rangle.P : ok$. By the rule (M-OUT):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a!\langle V\rangle.P : ok}$$ whenever $\Gamma(a) = T_a \wedge \mathcal{S}(T_a) = ch(M_v) \wedge \Gamma(v) = T_v \wedge \mathcal{S}(T_v) = M_v)$

but then: $\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$

B) We assume $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(x).Q : ok$. By the rule (M-IN):

$$\frac{\Gamma, x : U \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(x).Q : ok}$$ whenever $\Gamma(a) = T_a \wedge \mathcal{S}(T_a) = ch(U) \wedge \Gamma(x) = T_x \wedge \mathcal{S}(T_x) = U)$

Using the previous proof of the *substitution lemma*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok}{\Gamma, x : U \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}$$ whenever $\Gamma(v) = U$

Moreover: $\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q\{v/_x\}] : ok}$

We obtain the final conclusion joining both previous results :

$$\frac{\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok} \quad \dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q\{v/_x\} : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q\{v/_x\}] : ok}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] \mid l[Q\{v/_x\}] : ok}$$

(R-REP)

$$\frac{}{\Delta \triangleright l[*a?(X).P] \rightarrow \Delta \triangleright l[a?(X).(P \mid *a?(X).P)]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[*a?(X).P] : ok$. Then, by the rule (M-SITE):

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *a?(X).P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[*a?(X).P] : ok}$ by the rule (M-BANG):

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(X).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *a?(X).P : ok}$ by the rule (M-IN):

$\dfrac{\Gamma, x \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} a?(X).P : ok}$ whenever $\Gamma(a) = T_a \wedge \mathcal{S}(T_a) = ch(U) \wedge \Gamma(x) = T_x \wedge \mathcal{S}(T_x) = U)$

but then: $\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$

(R-FORK)

$$\frac{}{\Delta \triangleright l[P \mid Q] \rightarrow \Delta \triangleright l[P] \mid l[Q]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P \mid Q] : ok$. Then, by the rule (M-SITE):

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} l[P \mid Q] : ok}$ by the rule (M-PAR):

$\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \ ; \ \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$ but then:

$$\frac{\dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok} \quad \dfrac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q] : ok}}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] \mid l[Q] : ok}$$

(R-EQ)

$$\frac{}{\Delta \triangleright l[\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q] \rightarrow \Delta \triangleright l[P]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q] : ok$. Then, by the rule (M-SITE):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q] : ok}$$ by the rule (M-IF-THEN):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$ but then, in our case: $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$$

(R-NEQ)

$$\frac{}{\Delta \triangleright l[\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q] \rightarrow \Delta \triangleright l[P]}\, u \neq v$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q] : ok$. Then, by the rule (M-SITE):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q] : ok}$$ by the rule (M-IF-THEN):

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok \wedge \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$ but then, in our case: $$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q] : ok}$$

C.2.3 Network reduction rules

We need to prove all the network reduction rules by induction in order to prove the theorem 3.

(R-GO)

$$\frac{}{\Delta \triangleright l[\texttt{go}\ k.P] \rightarrow \Delta \triangleright k[P]}\, \Delta \vdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P] : ok$. Then, by the rule *(M-SITE)*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P] : ok}$$ by the rule *(M-GO)*:

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}$$ but then: $$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} k[P] : ok}$$

whenever $\Gamma(l) = T_l \wedge \mathcal{S}(T_l) = loc(\widetilde{K}) \wedge \Gamma(k) = T_k \wedge \mathcal{S}(T_k) = loc(\widetilde{K'}) \wedge \widetilde{K'} \subseteq \widetilde{K} \wedge \mathcal{D}(T_l, T_k)$

(R-NGO)

$$\frac{}{\Delta \triangleright l[\texttt{go}\ k.P] \rightarrow \Delta \triangleright k[0]} \Delta \nvdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P] : ok$. Then, by the rule *(M-SITE)*:

$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{go}\ k.P] : ok}$, but then, by the rule *(M-NGO)*:

