A Higher-Order Characterization of Probabilistic Polynomial Time

Ugo Dal Lago, Paolo Parisen

A Higher-Order Characterization of Probabilistic Polynomial Time

Ugo Dal Lago, Paolo Parisen Toldin

Dipartimento di Informatica, Università di Bologna
Équipe FOCUS, INRIA Sophia Antipolis
Mura Anteo Zamboni 7, 40127 Bologna, Italy

Abstract

We present RSLR, an implicit higher-order characterization of the class **PP** of those problems which can be decided in probabilistic polynomial time with error probability smaller than $1/2$. Analogously, a (less implicit) characterization of the class **BPP** can be obtained. RSLR is an extension of Hofmann's SLR with a probabilistic primitive, which enjoys basic properties such as subject reduction and confluence. Polynomial time soundness of RSLR is obtained by syntactical means, as opposed to the standard literature on SLR-derived systems, which use semantics in an essential way.

Keywords: Implicit computational complexity, Probabilistic classes, Lambda calculus, Linear types

1. Introduction

Implicit computational complexity (ICC) combines computational complexity, mathematical logic, and formal systems to give a machine independent account of complexity phenomena. It has been successfully applied to the characterization of a variety of complexity classes, especially in the sequential and parallel modes of computation (e.g., **FP** [4, 12], **PSPACE** [13], **LOGSPACE** [11], **NC** [5]). Its techniques, however, may be applied also to non-standard paradigms, like quantum computation [7] and concurrency [6]. Among the many characterizations of the class **FP** of functions computable in polynomial time, we can find Hofmann's *safe linear recursion* [10] (SLR in the following), a higher-order generalization of Bellantoni and Cook's *safe recursion* [3] in which linearity plays a crucial role.

Randomized computation is central to several areas of theoretical computer science, including cryptography, analysis of computation dealing with uncertainty and incomplete knowledge agent systems. In the context of computational complexity, probabilistic complexity classes like **BPP** [9] are nowadays considered as very closely corresponding to the

Email address: {dallago,parisent}@cs.unibo.it (Ugo Dal Lago, Paolo Parisen Toldin)
This work has been supported by project ANR 12IS02001 PACE.

informal notion of feasibility, since a solution to a problem in **BPP** can be computed in polynomial time up to any given degree of precision: **BPP** is the set of problems which can be solved by a probabilistic Turing machine working in polynomial time with a probability of error bounded by a constant *strictly* smaller than 1/2.

Probabilistic polynomial time computations, seen as oracle computations, were showed to be amenable to implicit techniques since the early days of ICC, by a relativization of Bellantoni and Cook's safe recursion [3]. They were then studied again in the context of formal systems for security, where probabilistic polynomial time computation plays a major role [14, 16]. These two systems are built on Hofmann's work SLR [10], by adding a random choice operator to the calculus. The system in [14], however, lacks higher-order recursion, and in both papers the characterization of probabilistic classes is obtained by semantic means. While this is fine for completeness, we think it is not completely satisfactory for soundness — we know from the semantics that for any term of a suitable type its normal form *may be* computed within the given bounds, but no notion of evaluation is given for which computation time is *guaranteed* to be bounded.

In this paper we propose RSLR, another probabilistic variation on SLR, and we show that it characterizes the class **PP** of those problems which can be solved in polynomial time by a Turing machine with error probability smaller than $\frac{1}{2}$ [9]. This is carried out by proving that any term in the language can be reduced in polynomial time, but also that any problems in **PP** can be represented in RSLR. A similar result, although in a less implicit form, is proved for **BPP**. Unlike [14], RSLR has higher-order recursion. Unlike [14] and [16], the bound on reduction time is obtained by syntactical means, giving an explicit notion of reduction which realizes that bound.

1.1. Related Work

More than ten years ago, Mitchell, Mitchell, and Scedrov [14] introduced OSLR, a type system that characterizes oracle polynomial time functionals. Even if inspired by SLR, OSLR does not admit primitive recursion on higher-order types, but only on base types. The main theorem shows that terms of type $\Box\mathbf{N}^m \to \mathbf{N}^n \to \mathbf{N}$ define precisely the *oracle polynomial time functionals*, which constitutes a class related but different from the ones we are interested in here. Finally, inclusion in the polynomial time class is proved without studying reduction from an operational viewpoint, but only via semantics: it is not clear for *which* notion of evaluation, computation time is guaranteed to be bounded.

Recently, Zhang's [16] introduced a further system (called CSLR) which builds on OSLR and allows higher-order recursion. The main interest of the paper are applications to the verification of security protocols. It is stated that CSLR defines exactly those functions that can be computed by probabilistic Turing machines in polynomial time, via a suitable variation of Hofmann's techniques as modified by Mitchell et al. This is again a purely semantic proof, whose details are missing in [16].

