Soft Session Types

Ugo Dal Lago, Paolo Di Giamberardino

▶ **To cite this version:**

Ugo Dal Lago, Paolo Di Giamberardino. Soft Session Types. 2011. hal-00616388v1

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Soft Session Types

Ugo Dal Lago[*]   Paolo Di Giamberardino[†]

June 24, 2011

Abstract

We show how systems of sessions types can enforce interaction to be bounded for all typable processes. The type system we propose is based on Lafont's soft linear logic and is strongly inspired by recent works about session types as intuitionistic linear logic formulas. Our main result is the existence, for every typable process, of a polynomial bound on the length of any reduction sequence starting from it and on the size of any of its reducts.

1   Introduction

Session types are one of the most successful paradigms around which communication can be disciplined in a concurrent or object-based environment. They can come in many different flavors, depending on the underlying programming language and on the degree of flexibility they allow when defining the structure of sessions. As an example, systems of session types for multi-party interaction have been recently introduced [5], while a form of higher-order session has been shown to be definable [11]. Recursive types, on the other hand, have been part of the standard toolset of session type theories since their inception [4].

The key property induced by systems of session types is the following: if two (or more) processes can be typed with "dual" session types, then they can interact with each other without "going wrong", i.e. avoiding situations where one party needs some data with a certain type and the other(s) offer something of a different, incompatible type. Sometimes, one would like to go beyond that and design a type system which guarantees stronger properties, including quantitative ones. An example of a property that we find particularly interesting is the following: suppose that two processes P and Q interact by creating a session having type A through which they communicate. Is this interaction guaranteed to be finite? How much would it last? Moreover, P and Q could be forced to interact with other processes in order to be able to offer A. The question could then become: can the global amount of interaction be kept under control? In other words, one could be interested in *proving the interaction induced by sessions to be bounded*. This problem has been almost neglected by the research community in the area of session types, although it is the *manifesto* of the so-called implicit computational complexity (ICC), where one aims at giving machine-free characterizations of complexity classes based on programming languages and logical systems.

Linear logic (LL in the following) has been introduced twenty-five years ago by Jean-Yves Girard [3]. One of its greatest merits has been to allow a finer analysis of the computational content of both intuitionistic and classical logic. In turn, this is possible by distinguishing multiplicative as well as additive connectives, by an involutive notion of negation, and by giving a new status to structural rules allowing them to be applicable only to modal formulas. One of the many consequences of this new, refined way of looking at proof theory has been the introduction of natural characterizations of complexity classes by fragments of linear logic. This is possible because linear logic somehow "isolates" complexity in the modal fragment of the logic (which is solely

---

[*]Università di Bologna & INRIA Sophia Antipolis, `dallago@cs.unibo.it`
[†]Laboratoire d'Informatique de Paris Nord, `digiambe@lipn.univ-paris13.fr`

responsible for the hyperexponential complexity of cut elimination in, say intuitionistic logic), which can then be restricted so as to get exactly the expressive power needed to capture small complexity classes. One of the simplest and most elegant of such systems is Lafont's soft linear logic (`SLL` in the following), which has been shown to correspond to polynomial time in the realm of classical [6], quantum [8] and higher-order concurrent computation [9].

Recently, Caires and Pfenning [1] have shown how a system of session types can be built around intuitionistic linear logic, by introducing π`DILL`, a type system for the π-calculus where types and rules are derived from the ones of intuitionistic linear logic. In their system, multiplicative connectives like $\otimes$ and $\multimap$ allow to model sequentiality in sessions, while the additive connectives $\&$ and $\oplus$ model external and internal choice, respectively. The modal connective !, on the other hand, allows to model a server of type $!A$ which can offer the functionality expressed by A multiple times.

In this paper, we study a restriction of π`DILL`, called π`DSLL`, which can be thought as being derived from π`DILL` in the same way as `SLL` is obtained from `LL`. In other words, the operator ! behaves in π`DSLL` in the same way as in `SLL`. The main result we prove about π`DSLL` is precisely about bounded interaction: whenever P can be typed in π`DSLL` and $P \to^n Q$, then both n and $|Q|$ (the size of the process Q, to be defined later) are polynomially related to $|P|$. This ensures an abstract but quite strong form of bounded interaction. Another, perhaps more "interactive" formulation of the same result is the following: if P and Q interact via a channel of type A, then the "complexity" of this interaction is bounded by a polynomial on $|P| + |Q|$, whose degree only depends on A.

We see this paper as the first successful attempt to bring techniques from implicit computational complexity into the realm of session types. Although proving bounded interaction has been technically nontrivial, due to the peculiarities of the π-calculus, we think the main contribution of this work lies in showing that bounded termination can be enforced by a natural adaptation of known systems of session types.

2 π`DILL`, an Informal Account

In this section, we will outline the main properties of π`DILL`, a session type system recently introduced by Caires and Pfenning [1, 2]. For more information, please consult the two cited papers.

In π`DILL`, session types are nothing more than formulas of (propositional) intuitionistic linear logic without atoms but with (multiplicative) constants:

$$A ::= \mathbf{1} \mid A \otimes A \mid A \multimap A \mid A \oplus A \mid A \& A \mid !A.$$

These types are assigned to channels (names) by a formal system deriving judgments in the form

$$\Gamma; \Delta \vdash P :: x : A,$$

where Γ and Δ are contexts assigning types to channels, and P is a process of the name-passing π calculus. The judgment above can be read as follows: the process P acts on the channel x according to the session type A *whenever* composed with processes behaving according to Γ and Δ (each on a specific channel). Informally, the various constructions on session types can be explained as follows:

- $\mathbf{1}$ is the type of an empty session channel. A process offering to communicate via a session channel typed this way simply synchronizes with another process through it without exchanging anything. This is meant to be an abstraction for all ground session types, e.g. natural numbers, lists, etc. In linear logic, this is the unit for $\otimes$.
- $A \otimes B$ is the type of a session channel x through which a message carrying another channel with type A is sent. After performing this action, the underlying process behaves according to B on the *same* channel x.

- $A \multimap B$ is the adjoint to $A \otimes B$: on a channel with this type, a process communicate by first performing an input and receiving a channel with type A, then acting according to B, again on x.
- $A \oplus B$ is the type of a channel on which a process either sends a special message `inl` and performs according to A or sends a special message `inr` and performs according to B.
- The type $A\&B$ can be assigned to a channel x on which the underlying process offers the possibility of choosing between proceeding according to A or to B, both on x. So, in a sense, & models external choice.
- Finally, the type $!A$ is attributed to a channel x only if a process can be replicated by receiving a channel y through x, then behaving on y according to A.

The assignments in Γ and Δ are of two different natures:

- An assignment of a type A to a channel x in Δ signals the need by P of a process offering a session of type A on the channel x; for this reason, Γ is called the *linear context*;
- An assignment of a type A to a channel x in Γ, on the other hand, represents the need by P of a process offering a session of type $!A$ on the channel x; thus, Δ is the *exponential context*.

Typing rules πDILL are very similar to the ones of DILL, itself one of the many possible formulations of linear logic as a sequent calculus. In particular, there are two cut rules, each corresponding to a different portion of the context:

$$\frac{\Gamma; \Delta_1 \vdash P :: x : A \quad \Gamma; \Delta_2, x : A \vdash Q :: T}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu x)(P||Q) :: T} \qquad \frac{\Gamma; \emptyset \vdash P :: y : A \quad \Gamma, x : A; \Delta \vdash Q :: T}{\Gamma; \Delta \vdash (\nu x)(!x(y).P||Q) :: T}$$

Please observe how cutting a process P against an assumption in the exponential context requires to "wrap" P inside a replicated input: this allows to *turn P into a server*.

In order to illustrate the intuitions above, we now give an example. Suppose that a process P models a service which acts on x as follows: it receives two natural numbers, to be interpreted as the number and secret code of a credit card and, if they correspond to a valid account, returns an MP3 file and a receipt code to the client. Otherwise, the session terminates. To do so, P needs to interact with another service (e.g. a banking service) Q through another channel y. The banking service, among others, provides a way to verify whether a given number and code correspond to a valid credit card. In πDILL, the process P would receive the type

$$\emptyset; y : (\mathbf{N} \multimap \mathbf{1} \oplus \mathbf{1})\&A \vdash P :: x : \mathbf{N} \multimap \mathbf{N} \multimap (\mathbf{S} \otimes \mathbf{N}) \oplus \mathbf{1},$$

where $\mathbf{N}$ and $\mathbf{S}$ are pseudo-types for natural numbers and MP3s, respectively. A is the type of all the other functionalities Q provides. As an example, P could be the following process:

$$x(nm_1).x(cd_1).y.\mathtt{inl};$$
$$(\nu nm_2)y\langle nm_2\rangle.(\nu cd_2)y\langle cd_2\rangle.$$
$$y.\mathtt{case}(x.\mathtt{inl}; (\nu mp)x\langle mp\rangle.(\nu rp)x\langle rp\rangle, x.\mathtt{inr}; 0)$$

Observe how the credit card number and secret code forwarded to Q are not the ones sent by the client: the flow of information happening inside a process is abstracted away in πDILL. Similarly, one can write a process Q and assign it a type as follows: $\emptyset; \emptyset \vdash Q :: y : (\mathbf{N} \multimap \mathbf{1} \oplus \mathbf{1})\&A$. Putting the two derivations together, we obtain $\emptyset; \emptyset \vdash (\nu x)(P||Q) :: x : \mathbf{N} \multimap \mathbf{N} \multimap (\mathbf{S} \otimes \mathbf{N}) \oplus \mathbf{1}$.

Let us now make an observation which will probably be appreciated by the reader familiar with linear logic. The processes P and Q can be typed in πDILL without the use of any exponential rule, nor of cut. What allows to type the parallel composition $(\nu x)(P||Q)$, on the other hand, is precisely the cut rule. The interaction between P and Q corresponds to the elimination of that cut. Since there isn't any exponential around, this process must be finite, since the size of the underlying process shrinks at every single reduction step. From a process-algebraic point of view, on the other hand, the finiteness of the interaction is an immediate consequence of the absence of any replication in P and Q.

The banking service Q can only serve one single session and would vanish at the end of it. To make it into a *persistent server* offering the same kind of session to possibly many different clients,

Q must be wrapped into a replication, obtaining $R =!z(y).Q$. In R, the channel z can be given type $!((\mathbf{N} \multimap \mathbf{1} \oplus \mathbf{1}) \& A)$ in the empty context. The process P should be somehow adapted to be able to interact with R: before performing the two outputs on y, it's necessary to "spawn" R by performing an output on z and passing y to it. This way we obtain a process S such that

$$\emptyset; z :!((\mathbf{N} \multimap \mathbf{1} \oplus \mathbf{1}) \& A) \vdash S :: x : \mathbf{N} \multimap \mathbf{N} \multimap (\mathbf{S} \otimes \mathbf{N}) \oplus \mathbf{1},$$

and the composition $(\nu z)(S||R)$ can be given the same type as $(\nu x)(P||Q)$. Of course, S could have used the channel z more than once, initiating different sessions. This is meant to model a situation in which the same client interacts with the same server by creating more than one session with the same type, itself done by performing *more than one output* on the same channel. Of course, servers can themselves depend on other servers. And these dependencies are naturally modeled by the exponential modality of linear logic.

3 On Bounded Interaction

In $\pi\texttt{DILL}$, the possibility of modeling persistent servers which in turn depend on other servers makes it possible to type processes which exhibit a very complex and combinatorially heavy interactive behavior.

Consider the following processes, the first one parameterized on any $i \in \mathbb{N}$:

$$dupser_i \doteq !x_i(y).(\nu z)x_{i+1}\langle z\rangle.(\nu w)x_{i+1}\langle w\rangle.;$$
$$dupclient \doteq (\nu y)x_0\langle y\rangle.$$

In $\pi\texttt{DILL}$, these processes can be typed as follows:

$$\emptyset; x_{i+1} :!\mathbf{1} \vdash dupser_i :: x_i :!\mathbf{1};$$
$$\emptyset; x_0 :!\mathbf{1} \vdash dupclient :: z : \mathbf{1}.$$

Then, for every $n \in \mathbb{N}$ one can type the parallel composition

$$mulser_n \doteq (\nu x_1 \ldots x_n)(dupser_n|| \ldots ||dupser_0)$$

as follows

$$\emptyset; x_n :!\mathbf{1} \vdash mulser_n :: x_0 :!\mathbf{1}.$$

Informally, $mulser_n$ is a persistent server which offers a session type $\mathbf{1}$ on a channel x_0, provided a server with the same functionality is available on x_n. The process $mulser_n$ is the parallel composition of n servers in the form $dupser_i$, each spawning two different sessions provided by $dupser_{i+1}$ on the same channel x_{i+1}.

The process $mulser_n$ cannot be further reduced. But notice that, once $mulser_n$ and $dupclient$ are composed, the following exponential blowup happens:

$$
\begin{aligned}
(\nu x_0)(mulser_n||dupclient) &\equiv (\nu x_0 \ldots x_n)(dupser_n|| \ldots ||dupser_0||dupclient) \\
&\to (\nu x_0 \ldots x_n)(dupser_n|| \ldots ||dupser_1||P_1) \\
&\to^2 (\nu x_1 \ldots x_n)(dupser_n|| \ldots ||dupser_2||P_2||P_2) \\
&\to^4 (\nu x_2 \ldots x_n)(dupser_n|| \ldots ||dupser_3||\underbrace{P_3|| \ldots ||P_3}_{4 \text{ times}}) \\
&\to^* (\nu x_n)(dupser_n||\underbrace{P_n|| \ldots ||P_n}_{2^n \text{ times}}) \\
&\to^{2^n} 0
\end{aligned}
$$

where, for every $i \in \mathbb{N}$, the process P_i is simply $(\nu y)x_i\langle y\rangle.(\nu z)x_i\langle z\rangle$. Notice that *both* the number or reduction steps *and* the size of intermediate processes are exponential in n, while the size of the

4

initial process is linear in n. This is a perfectly legal process in πDILL. Moreover the type $!\mathbf{1}$ of the channel x_0 through which *dupclient* and *mulser$_n$* communicate do not contain any information about the "complexity" of the interaction: it is the same for every n.

The deep reasons why this phenomenon can happen are in the very general (and "generous") rules governing the behavior of the exponential modality $!$ in linear logic. It is this generality that allows the embedding of propositional intuitionistic logic into linear logic. Since the complexity of normalization for the former [12, 10] is nonelementary, the exponential blowup described above is not a surprise.