$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}$ whenever $\Gamma(l) = T_l \wedge \Gamma(k) = T_k \wedge \neg\mathcal{D}(T_l, T_k)$

(R-PING)

$$\frac{}{\Delta \triangleright l[\texttt{ping}\ k.P\ \texttt{else}\ Q] \rightarrow \Delta \triangleright l[P]} \Delta \vdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok$. Then, by the rule *(M-SITE)*:

$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok}$ by the rule *(M-PING-OK)*:

$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}$ whenever $\Gamma(l) = T_l \wedge \Gamma(k) = T_k \wedge \mathcal{D}(T_l, T_k)$

but then: $\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$

(R-NPING)

$$\frac{}{\Delta \triangleright l[\texttt{ping}\ k.P\ \texttt{else}\ Q] \rightarrow \Delta \triangleright l[Q]} \Delta \nvdash k \leftarrow l$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok$. Then, by the rule *(M-SITE)*:

$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[\texttt{ping}\ k.P\ \texttt{else}\ Q] : ok}$ by the rule *(M-PING-NOT-OK)*:

$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}$ whenever $\Gamma(l) = T_l \wedge \Gamma(k) = T_k \wedge \neg\mathcal{D}(T_l, T_k)$

but then: $\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[Q] : ok}$

(R-KILL)

$$\frac{}{\Delta \triangleright l[\texttt{kill}] \rightarrow (\Delta - l) \triangleright l[0]}$$

As we know, any nil process is *ok* in any location: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

Also, $\Delta - l$ conforms to Δ because it is a smaller set.

(R-BRK)

$$\frac{}{\Delta \triangleright l[\texttt{break}\ k] \rightarrow (\Delta - l \leftrightarrow k) \triangleright l[0]} \Delta \vdash l \leftrightarrow k$$

As we know, any nil process is *ok* in any location: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[0] : ok$

Also, $\Delta - l \leftrightarrow k$ conforms to Δ because it is a smaller set.

(R-NEW)

$$\frac{}{\Delta \triangleright l[(\nu c)P] \rightarrow \Delta \triangleright (\nu c)l[P]}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[(\nu c)P]] : ok$. Then, by the rule *(M-SITE)*:

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu c)P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[(\nu c)P] : ok}$$ by the rule *(M-NEW2)*:

$$\frac{\Gamma, c \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu c)P : ok}$$ but then: $$\frac{\Gamma, c \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu c)l[P] : ok}$$

C.2.4 Contextual reduction rules

We need to prove all the contextual reduction rules by induction in order to prove the theorem 3.

(R-STR)

$$\frac{\Delta \triangleright N' \equiv \Delta \triangleright N \quad \Delta \triangleright N \rightarrow \Delta' \triangleright M \quad \Delta' \triangleright M \equiv \Delta' \triangleright M'}{\Delta \triangleright N' \rightarrow \Delta' \triangleright M'}$$

Using the proof of the structural rules in C.1.1, any structural congruence is safe *(ok)*, so:

$\Gamma \vdash^{\mathcal{D},\mathcal{S}} (N' \equiv N) : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} (M \equiv M') : ok$

Furthermore, it was proof that any reduction rule keep the *safety* property, so:

if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$

Then, it can be deduced the second part of the (R-STR) rule:

if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} M' : ok$

(R-CTXT-REST)

$$\frac{\Delta + n : U \triangleright N \rightarrow \Delta' + n : W \triangleright M}{\Delta \triangleright (\nu n : U)N \rightarrow \Delta' \triangleright (\nu n : W)M}$$

We assume $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)N : ok$ and Δ conforms to $\mathcal{D}$

We need to show that $\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu n : U)M : ok$

By Induction Hypothesis, and using rule *(M-NEW)*:

if $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma, n : U \vdash^{\mathcal{D},\mathcal{S}} M : ok$

(R-CTXT-PAR)

$$\frac{\Delta \triangleright N \rightarrow \Delta' \triangleright N'}{\Delta \triangleright N \mid M \rightarrow \Delta' \triangleright N' \mid M} \Delta \vdash M$$