Finally, both works are derived from Hofmann's one, and as a consequence they both have potential problems with subject reduction. Indeed, as Hofmann showed in his work [10], subject reduction does not hold in SLR, and hence is problematic in both OSLR and CSLR.

1.2. RSLR*: An Informal Account*

Our system is called RSLR, which stands for Random Safe Linear Recursion. RSLR can be thought of as the system obtained by endowing SLR with a new primitive for random binary choice. Some restrictions have to be made to SLR if one wants to be able to prove polynomial time soundness operationally. And what one obtains at the end is indeed quite similar to (a probabilistic variation of) Bellantoni, Niggl and Schwichtenberg calculus RA [2, 15]. Actually, the main difference between RSLR and SLR has to do with linearity: keeping the size of reducts under control during normalization is very difficult in presence of higher-order duplication. For this reason, the two function spaces $A \to B$ and $A \multimap B$ of SLR collapse to just one in RSLR, and arguments of a higher-order type can *never* be duplicated. This constraint allows us to avoid an exponential blowup in the size of terms and results in a reasonably simple system for which polytime soundness can be proved explicitly, by studying the combinatorics of reduction. Another consequence of the just described modification is Subject Reduction, which can be easily proved in our system, contrarily to what happens in SLR [10].

1.3. On the Difficulty of Probabilistic ICC

Differently from most well-known complexity classes such as **P**, **NP** and **LOGSPACE**, interesting probabilistic complexity classes, like **BPP** and **ZPP** [9], are *semantic*. A semantic class is a complexity class defined on top of a class of algorithms which cannot be easily enumerated: a probabilistic polynomial time Turing machine does not *necessarily* solve a problem in **BPP** nor in **ZPP**. For most semantic classes, including **BPP** and **ZPP**, the existence of complete problems and the possibility to prove hierarchy theorems are both open question. Indeed, researchers in the area have proved the existence of such results for other probabilistic classes, but not for those we are interested in [8].

Now, having a "truly implicit" system I for a complexity class C means that we have a way to enumerate programs solving all problems in C (for every problem there is at least one program that solves it). The presence of complete problems, in other words, is deeply linked to the possibility of characterizing the class in the spirit of ICC. In our case the "semantic information" in **BPP** and **ZPP**, i.e., the error probability, seems to be an information that is impossible to capture by way of (recursively enumerable) syntactical restrictions. We need to execute the program on infinitely many inputs in order to check if the error probability is within bounds or not.

2. The Syntax and Basic Properties of RSLR

RSLR is a fairly standard Curry-style λ-calculus with constants for the natural numbers, branching and recursion. Its type system, on the other hand, is based on ideas coming from linear logic (variables of certain types can appear *at most once* in terms) and on a distinction between modal and non modal variables.

Let us introduce the category of types first:

Definition 2.1 (Types)**.** The *types* of RSLR are generated by the following grammar:

$$A ::= \mathbf{N} \mid \Box A \to A \mid \blacksquare A \to A.$$

Types different from $\mathbf{N}$ are denoted with metavariables like H or G. $\mathbf{N}$ is the only *base type*.

There are two function spaces in RSLR. Terms which can be typed with $\blacksquare A \to B$ are such that the result (of type B) can be computed in constant time, independently on the size of the argument (of type A). On the other hand, computing the result of functions in $\Box A \to B$ requires polynomial time in the size of their argument.

A notion of subtyping is used in RSLR to capture the intuition above by stipulating that the type $\blacksquare A \to B$ is a subtype of $\Box A \to B$. Subtyping is best formulated by introducing aspects:

Definition 2.2 (Aspects)**.** An *aspect* is either $\Box$ or $\blacksquare$: the first is the *modal* aspect, while the second is the *non-modal* one. Aspects are partially ordered by the binary relation $\{(\Box, \Box), (\Box, \blacksquare), (\blacksquare, \blacksquare)\}$, noted $<:$.

Defining subtyping, then, merely consists in generalizing $<:$ to a partial order on types in which only structurally identical types can be compared. Subtyping rules are in Figure 1. Please observe that (S-Sub) is contravariant in the aspect a.

$$\frac{}{A <: A}\ (\text{S-Refl}) \qquad \frac{A <: B \qquad B <: C}{A <: C}\ (\text{S-Trans})$$

$$\frac{B <: A \qquad C <: D \qquad b <: a}{aA \to C <: bB \to D}\ (\text{S-Sub})$$

Figure 1: Subtyping Rules.