It would be desirable, on the other hand, to be sure that the interaction caused by any process P is bounded: whenever $P \to^n Q$, then there's a *reasonably low* upper bound to both n and $|Q|$. This is precisely what we achieve by restricting πDILL into πDSLL.

4 πDSLL: Syntax and Main Properties

In this section, the syntax of πDSLL will be introduced. Moreover, some basic operational properties will be given.

4.1 The Process Algbera

πDSLL is a type system for a fairly standard π-calculus, exactly the one on top of which πDILL is defined:

Definition 1 (Processes) *Given an infinite set of* names *or* channels $x, y, z, \ldots$, *the set of* processes *is defined as follows:*

$$P ::= 0 \mid P||Q \mid (\nu x)P \mid x(y).P \mid x\langle y\rangle.P \mid !x(y).P \mid x.\mathtt{inl}; P \mid x.\mathtt{inr}; P \mid x.\mathtt{case}(P, Q)$$

The only non-standard constructs are the last three, which allow to define a choice mechanism: the process $x.\mathtt{case}(P, Q)$ can evolve as P or as Q *after* having received a signal in the form $\mathtt{inl}$ o $\mathtt{inr}$ through x. Processes sending such a signal through the channel x, then continuing like P are, respectively, $x.\mathtt{inl}; P$ and $x.\mathtt{inr}; P$. The set of names occurring free in the process P (hereby denoted $fn(P)$) is defined as usual. The same holds for the capture avoiding substitution of a name x for y in a process P (denoted $P\{x/y\}$), and for α-equivalence between processes (denoted $\equiv_\alpha$).

Structural congruence is an equivalence relation identifying those processes which are syntactically different but can be considered equal for very simple structural reasons:

Definition 2 (Structural Congruence) *The relation $\equiv$, called* structural congruence, *is the least congruence on processes satisfying the following seven axioms:*

$$P \equiv Q \quad whenever\ P \equiv_\alpha Q \qquad\qquad (\nu x)0 \equiv 0$$
$$P||0 \equiv P \qquad\qquad (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$$
$$P||Q \equiv Q||P \qquad\qquad ((\nu x)P)||Q \equiv (\nu x)(P||Q) \quad whenever\ x \notin fn(Q)$$
$$P||(Q||R) \equiv (P||Q)||R$$

Formal systems for reduction and labelled semantics can be defined in a standard way. We refer the reader to [1] for more details.

A quantitative attribute of processes which is delicate to model in process algebras is their *size*: how can we measure the size of a process? In particular, it is not straightforward to define a measure which both reflects the "number of symbols" in the process and is invariant under structural congruence (this way facilitating all proofs). A good compromise is the following:

Definition 3 (Process Size) *The* size $|P|$ *of a process P is defined by induction on the structure of P as follows:*

$$|0| = 0; \qquad |x(y).P| = |P| + 1; \qquad |x.\textbf{\textit{inl}}; P| = |P| + 1;$$
$$|P||Q| = |P| + |Q|; \qquad |x\langle y\rangle.P| = |P| + 1; \qquad |x.\textbf{\textit{inr}}; P| = |P| + 1;$$
$$|(\nu x)P| = |P|; \qquad |!x(y).P| = |P| + 1; \qquad |x.\textbf{\textit{case}}(P, Q)| = |P| + |Q| + 1.$$

According to the definition above, the empty process 0 has null size, while restriction does not increase the size of the underlying process. This allows for a definition of size which remains invariant under structural congruence. The price to pay is the following: the "number of symbols" of a process P can be arbitrarily bigger than $|P|$ (e.g. for every $n \in \mathbb{N}$, $|(\nu x)^n P| = |P|$). However, we have the following:

Lemma 1 *For every P, Q, $|P| = |Q|$ whenever $P \equiv Q$. Moreover, there is a polynomial p such that for every P, there is Q with $P \equiv Q$ and the number of symbols in Q is at most $p(|Q|)$.*

4.2   The Type System

The language of types of πDSLL is exactly the same as the one of πDILL, and the interpretation of type constructs does not change (see Section 2 for some informal details). Typing judgments and typing rules, however, are significantly different, in particular, in the treatment of the exponential connective !.

Typing judgments become syntactical expressions in the form

$$\Gamma; \Delta; \Theta \vdash P :: x : A.$$

First of all, observe how the context is divided into *three* chunks now: Γ and Δ have to be interpreted as exponential contexts, while Θ is the usual linear context from πDILL. The necessity of having *two* exponential contexts depends on the finer, less canonical exponential discipline of SLL compared to the one of LL. We use the following terminology: Γ is said to be the *auxiliary* context, while Δ is the *multiplexor* context.

Typing rules are in Figure 1. The rules governing the typing constant $\mathbf{1}$, the multiplicatives ($\otimes$ and $\multimap$) and the additives ($\oplus$ and $\&$) are exact analogues of the ones from πDILL. The only differences come from the presence of two exponential contexts: in binary rules ($\otimes$R, $\multimap$ L, $\&$R and $\oplus$L) the auxiliary context is treated multiplicatively, while the multiplexor context is treated additively, as in πDILL[1]. Now, consider the rules governing the exponential connective !, which are $\flat_!$, $\flat_\#$, $!$L$_!$, $!$L$_\#$ and $!$R:

- The rules $\flat_!$ and $\flat_\#$ both allow to spawn a server. This corresponds to turning an assumption $x : A$ in the linear context into one $y : A$ in one of the exponential contexts;
- The rules $!$L$_!$ and $!$L$_\#$ lift an assumption in the exponential contexts to the linear context; this requires changing its type from A to $!A$;
- The rule $!$R allows to turn an ordinary process into a server, by packaging it into a replicated input and modifying its type.

Finally there are *three* cut rules in the system, namely cut, cut$_!$ and cut$_\#$:

- cut is the usual linear cut rule, i.e. the natural generalization of the one from πDILL.
- cut$_!$ and cut$_\#$ allow to eliminate an assumption in the exponential contexts. In both cases, the process which allows to do that must be typable with empty linear and multiplexor contexts.

We state the following two lemmas, concerning structural properties of the type system, which we need for technical reasons.

---

[1]The reader familiar with linear logic and proof nets will recognize in the different treatment of the auxiliary and multiplexor contexts, one of the basic principle of SLL: *contraction is forbidden on the auxiliary doors of exponential boxes*. The channel names contained in the auxiliary context correspond to the auxiliary doors of exponential boxes, so we treat them multiplicatively. The contraction effect induced by the additive treatment of the channel names in the multiplexor context corresponds to the multiplexing rule of SLL.

$$\frac{\Gamma;\Delta;\Theta \vdash P :: T}{\Gamma;\Delta;\Theta, x : \mathbf{1} \vdash P :: T} \; \mathsf{1L} \qquad \frac{}{\Gamma;\Delta;\emptyset \vdash 0 :: x : \mathbf{1}} \; \mathsf{1R}$$

$$\frac{\Gamma;\Delta;\Theta, y : A, x : B \vdash P :: T}{\Gamma;\Delta;\Theta, x : A \otimes B \vdash x(y).P :: T} \; \otimes\mathsf{L} \qquad \frac{\Gamma_1;\Delta;\Theta_1 \vdash P :: y : A \quad \Gamma_2;\Delta;\Theta_2 \vdash Q :: x : B}{\Gamma_1, \Gamma_2;\Delta;\Theta_1, \Theta_2 \vdash (\nu y)x\langle y\rangle.(P\|Q) :: x : A \otimes B} \; \otimes\mathsf{R}$$

$$\frac{\Gamma_1;\Delta;\Theta_1, y : A \vdash P :: T \quad \Gamma_1;\Delta;\Theta_2, x : B \vdash Q :: T}{\Gamma_1, \Gamma_2;\Delta;\Theta_1, \Theta_2, x : A \multimap B \vdash (\nu y)x\langle y\rangle.(P\|Q) :: T} \; \multimap \mathsf{L} \qquad \frac{\Gamma;\Delta;\Theta, y : A \vdash P :: x : B}{\Gamma;\Delta;\Theta \vdash x(y).P :: x : A \multimap B} \; \multimap \mathsf{R}$$

$$\frac{\Gamma;\Delta;\Theta, x : A \vdash P :: T \quad \Gamma;\Delta;\Theta, x : B \vdash P :: T}{\Gamma;\Delta; x : A \oplus B, \Theta \vdash y.\mathtt{case}(P, Q) :: T} \; \oplus\mathsf{L} \qquad \frac{\Gamma;\Delta;\Theta \vdash P :: x : A}{\Gamma;\Delta;\Theta \vdash x.\mathtt{inl}; P :: x : A \oplus B} \; \oplus\mathsf{R}_1$$

$$\frac{\Gamma;\Delta;\Theta \vdash P :: x : B}{\Gamma;\Delta;\Theta \vdash x.\mathtt{inr}; P :: x : A \oplus B} \; \oplus\mathsf{R}_2 \qquad \frac{\Gamma;\Delta;\Theta, x : A \vdash P :: T}{\Gamma;\Delta;\Theta, x : A\&B \vdash x.\mathtt{inl}; P :: T} \; \&\mathsf{L}_1$$

$$\frac{\Gamma;\Delta;\Theta, x : B \vdash P :: T}{\Gamma;\Delta;\Theta, x : A\&B \vdash x.\mathtt{inr}; P :: T} \; \&\mathsf{L}_2 \qquad \frac{\Gamma;\Delta;\Theta \vdash P :: x : A \quad \Gamma;\Delta;\Theta \vdash P :: x : B}{\Gamma;\Delta;\Theta \vdash y.\mathtt{case}(P, Q) :: x : A\&B} \; \&\mathsf{R}$$

$$\frac{\Gamma;\Delta;\Theta, y : A \vdash P :: T}{\Gamma;\Delta, x : A;\Theta \vdash (\nu y)x\langle y\rangle.P :: T} \; \flat_{\#} \qquad \frac{\Gamma;\Delta;\Theta, y : A \vdash P :: T}{\Gamma, x : A;\Delta;\Theta \vdash (\nu y)x\langle y\rangle.P :: T} \; \flat_{!}$$

$$\frac{\Gamma;\Delta, x : A;\Theta \vdash P :: T}{\Gamma;\Delta;\Theta, x :!A \vdash P :: T} \; \mathsf{!L}_{\#} \qquad \frac{\Gamma, x : A;\Delta;\Theta \vdash P :: T}{\Gamma;\Delta;\Theta, x :!A \vdash P :: T} \; \mathsf{!L}_{!} \qquad \frac{\Gamma;\emptyset;\emptyset \vdash \emptyset :: Q y : A}{\emptyset;\Delta;!\Gamma \vdash !x(y).Q :: x :!A} \; \mathsf{!R}$$

$$\frac{\Gamma_1;\Delta;\Theta_1 \vdash P :: x : A \quad \Gamma_2;\Delta;\Theta_2, x : A \vdash Q :: T}{\Gamma_1, \Gamma_2;\Delta;\Theta_1, \Theta_2 \vdash (\nu x)(P\|Q) :: T} \; \mathsf{cut}$$

$$\frac{\Delta;\emptyset;\emptyset \vdash \emptyset :: P y : A \quad \Gamma;\Delta, x : A;\Theta \vdash Q :: T}{\Gamma;\Delta;\Theta \vdash (\nu x)(!x(y).P\|Q) :: T} \; \mathsf{cut}_{\#}$$

$$\frac{\Gamma_1;\emptyset;\emptyset \vdash \emptyset :: P y : A \quad \Gamma_2, x : A;\Delta;\Theta \vdash Q :: T}{\Gamma_1, \Gamma_2;\Delta;\Theta \vdash (\nu x)(!x(y).P\|Q) :: T} \; \mathsf{cut}_{!}$$

Figure 1: Typing rules for πDSLL.

Lemma 2 (Weakening lemma) *For any proof π of $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$, there exists a proof π' of $\Gamma'; \Delta'; \Theta \vdash \mathsf{D} :: T$, where $\Gamma \subseteq \Gamma', \Delta \subseteq \Delta'$.*

Proof. By a simple induction on the height of π. ☐

Lemma 3 (Lifting lemma) *If $\Gamma; \Delta; \Theta \vdash D$, then there exists an E such that $\emptyset; \Gamma, \Delta; \Theta \vdash E$, where $\widehat{E} = \widehat{D}$. We denote E by $D_{\Downarrow}$*

Proof. By a simple induction on the height of the proof. ☐

4.3 Back to Our Example

Let us now reconsider the example processes introduced in Section 3. The basic building block over which everything is built was the process $dupser_i$. It is still a *typable* process in $\pi\mathsf{DSLL}$ but, as can be easily checked, it cannot receive *the same* types it has in $\pi\mathsf{DILL}$. Indeed, it can be typed as follows:

$$\emptyset; \emptyset; x_{i+1} :!^{i+1}\mathbf{1} \vdash dupser_i :: x_i :!^i\mathbf{1};$$

The parallel composition $mulser_n \doteq (\nu x_1 \ldots x_n)(dupser_n || \ldots || dupser_0)$ can still be typed, but it receives a different type itself:

$$\emptyset; \emptyset; x_n :!^n\mathbf{1} \vdash mulser_n :: x_0 : \mathbf{1}$$

As a consequence, the exponential behavior caused by allowing $mulser_n$ to interact with a client can *still* take place. However, and this is what makes $\pi\mathsf{DSLL}$ different from $\pi\mathsf{DILL}$, the type $!^n\mathbf{1}$ assigned to x_n in the typing judgment for $mulser_n$ somehow reflects the exponent of a polynomial governing the interaction complexity of $mulser_n$. Please notice how in $\pi\mathsf{DILL}$, every process in the form $mulser_n$ received *exactly* the same type.

4.4 Subject Reduction

A basic property most type systems for functional languages satisfy is subject reduction: typing is preserved along reduction. For processes, this is often true for internal reduction: if $P \to Q$ and $\vdash P : A$, then $\vdash Q : A$. In this section, a subject reduction result for $\pi\mathsf{DSLL}$ will be given and some ideas on the underlying proof will be described. Some concepts outlined here will become necessary ingredients in the proof of bounded interaction, to be done in Section 5 below. In doing this, we will closely follow the path traced by Caires and Pfenning and proceed quite quickly, concentrating our attention on the differences with their proof.