It was proof that any reduction rule keep the *ok* typing, so:

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' : ok$

By the rule (M-PAR-NET): if $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok \wedge \Gamma \vdash^{\mathcal{D},\mathcal{S}} M : ok$ then $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N \mid M : ok$

So: $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N' \mid M : ok$

C.3 Safety theorem

Theorem 4:

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N : ok$ *and* Δ *conforms to* $\mathcal{D}$ *then* $\Delta \triangleright N$ *is safe.*

PROOF:

At this moment, we have ensured that a network well-typed will be also well-typed after any semantic reduction. Using the previous theorem we will prove that is a network is well-typed, the network will be kill-safe. In order to do that, the different typing rules will be analyzed, by induction.

If we have the definition of the safety property taken:

$l_{\mathcal{K}}$ *is the set of location types that can be killed by location* l

T_k *is the type of location* k

$k_{\mathcal{K}}$ *is the set of location types that can be killed by location* k

A network N is *now-safe* if whenever $N \equiv l[P] \mid N'$ then
if $l[P] \equiv l[\texttt{kill}\ k \mid P']$ then $T_k \in l_{\mathcal{K}}$
else if $l[P] \equiv l[\texttt{go}\ k.Q \mid P'']$ then $k_{\mathcal{K}} \subseteq l_{\mathcal{K}}$.

Using the previous proved theorem 3, if we prove that the network is ***now-safe*** in a concrete situation, then the network will be *safe* in any situation.

In order to prove it, the different typing rules of the proposed type system will be analyzed by induction, to analyze if the *ok* typing represents the migration-safety and the kill-safety of the network. If it is made, finally we can ensure that if a network is well-typed, it will be *safe*.

C.3.1 Network typing rules

(M-NIL-NET)

$$\overline{\Gamma \vdash^{\mathcal{D},\mathcal{S}} 0 : ok}$$

Any nil network will be *now-safe*.

(M-PAR-NET)

$$\frac{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 : ok; \Gamma \vdash^{\mathcal{D},\mathcal{S}} N_2 : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 \mid N_2 : ok}$$

If $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 : ok$ then all the murders in N_1 respect the safety property, so it is now-safe. On the other hand, $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_2 : ok$ then all the murders in N_2 also respect the safety property.

So, the parallel composition of them $\Gamma \vdash^{\mathcal{D},\mathcal{S}} N_1 \mid N_2 : ok$ will be also *now-safe*.

(M-SITE)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} l[P] : ok}$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$, it is because, using the lemma 4 that is proved in section C.3.2, P is *now-safe*.

So the location of P in l will be also *now-safe*.

(M-NEW)

$$\frac{\Gamma, l : U \vdash^{\mathcal{D},\mathcal{S}} N : ok}{\Gamma \vdash^{\mathcal{D},\mathcal{S}} (\nu l : U)N : ok}$$

If a network N is *ok*, with a location l included in the network configuration, it is because all the murders in the network respect the safety property. Then, the network is now-safe.

So, the declaration of a new location l will be also *now-safe*.

C.3.2 Process typing rules

Analyzing the process typing rules of the type system, the lemma shown in lemma 4 have to be proved:

Lemma 4

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ *where* $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$ *then*

if $l[P] \equiv l[\texttt{kill}\ k \mid P']$ *where* $\Gamma(k) = T_k$ *then* $T_k \in \widetilde{K}$

else if $l[P] \equiv l[\texttt{go}\ k.Q \mid P'']$ *where* $\Gamma(k) = T_k$, $\mathcal{S}(T_k) = loc(\widetilde{K'})$ *then* $\widetilde{K'} \subseteq \widetilde{K}$

In order to prove it, the different process typing rules will be analyzed by induction. If it is made, we can ensure that if any process is well-typed (*ok*) then it will be *safe*.

(M-NIL-PROC)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} 0 : ok}$$

Any nil process is *ok* and then, it will be *now-safe*.

(M-PAR)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok}$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok$ then if P includes any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T_k$ and $T_k \in \widetilde{K}$. So it is *now-safe*.