RSLR's terms are those of an applied λ-calculus with primitive recursion and branching, in the style of Gödel's T:

Definition 2.3 (Terms)**.** *Terms* and *constants* are defined as follows:

$$t ::= x \mid c \mid ts \mid \lambda x : aA.t \mid \mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q \mid \mathtt{recursion}_A\ t\ s\ r;$$

$$c ::= n \mid \mathtt{S}_0 \mid \mathtt{S}_1 \mid \mathtt{P} \mid \mathtt{rand}.$$

Here, x ranges over a denumerable set of variables and n ranges over the natural numbers seen as constants of base type. Every constant c has its naturally defined type, that we indicate with $\mathit{type}(c)$. Formally, $\mathit{type}(n) = \mathbf{N}$ for every n, $\mathit{type}(\mathtt{rand}) = \mathbf{N}$, while

$\mathit{type}(\mathtt{S}_0) = \mathit{type}(\mathtt{S}_1) = \mathit{type}(\mathtt{P}) = \blacksquare\mathbf{N} \to \mathbf{N}$. The size $|t|$ of any term t can be easily defined by induction on t (where, by convention, we stipulate that $\log_2(0) = 0$):

$$\begin{aligned}
|x| &= 1;\\
|n| &= \lfloor \log_2(n) \rfloor + 1;\\
|\mathtt{S}_0| = |\mathtt{S}_1| = |\mathtt{P}| = |\mathtt{rand}| &= 1;\\
|ts| &= |t| + |s|;\\
|\lambda x : aA.t| &= |t| + 1;\\
|\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q| &= |t| + |s| + |r| + |q| + 1;\\
|\mathtt{recursion}_A\ t\, s\, r| &= |t| + |s| + |r| + 1.
\end{aligned}$$

Notice that the size of n is exactly the length of the number n in binary representation. Size of 5, as an example, is $\lfloor \log_2(5) \rfloor + 1 = 3$, while 0 only requires one binary digit to be represented, and its size is thus 1. As usual, terms are considered modulo α-conversion. Free (occurrences of) variables and capture-avoiding substitution can be defined in a standard way.

Definition 2.4 (Explicit term)**.** A term is said to be *explicit* if it does not contain any instance of `recursion`.

The main peculiarity of RSLR with respect to similar calculi is the presence of an operator for probabilistic, binary choice, called `rand`, which evolves to either 0 or 1 with probability $\frac{1}{2}$. Although the calculus is in Curry-style, variables are explicitly assigned a type and an aspect in abstractions. This is for technical reasons that will become apparent soon.

Note 2.5. The presence of terms which can (probabilistically) evolve in different ways makes it harder to define a confluent notion of reduction for RSLR. To see why, consider a term like $t = (\lambda x : \blacksquare\mathbf{N}.(t_\oplus x x))\mathtt{rand}$, where $t_\oplus$ is a term computing $\oplus$ on natural numbers seen as booleans (0 stands for "false" and everything else stands for "true"):

$$\begin{aligned}
t_\oplus &= \lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus;\\
s_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 0\ \mathtt{even}\ 1\ \mathtt{odd}\ 1;\\
r_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 1\ \mathtt{even}\ 0\ \mathtt{odd}\ 0.
\end{aligned}$$

If we evaluate t in a call-by-value fashion, `rand` will be fired *before* being passed to $t_\oplus$ and, as a consequence, the latter will be fed with two identical natural numbers, returning 0 with probability 1. If, on the other hand, `rand` is passed unevaluated to $t_\oplus$, the four possible combinations on the truth table for $\oplus$ will appear with equal probabilities and the outcome will be 0 or 1 with probability $\frac{1}{2}$. In other words, we need to somehow restrict our notion of reduction if we want it to be consistent, i.e. confluent. For the just explained reasons, arguments are passed to functions following a mixed scheme in RSLR: arguments of base type are evaluated before being passed to functions, while arguments of an higher-order type are passed to functions possibly unevaluated, in a call-by-name fashion. This way,

higher-order terms cannot be duplicated and this guarantees that if a term is duplicated, then it has no `rand` inside. The counterexample above, as a consequence, no longer works.

Let's first of all define the one-step reduction relation:

Definition 2.6 (Reduction)**.** The *one-step reduction relation* $\to$ is a binary relation between terms and sequences of terms. It is defined by the rules in Figure 2, which can be applied in any contexts except in the second and third argument of a recursion. Notice how the last but one rule is defined: in some cases, we allow to swap some arguments. Finally, we say that a term t is in *normal form* if t cannot appear as the left-hand side of a pair in $\to$. NF is the set of terms in normal form.

$$
\begin{aligned}
\mathtt{case}_A\ 0\ \mathtt{zero}\ t\ \mathtt{even}\ s\ \mathtt{odd}\ r &\to t;\\
\mathtt{case}_A\ (2\cdot(n+1))\ \mathtt{zero}\ t\ \mathtt{even}\ s\ \mathtt{odd}\ r &\to s;\\
\mathtt{case}_A\ (2\cdot n+1)\ \mathtt{zero}\ t\ \mathtt{even}\ s\ \mathtt{odd}\ r &\to r;\\
\mathtt{recursion}_A\ 0\, g\, f &\to g;\\
\mathtt{recursion}_A\ (n+1)\, g\, f &\to f(n+1)(\mathtt{recursion}_A\ \lfloor\frac{n+1}{2}\rfloor\, g\, f);\\
\mathtt{S}_0 n &\to 2\cdot n;\\
\mathtt{S}_1 n &\to 2\cdot n+1;\\
\mathtt{P} n &\to \lfloor\frac{n}{2}\rfloor;\\
(\lambda x : a\mathbf{N}.t)n &\to t[x/n];\\
(\lambda x : aH.t)s &\to t[x/s];\\
(\lambda x : aA.t)sr &\to (\lambda x : aA.tr)s;\\
\mathtt{rand} &\to 0, 1.
\end{aligned}
$$

Figure 2: One-step Reduction Rules.