When proving subject reduction, one constantly work with type derivations. This is particularly true here, where (internal) reduction corresponds to the cut-elimination process. A linear notation for proofs in the form of *proof terms* can be easily defined, allowing for more compact descriptions. As an example, a proof in the form

$$\frac{\pi : \Gamma_1; \Delta; \Theta_1 \vdash P :: x : A \quad \rho : \Gamma_2; \Delta; \Theta_2, x : A \vdash Q :: T}{\Gamma_1, \Gamma_2; \Delta; \Theta_1, \Theta_2 \vdash (\nu x)(P||Q) :: T} \ \mathsf{cut}$$

corresponds to the proof term $\mathsf{cut}(\mathsf{D}, x.\mathsf{E})$, where D is the proof term for π and E is the proof term for ρ. If D is a proof term corresponding to a type derivation for the process P, we write $\widehat{\mathsf{D}} = P$. From now on, proof terms will often take the place of processes: $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$ stands for the existence of a type derivation D with conclusion $\Gamma; \Delta; \Theta \vdash \widehat{\mathsf{D}} :: T$. A proof term D is said to be normal if it does not contain any instances of cut rules.

In Fig 2 we show in detail how processes are associated with proof terms.

Subject reduction will be proved by showing that if P is typable by a type derivation D and $P \to Q$, then a type derivation E for Q exists. Actually, E can be obtained by manipulating D using techniques derived from cut-elimination. Noticeably, not every cut-elimination rule is

$$
\begin{array}{lcl}
\mathbf{1}\mathsf{L}(x,\mathsf{D}) & \rightsquigarrow & \widehat{\mathsf{D}}^z \\
\mathbf{1}\mathsf{R} & \rightsquigarrow & 0 \\
\otimes\mathsf{L}(x,y.z.\mathsf{E}) & \rightsquigarrow & x(y).\widehat{\mathsf{E}}^z \\
\otimes\mathsf{R}(\mathsf{D},\mathsf{E}) & \rightsquigarrow & (\nu y)x\langle y\rangle.(\widehat{\mathsf{D}}^y\|\widehat{\mathsf{E}}^x) \\
\multimap\mathsf{L}(x,\mathsf{D},y.\mathsf{E}) & & (\nu y)x\langle y\rangle.(\widehat{\mathsf{D}}^y\|\widehat{\mathsf{E}}^z) \\
\multimap\mathsf{R}(x.\mathsf{D}) & \rightsquigarrow & x(y).\widehat{\mathsf{E}}^x \\
\mathsf{cut}(\mathsf{D},x.\mathsf{E}) & \rightsquigarrow & (\nu x)(\widehat{\mathsf{D}}^x\|\widehat{\mathsf{E}}^z) \\
\mathsf{cut}_!(\mathsf{D},x.\mathsf{E}) & \rightsquigarrow & (\nu x)(!x(y).\widehat{\mathsf{D}}^y\|\widehat{\mathsf{E}}^z) \\
\mathsf{cut}_\#(\mathsf{D},x.\mathsf{E}) & \rightsquigarrow & (\nu x)(!x(y).\widehat{\mathsf{D}}^y\|\widehat{\mathsf{E}}^z) \\
\flat_!(x,y.\mathsf{E}) & \rightsquigarrow & (\nu y)x\langle y\rangle.\widehat{\mathsf{E}}^z \\
\flat_\#(x,y.\mathsf{E}) & \rightsquigarrow & (\nu y)x\langle y\rangle.\widehat{\mathsf{E}}^z \\
!\mathsf{R}(\mathsf{D},x_1,\ldots,x_n) & \rightsquigarrow & !x(y).\widehat{\mathsf{D}}^y \\
!\mathsf{L}_!(x.\mathsf{D}) & \rightsquigarrow & \widehat{\mathsf{D}}^z \\
!\mathsf{L}_\#(x.\mathsf{D}) & \rightsquigarrow & \widehat{\mathsf{D}}^z \\
\oplus\mathsf{L}(x,y.\mathsf{D},z.\mathsf{E}) & \rightsquigarrow & y.\mathtt{case}(\widehat{\mathsf{D}}^x,\widehat{\mathsf{E}}^z) \\
\oplus\mathsf{R}_1(\mathsf{D}) & \rightsquigarrow & x.\mathtt{inl};\widehat{\mathsf{D}}^x \\
\oplus\mathsf{R}_2(\mathsf{D}) & \rightsquigarrow & y.\mathtt{inr};\widehat{\mathsf{D}}^y \\
\&\mathsf{L}_1(x,y.\mathsf{E}) & \rightsquigarrow & x.\mathtt{inl};\widehat{\mathsf{D}}^z \\
\&\mathsf{L}_2(x,y.\mathsf{D}) & \rightsquigarrow & y.\mathtt{inr};\widehat{\mathsf{D}}^z \\
\&\mathsf{R}(\mathsf{D},\mathsf{E}) & \rightsquigarrow & z.\mathtt{case}(\widehat{\mathsf{D}}^z,\widehat{\mathsf{E}}^z)
\end{array}
$$

Figure 2: Extraction of processes from proof terms.

necessary to prove subject reduction. In other words, we are in presence of a weak correspondence between proof terms and processes, and remain far from a genuine Curry-Howard correspondence.

Those manipulations of proof-terms which are necessary to prove subject reduction can be classified as follows:

- First of all, a binary relation $\Longrightarrow$ on proof terms called *computational reduction* can be defined. At the logical level, this corresponds to proper cut-elimination steps, i.e. those cut-elimination steps in which two rules introducing the same connective interact. At the process level, computational reduction correspond to internal reduction. $\Longrightarrow$ is not symmetric. Computational reduction rules are given in Figure 3.

- A binary relation $\longmapsto$ on proof terms called *shift reduction*, distinct from $\Longrightarrow$ must be introduced. At the process level, it corresponds to structural congruence. As $\Longrightarrow$, $\longmapsto$ is not a symmetric relation. Shift reduction rules are given in Figure 4.

- Finally, an equivalence relation $\equiv$ on proof terms called *proof equivalence* is necessary. At the logical level, this corresponds to the so-called commuting conversions, while at the process level, the induced processes are either structurally congruent or strongly bisimilar. Equivalence rules are given in Figure 5.

The reflexive and transitive closure of $\longmapsto \cup \equiv$ is denoted with $\hookrightarrow$, i.e. $\hookrightarrow = (\longmapsto \cup \equiv)^*$. To help the reader understand the rules defining $\Longrightarrow$, $\longmapsto$ and $\equiv$, let us give some relevant examples:

- Let us consider the proof term $\mathsf{D} = \mathsf{cut}((\otimes\mathsf{R}(\mathsf{D}_1,\mathsf{D}_2)),x.\otimes\mathsf{L}(x,y.x.\mathsf{D}_3))$ which corresponds to the $\otimes$-case of cut elimination. By a computational reduction rule, $\mathsf{D} \Longrightarrow \mathsf{E} = \mathsf{cut}(\mathsf{D}_1,y.\mathsf{cut}(\mathsf{D}_2,x.\mathsf{D}_3))$. From the process side, $\widehat{\mathsf{D}} = (\nu x)(((\nu y)x\langle y\rangle.(\widehat{\mathsf{D}_1}\|\widehat{\mathsf{D}_2}))\|x(y).\widehat{\mathsf{D}_3})$ and $\widehat{\mathsf{E}} = (\nu x)(\nu y)((\widehat{\mathsf{D}_1}\|\widehat{\mathsf{D}_2})\|\widehat{\mathsf{D}_3})$, where $\widehat{\mathsf{E}}$ is the process obtained from $\widehat{\mathsf{D}}$ by internal passing the channel y through the channel x.

- Let $\mathsf{D} = \mathsf{cut}(!\mathsf{R}(\mathsf{D}_1,x_1,\ldots,x_n),x.!\mathsf{L}_!(x.\mathsf{D}_2))$ be the proof obtained by composing a proof D_1 (whose last rule is $!\mathsf{R}$) with a proof D_2 (whose last rule is $!\mathsf{L}_!$) through a cut rule. A shift reduction rule tells us that $\mathsf{D} \longmapsto \mathsf{E} = !\mathsf{L}_!(x_1.!\mathsf{L}_!(x_2.\ldots.!\mathsf{L}_!(x_n.\mathsf{cut}_!(\mathsf{D}_1,y.\mathsf{D}_2))\ldots))$, which corre-

$(\mathsf{cut}/\otimes\mathsf{R}/\otimes\mathsf{L}):$	$\mathsf{cut}((\otimes\mathsf{R}(\mathsf{D},\mathsf{E})), x.\otimes\mathsf{L}(x, y.x.\mathsf{F}))$	$\implies$	$\mathsf{cut}(\mathsf{D}, y.\mathsf{cut}(\mathsf{E}, x.\mathsf{F}))$
$(\mathsf{cut}/\multimap\mathsf{L}/\multimap\mathsf{R}):$	$\mathsf{cut}(\multimap\mathsf{R}(y.\mathsf{D}), x.\multimap\mathsf{L}(x, \mathsf{E}, x.\mathsf{F}))$	$\implies$	$\mathsf{cut}(\mathsf{cut}(\mathsf{E}, y.\mathsf{D}), x.\mathsf{F})$
$(\mathsf{cut}/\&\mathsf{R}/\&\mathsf{L}_1):$	$\mathsf{cut}(\&\mathsf{R}(\mathsf{D}, \mathsf{E}), x.\&\mathsf{L}_1(x, y.\mathsf{F}))$	$\implies$	$\mathsf{cut}(\mathsf{D}, x.\mathsf{F})$
$(\mathsf{cut}/\&\mathsf{R}/\&\mathsf{L}_2):$	$\mathsf{cut}(\&\mathsf{R}(\mathsf{D}, \mathsf{E}), x.\&\mathsf{L}_2(x, y.\mathsf{F}))$	$\implies$	$\mathsf{cut}(\mathsf{E}, x.\mathsf{F})$
$(\mathsf{cut}/\oplus\mathsf{R}_1/\oplus\mathsf{L}):$	$\mathsf{cut}(\oplus\mathsf{R}_1(\mathsf{D}), x.\oplus\mathsf{L}(x, y.\mathsf{E}, z.\mathsf{F}))$	$\implies$	$\mathsf{cut}(\mathsf{D}, x.\mathsf{E})$
$(\mathsf{cut}/\oplus\mathsf{R}_2/\oplus\mathsf{L}):$	$\mathsf{cut}(\oplus\mathsf{R}_2(\mathsf{D}), x.\oplus\mathsf{L}(x, y.\mathsf{E}, z.\mathsf{F}))$	$\implies$	$\mathsf{cut}(\mathsf{D}, x.\mathsf{F})$
$(\mathsf{cut}_!/-/\flat_!):$	$\mathsf{cut}_!(\mathsf{D}, x.\flat_!(x, y.\mathsf{E}))$	$\implies$	$\mathsf{cut}(\mathsf{D}_\Downarrow, y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E}_\Downarrow))$ where $x \notin FV(\widehat{\mathsf{E}})$
$(\mathsf{cut}_\#/-/\flat_\#):$	$\mathsf{cut}_\#(\mathsf{D}, x.\flat_\#(x, y.\mathsf{E}))$	$\implies$	$\mathsf{cut}(\mathsf{D}_\Downarrow, y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E}))$

Figure 3: Computational reduction rules

$(\mathsf{cut}/!\mathsf{R}/!\mathsf{L}_!):$	$\mathsf{cut}(!\mathsf{R}(\mathsf{D}, x_1, \ldots, x_n), x.!\mathsf{L}_!(x.\mathsf{E}))$	$\longmapsto$	$!\mathsf{L}_!(x_1.!\mathsf{L}_!(x_2.\ldots.!\mathsf{L}_!(x_n.\mathsf{cut}_\#(\mathsf{D}, y.\mathsf{E}))\ldots))$
$(\mathsf{cut}/!\mathsf{R}/!\mathsf{L}_\#):$	$\mathsf{cut}(!\mathsf{R}(\mathsf{D}, x_1, \ldots, x_n), x.!\mathsf{L}_\#(x.\mathsf{E}))$	$\longmapsto$	$!\mathsf{L}_!(x_1.!\mathsf{L}_!(x_2.\ldots.!\mathsf{L}_!(x_n.\mathsf{cut}_\#(\mathsf{D}, y.\mathsf{E}))\ldots))$

Figure 4: Shift reduction rules

sponds to the "soft promotion/dereliction" reduction step of SLL. The shift reduction does not have a corresponding reduction step at process level, since $\widehat{\mathsf{D}} \equiv \widehat{\mathsf{E}}$; nevertheless, it is defined as an asymmetric relation, for technical reasons connected to the proof of bounded interaction.

- Let $\mathsf{D} = \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}(\mathsf{D}_2, y.\mathsf{D}_3))$. A defining rule for proof equivalence $\equiv$, states that in D the $\mathsf{cut}_\#$ rule can be permuted over the cut rule, by duplicating D_1; namely $\mathsf{D} \equiv \mathsf{E} = \mathsf{cut}(\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2), y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_3))$. This is possible because the channel x belongs to the multiplexor contexts of both $\mathsf{D}_2, \mathsf{D}_3$, such contexts being treated additively. At the process level, $\widehat{\mathsf{D}} = (\nu x)((!x(y).\widehat{\mathsf{D}_1})||(\nu y)(\widehat{\mathsf{D}_2}||\widehat{\mathsf{D}_3}))$, while $\widehat{\mathsf{E}} = (\nu y)(((\nu x)(!x(y).\widehat{\mathsf{D}_1}||\widehat{\mathsf{D}_2}))||((\nu x)(!x(y).\widehat{\mathsf{D}_1}||\widehat{\mathsf{D}_3})))$, $\widehat{\mathsf{D}}$ and $\widehat{\mathsf{E}}$ being strongly bisimilar.

The following propositions state the correspondences between the proof terms manipulation rules described above and relations over processes: we omit the proofs, leaving to the reader the verification of each case.