On the other hand, $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok$ then if Q has any `kill` k' then $\Gamma(k') = T_2$ and $T_2 \in \widetilde{K}$. So it is also *now-safe*.

Then, the parallel composition of them $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P \mid Q : ok$ will be also *now-safe*.

(M-BANG)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} *u?(v : U).P : ok}$$

If $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok$, it is because it does not have any murder. This way, it will be *now-safe*.

So the replication of multiple $u?(v : U).P$ in l will be also *now-safe*.

(M-NEW2)

$$\frac{\Gamma, n_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} (\nu n : U)P : ok}$$

If a process P is *ok*, with a location l included in the network configuration, it is because if

the process P has any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$. Then, the process P in l is *now-safe*.

So, the declaration of a new variable l will be also *now-safe*.

(M-OUT)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u!\langle v : U\rangle.P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = T_v,\ \mathcal{S}(T_v) = U)$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$. Then, the process is *now-safe*.

This way, if an output is included, it will continue being *now-safe* because it does not contain any `kill` k.

(M-IN)

$$\frac{\Gamma, v_l : U \vdash_l^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} u?(v : U).P : ok}(\Gamma(u) = T_u,\ \mathcal{S}(T_u) = ch(U),\ \Gamma(v) = T_v,\ \mathcal{S}(T_v) = U)$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$. Then, the process is *now-safe*.

If an input is included, it will continue being *now-safe* because it does not contain any `kill` k.

(M-GO)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \mathtt{go}\ k.P : ok}(\Gamma(l) = T_l; \mathcal{S}(T_l) = loc(\widetilde{K}); \Gamma(k) = T_k; \mathcal{S}(T_k) = loc(\widetilde{K'}); \mathcal{D}(T_l, T_k); \widetilde{K'} \subseteq \widetilde{K})$$

If a process P is *ok*, with a location k included in the network configuration, it is because if the process P has any `kill` k' then $\Gamma(k) = T_2$,$\mathcal{S}(T_2) = loc(\widetilde{K'})$, $\Gamma(k') = T_3$ and $T_3 \in \widetilde{K'}$. Then, the process is *now-safe*.

If a `go` $k.P$ is included, it will continue being *now-safe* whether the type of the locations l and k were related under $\mathcal{D}$ and $\widetilde{K'} \subseteq \widetilde{K}$ with $\Gamma(l) = T_l$ and $\mathcal{S}(T_l) = loc(\widetilde{K})$.

(M-NGO)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{go}\ k.P : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \neg\mathcal{D}(T_l, T_k))$$

It is always *now-safe* since the types of the locations l and k are not related under $\mathcal{D}$.

(M-IF-THEN)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} P : ok; \Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{if}\ [v = u]\ \texttt{then}\ P\ \texttt{else}\ Q : ok}$$

If a process P is *ok*, with a location l included in the network configuration, it is because if the process P has any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$. Then, the process is *now-safe*.

Moreover, if a process Q is *ok*, with a location l included in the network configuration, it is because if the process Q has any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$. Then, the process is *now-safe*.

So if a matching is ”true”, it will be also *now-safe*, because the process will continue being the same P. Otherwise, if a matching is ”false”, it will be also *now-safe*, because the process will continue being the same Q.

(M-KILL)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill} : ok}(\Gamma(l) = T; \mathcal{S}(T_l) = loc(\widetilde{K}); T \in \widetilde{K})$$

If a $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}}$ `kill` $: ok$, it is because the `kill` is placed in a location l with $\Gamma(l) = T$, $\mathcal{S}(T_l) = loc(\widetilde{K})$ and $T \in \widetilde{K}$, i.e. the type of the location l is inside the set of possible location types that l could kill. Then, it is *now-safe*.

(M-BREAK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{break}\ k : ok}(\Gamma(l) = T_l; \Gamma(k) = T_k; \mathcal{D}(T_l, T_k))$$

If a `break` k is *now-safe* it is because it does not have any `kill` k, whenever the types of the locations l and k were related under $\mathcal{D}$.