Informally, $t \to s_1, \ldots, s_n$ means that t can evolve in one-step to each of $s_1, \ldots, s_n$ with the same probability $\frac{1}{n}$. As a matter of fact, n can be either 1 or 2.

A multistep reduction relation will not be defined by simply taking the transitive and reflective closure of $\to$, since a term can reduce in multiple steps to many terms with different probabilities. Multistep reduction puts in relation a term t to a probability distribution on terms $\mathscr{D}_t$ such that $\mathscr{D}_t(s) > 0$ only if s is a normal form to which t reduces. Of course, if t is itself a normal form, $\mathscr{D}_t$ is well defined, since the only normal form to which t reduces is t itself, so $\mathscr{D}_t(t) = 1$. But what happens when t is *not* in normal form? Is $\mathscr{D}_t$ a well-defined concept? Let us start by formally defining $\rightsquigarrow$:

Definition 2.7 (Multistep Reduction)**.** A *probability distribution* $\mathscr{D}$ should be understood here as a function from NF to $[0,1]$ such that $\sum_{t\in NF}\mathscr{D}(t)=1$. The binary relation $\rightsquigarrow$ between terms and probability distributions is defined by the rules in Figure 3. $\mathscr{I}_t$ is Dirac distribution on $t\in NF$, namely the function returning 1 on t and 0 on any other normal form. A finite distribution on terms $\mathscr{D}$ can be denoted as $\{t_1^{\alpha_1},\ldots,t_n^{\alpha_n}\}$ where $\mathscr{D}(s)=\sum_{t_i=s}\alpha_i$ (observe that the terms $t_1,\ldots,t_n$ are not necessarily distinct).

$$\frac{t\to t_1,\ldots,t_n \qquad t_i\rightsquigarrow\mathscr{D}_i}{t\rightsquigarrow\sum_{i=1}^{n}\frac{1}{n}\mathscr{D}_i} \qquad\qquad \frac{t\in NF}{t\rightsquigarrow\mathscr{I}_t}$$

Figure 3: Multistep Reduction: Inference Rules

In Section 2.2, we will prove that for every t there is at most one $\mathscr{D}$ such that $t\rightsquigarrow\mathscr{D}$. For the sake of clarifying how multistep reduction works, let us consider an example. Let $\texttt{ifz}_A\ t\ \texttt{then}\ s\ \texttt{else}\ r$ be syntactic sugar for $\texttt{case}_A\ t\ \texttt{zero}\ s\ \texttt{even}\ r\ \texttt{odd}\ r$. Now, consider the term

$$t=\texttt{ifz}_\mathbf{N}\ (\texttt{rand})\ \texttt{then}\ (\texttt{ifz}_\mathbf{N}\ (\texttt{rand})\ \texttt{then}\ 1\ \texttt{else}\ 2)\ \texttt{else}\ 2.$$

The following one-step reduction can be derived from the last rule in Figure 2, applying it in a proper context:

$$\begin{aligned} t\to\ &\texttt{ifz}_\mathbf{N}\ 0\ \texttt{then}\ (\texttt{ifz}_\mathbf{N}\ (\texttt{rand})\ \texttt{then}\ 1\ \texttt{else}\ 2)\ \texttt{else}\ 2,\\ &\texttt{ifz}_\mathbf{N}\ 1\ \texttt{then}\ (\texttt{ifz}_\mathbf{N}\ (\texttt{rand})\ \texttt{then}\ 1\ \texttt{else}\ 2)\ \texttt{else}\ 2. \end{aligned}$$

Obviously, $\texttt{ifz}_\mathbf{N}\ 1\ \texttt{then}\ (\texttt{ifz}_\mathbf{N}\ (\texttt{rand})\ \texttt{then}\ 1\ \texttt{else}\ 2)\ \texttt{else}\ 2\to 2$ (remember that the `ifz` construct is nothing more than syntactic sugar for a `case`). Let us examine the first of the two terms t reduces to. It one-step reduces to $s=\texttt{ifz}_\mathbf{N}\ (\texttt{rand})\ \texttt{then}\ 1\ \texttt{else}\ 2$, and it is quite easy to realize that:

$$s\to(\texttt{ifz}_\mathbf{N}\ 0\ \texttt{then}\ 1\ \texttt{else}\ 2),(\texttt{ifz}_\mathbf{N}\ 1\ \texttt{then}\ 1\ \texttt{else}\ 2),$$

again by applying the last rule in Figure 2. Finally,

$$\begin{aligned} &\texttt{ifz}_\mathbf{N}\ 0\ \texttt{then}\ 1\ \texttt{else}\ 2\to 1;\\ &\texttt{ifz}_\mathbf{N}\ 1\ \texttt{then}\ 1\ \texttt{else}\ 2\to 2. \end{aligned}$$