Proposition 1 *Let $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$ and $\Gamma'; \Delta'; \Theta' \vdash \mathsf{E} :: T'$. If $\mathsf{D} \implies \mathsf{E}$, then $\widehat{\mathsf{D}} \to \widehat{\mathsf{E}}$.*

Proposition 2 *Let $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$ and $\Gamma'; \Delta'; \Theta' \vdash \mathsf{E} :: T'$. If $\mathsf{D} \longmapsto \mathsf{E}$, then $\widehat{\mathsf{D}}$ is equivalent to $\widehat{\mathsf{E}}$ modulo structural congruence.*

Proposition 3 *Let $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$ and $\Gamma'; \Delta'; \Theta' \vdash \mathsf{E} :: T'$. If $\mathsf{D} \equiv \mathsf{E}$, then $\widehat{\mathsf{D}}$ is equivalent to $\widehat{\mathsf{E}}$ modulo structural congruence or strong bisimilarity.*

Subject reduction takes the following form:

Theorem 1 *Let $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$. Suppose that $\widehat{\mathsf{D}} = P \to Q$. Then there is E such that $\widehat{\mathsf{E}} = Q$, $\mathsf{D} \hookrightarrow \implies \hookrightarrow \mathsf{E}$ and $\Phi; \Psi; \Theta \vdash \mathsf{E} :: T$, where $\Gamma, \Delta = \Phi, \Psi$.*

In order to prove Theorem 1 we need the followings auxiliary results:

Lemma 4 *Let $\Gamma; \Delta; \Theta \vdash \mathsf{D} \rightsquigarrow P :: x : T$.*

1. *If $P \xrightarrow{\alpha} Q$ and $T = \mathbf{1}$ then $s(\alpha) \neq x$.*

2. *If $P \xrightarrow{\alpha} Q$ and $y : \mathbf{1} \in \Theta$ then $s(\alpha) \neq y$.*

3. *If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $T = A \otimes B$ then $\alpha = \overline{(\nu y)x\langle y\rangle}$.*

4. *If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = y$ and $y : A \otimes B \in \Theta$ then $\alpha = y(z)$.*

- Structural conversions

$(\mathsf{cut}/-/\mathsf{cut}_1):$ $\quad\mathsf{cut}(\mathsf{D}, x.\mathsf{cut}(\mathsf{E}_x, y.\mathsf{F}_y))$ $\quad\equiv\quad \mathsf{cut}(\mathsf{cut}(\mathsf{D}, x.\mathsf{E}_x), y.\mathsf{F}_y)$
$(\mathsf{cut}/-/\mathsf{cut}_2):$ $\quad\mathsf{cut}(\mathsf{D}, x.\mathsf{cut}(\mathsf{E}, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}(\mathsf{E}, x.\mathsf{cut}(\mathsf{D}, y.\mathsf{F}_{xy}))$
$(\mathsf{cut}/-/\mathsf{cut}_!):$ $\quad\mathsf{cut}(\mathsf{D}, x.\mathsf{cut}_!(\mathsf{E}, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}_!(\mathsf{E}, y.\mathsf{cut}(\mathsf{D}, x.\mathsf{F}_{xy}))$
$(\mathsf{cut}/\mathsf{cut}_!/-):$ $\quad\mathsf{cut}(\mathsf{cut}_!(\mathsf{D}, y.\mathsf{E}_y), x.\mathsf{F}_x)$ $\quad\equiv\quad \mathsf{cut}_!(\mathsf{D}, y.\mathsf{cut}(\mathsf{E}_y, x.\mathsf{F}_x))$
$(\mathsf{cut}/-/\mathsf{cut}_\#):$ $\quad\mathsf{cut}(\mathsf{D}, x.\mathsf{cut}_\#(\mathsf{E}, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}_\#(\mathsf{E}, y.\mathsf{cut}(\mathsf{D}, x.\mathsf{F}_{xy}))$
$(\mathsf{cut}/\mathsf{cut}_\#/-):$ $\quad\mathsf{cut}(\mathsf{cut}_\#(\mathsf{D}, y.\mathsf{E}_y), x.\mathsf{F}_x)$ $\quad\equiv\quad \mathsf{cut}_\#(\mathsf{D}, y.\mathsf{cut}(\mathsf{E}_y, x.\mathsf{F}_x))$
$(\mathsf{cut}/\mathbf{1R}/\mathbf{1L}):$ $\quad\mathsf{cut}(\mathbf{1R}, x.\mathbf{1L}(x, \mathsf{D}))$ $\quad\equiv\quad \mathsf{D}$

- Strong bisimilarities

$(\mathsf{cut}_\#/-/\mathsf{cut}):$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{cut}(\mathsf{E}_x, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}(\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E}_x), y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{F}_{xy}))$
$(\mathsf{cut}_\#/-/\mathsf{cut}_\#):$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{cut}_\#(\mathsf{E}_x, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}_\#(\mathsf{D}, x.\mathsf{cut}_\#(\mathsf{E}_x, y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{F}_{xy})))$
$(\mathsf{cut}_\#/-/\mathsf{cut}_!):$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{cut}_!(\mathsf{E}_x, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}_!(\mathsf{E}_x, y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{F}_{xy}))$
$(\mathsf{cut}_!/-/\mathsf{cut}_1):$ $\quad\mathsf{cut}_!(\mathsf{D}, x.\mathsf{cut}(\mathsf{E}_x, y.\mathsf{F}_y))$ $\quad\equiv\quad \mathsf{cut}(\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E}_x), y.\mathsf{F}_y)$
$(\mathsf{cut}_!/-/\mathsf{cut}_2):$ $\quad\mathsf{cut}_!(\mathsf{D}, x.\mathsf{cut}(\mathsf{E}, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}(\mathsf{E}, y.\mathsf{cut}_!(\mathsf{D}, x.\mathsf{F}_{xy}))$
$(\mathsf{cut}_!/-/\mathsf{cut}_!)_1:$ $\quad\mathsf{cut}_!(\mathsf{D}, x.\mathsf{cut}_!(\mathsf{E}_x, y.\mathsf{F}_y))$ $\quad\equiv\quad \mathsf{cut}_!(\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E}_x), y.\mathsf{F}_y)$
$(\mathsf{cut}_!/-/\mathsf{cut}_!)_2:$ $\quad\mathsf{cut}_!(\mathsf{D}, x.\mathsf{cut}_!(\mathsf{E}, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}_!(\mathsf{E}, x.\mathsf{cut}_!(\mathsf{D}, y.\mathsf{F}_{xy}))$
$(\mathsf{cut}_!/-/\mathsf{cut}_\#):$ $\quad\mathsf{cut}_!(\mathsf{D}, x.\mathsf{cut}_\#(\mathsf{E}_x, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}_\#(\mathsf{E}_x, y.\mathsf{cut}_!(\mathsf{D}, x.\mathsf{F}_{xy}))$
$(\mathsf{cut}_\#/-/\mathsf{cut}_\#)_0:$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{cut}_\#(\mathsf{E}_x, y.\mathsf{F}_{xy}))$ $\quad\equiv\quad \mathsf{cut}_\#(\mathsf{E}_x, y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{F}_{xy}))$ if $y \notin FV(\widehat{\mathsf{F}})$
$(\mathsf{cut}_\#/-/-_0):$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E})$ $\quad\equiv\quad \mathsf{E}$ for $x \notin FN(\widehat{\mathsf{E}})$

- Commuting conversions

$(\mathsf{cut}/-/\mathbf{1L}):$ $\quad\mathsf{cut}(\mathsf{D}, x.\mathbf{1L}(y, \mathsf{E}_x))$ $\quad\equiv\quad \mathbf{1L}(y, \mathsf{cut}(\mathsf{D}, x.\mathsf{E}_x))$
$(\mathsf{cut}/-/!\mathsf{L}_!):$ $\quad\mathsf{cut}(\mathsf{D}, x.!\mathsf{L}_!(y.\mathsf{E}_{xz}))$ $\quad\equiv\quad !\mathsf{L}_!(y.\mathsf{cut}(\mathsf{D}, x.\mathsf{E}_{xz}))$
$(\mathsf{cut}/-/!\mathsf{L}_\#):$ $\quad\mathsf{cut}(\mathsf{D}, x.!\mathsf{L}_\#(y.\mathsf{E}_{xz}))$ $\quad\equiv\quad !\mathsf{L}_\#(y.\mathsf{cut}(\mathsf{D}, x.\mathsf{E}_{xz}))$
$(\mathsf{cut}/\mathbf{1L}/-):$ $\quad\mathsf{cut}(\mathbf{1L}(y, \mathsf{D}), x.\mathsf{E}_x)$ $\quad\equiv\quad \mathbf{1L}(y, \mathsf{cut}(\mathsf{D}, x.\mathsf{E}_x))$
$(\mathsf{cut}/!\mathsf{L}_!/-):$ $\quad\mathsf{cut}(!\mathsf{L}_!(y.\mathsf{D}_z), x.\mathsf{E}_x)$ $\quad\equiv\quad !\mathsf{L}_!(y.\mathsf{cut}(\mathsf{D}_z, x.\mathsf{E}_{xz}))$
$(\mathsf{cut}/!\mathsf{L}_\#/-):$ $\quad\mathsf{cut}(!\mathsf{L}_\#(y.\mathsf{D}_z), x.\mathsf{E}_x)$ $\quad\equiv\quad !\mathsf{L}_\#(y.\mathsf{cut}(\mathsf{D}_z, x.\mathsf{E}_{xz}))$
$(\mathsf{cut}_!/-/\mathbf{1L}):$ $\quad\mathsf{cut}_!(\mathsf{D}, x.\mathbf{1L}(y, \mathsf{E}_x))$ $\quad\equiv\quad \mathbf{1L}(y, \mathsf{cut}_!(\mathsf{D}, x.\mathsf{E}_x))$
$(\mathsf{cut}_!/-/!\mathsf{L}_!):$ $\quad\mathsf{cut}_!(\mathsf{D}, x.!\mathsf{L}_!(y.\mathsf{E}_{xz}))$ $\quad\equiv\quad !\mathsf{L}_!(y.\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E}_{xz}))$
$(\mathsf{cut}_!/-/!\mathsf{L}_\#):$ $\quad\mathsf{cut}_!(\mathsf{D}, x.!\mathsf{L}_\#(y.\mathsf{E}_{xz}))$ $\quad\equiv\quad !\mathsf{L}_\#(y.\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E}_{xz}))$
$(\mathsf{cut}_\#/-/\mathbf{1L}):$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.\mathbf{1L}(y, \mathsf{E}_x))$ $\quad\equiv\quad \mathbf{1L}(y, \mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E}_x))$
$(\mathsf{cut}_\#/-/!\mathsf{L}_!):$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.!\mathsf{L}_!(y.\mathsf{E}_{xz}))$ $\quad\equiv\quad !\mathsf{L}_!(y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E}_{xz}))$
$(\mathsf{cut}_\#/-/!\mathsf{L}_\#):$ $\quad\mathsf{cut}_\#(\mathsf{D}, x.!\mathsf{L}_\#(y.\mathsf{E}_{xz}))$ $\quad\equiv\quad !\mathsf{L}_\#(y.\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E}_{xz}))$

Figure 5: Equivalence rules

5. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $T = A \multimap B$ then $\alpha = x(y)$.

6. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = y$ and $y : A \multimap B \in \Theta$ then $\alpha = \overline{(\nu z)y\langle z\rangle}$.

7. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $T = A \& B$ then $\alpha = x.\textbf{\textit{inl}}$; or $\alpha = x.\textbf{\textit{inr}}$;.

8. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = y$ and $y : A \& B \in \Theta$ then $\alpha = \overline{y.\textbf{\textit{inl}}}$; or $\alpha = \overline{y.\textbf{\textit{inr}}}$;.

9. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $T = A \oplus B$ then $\alpha = \overline{x.\textbf{\textit{inl}}}$; or $\alpha = \overline{x.\textbf{\textit{inr}}}$;.

10. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = y$ and $y : A \oplus B \in \Theta$ then $\alpha = y.\textbf{\textit{inl}}$; or $\alpha = y.\textbf{\textit{inr}}$;

11. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $T = !A$ then $\alpha = x(y)$.

12. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = y$ and $y : !A$ or $y \in \Gamma$ or $y \in \Delta$ or $y \in \Phi$ then $\alpha = \overline{(\nu z)y\langle z\rangle}$.

Proof. Trivial from definitions. $\qquad\square$

Lemma 5 *Assume that:*
  1. $\Gamma_1; \Delta; \Theta_1 \vdash \mathsf{D} :: x : A \otimes B$ with $\widehat{\mathsf{D}} = P \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q$;
  2. $\Gamma_2; \Delta; \Theta_2, x : A \otimes B \vdash \mathsf{E} :: z : C$ with $\widehat{\mathsf{E}} = R \xrightarrow{x(y)} S$.
Then:
  1. $\mathsf{cut}(\mathsf{D}, x.\mathsf{E}) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{F}$ *for some* F;
  2. $\Gamma_1, \Gamma_2; \Delta; \Theta_1, \Theta_2 \vdash \mathsf{F} :: z : C$, *where* $\widehat{\mathsf{F}} \equiv (\nu x)(Q||S)$.

Proof. By simultaneous induction on $\mathsf{D}_1, \mathsf{D}_2$. The property stated in the lemma holds also for the system $\pi\mathtt{DILL}$ (see [1]); since the proof technique is essentialy the same modulo some minor details, we omit the proof. $\qquad\square$

Lemma 6 *Assume*

  1. $\Gamma_1; \Delta; \Theta_1 \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : A \multimap B$ with $P_1 \xrightarrow{x(y)} P_1'$

  2. $\Gamma_2; \Delta; \Theta_2, x : A \multimap B \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ with $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q_1'$

Then

  1. $\mathsf{cut}(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{D}$ *for some* D;

  2. $\Gamma_1, \Gamma_2; \Delta; \Theta_1, \Theta_2 \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu x)(\nu y)(P_1'||Q_1')$.

Proof. See the proof of Lemma 5. $\qquad\square$

Lemma 7 *Assume*

  1. $\Gamma_1; \Delta; \Theta_1 \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : !A$ with $P_1 \xrightarrow{x(y)} P_1'$

  2. $\Gamma_2; \Delta; \Theta_2, x : !A \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ with $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q_1'$

Then

  1. $\mathsf{cut}(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{D}$ *for some* D;

  2. $\Gamma_1, \Gamma_2; \Delta; \Theta_1, \Theta_2 \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu x)(\nu y)(P_1'||Q_1')$.

Proof. See the proof of Lemma 5. $\qquad\square$

Lemma 8 *Assume*

1. $\Gamma_1; \Delta; \Theta_1 \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : A \& B$ *with* $P_1 \xrightarrow{x.inl;} P_1'$

2. $\Gamma_2; \Delta; \Theta_2, x : A \& B \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ *with* $Q_1 \xrightarrow{\overline{x.inl;}} Q_1'$

Then

1. $\mathsf{cut}(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{D}$ *for some* D;

2. $\Gamma_1, \Gamma_2; \Delta; \Theta_1, \Theta_2 \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu x)(P_1' \| Q_1')$.

Proof. See the proof of Lemma 5. □

Lemma 9 *Assume*

1. $\Gamma_1; \Delta; \Theta_1 \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : A \oplus B$ *with* $P_1 \xrightarrow{\overline{x.inl;}} P_1'$

2. $\Gamma_2; \Delta; \Theta_2, x : A \oplus B \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ *with* $Q_1 \xrightarrow{x.inl;} Q_1'$

Then

1. $\mathsf{cut}(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{D}$ *for some* D;

2. $\Gamma_1, \Gamma_2; \Delta; \Theta_1, \Theta_2 \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu x)(P_1' \| Q_1')$.