(M-PING-OK)

$$\frac{\Gamma \vdash_k^{\mathcal{D},\mathcal{S}} P : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ k.P\ \texttt{else}\ Q : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \mathcal{D}(T_l, T_k))$$

We know that process P is *ok* with a location l included in the network configuration, and it is because if the process P has any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$. Then, the process is *now-safe*.

If a `ping` $k.P$ `else` Q is included, it will continue being *now-safe* whether the type of the locations l and k were related under $\mathcal{D}$, because the process will be the same than the commented previously, i.e. P located in k.

(M-PING-NOT-OK)

$$\frac{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} Q : ok}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{ping}\ u.P\ \texttt{else}\ Q : ok}(\Gamma(u) = T_u; \Gamma(l) = T_l; \neg\mathcal{D}(T_l, T_u))$$

We know that a process Q is *ok* with a location l included in the network configuration, and it is because if the process Q any `kill` k then $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$. Then, the process is *now-safe*.

If a `ping` $k.P$ `else` Q is included, it will continue being *now-safe* whether the type of the locations l and k were not related under $\mathcal{D}$, because the process will be the same than the commented previously, i.e. Q located in l.

(M-MURDER-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \mathcal{S}(T_l) = loc(\widetilde{K}); \Gamma(l) = T_l; \mathcal{D}(T_l, T_k); T_k \in \widetilde{K})$$

If a $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}}$ `kill` $k : ok$, it is because $\Gamma(l) = T_l$, $\mathcal{S}(T_l) = loc(\widetilde{K})$, $\Gamma(k) = T$ and $T \in \widetilde{K}$, i.e. the type of the location k is inside the set of types related to locations that l could kill. Then, it is *now-safe*.

(M-MURDER-NOT-OK)

$$\frac{}{\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok}(\Gamma(k) = T_k; \Gamma(l) = T_l; \neg\mathcal{D}(T_l, T_k))$$

If a $\Gamma \vdash_l^{\mathcal{D},\mathcal{S}} \texttt{kill}\ k : ok$ with $\neg\mathcal{D}(T_l, T_k)$, it is because these locations are not connected and any murder between them is impossible. Then, it is *now-safe*.

Bibliography

[1] The nordugrid homepage. http://www.nordugrid.org.

[2] Understanding file permissions on unix: a brief tutorial. http://www.dartmouth.edu/~rc/help/faq/permissions.html.

[3] The world community grid homepage. http://www.worldcommunitygrid.org.

[4] Roberto Amadio, Grard Boudol, and Cdric Lhoussaine. The receptive distributed π-calculus. In *In FST&TCS, number 1738 in Lecture Notes in Computer Science*, pages 304–315. Springer-Verlag, 1999.

[5] Adrian Francalanza and Matthew Hennessy. A theory of system behaviour in the presence of node and link failure. *Inf. Comput.*, 206(6):711–759, 2008.

[6] Matthew Hennessy. *A Distributed Pi-Calculus*. Cambridge University Press, New York, NY, USA, 2007.

[7] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf. Comput.*, 173(1):82–120, 2002.

[8] Hans Hüttel and Morten Kühnrich. Authentication and sandboxing in a distributed π-calculus. In Jan Jürjens Dieter Gollmann, editor, *WITS 2006*, Department of Computer Science, Aalborg University, Denmark, 2006.

[9] R. Milner. *The polyadic pi-calculus: a tutorial*, pages 203–246. Springer-Verlag, 1993.

[10] Joachim Parrow. An introduction to the π-calculus. In *Handbook of Process Algebra*. Bergstra, Ponse and Smolka, Elsevier, 2000.

[11] Joachim Parrow Robin Milner and David Walker. A calculus of mobile processes. Technical Report ECS-LFCS-89-85 and ECS-LFCS-89-86, University of Edinburgh, 1989.

[12] Pawel T. Wojciechowski and Peter Sewell. Nomadic pict: Language and infrastructure design for mobile agents. *Agent Systems and Applications, International Symposium on / International Symposium on Mobile Agents*, 0:2, 1999.

List of Tables