Let us derive appropriate multi-step judgments. Of course, $1\rightsquigarrow\mathscr{I}_1$ and $2\rightsquigarrow\mathscr{I}_2$. As a consequence, $\texttt{ifz}_\mathbf{N}\ 0\ \texttt{then}\ 1\ \texttt{else}\ 2\rightsquigarrow\mathscr{I}_1$ and $\texttt{ifz}_\mathbf{N}\ 1\ \texttt{then}\ 1\ \texttt{else}\ 2\rightsquigarrow\mathscr{I}_2$. s one-step reduces to the two terms we have just considered, so $s\rightsquigarrow\frac{1}{2}\mathscr{I}_1+\frac{1}{2}\mathscr{I}_2$. Now, t one-step reduces to either s or 2 and, as a consequence,

$$t\rightsquigarrow\frac{1}{2}\left(\frac{1}{2}\mathscr{I}_1+\frac{1}{2}\mathscr{I}_2\right)+\frac{1}{2}\mathscr{I}_2=\frac{1}{4}\mathscr{I}_1+\frac{3}{4}\mathscr{I}_2.$$

We are finally able to present the type system. Preliminary to that is the definition of a proper notion of a context.

Definition 2.8 (Contexts)**.** A *context* Γ is a finite set of assignments of types and aspects to variables, i.e., of expressions in the form $x : aA$. As usual, we require contexts not to contain assignments of distinct types and aspects to the same variable. The union of two disjoint contexts Γ and Δ is denoted as Γ, Δ. In doing so, we implicitly assume that the variables in Γ and Δ are pairwise distinct. The expression $\Gamma; \Delta$ denotes the union Γ, Δ, but is only defined when all types appearing in Γ are base types. As an example, it is perfectly legitimate to write $x : a\mathbf{N}; y : b\mathbf{N}$, while the following is an ill-defined expression:

$$x : a(b\mathbf{N} \to \mathbf{N}); y : c\mathbf{N},$$

the problem being the first assignment, which appears on the left of ";" but which assigns the *higher-order* type $b\mathbf{N} \to \mathbf{N}$ (and the aspect a) to x. This notation is particularly helpful when giving typing rules. With the expression $\Gamma <: a$ we mean that any aspect b appearing in Γ is such that $b <: a$.

Typing rules are in Figure 4.

$$\frac{x : aA \in \Gamma}{\Gamma \vdash x : A}\ (\text{T-Var-Aff}) \qquad \frac{\Gamma \vdash t : A \qquad A <: B}{\Gamma \vdash t : B}\ (\text{T-Sub})$$

$$\frac{\Gamma, x : aA \vdash t : B}{\Gamma \vdash \lambda x : aA.t : aA \to B}\ (\text{T-Arr-I}) \qquad \frac{}{\Gamma \vdash c : \mathit{type}(c)}\ (\text{T-Const-Aff})$$

$$\frac{\begin{array}{cc}\Gamma; \Delta_1 \vdash t : \mathbf{N} & \Gamma; \Delta_3 \vdash r : A \\ \Gamma; \Delta_2 \vdash s : A & \Gamma; \Delta_4 \vdash q : A \qquad A \text{ is } \Box\text{-free}\end{array}}{\Gamma; \Delta_1, \Delta_2, \Delta_3, \Delta_4 \vdash \mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q : A}\ (\text{T-Case})$$

$$\frac{\begin{array}{c}\Gamma_1; \Delta_1 \vdash t : \mathbf{N} \\ \Gamma_1, \Gamma_2; \Delta_2 \vdash s : A \qquad\qquad \Gamma_1, \Delta_1 <: \Box \\ \Gamma_1, \Gamma_2; \vdash r : \Box\mathbf{N} \to \blacksquare A \to A \qquad A \text{ is } \Box\text{-free}\end{array}}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash \mathtt{recursion}_A\ t\, s\, r : A}\ (\text{T-Rec})$$

$$\frac{\Gamma; \Delta_1 \vdash t : aA \to B \qquad \Gamma; \Delta_2 \vdash s : A \qquad \Gamma, \Delta_2 <: a}{\Gamma; \Delta_1, \Delta_2 \vdash (ts) : B}\ (\text{T-Arr-E})$$

Figure 4: Typing Rules

Note 2.9. Observe how rules with more than one premise are designed in such a way as to guarantee that whenever $\Gamma \vdash t : A$ can be derived and $x : aH$ is in Γ, then x can appear

free *at most once* in t. If $y : a\mathbf{N}$ is in Γ, on the other hand, then y can appear free in t an arbitrary number of times. This can be proved by induction on the structure of any derivation for $\Gamma \vdash t : A$.