Proof. See the proof of Lemma 5. □

Lemma 10 *Assume*

1. $\Gamma_1; \emptyset; \emptyset \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : A$

2. $\Gamma_2, x : A; \Delta; \Theta \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ *with* $Q_1 \xrightarrow{\overline{(\nu y)x\langle y \rangle}} Q_1'$

Then

1. $\mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D})$ *for some* D *where* $x \notin FV(\widehat{\mathsf{D}})$;

2. $\Gamma; \Phi; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu y)(P_1 \| Q_1')$, *where* $\Gamma, \Phi = \Gamma_1, \Gamma_2, x : A, \Delta$.

Proof. By induction on D_2. We have different cases, depending from the last rules of D_2. Let us just write down some relevant case:

- Suppose $\mathsf{D}_2 = \flat_!(x, y.\mathsf{D}_2')$; then $Q_1 \equiv (\nu y)x\langle y \rangle.Q_1'$ and $\Gamma_2, x : A; \Delta; \Theta \vdash \mathsf{D}_2' \rightsquigarrow Q_1' :: z : C$ by inversion. Now $\mathsf{cut}_!(\mathsf{D}_1, x.\flat_!(x, y.\mathsf{D}_2')) \Longrightarrow \mathsf{cut}(\mathsf{D}_{1\Downarrow}, y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_{2\Downarrow}'))$ by $(\mathsf{cut}_!/-/\flat_!) \equiv \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}(\mathsf{D}_{1\Downarrow}, y.\mathsf{D}_{2\Downarrow}'))$ by $(\mathsf{cut}/-/\mathsf{cut}_\#)$. We pick $\mathsf{D} = \mathsf{cut}(\mathsf{D}_{1\Downarrow}, y.\mathsf{D}_{2\Downarrow}')$; then $\Gamma; \Phi; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ for some $Q_2 \equiv (\nu y)(P_1 \| Q_1')$, where $\Gamma, \Phi = \Gamma_1, \Gamma_2, x : A, \Delta$ q.e.d.

- Suppose $\mathsf{D}_2 = \mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}_2')$; then $\Delta; \emptyset; \emptyset \vdash \mathsf{D}_1' \rightsquigarrow R_1 :: w : C$ and $\Gamma_2, x : A; \Delta; \Theta \vdash \mathsf{D}_2' \rightsquigarrow R_2' :: z : B$ with $Q_1 \xrightarrow{\overline{(\nu y)x\langle y \rangle}} R_1 \| R_2'$, by inversion. Now by induction hypothesis, $\mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{D}_2') \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*)$ for some D^* (where $x \notin FV(\widehat{\mathsf{D}^*})$, and $\Gamma; \Phi; \Theta_2 \vdash \mathsf{D}^* \rightsquigarrow S :: z : B$ for some $S = (\nu y)(P_1 \| R_2')$. $\mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}_2')) \equiv \mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{D}_2'))$ by $(\mathsf{cut}_!/-/\mathsf{cut}_\#)$, $\hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*))$ by congruence, $\equiv \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}^*))$ by $(\mathsf{cut}_\#/-/\mathsf{cut}_\#)_0$. Pick $\mathsf{D} = \mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}^*)$. Then $Q_2 = (\nu y)R_1 \| S$ by cut, and $\Gamma; \Phi; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ for some $Q_2 \equiv (\nu y)(P_1 \| Q_1')$. q.e.d.

□

Corollary 1 *Assume*

1. $\Gamma_1; \emptyset; \emptyset \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : A$

2. $\Gamma_2, x : A; \Delta; \Theta \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ *with* $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q_1'$

Then

1. $\mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{D}$ *for some* D;

2. $\Gamma; \Phi; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu x)(!x(y).P_1 || (\nu y)(P_1 || Q_1'))$, *where* $\Gamma, \Phi = \Gamma_1, \Gamma_2, \Delta$

Proof. Follows from Lemma 10. $\square$

Lemma 11 *Assume*

1. $\Delta; \emptyset; \emptyset \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : A$

2. $\Gamma; \Delta, x : A; \Theta \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ *with* $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q_1'$

Then :

1. $\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D})$ *for some* D;

2. $\Phi; \Psi, x : A; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu x)(\nu y)(P_1 || Q_1')$, *where* $\Phi, \Psi = \Gamma, \Delta$.

Proof. By induction on D_2. We have different cases, depending from the last rules of D_2. Let us just write down some relevant case:

- $\mathsf{D}_2 = \mathsf{cut}(\mathsf{D}_1', y.\mathsf{D}_2')$. Assume $\Gamma = \Gamma_1, \Gamma_2$ and $\Theta = \Theta_1, \Theta_2$. Now $\Gamma_1; \Delta, x : A; \Theta_1 \vdash \mathsf{D}_1' \rightsquigarrow R_1 ::$ $w : B$ and $\Gamma_2; \Delta, x : A; \Theta, w : B \vdash \mathsf{D}_2' \rightsquigarrow R_2 :: z : C$ by inversion. We have two cases:either $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} R_1 || R_2'$, or $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} R_1' || R_2$.

  First case:

  $\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_1') \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*)$ for some D^*; then $\Gamma_1; \Delta, x : A; \Theta_1 \vdash \mathsf{D}^* \rightsquigarrow S ::$ $w : B$ for some $S = (\nu y)(P_1 || R_1')$ by induction hypothesis; $\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}(\mathsf{D}_1', y.\mathsf{D}_2')) \equiv$ $\mathsf{cut}(\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_1'), y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2'))$ by $(\mathsf{cut}_\#/-/\mathsf{cut})$, $\hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}(\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*), y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2'))$ by congruence $\equiv \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}(\mathsf{D}^*, y.\mathsf{D}_2'))$ by $(\mathsf{cut}_\#/-/\mathsf{cut})$. Pick $\mathsf{D} = \mathsf{cut}(\mathsf{D}^*, y.\mathsf{D}_2')$; then $Q_2 = (\nu y)S || R_2$ by cut. Then $\Gamma; \Delta, x : A; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ for some $Q_2 \equiv (\nu y)(P_1 || Q_1')$.

  Second case:

  $\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2') \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*)$ for some D^*; then $\Gamma_2; \Delta, x : A; \Theta_2 \vdash \mathsf{D}^* \rightsquigarrow S ::$ $w : B$ for some $S = (\nu y)(P_1 || R_2')$ by induction hypothesis; $\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}(\mathsf{D}_1', y.\mathsf{D}_2')) \equiv$ $\mathsf{cut}(\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_1'), y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2'))$ by $(\mathsf{cut}_\#/-/\mathsf{cut})$, $\hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}(\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_1'), y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*))$ by congruence, $\equiv \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}(\mathsf{D}_1', y.\mathsf{D}^*))$ by $(\mathsf{cut}_\#/-/\mathsf{cut})$. Pick $\mathsf{D} = \mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}^*)$; then $Q_2 = (\nu y)R_1 || S$ by cut. Then $\Gamma; \Delta, x : A; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ for some $Q_2 \equiv (\nu y)(P_1 || Q_1')$.

- $\mathsf{D}_2 = \mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}_2')$. $\Delta; \emptyset; \emptyset \vdash \mathsf{D}_1' \rightsquigarrow R_1 :: w : B$ $\Gamma; \Delta, x : A, w : B; \Theta \vdash \mathsf{D}_2' \rightsquigarrow R_2 :: z : C$ by inversion. Now $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} R_1 || R_2'$; $\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2') \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*)$ for some D^* and $\Gamma; \Delta, x : A, w : B; \Theta \vdash \mathsf{D}^* \rightsquigarrow S :: w : B$ for some $S = (\nu y)(P_1 || R_2')$ by induction hypothesis. $\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}_2')) \equiv \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}_2')))$ by $(\mathsf{cut}_\#/-/\mathsf{cut}_\#)$ $\hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{D}^*)))$ by congruence, $\equiv \mathsf{cut}_\#(\mathsf{D}_1, x.\mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}^*))$ by $(\mathsf{cut}_\#/-/\mathsf{cut}_\#)$. Pick $\mathsf{D} = \mathsf{cut}_\#(\mathsf{D}_1', y.\mathsf{D}^*)$; then $Q_2 = (\nu y)R_1 || S$ by cut. Then $\Gamma; \Delta, x : A; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ for some $Q_2 \equiv (\nu y)(P_1 || Q_1')$.

$\square$

Corollary 2 *Assume*

1. $\Delta; \emptyset; \emptyset \vdash \mathsf{D}_1 \rightsquigarrow P_1 :: x : A$

2. $\Gamma; x : A, \Delta; \Theta \vdash \mathsf{D}_2 \rightsquigarrow Q_1 :: z : C$ *with* $Q_1 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q_1'$

Then

1. $\mathsf{cut}_{\#}(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{D}$ *for some* D;

2. $\Phi; \Psi; \Theta \vdash \mathsf{D} \rightsquigarrow Q_2 :: z : C$ *for some* $Q_2 \equiv (\nu x)(!x(y).P_1 || (\nu y)(P_1 || Q_1'))$, *where* $\Phi, \Psi = \Gamma, \Delta$.

Proof. Follows from Lemma 11. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Proof of Theorem 1

Proof. We reason by induction on the structure of D. Since $\widehat{\mathsf{D}} = P \to Q$ the only possible last rules of D can be: $\mathbf{1L}, !\mathsf{L}_!, !\mathsf{L}_{\#}$, a linear cut ($\mathsf{cut}$) or an exponential cut ($\mathsf{cut}_!$ or $\mathsf{cut}_{\#}$). In all the other cases, the underlying process can only perform a visible action, as can be easily verified by inspecting the rules from Figure 1. With this observation in mind, let us inspect the operational semantics derivation proving that $P \to Q$. At some point we will find two subprocesses of P, call them R and S, which communicate, causing an internal reduction. We here claim that this can only happen in presence of a cut, and only the communication between R and S must occur along the channel involved in the cut. Now, it's only a matter of showing that the just described situation can be "resolved" preserving types, and this can be done using the previous lemmas. Some relevant case:

- $\mathsf{D} = \mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{D}_2)$; assume $\Gamma = \Gamma_1, \Gamma_2$ and $P \equiv (\nu x)!x(w).P_1 || P_2$. Now $\Gamma_1; \emptyset; \emptyset \vdash \emptyset :: \mathsf{D}_1 \rightsquigarrow P_1 x : C$ and $\Gamma_2, x : A; \Delta; \Theta \vdash \mathsf{D}_2 \rightsquigarrow P_2 :: z : A$, by inversion; from $P \to Q$ either $P_2 \to Q_2$ and $Q = (\nu x)!x(w).P_1 || Q_2$ or $Q_2 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q_2$ and $(\nu x)!x(w).P_1 || (\nu y)P_1 || Q_2$.

  First case:

  $\Gamma_2, x : A; \Delta; \Theta \vdash \mathsf{E}_2 \rightsquigarrow Q_2 :: z : A$ for some E_2 with $\mathsf{D}_2 \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{E}_2$ by i.h.; $\mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{E}_2)$ by congruence. Pick $\mathsf{E} = \mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{E}_2)$; then $\Gamma; \Delta; \Theta \vdash \mathsf{E} \rightsquigarrow Q :: z : A$ by $\mathsf{cut}_!$.

  Second case:

  $\mathsf{cut}_!(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{E}$ for some E; then $\Gamma; \Delta; \Theta \vdash \mathsf{E} \rightsquigarrow R :: z : A$ for some $R \equiv Q$ by Corollary 1.

- $\mathsf{D} = \mathsf{cut}_{\#}(\mathsf{D}_1, x.\mathsf{D}_2)$. Now, $P \equiv (\nu x)!x(w).P_1 || P_2$ and $\Delta; \emptyset; \emptyset \vdash \emptyset :: \mathsf{D}_1 \rightsquigarrow P_1 x : C$, $\Gamma; \Delta, x : A; \Theta \vdash \mathsf{D}_2 \rightsquigarrow P_2 :: z : A$, by inversion; from $P \to Q$ either $P_2 \to Q_2$ and $Q = (\nu x)!x(w).P_1 || Q_2$ or $Q_2 \xrightarrow{\overline{(\nu y)x\langle y\rangle}} Q_2$ and $(\nu x)!x(w).P_1 || (\nu y)P_1 || Q_2$

  First case:

  $\Gamma; \Delta, x : A; \Theta \vdash \mathsf{E}_2 \rightsquigarrow Q_2 :: z : A$ for some E_2 with $\mathsf{D}_2 \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{E}_2$ by i.h. and $\mathsf{cut}_{\#}(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{cut}_{\#}(\mathsf{D}_1, x.\mathsf{E}_2)$ by congruence. Pick $\mathsf{E} = \mathsf{cut}_{\#}(\mathsf{D}_1, x.\mathsf{E}_2)$; then $\Gamma; \Delta; \Theta \vdash \mathsf{E} \rightsquigarrow Q :: z : A$ by $\mathsf{cut}_{\#}$

  Second case:

  $\mathsf{cut}_{\#}(\mathsf{D}_1, x.\mathsf{D}_2) \hookrightarrow \Longrightarrow \hookrightarrow \mathsf{E}$ for some E; then $\Gamma; \Delta; \Theta \vdash \mathsf{E} \rightsquigarrow R :: z : A$ for some $R \equiv Q$ by Corollary 2.

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

5 Proving Polynomial Bounds

In this section, we prove the main result of this paper, namely some polynomial bounds on the length of internal reduction sequences and on the size of intermediate results for processes typable in πDILL. In other words, interaction will be shown to be bounded. The simplest formulation of this result is the following:

Theorem 2 *For every type A, there is a polynomial p_A such that whenever $\emptyset; \emptyset; x : A \vdash \mathsf{D} :: y : \mathbf{1}$ and $\emptyset; \emptyset; \emptyset \vdash \mathsf{E} :: x : A$ where D and E are normal and $(\nu x)(\widehat{\mathsf{D}} || \widehat{\mathsf{E}}) \to^n P$, it holds that $n, |P| \leq p_A(|\widehat{\mathsf{D}}| + |\widehat{\mathsf{E}}|)$*

Intuitively, what Theorem 2 says is that the complexity of the interaction between two processes typable without cuts and communicating through a channel with session type A is polynomial in their sizes, where the specific polynomial involved only depends on A itself. In other words, the complexity of the interaction is not only bounded, but can be somehow "read off" from the types of the communicating parties.