Definition 2.10. A *first-order term* of arity k is a closed, well-typed, term of type $a_1\mathbf{N} \to a_2\mathbf{N} \to \ldots a_k\mathbf{N} \to \mathbf{N}$ for some $a_1, \ldots, a_k$.

Example 2.1. Let's see some examples, namely two terms that we are able to type in our system, and one that is *not* possible to type. As we will see in Chapter 4.1 we are able to type addition and multiplication. Addition gives in output a number (recall that we are in unary notation) such that the resulting length is the sum of the input lengths.

$$\begin{aligned}\mathsf{add} =& \lambda x : \square\mathbf{N}.\lambda y : \blacksquare\mathbf{N}.\\ & \mathtt{recursion}_{\mathbf{N}}\ x\, y\, (\lambda x : \square\mathbf{N}.\lambda y : \blacksquare\mathbf{N}.\mathtt{S}_1 y) : \square\mathbf{N} \to \blacksquare\mathbf{N} \to \mathbf{N}\end{aligned}$$

We are also able to define multiplication. The operator is, as usual, defined by iterating addition:

$$\begin{aligned}\mathsf{mult} =& \lambda x : \square\mathbf{N}.\lambda y : \square\mathbf{N}.\\ & \mathtt{recursion}_{\mathbf{N}}\ (\mathtt{P}x)\, y\, (\lambda x : \square\mathbf{N}.\lambda z : \blacksquare\mathbf{N}.\mathsf{add} yz) : \square\mathbf{N} \to \square\mathbf{N} \to \mathbf{N}.\end{aligned}$$

Now that we have multiplication, why not iterate it and get an exponential? As it will be clear from the next example, the restriction on the aspect of the iterated function save us from having an exponential growth. Are we able to type the following term?

$$\lambda h : \square\mathbf{N}.\mathtt{recursion}_{\mathbf{N}}\ h\, (11)\, (\lambda x : \square\mathbf{N}.\lambda y : \blacksquare\mathbf{N}.\mathsf{mult}(y, y))$$

The answer is negative: the operator mult requires an input of aspect $\square$, while the iterated function necessarily has type $\square\mathbf{N} \to \blacksquare\mathbf{N} \to \mathbf{N}$.

2.1. Subject Reduction

The first property we are going to prove about RSLR is preservation of types under reduction, the so-called Subject Reduction Theorem. The proof of it is going to be very standard and, as usual, amounts to proving substitution lemmas. Preliminary to that is a technical lemma saying that weakening is derivable (since the type system is affine):

Lemma 2.1 (Weakening Lemma)**.** *If $\Gamma \vdash t : A$, then $\Gamma, x : bB \vdash t : A$ whenever x does not appear in Γ.*

Proof. By induction on the structure of the typing derivation for t.

- If last rule is (T-Var-Aff) or (T-Const-Aff), we are allowed to add whatever we want to the context.
- If last rule is (T-Sub) or (T-Arr-I), the thesis is proved by applying the induction hypothesis to the premise.

- Suppose that the last rule was:

$$\frac{\begin{array}{ccc}\Gamma;\Delta_1 \vdash u : N & \Gamma;\Delta_3 \vdash r : A & \\ \Gamma;\Delta_2 \vdash s : A & \Gamma;\Delta_4 \vdash q : A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ u\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q : A}\ (\text{T-CASE})$$

  If $B = \mathbf{N}$ we can easily proceed by applying the induction hypothesis to every premises and add x to Γ. Otherwise, we can proceed by applying induction hypothesis to just one premise.
- Suppose that the last rule is:

$$\frac{\begin{array}{cc}\Gamma_1;\Delta_1 \vdash q : \mathbf{N} & \\ \Gamma_1,\Gamma_2;\Delta_2 \vdash s : A & \Gamma_1,\Delta_1 <: \Box \\ \Gamma_1,\Gamma_2;\vdash r : \Box\mathbf{N} \to \blacksquare A \to A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_A\ q\ s\ r : A}\ (\text{T-REC})$$

  Suppose that $B = \mathbf{N}$. We have the following cases:
  - If $b = \Box$, we can do it by applying induction hypothesis to all the premises and add x in Γ_1.
  - If $b = \blacksquare$ we apply induction hypothesis on $\Gamma_1,\Gamma_2;\Delta_2 \vdash s : A$ and on $\Gamma_1,\Gamma_2;\vdash r : \Box\mathbf{N} \to \blacksquare A \to A$.

  Otherwise we apply induction hypothesis on $\Gamma_1;\Delta_1 \vdash q : \mathbf{N}$ or on $\Gamma_1,\Gamma_2;\Delta_2 \vdash s : A$ and we are done.

This concludes the proof. □

Two substitution lemmas are needed in RSLR. The first one applies when the variable to be substituted has a non-modal type:

Lemma 2.2 ($\blacksquare$-Substitution Lemma)**.** *Let* $\Gamma;\Delta \vdash t : A$*. Then*

1. *if* $\Gamma = x : \blacksquare\mathbf{N},\Theta$*, then* $\Theta;\Delta \vdash t[x/n] : A$ *for every* n*;*
2. *if* $\Delta = x : \blacksquare H,\Theta$ *and* $\Gamma;\Xi \vdash s : H$*, then* $\Gamma;\Theta,\Xi \vdash t[x/s] : A$*.*

Proof. By induction on a type derivation of t. Some interesting cases:

- If the last rule is (T-REC), our derivation will have the following shape:

$$\frac{\begin{array}{cc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & \Gamma_2,\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\ s\ r : B}\ (\text{T-REC})$$

  By definition, $x : \blacksquare A$ cannot appear in $\Gamma_2;\Delta_4$. If it appears in Δ_5 we can simply apply induction hypothesis and prove the thesis. We will focus on the most interesting case: it appears in Γ_3 and so $A = \mathbf{N}$. In that case, by the induction hypothesis applied to (type derivations for) s and r, we obtain that:

$$\begin{array}{c}\Gamma_2,\Gamma_4;\Delta_5 \vdash s[x/n] : B; \\ \Gamma_2,\Gamma_4;\vdash r[x/n] : \Box\mathbf{N} \to \blacksquare B \to B;\end{array}$$

  where $\Gamma_3 = \Gamma_4, x : \blacksquare\mathbf{N}$.

- If the last rule is (T-Arr-E),

$$\frac{\Gamma;\Delta_4 \vdash t : aC \to B \qquad \Gamma;\Delta_5 \vdash s : C \qquad \Gamma,\Delta_5 <: a}{\Gamma,\Delta_4,\Delta_5 \vdash (ts) : B}\ (\text{T-Arr-E})$$

  If $x : A$ is in Γ then we apply induction hypothesis on both branches, otherwise it is either in Δ_4 or in Δ_5 and we apply induction hypothesis on the corresponding branch. We arrive to the thesis by applying (T-Arr-E) at the end.

This concludes the proof. □

Substituting a variable of a *modal* type requires an additional hypothesis on the term being substituted:

Lemma 2.3 (□-Substitution Lemma)**.** *Let* $\Gamma;\Delta \vdash t : A$*. Then*

1. *if* $\Gamma = x : \Box\mathbf{N}, \Theta$*, then* $\Theta;\Delta \vdash t[x/n] : A$ *for every* n*;*
2. *if* $\Delta = x : \Box H, \Theta$ *and* $\Gamma;\Xi \vdash s : H$ *where* $\Gamma,\Xi <: \Box$*, then* $\Gamma;\Theta,\Xi \vdash t[x/s] : A$*.*

Proof. By induction on the structure of a type derivation for t. Some cases:

- If last rule is (T-Rec), our derivation will have the following shape:

$$\frac{\begin{array}{cc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & \Gamma_2,\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\, s\, r : B}\ (\text{T-Rec})$$

  By definition $x : \Box A$ can appear in $\Gamma_1;\Delta_4$. If so, by applying induction hypothesis we can derive easily the proof. In the other cases, we can proceed as in Lemma 2.2. We will focus on the most interesting case, where $x : \Box A$ appears in Γ_2 and so $A = \mathbf{N}$. In that case, by the induction hypothesis applied to (type derivations for) s and r, we obtain that:

$$\begin{array}{c}\Gamma_4,\Gamma_3;\Delta_5 \vdash s[x/n] : B \\ \Gamma_4,\Gamma_3;\vdash r[x/n] : \Box\mathbf{N} \to \blacksquare B \to B\end{array}$$

  where $\Gamma_2 = \Gamma_4, x : \Box\mathbf{N}$.

- If last rule is (T-Arr-E),

$$\frac{\Gamma;\Delta_4 \vdash t : aC \to B \qquad \Gamma;\Delta_5 \vdash s : C \qquad \Gamma,\Delta_5 <: a}{\Gamma,\Delta_4,\Delta_5 \vdash (ts) : B}\ (\text{T-Arr-E})$$

  If $x : A$ is in Γ then we apply induction hypothesis on both branches, otherwise it is either in Δ_4 or in Δ_5 and we apply induction hypothesis on the relative branch. We prove our thesis by applying (T-Arr-E) at the end.