How does the proof of Theorem 2 look like? Conceptually, it can be thought of as being structured into four steps:

1. First of all, a natural number $\mathbb{W}(\mathsf{D})$ is attributed to any proof term D. $\mathbb{W}(\mathsf{D})$ is said to be the *weight* of D.
2. Secondly, the weight of any proof term is shown to strictly decrease along computational reduction, not to increase along shifting reduction and to stay the same for equivalent proof terms.
3. Thirdly, $\mathbb{W}(\mathsf{D})$ is shown to be bounded by a polynomial on $|\widehat{\mathsf{D}}|$, where the exponent only depends on the nesting depth of boxes of D, denoted $\mathbb{B}(\mathsf{D})$.
4. Finally, the box depth $\mathbb{B}(\mathsf{D})$ of any proof term D is shown to be "readable" from its type interface.

This is exactly what we are going to do in the rest of this section. Please observe how points 1–3 above allow to prove the following stronger result, from which Theorem 2 easily follows, given point 4:

Proposition 4 *For every $n \in \mathbb{N}$, there is a polynomial p_n such that for every process P with $\Gamma; \Delta; \Theta \vdash P :: T$, if $P \to^m Q$, then $m, |Q| \leq p_{\mathbb{B}(P)}(|P|)$.*

5.1 Preliminary Definitions

Some concepts have to be precisely defined before we can embark in the proof of Proposition 4. First of all, we need to define what the box-depth of a process is. Simply, given a process P, its *box-depth* $\mathbb{B}(P)$ is the nesting-level of replications[2] in P. As an example, the box-depth of $!x(y).!z(w).0$ is 2, while the one of $(\nu x)y(z)$ is 0. Formally,

$$
\begin{aligned}
\mathbb{B}(\mathbf{1}\mathsf{L}(x, \mathsf{D})) &= \mathbb{B}(\mathsf{D}) & \mathbb{B}(\oplus\mathsf{R}_1(\mathsf{D})) &= \mathbb{B}(\mathsf{D}) \\
\mathbb{B}(\mathbf{1}\mathsf{R}) &= 0 & \mathbb{B}(\oplus\mathsf{R}_2(\mathsf{D})) &= \mathbb{B}(\mathsf{D}) \\
\mathbb{B}(\otimes\mathsf{L}(x, y.z.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) & \mathbb{B}(\flat_!(x, y.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) \\
\mathbb{B}(\otimes\mathsf{R}(\mathsf{D}, \mathsf{E})) &= \max\{\mathbb{B}(\mathsf{D}), \mathbb{B}(\mathsf{E})\} & \mathbb{B}(\flat_\#(x, y.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) \\
\mathbb{B}(\multimap\mathsf{L}(x, \mathsf{D}, y.\mathsf{E})) &= \max\{\mathbb{B}(\mathsf{D}), \mathbb{B}(\mathsf{E})\} & \mathbb{B}(!\mathsf{L}_!(x.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) \\
\mathbb{B}(\multimap\mathsf{R}(x.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) & \mathbb{B}(!\mathsf{L}_\#(x.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) \\
\mathbb{B}(\&\mathsf{L}_1(x, y.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) & \mathbb{B}(!\mathsf{R}(x_1, \ldots, x_n, \mathsf{D})) &= 1 + \mathbb{B}(\mathsf{D}) \\
\mathbb{B}(\&\mathsf{L}_2(x, y.\mathsf{D})) &= \mathbb{B}(\mathsf{D}) & \mathbb{B}(\mathsf{cut}(\mathsf{D}, x.\mathsf{E})) &= \max\{\mathbb{B}(\mathsf{D}), \mathbb{B}(\mathsf{E})\} \\
\mathbb{B}(\&\mathsf{R}(\mathsf{D}, \mathsf{E})) &= \max\{\mathbb{B}(\mathsf{D}), \mathbb{B}(\mathsf{E})\} & \mathbb{B}(\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})) &= \max\{\mathbb{B}(\mathsf{D}) + 1, \mathbb{B}(\mathsf{E})\} \\
\mathbb{B}(\oplus\mathsf{L}(x, y.\mathsf{D}, z.\mathsf{E})) &= \max\{\mathbb{B}(\mathsf{D}), \mathbb{B}(\mathsf{E})\} & \mathbb{B}(\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E})) &= \max\{\mathbb{B}(\mathsf{D}) + 1, \mathbb{B}(\mathsf{E})\}
\end{aligned}
$$

---

[2]This terminology is derived from linear logic, where proofs obtained by the promotion rule are usually called boxes

Analogously, the box-depth of a proof term D is simply $\mathbb{B}(\widehat{\mathsf{D}})$.

Now, suppose that $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$ and that $x : A$ belongs to either Γ or Δ, i.e. that x is an "exponential" channel in D. A key parameter is the *virtual number of occurrences* of x in D, which is denoted as $\mathbb{FO}(x, \mathsf{D})$. This parameter, as its name suggests, is not simply the number of literal occurrences of x in D, but takes into account possible duplications derived from cuts. So, for example, $\mathbb{FO}(w, \mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})) = \mathbb{FO}(x, \mathsf{E}) \cdot \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$, while $\mathbb{FO}(w, \otimes\mathsf{R}(\mathsf{D}, \mathsf{E}))$ is merely $\mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$. Obviously, $\mathbb{FO}(w, \flat_!(x, w.\mathsf{D})) = 1$ and $\mathbb{FO}(w, \flat_\#(x, w.\mathsf{D})) = 1$. Formally:

$$\mathbb{FO}(w, \mathbf{1}\mathsf{L}(x, \mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, \mathbf{1}\mathsf{R}) = 0$$
$$\mathbb{FO}(w, \otimes\mathsf{L}(x, y.z.\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, \otimes\mathsf{R}(\mathsf{D}, \mathsf{E})) = \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$$
$$\mathbb{FO}(w, \multimap \mathsf{L}(x, \mathsf{D}, y.\mathsf{E})) = \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$$
$$\mathbb{FO}(w, \multimap \mathsf{R}(x.\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, \mathsf{cut}(\mathsf{D}, x.\mathsf{E})) = \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$$
$$\mathbb{FO}(w, \mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})) = \mathbb{FO}(x, \mathsf{E}) \cdot \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$$
$$\mathbb{FO}(w, \mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E})) = \mathbb{FO}(x, \mathsf{E}) \cdot \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$$
$$\mathbb{FO}(w, \flat_!(x, w.\mathsf{D})) = 1$$
$$\mathbb{FO}(w, \flat_\#(x, w.\mathsf{D})) = 1$$
$$\mathbb{FO}(w, \flat_!(x, y.\mathsf{D})) = 0$$
$$\mathbb{FO}(w, \flat_\#(x, y.\mathsf{D})) = 0$$
$$\mathbb{FO}(w, !\mathsf{L}_!(x.\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, !\mathsf{L}_\#(x.\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, !\mathsf{R}(x_1, \ldots, x_n, \mathsf{D})) = 0$$
$$\mathbb{FO}(w, \oplus\mathsf{L}(x, y.\mathsf{D}, z.\mathsf{E})) = \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$$
$$\mathbb{FO}(w, \oplus\mathsf{R}_1(\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, \oplus\mathsf{R}_2(\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, \&\mathsf{L}_1(x, y.\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, \&\mathsf{L}_2(x, y.\mathsf{D})) = \mathbb{FO}(w, \mathsf{D})$$
$$\mathbb{FO}(w, \&\mathsf{R}(\mathsf{D}, \mathsf{E})) = \mathbb{FO}(w, \mathsf{D}) + \mathbb{FO}(w, \mathsf{E})$$

A channel in either the auxiliary or the exponential context can "float" to the linear context as an effect of rules $!\mathsf{L}_!$ or $!\mathsf{L}_\#$. From that moment on, it can only be treated as a linear channel. As a consequence, it makes sense to define the *duplicability factor* of a proof term D, written $\mathbb{D}(\mathsf{D})$, simply as the maximum of $\mathbb{FO}(x, \mathsf{D})$ over all instances of the rules $!\mathsf{L}_!$ or $!\mathsf{L}_\#$ in D, where x is the involved channel. For example, $\mathbb{D}(!\mathsf{L}_!(x.\mathsf{D})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{FO}(y, \mathsf{D})\}$ and $\mathbb{D}(\multimap \mathsf{L}(x, \mathsf{D}, y.\mathsf{E})) =$

max$\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\}$. Formally, the duplicability factor $\mathbb{D}(\mathsf{D})$ of D is defined as follows

$$\mathbb{D}(\mathbf{1}\mathsf{L}(x, \mathsf{D})) = \mathbb{D}(\mathsf{D}) \qquad\qquad \mathbb{D}(\oplus\mathsf{R}_1(\mathsf{D})) = \mathbb{D}(\mathsf{D})$$
$$\mathbb{D}(\mathbf{1}\mathsf{R}) = 0 \qquad\qquad \mathbb{D}(\oplus\mathsf{R}_2(\mathsf{D})) = \mathbb{D}(\mathsf{D})$$
$$\mathbb{D}(\otimes\mathsf{L}(x, y.z.\mathsf{D})) = \mathbb{D}(\mathsf{D}) \qquad\qquad \mathbb{D}(\flat_!(x, y.\mathsf{D})) = \mathbb{D}(\mathsf{D})$$
$$\mathbb{D}(\otimes\mathsf{R}(\mathsf{D}, \mathsf{E})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\} \qquad\qquad \mathbb{D}(\flat_\#(x, y.\mathsf{D})) = \mathbb{D}(\mathsf{D})$$
$$\mathbb{D}(\multimap\mathsf{L}(x, \mathsf{D}, y.\mathsf{E})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\} \qquad\qquad \mathbb{D}(!\mathsf{L}_!(x.\mathsf{D})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{FO}(y, \mathsf{D})\}$$
$$\mathbb{D}(\multimap\mathsf{R}(x.\mathsf{D})) = \mathbb{D}(\mathsf{D}) \qquad\qquad \mathbb{D}(!\mathsf{L}_\#(x.\mathsf{D})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{FO}(y, \mathsf{D})\}$$
$$\mathbb{D}(\&\mathsf{L}_1(x, y.\mathsf{D})) = \mathbb{D}(\mathsf{D}) \qquad\qquad \mathbb{D}(!\mathsf{R}(x_1, \ldots, x_n, \mathsf{D})) = \mathbb{D}(\mathsf{D})$$
$$\mathbb{D}(\&\mathsf{L}_2(x, y.\mathsf{D})) = \mathbb{D}(\mathsf{D}) \qquad\qquad \mathbb{D}(\mathsf{cut}(\mathsf{D}, x.\mathsf{E})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\}$$
$$\mathbb{D}(\&\mathsf{R}(\mathsf{D}, \mathsf{E})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\} \qquad\qquad \mathbb{D}(\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\}$$
$$\mathbb{D}(\oplus\mathsf{L}(x, y.\mathsf{D}, z.\mathsf{E})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\} \qquad\qquad \mathbb{D}(\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E})) = \max\{\mathbb{D}(\mathsf{D}), \mathbb{D}(\mathsf{E})\}$$

It's now possible to give the definition of $\mathbb{W}(\mathsf{D})$, namely the *weight* of the proof term D. Before doing that, however, it is necessary to give a parameterized notion of weight, denoted $\mathbb{W}_n(\mathsf{D})$. Intuitively, $\mathbb{W}_n(\mathsf{D})$ is defined similarly to $|\widehat{\mathsf{D}}|$. However, every input and output action in $\widehat{\mathsf{D}}$ can possibly count more than one:

- Everything inside D in $!\mathsf{R}(x_1, \ldots, x_n, \mathsf{D})$ counts for n;
- Everything inside D in either $\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})$ or $\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E})$ counts for $\mathbb{FO}(x, \mathsf{E})$.

For example, $\mathbb{W}_n(\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E})) = \mathbb{FO}(x, \mathsf{E}) \cdot \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$, while $\mathbb{W}_n(\&\mathsf{L}_2(x, y.\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$. Formally:

$$\mathbb{W}_n(\mathbf{1}\mathsf{L}(x, \mathsf{D})) = \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\mathbf{1}\mathsf{R}) = 0$$
$$\mathbb{W}_n(\otimes\mathsf{L}(x, y.z.\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\otimes\mathsf{R}(\mathsf{D}, \mathsf{E})) = 1 + \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$$
$$\mathbb{W}_n(\multimap\mathsf{L}(x, \mathsf{D}, y.\mathsf{E})) = 1 + \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$$
$$\mathbb{W}_n(\multimap\mathsf{R}(x.\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\mathsf{cut}(\mathsf{D}, x.\mathsf{E})) = \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$$
$$\mathbb{W}_n(\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})) = \mathbb{FO}(x, \mathsf{E}) \cdot \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$$
$$\mathbb{W}_n(\mathsf{cut}_\#(\mathsf{D}, x.\mathsf{E})) = \mathbb{FO}(x, \mathsf{E}) \cdot \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$$
$$\mathbb{W}_n(\flat_!(x, y.\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\flat_\#(x, y.\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(!\mathsf{L}_!(x.\mathsf{D})) = \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(!\mathsf{L}_\#(x.\mathsf{D})) = \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(!\mathsf{R}(x_1, \ldots, x_n, \mathsf{D})) = n \cdot (\mathbb{W}_n(\mathsf{D}) + 1)$$
$$\mathbb{W}_n(\oplus\mathsf{L}(x, y.\mathsf{D}, z.\mathsf{E})) = 1 + \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$$
$$\mathbb{W}_n(\oplus\mathsf{R}_1(\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\oplus\mathsf{R}_2(\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\&\mathsf{L}_1(x, y.\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\&\mathsf{L}_2(x, y.\mathsf{D})) = 1 + \mathbb{W}_n(\mathsf{D})$$
$$\mathbb{W}_n(\&\mathsf{R}(\mathsf{D}, \mathsf{E})) = 1 + \mathbb{W}_n(\mathsf{D}) + \mathbb{W}_n(\mathsf{E})$$

Now, $\mathbb{W}(\mathsf{D})$ is simply $\mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{D})$.

5.2 Monotonicity Results

The crucial ingredient for proving polynomial bounds are a series of results about how the weight D evolves when D is put in relation with another proof term E by way of either $\implies$, $\longmapsto$ or $\equiv$.

Lemma 12 *For every* D, $\mathbb{D}(\mathsf{D}) = \mathbb{D}(\mathsf{D}_\Downarrow)$ *and for every* n, $\mathbb{W}_n(\mathsf{D}) = \mathbb{W}_n(\mathsf{D}_\Downarrow)$.

Whenever a proof term D computationally reduces to E, the underlying weight is guaranteed to strictly decrease:

Proposition 5 *If* $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$ *and* $\mathsf{D} \implies \mathsf{E}$, *then* $\Phi; \Psi; \Theta \vdash \mathsf{E} :: T$ *(where* $\Gamma, \Delta = \Phi, \Psi$*),* $\mathbb{D}(\mathsf{E}) \le \mathbb{D}(\mathsf{D})$ *and* $\mathbb{W}(\mathsf{E}) < \mathbb{W}(\mathsf{D})$.

Proof. By induction on the proof that $\mathsf{D} \implies \mathsf{E}$. Some interesting cases:

- Suppose that $\mathsf{D} = \mathsf{cut}(\multimap \mathsf{R}(y.\mathsf{F}), x. \multimap \mathsf{L}(x, \mathsf{G}, x.\mathsf{H})) \implies \mathsf{cut}(\mathsf{cut}(\mathsf{G}, y.\mathsf{F}), x.\mathsf{H}) = \mathsf{E}$. Then

$$\mathbb{D}(\mathsf{D}) = \max\{\mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{G}), \mathbb{D}(\mathsf{H})\} = \mathbb{D}(\mathsf{E});$$
$$\mathbb{W}(\mathsf{D}) = \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{D}) = 3 + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{G}) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{H})$$
$$> 2 + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{G}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{H}) = \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{E}) = \mathbb{W}(\mathsf{E}).$$

- Suppose that $\mathsf{D} = \mathsf{cut}(\&\mathsf{R}(\mathsf{F}, \mathsf{G}), x.\&\mathsf{L}_1(x, y.\mathsf{H})) \implies \mathsf{cut}(\mathsf{F}, x.\mathsf{H}) = \mathsf{E}$. Then

$$\mathbb{D}(\mathsf{D}) = \max\{\mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{G}), \mathbb{D}(\mathsf{H})\} = \mathbb{D}(\mathsf{E});$$
$$\mathbb{W}(\mathsf{D}) = \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{D}) = 3 + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{G}) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{H})$$
$$> 2 + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{G}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{H}) = \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{E}) = \mathbb{W}(\mathsf{E}).$$

- Suppose that $\mathsf{D} = \mathsf{cut}_!(\mathsf{F}, x.\flat_!(x, y.\mathsf{G})) \implies \mathsf{cut}(\mathsf{F}_\Downarrow, y.\mathsf{cut}_\#(\mathsf{F}, x.\mathsf{G}_\Downarrow)) = \mathsf{E}$. Then,

$$\mathbb{D}(\mathsf{D}) = \max\{\mathbb{D}(\mathsf{F}_\Downarrow), \mathbb{D}(\mathsf{G}_\Downarrow)\} = \max\{\mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{G})\} = \mathbb{D}(\mathsf{E});$$
$$\mathbb{W}(\mathsf{D}) = \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{D}) = \mathbb{F}\mathbb{O}(x, \flat_!(x, y.\mathsf{G})) \cdot \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{F}_\Downarrow) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\flat_!(x, y.\mathsf{G}))$$
$$= \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\flat_!(x, y.\mathsf{G})) = \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{F}) + 1 + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{G}) \ge \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + 1 + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{G})$$
$$> \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{G}) = \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + 0 \cdot \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{G})$$
$$= \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + \mathbb{F}\mathbb{O}(x, \mathsf{G}) \cdot \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{G})$$
$$= \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{E}) = \mathbb{W}(\mathsf{E}).$$

- Suppose that
$$\mathsf{D} = \mathsf{cut}_\#(\mathsf{F}, x.\flat_\#(x, y.\mathsf{G})) \implies \mathsf{cut}(\mathsf{F}_\Downarrow, y.\mathsf{cut}_\#(\mathsf{F}, x.\mathsf{G})) = \mathsf{E}.$$

Then we can proceed exactly as in the previous case.

This concludes the proof. $\qquad\qquad\square$

Shift reduction, on the other hand, is *not* guaranteed to induce a strict decrease on the underlying weight which, however, cannot increase:

Proposition 6 *If* $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$ *and* $\mathsf{D} \longmapsto \mathsf{E}$, *then* $\Gamma; \Delta; \Theta \vdash \mathsf{E} :: T$, $\mathbb{D}(\mathsf{E}) \le \mathbb{D}(\mathsf{D})$ *and* $\mathbb{W}(\mathsf{E}) \le \mathbb{W}(\mathsf{D})$.

Proof. By induction on the proof that $\mathsf{D} \longmapsto \mathsf{E}$. Some interesting cases:

- Suppose that
$$\mathsf{D} = \mathsf{cut}(!\mathsf{R}(x_1, \ldots, x_n, \mathsf{F}), x.!\mathsf{L}_!(x.\mathsf{G})) \longmapsto !\mathsf{L}_!(x_1.!\mathsf{L}_!(x_2. \ldots .!\mathsf{L}_!(x_n.\mathsf{cut}_!(\mathsf{F}, y.\mathsf{G})))) = \mathsf{E}.$$

Then

$$\mathbb{D}(\mathsf{D}) = \max\{\mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{G})\} = \mathbb{D}(\mathsf{E})$$
$$\mathbb{W}(\mathsf{D}) = \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{D}) = \mathbb{D}(\mathsf{D}) \cdot \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{G})$$
$$\ge \mathbb{F}\mathbb{O}(y, \mathsf{G}) \cdot \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{D})}(\mathsf{G}) = \mathbb{F}\mathbb{O}(y, \mathsf{G}) \cdot \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{G})$$
$$= \mathbb{W}_{\mathbb{D}(\mathsf{E})}(\mathsf{E}) = \mathbb{W}(\mathsf{E}).$$

- Suppose that

$$D = \mathsf{cut}(!\mathsf{R}(x_1, \ldots, x_n, \mathsf{F}), x.!\mathsf{L}_\#(x.\mathsf{G})) \longmapsto \; !\mathsf{L}_\#(x_1.!\mathsf{L}_\#(x_2.\ldots.!\mathsf{L}_\#(x_n.\mathsf{cut}_\#(\mathsf{F}, y.\mathsf{G})))) = E.$$

Then we can proceed as in the previous case.
This concludes the proof. □

Finally, equivalence leaves the weight unchanged:

Proposition 7 *If* $\Gamma; \Delta; \Theta \vdash D :: T$ *and* $D \equiv E$, *then* $\Gamma; \Delta; \Theta \vdash E :: T$, $\mathbb{D}(E) = \mathbb{D}(D)$ *and* $\mathbb{W}(E) = \mathbb{W}(D)$.

Proof. By induction on the proof that $D \equiv E$. Some interesting cases:
- Suppose that
$$D = \mathsf{cut}(\mathsf{F}, x.\mathsf{cut}(\mathsf{G}_x, y.\mathsf{H}_y)) \equiv \mathsf{cut}(\mathsf{cut}(\mathsf{F}, x.\mathsf{G}_x), y.\mathsf{H}_y) = E.$$

Then:

$$\mathbb{D}(D) = \max\{\mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{G}_x), \mathbb{D}(\mathsf{H}_y)\} = \mathbb{D}(E)$$
$$\mathbb{W}(D) = \mathbb{W}_{\mathbb{D}(D)}(D) = \mathbb{W}_{\mathbb{D}(D)}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{G}_x) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{H}_y)$$
$$= \mathbb{W}_{\mathbb{D}(E)}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(E)}(\mathsf{G}_x) + \mathbb{W}_{\mathbb{D}(E)}(\mathsf{H}_y) = \mathbb{W}_{\mathbb{D}(E)}(E) = \mathbb{W}(E).$$

- Suppose that
$$D = \mathsf{cut}(\mathsf{F}, x.\mathsf{cut}(\mathsf{G}, y.\mathsf{H}_{xy})) \equiv \mathsf{cut}(\mathsf{G}, x.\mathsf{cut}(\mathsf{F}, y.\mathsf{H}_{xy})) = E.$$

Then we can proceed as in the previous case.
- Suppose that
$$D = \mathsf{cut}(\mathsf{F}, x.\mathsf{cut}_!(\mathsf{G}, y.\mathsf{H}_{xy})) \equiv \mathsf{cut}_!(\mathsf{G}, y.\mathsf{cut}(\mathsf{F}, x.\mathsf{H}_{xy})) = E.$$

Then, since $\mathbb{FO}(y, \mathsf{F}) = 0$

$$\mathbb{D}(D) = \max\{\mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{G}), \mathbb{D}(\mathsf{H}_{xy})\} = \mathbb{D}(E)$$
$$\mathbb{W}(D) = \mathbb{W}_{\mathbb{D}(D)}(D) = \mathbb{W}_{\mathbb{D}(D)}(\mathsf{F}) + \mathbb{FO}(y, \mathsf{H}_{xy}) \cdot \mathbb{W}_{\mathbb{D}(D)}(\mathsf{G}) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{H}_{xy})$$
$$= \mathbb{W}_{\mathbb{D}(D)}(\mathsf{F}) + \mathbb{FO}(y, \mathsf{cut}(\mathsf{F}, x.\mathsf{H}_{xy})) \cdot \mathbb{W}_{\mathbb{D}(D)}(\mathsf{G}) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{H}_{xy})$$
$$= \mathbb{W}_{\mathbb{D}(E)}(\mathsf{F}) + \mathbb{FO}(y, \mathsf{cut}(\mathsf{F}, x.\mathsf{H}_{xy})) \cdot \mathbb{W}_{\mathbb{D}(E)}(\mathsf{G}) + \mathbb{W}_{\mathbb{D}(E)}(\mathsf{H}_{xy})$$
$$= \mathbb{W}_{\mathbb{D}(E)}(E) = \mathbb{W}(E)$$

- Suppose that

$$D = \mathsf{cut}_\#(\mathsf{F}, x.\mathsf{cut}(\mathsf{G}_x, y.\mathsf{H}_{xy})) \equiv \mathsf{cut}(\mathsf{cut}_\#(\mathsf{F}, x.\mathsf{G}_x), y.\mathsf{cut}_\#(\mathsf{F}, x.\mathsf{H}_{xy})) = E.$$

Then:

$$\mathbb{D}(D) = \max\{\mathbb{D}(\mathsf{F}), \mathbb{D}(\mathsf{G}_x), \mathbb{D}(\mathsf{H}_{xy})\} = \mathbb{D}(E)$$
$$\mathbb{W}(D) = \mathbb{FO}(x, \mathsf{cut}(\mathsf{G}_x, y.\mathsf{H}_{xy})) \cdot \mathbb{W}_{\mathbb{D}(D)}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{G}_x) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{H}_{xy})$$
$$= (\mathbb{FO}(x, \mathsf{G}_x) + \mathbb{FO}(x, \mathsf{H}_{xy})) \cdot \mathbb{W}_{\mathbb{D}(D)}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{G}_x) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{H}_{xy})$$
$$= (\mathbb{FO}(x, \mathsf{G}_x) \cdot \mathbb{W}_{\mathbb{D}(D)}(\mathsf{F}) + \mathbb{FO}(x, \mathsf{H}_{xy})) \cdot \mathbb{W}_{\mathbb{D}(D)}(\mathsf{F}) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{G}_x) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{H}_{xy})$$
$$= \mathbb{W}_{\mathbb{D}(D)}(\mathsf{cut}_\#(\mathsf{F}, x.\mathsf{G}_x)) + \mathbb{W}_{\mathbb{D}(D)}(\mathsf{cut}_\#(\mathsf{F}, x.\mathsf{H}_{xy}))$$
$$= \mathbb{W}_{\mathbb{D}(D)}(E) = \mathbb{W}_{\mathbb{D}(E)}(E) = \mathbb{W}(E).$$

This concludes the proof. □

Now, consider again the subject reduction theorem (Theorem 1): what it guarantees is that whenever $P \to Q$ and $\widehat{D} = P$, there is E with $\widehat{E} = Q$ and $D \hookrightarrow \Longrightarrow \hookrightarrow E$. In view of the three propositions we have just stated and proved, it's clear that $\mathbb{W}(D) > E$. Altogether, this implies that $\mathbb{W}(D)$ is an upper bound on the number or internal reduction steps $\widehat{D}$ can perform. But is $\mathbb{W}(D)$ itself bounded?

5.3 Bounding the Weight

What kind of bounds can we expect to prove for $\mathbb{W}(\mathsf{D})$? More specifically, how related are $\mathbb{W}(\mathsf{D})$ and $|\widehat{\mathsf{D}}|$?

Lemma 13 *Suppose* $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$. *Then*

1. *If* $x \in \Gamma$, *then* $\mathbb{FO}(x, \mathsf{D}) \leq 1$;
2. *If* $x \in \Delta$, *then* $\mathbb{FO}(x, \mathsf{D}) \leq |\mathsf{D}|$;
3. *If* $x \in \Theta$, *then* $\mathbb{FO}(x, \mathsf{D}) = 0$;

Proof. By induction on the structure of a type derivation π for $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$. Some interesting cases:

- If π is

$$\frac{\rho_1 : \Gamma_1; \Delta; \Theta_1 \vdash \mathsf{D}_1 :: z : A \quad \rho_2 : \Gamma_2; \Delta; \Theta_2 \vdash \mathsf{D}_2 :: y : B}{\Gamma_1, \Gamma_2; \Delta; \Theta_1, \Theta_2 \vdash \otimes\mathsf{R}(\mathsf{D}_1, \mathsf{D}_2) :: y : A \otimes B} \ \otimes\mathsf{R}$$

  then

$$\begin{aligned}
\mathbb{FO}(x, \otimes\mathsf{R}(\mathsf{D}_1, \mathsf{D}_2)) &= \mathbb{FO}(x, \mathsf{D}_1) \leq 1 && \text{if } x \in \Gamma_1 \\
\mathbb{FO}(x, \otimes\mathsf{R}(\mathsf{D}_1, \mathsf{D}_2)) &= \mathbb{FO}(x, \mathsf{D}_2) \leq 1 && \text{if } x \in \Gamma_2 \\
\mathbb{FO}(x, \otimes\mathsf{R}(\mathsf{D}_1, \mathsf{D}_2)) &= \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_1) \\
&\leq |\mathsf{D}_1| + |\mathsf{D}_2| \leq |\otimes\mathsf{R}(\mathsf{D}_1, \mathsf{D}_2)| && \text{if } x \in \Delta \\
\mathbb{FO}(x, \otimes\mathsf{R}(\mathsf{D}_1, \mathsf{D}_2)) &= \mathbb{FO}(x, \mathsf{D}_1) = 0 && \text{if } x \in \Theta_1 \\
\mathbb{FO}(x, \otimes\mathsf{R}(\mathsf{D}_1, \mathsf{D}_2)) &= \mathbb{FO}(x, \mathsf{D}_2) = 0 && \text{if } x \in \Theta_2
\end{aligned}$$

- If π is

$$\frac{\Gamma_1; \emptyset; \emptyset \vdash \emptyset :: \mathsf{D}_1 z : A \quad \Gamma_2; \Gamma_1, y : A; \Theta \vdash \mathsf{D}_2 :: T}{\Gamma_2; \Gamma_1; \Theta \vdash \mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{D}_2) :: T} \ \mathsf{cut}_\#$$

  then:

$$\begin{aligned}
\mathbb{FO}(x, \mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{D}_2)) &= \mathbb{FO}(y, \mathsf{D}_2) \cdot \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_2) \\
&\leq |\mathsf{D}_2| \cdot 1 + |\mathsf{D}_1| \leq |\mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{E}_2)| && \text{if } x \in \Gamma_1 \\
\mathbb{FO}(x, \mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{D}_2)) &= \mathbb{FO}(y, \mathsf{D}_2) \cdot \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_2) \\
&\leq |\mathsf{D}_2| \cdot 0 + 1 = 1 && \text{if } x \in \Gamma_2 \\
\mathbb{FO}(x, \mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{E}_2)) &= \mathbb{FO}(y, \mathsf{D}_2) \cdot \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_2) \\
&\leq |\mathsf{D}_2| \cdot 0 + 1 = 1 && \text{if } x \in \Theta
\end{aligned}$$

- If π is

$$\frac{\Gamma_1; \emptyset; \emptyset \vdash \emptyset :: \mathsf{D}_1 z : A \quad \Gamma_2; \Delta; \Theta \vdash \mathsf{D}_2 :: T}{\Gamma_2; \Delta; \Theta \vdash \mathsf{cut}_w(\mathsf{D}_1, y.\mathsf{D}_2) :: T} \ \mathsf{cut}_w$$

  then:

$$\begin{aligned}
\mathbb{FO}(x, \mathsf{cut}_w(\mathsf{D}_1, y.\mathsf{D}_2)) &= \mathbb{FO}(y, \mathsf{D}_2) \cdot \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_2) \\
&\leq 0 \cdot 1 + 0 = 0 && \text{if } x \in \Gamma_1 \\
\mathbb{FO}(x, \mathsf{cut}_w(\mathsf{D}_1, y.\mathsf{D}_2)) &= \mathbb{FO}(y, \mathsf{D}_2) \cdot \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_2) \\
&\leq 0 \cdot 0 + 1 = 1 && \text{if } x \in \Gamma_2 \\
\mathbb{FO}(x, \mathsf{cut}_w(\mathsf{D}_1, y.\mathsf{E}_2)) &= \mathbb{FO}(y, \mathsf{D}_2) \cdot \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_2) \\
&\leq 0 \cdot 0 + |\mathsf{D}_2| \leq |\mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{E}_2)| && \text{if } x \in \Delta \\
\mathbb{FO}(x, \mathsf{cut}_w(\mathsf{D}_1, y.\mathsf{E}_2)) &= \mathbb{FO}(y, \mathsf{D}_2) \cdot \mathbb{FO}(x, \mathsf{D}_1) + \mathbb{FO}(x, \mathsf{D}_2) \\
&\leq 0 \cdot 0 + 1 = 1 && \text{if } x \in \Theta
\end{aligned}$$

This concludes the proof. □

Lemma 14 *Suppose* $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$. *Then* $\mathbb{D}(\mathsf{D}) \leq |\mathsf{D}|$.

Proof. An easy induction on the structure of a type derivation π for $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$. Some interesting cases:

- If π is

$$\frac{\Gamma_1; \emptyset; \emptyset \vdash \emptyset :: \mathsf{D}_1 z : A \quad \Gamma_2; \Delta, y : A; \Theta \vdash \mathsf{D}_2 :: T}{\Gamma_2; \Delta, \Gamma_1; \Theta \vdash \mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{D}_2) :: T} \ \mathsf{cut}_\#$$

  then, by Lemma 13 and by induction hypothesis:

$$\begin{aligned}
\mathbb{D}(\mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{D}_2)) &= \max\{\mathbb{D}(\mathsf{D}_1), \mathbb{D}(\mathsf{D}_2)\} \\
&\leq \max\{|\mathsf{D}_1|, |\mathsf{D}_2|\} \\
&\leq |\mathsf{cut}_\#(\mathsf{D}_1, y.\mathsf{D}_2)|
\end{aligned}$$

This concludes the proof. □

Lemma 15 *If* $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: T$, *then for every* $n \geq \mathbb{D}(\mathsf{D})$, $\mathbb{W}_n(\mathsf{D}) \leq |\widehat{\mathsf{D}}| \cdot n^{\mathbb{B}(\widehat{\mathsf{D}})+1}$.

Proof. By induction on the structure of D. Some interesting cases:

- If $\mathsf{D} = \otimes\mathsf{R}(\mathsf{E}, \mathsf{F})$, then:

$$\begin{aligned}
\mathbb{W}_n(\otimes\mathsf{R}(\mathsf{E}, \mathsf{F})) &= 1 + \mathbb{W}_n(\mathsf{E}) + \mathbb{W}_n(\mathsf{F}) \\
&\leq 1 + |\mathsf{E}| \cdot n^{\mathbb{B}(\mathsf{E})+1} + |\mathsf{F}| \cdot n^{\mathbb{B}(\mathsf{F})+1} \\
&\leq 1 + (|\mathsf{E}| + |\mathsf{F}|) \cdot n^{\max\{\mathbb{B}(\mathsf{E})+1, \mathbb{B}(\mathsf{F})+1\}} \\
&\leq (1 + |\mathsf{E}| + |\mathsf{F}|) \cdot n^{\max\{\mathbb{B}(\mathsf{E})+1, \mathbb{B}(\mathsf{F})+1\}} \\
&\leq |\otimes\mathsf{R}(\mathsf{E}, \mathsf{F})| \cdot n^{\mathbb{B}(\otimes\mathsf{R}(\mathsf{E},\mathsf{F}))+1}
\end{aligned}$$

- If $\mathsf{D} = \mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})$, then:

$$\begin{aligned}
\mathbb{W}_n(\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})) &= \mathbb{FO}(x, \mathsf{E}) \cdot (\mathbb{W}_n(\mathsf{D}) + 1) + \mathbb{W}_n(\mathsf{E}) \\
&\leq \mathbb{FO}(x, \mathsf{E}) \cdot (|\mathsf{D}| \cdot n^{\mathbb{B}(\mathsf{D})+1} + 1) + |\mathsf{E}| \cdot n^{\mathbb{B}(\mathsf{E})+1} \\
&\leq n \cdot |\mathsf{D}| \cdot n^{\mathbb{B}(\mathsf{D})+1} + n + |\mathsf{E}| \cdot n^{\mathbb{B}(\mathsf{E})+1} \\
&\leq |\mathsf{D}| \cdot n^{\mathbb{B}(\mathsf{D})+2} + n^{\mathbb{B}(\mathsf{E})+1} + |\mathsf{E}| \cdot n^{\mathbb{B}(\mathsf{E})+1} \\
&\leq (|\mathsf{D}| + |\mathsf{E}| + 1) \cdot n^{\max\{\mathbb{B}(\mathsf{D})+2, \mathbb{B}(\mathsf{E})+1\}} \\
&= (|\mathsf{cut}_!(\mathsf{D}, x.\mathsf{E})|) \cdot n^{\mathbb{B}(\mathsf{cut}_!(\mathsf{D},x.\mathsf{E}))}
\end{aligned}$$

- If $\mathsf{D} = !\mathsf{R}(x_1, \ldots, x_n, \mathsf{E})$, then:

$$\begin{aligned}
\mathbb{W}_n(!\mathsf{R}(x_1, \ldots, x_n, \mathsf{E})) &= n \cdot (\mathbb{W}_n(\mathsf{E}) + 1) \\
&\leq n \cdot |\mathsf{E}| \cdot n^{\mathbb{B}(\mathsf{E})+1} + n \\
&\leq |\mathsf{E}| \cdot n^{\mathbb{B}(\mathsf{E})+2} + n^{\mathbb{B}(\mathsf{E})+2} \\
&= (1 + |\mathsf{E}|) \cdot n^{\mathbb{B}(!\mathsf{R}(x_1,\ldots,x_n,\mathsf{E}))+1} \\
&= |!\mathsf{R}(x_1, \ldots, x_n, \mathsf{E})| \cdot n^{\mathbb{B}(!\mathsf{R}(x_1,\ldots,x_n,\mathsf{E}))+1}
\end{aligned}$$

This concludes the proof. □

5.4 Putting Everything Together

We now have almost all the necessary ingredients to obtain a proof of Proposition 4: the only missing tale are the bounds on the size of any reducts, since the polynomial bounds on the length of internal reduction are exactly the ones from Lemma 15. Observe, however, that the latter induces the former:

Lemma 16 *Suppose that $P \to^n Q$. Then $|Q| \le n \cdot |P|$.*

Proof. By induction on n, enriching the statement as follows: whenever $P \to^n Q$, both $|Q| \le n \cdot |P|$ and $|R| \le |P|$ for every subprocess R of Q in the form $!x(y).S$. $\square$

Lemma 17 *For every D, $\mathbb{B}(\mathsf{D}) = \mathbb{B}(\widehat{\mathsf{D}})$ and $|\mathsf{D}| = |\widehat{\mathsf{D}}|$.*

Finally:

Proof. [Proposition 4] Let $\{q_n\}_{n \in \mathbb{N}}$ the polynomials coming from Lemma 15. The polynomials we are looking for are defined as follows:

$$p_n(x) = q_n(x) + x \cdot q_n(x).$$

Now, suppose that $P \to^m Q$. By Theorem 1, there are proof terms D, E such that $P = \widehat{\mathsf{D}}$, $Q = \widehat{\mathsf{E}}$ and

$$\mathsf{D}(\hookrightarrow \Longrightarrow \hookrightarrow)^m \mathsf{E}.$$

Now, from propositions 5, 6 and 7, it follows that

$$\mathbb{W}(\mathsf{D}) \ge m + \mathbb{W}(\mathsf{E}) \ge m.$$

As a consequence, by Lemma 15 and Lemma 17

$$m \le q_{\mathbb{B}(\mathsf{D})}(|\mathsf{D}|) \le q_{\mathbb{B}(P)}(|P|) \le p_{\mathbb{B}(P)}(|P|).$$

By Lemma 16, it follows that

$$|Q| \le m \cdot |P| \le q_{\mathbb{B}(P)}(|P|) \cdot |P| \le p_{\mathbb{B}(P)}(|P|).$$

This concludes the proof. $\square$

Let us now consider Theorem 2: how can we deduce it from Proposition 4? Everything boils down to show that for normal processes, the box-depth can be read off from their type. In the following lemma, $\mathbb{B}(A)$ and $\mathbb{B}(\Gamma)$ are the nesting depth of $!$ inside the type A and inside the types appearing in Γ (for every type A and context Γ).

Lemma 18 *Suppose that $\Gamma; \Delta; \Theta \vdash \mathsf{D} :: x : A$ and that $\widehat{\mathsf{D}}$ is normal. Then $\mathbb{B}(\widehat{\mathsf{D}}) = \max\{\mathbb{B}(\Gamma), \mathbb{B}(\Delta), \mathbb{B}(\Theta), \mathbb{B}(A)\}$.*

Proof. An easy induction on D. $\square$

6 Conclusions

In this paper, we introduced a variation on Caires and Pfenning's $\pi\mathtt{DILL}$, called $\pi\mathtt{DSLL}$, being inspired by Lafont's soft linear logic. The key feature of $\pi\mathtt{DSLL}$ is the fact that the amount of interaction induced by allowing two processes to interact with each other is bounded by a polynomial whose degree can be "read off" from the type of the session channel through which they communicate.

What we consider the main achievement of this paper is definitely *not* the proof of these polynomial bounds, which can be obtained by adapting the ones in [9] or in [8], although this anyway presents some technical difficulties due to the low-level nature of the π-calculus compared

to the lambda calculus or to higher-order π-calculus. Instead, what we found very interesting is that the operational properties induced by typability in πDSLL, bounded interaction *in primis*, are not only very interesting and useful in practice, but different from the ones one obtains in soft lambda calculi: in the latter, it's the *normalization* time which is bounded, while here it's the *interaction* time. Another aspect that we find interesting is the following: it seems that the constraints on processes induced by the adoption of the more stringent typing discipline πDSLL, as opposed to πDILL, are quite natural and do not rule out too many interesting examples. In particular, the way sessions can be defined remains essentially untouched: what changes is the way sessions can be offered, i.e. the discipline governing the offering of multiple sessions by servers. All the examples in [1] and the one from Section 2 are indeed typable in πDSLL.

Topics for future work include the accommodation of recursive types into πDSLL. This could be easier than expected, due to the robustness of light logics to the presence of recursive types [7]

References

[1] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: *CONCUR 2010*, *LNCS* 6269. Springer, pp. 222–236.

[2] Luís Caires, Bernardo Toninho & Frank Pfenning (2011): *Dependent Session Types via Intuitionistic Linear Type Theory*. In: *PPDP 2011*. ACM Press. To appear.

[3] Jean-Yves Girard (1987): *Linear Logic*. *Theor. Comput. Sci.* 50, pp. 1–102.

[4] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In: *ESOP*, *LNCS* 1381. pp. 122–138.

[5] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In: *POPL 2008*. ACM Press, pp. 273–284.

[6] Yves Lafont (2004): *Soft linear logic and polynomial time*. *Theor. Comput. Sci.* 318(1-2), pp. 163–180.

[7] Ugo Dal Lago & Patrick Baillot (2006): *On light logics, uniform encodings and polynomial time*. *Mathematical Structures in Computer Science* 16(4), pp. 713–733.

[8] Ugo Dal Lago, Simone Martini & Davide Sangiorgi (2010): *Light Logics and Higher-Order Processes*. In: *EXPRESS'10*, *EPTCS* 41. pp. 46–60.

[9] Ugo Dal Lago, Andrea Masini & Margherita Zorzi (2010): *Quantum implicit computational complexity*. *Theor. Comput. Sci.* 411(2), pp. 377–409.

[10] Harry G. Mairson (1992): *A Simple Proof of a Theorem of Statman*. *Theor. Comput. Sci.* 103(2), pp. 387–394.

[11] Dimitris Mostrous & Nobuko Yoshida (2007): *Two Session Typing Systems for Higher-Order Mobile Processes*. In: *TLCA 2007*, *LNCS* 4583. pp. 321–335.

[12] Richard Statman (1979): *The Typed lambda-Calculus is not Elementary Recursive*. *Theor. Comput. Sci.* 9, pp. 73–81.