This concludes the proof. □

Substitution lemmas are necessary ingredients when proving Subject Reduction. In particular, they allow to prove that types are preserved along β-reduction steps, the other reduction steps being very easy. We get:

Theorem 2.4 (Subject Reduction)**.** *Suppose that* $\Gamma \vdash t : A$*. If* $t \to t_1 \dots t_j$*, then for every* $i \in \{1, \dots, j\}$*, it holds that* $\Gamma \vdash t_i : A$*.*

Proof. By induction on the structure of a derivation for t. Some interesting cases:

- If last rule is (T-Case).

$$\frac{\begin{array}{cccc}\Gamma;\Delta_1 \vdash s : N & \Gamma;\Delta_3 \vdash q : A & & \\ \Gamma;\Delta_2 \vdash r : A & \Gamma;\Delta_4 \vdash u : A & A \text{ is } \Box\text{-free} & \end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  Our final term could reduce in two ways. Either we do β-reduction on s, r, q or u, or we choose one of branches in the case. In all the cases, the proof is trivial.

- If last rule is (T-Rec).

$$\frac{\begin{array}{cc}\rho : \Gamma_1;\Delta_1 \vdash s : \mathbf{N} & \\ \mu : \Gamma_1,\Gamma_2;\Delta_2 \vdash r : A & \Gamma_1,\Delta_1 <: \Box \\ \nu : \Gamma_1,\Gamma_2;\vdash q : \Box\mathbf{N} \to \blacksquare A \to A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_A\ s\, r\, q : A}\ (\text{T-Rec})$$

  Our term could reduce in three ways. We could evaluate s (trivial), we could be in the case where $s = 0$ (trivial) and the other case is where we unroll the recursion (so, where s is a value $n \geq 1$). We are going to focus on this last option. The term rewrites to $qn(\mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q)$. We could define the following derivations π and σ:

$$\dfrac{\dfrac{}{\Gamma_1;\Delta_1 \vdash \lfloor \frac{n}{2} \rfloor : \mathbf{N}}\ (\text{T-Const-Aff}) \qquad \mu : \Gamma_1,\Gamma_2;\Delta_2 \vdash r : A}{\nu : \Gamma_1,\Gamma_2;\vdash q : \Box\mathbf{N} \to \blacksquare A \to A \qquad \Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q : A}\ (\text{T-Rec})$$

$$\dfrac{\nu : \emptyset;\Gamma_1,\Gamma_2 \vdash q : \Box\mathbf{N} \to \blacksquare A \to A \qquad \dfrac{}{\emptyset;\emptyset \vdash n : \mathbf{N}}\ (\text{T-Const-Aff})}{\emptyset;\Gamma_1,\Gamma_2 \vdash qn : \blacksquare A \to A}\ (\text{T-Arr-E})$$

  By gluing the two derivation with the rule (T-Arr-E) we obtain:

$$\frac{\begin{array}{c}\sigma : \Gamma_1,\Gamma_2;\vdash qn : \blacksquare A \to A \\ \pi : \Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q : A\end{array}}{\Gamma_1,\Gamma_2,\Gamma_3;\Delta_1,\Delta_2 \vdash qn(\mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q) : A}\ (\text{T-Arr-E})$$

  Notice that in the derivation ν we put Γ_1, Γ_2 on the left side of ";" and also on the right side. Recall Definition 2.8.

- If last rule was (T-Sub) we have the following derivation:

$$\frac{\Gamma \vdash s : A \qquad A <: B}{\Gamma \vdash s : B}\ (\text{T-Sub})$$

  If s reduces to r we can apply induction hypothesis on the premises and having the following derivation:

$$\frac{\Gamma \vdash r : A \qquad A <: B}{\Gamma \vdash r : B}\ (\text{T-Sub})$$

- If last rule was (T-Arr-E), we could have different cases.
  - Cases where on the left part of our application we have $\mathtt{S}_i$, $\mathtt{P}$ is trivial.
  - Let's focus on the case where on the left part we find a λ-abstraction. We will only consider the case where we apply the substitution. The other cases are trivial. We could have two possibilities:
    - First of all, we can be in the following situation:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : \blacksquare A.r : aC \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: a}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : \blacksquare A.r)s : B}\ (\text{T-Arr-E})$$

      where $C <: A$ and $a <: \blacksquare$. We have that $(\lambda x : \blacksquare A.r)s$ rewrites to $r[x/s]$. By looking at rules in Figure 4 we can deduce that $\Gamma;\Delta_1 \vdash \lambda x : \blacksquare A.r : aC \to B$ derives from $\Gamma; x : \blacksquare A, \Delta_1 \vdash r : D$ (with $D <: B$). For the reason that $C <: A$ we can apply (T-Sub) rule to $\Gamma;\Delta_2 \vdash s : C$ and obtain $\Gamma;\Delta_2 \vdash s : A$ By applying Lemma 2.2, we get to

$$\Gamma,\Delta_1,\Delta_2 \vdash r[x/s] : D$$

      from which the thesis follows by applying (T-Sub).
    - But we can even be in the following situation:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : \square A.r : \square C \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: \square}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : \square A.r)s : B}\ (\text{T-Arr-E})$$

      where $C <: A$. We have that $(\lambda x : \square A.r)s$ rewrites in $r[x/s]$. We behave as in the previous point, by applying Lemma 2.3, and we are done.
  - Another interesting case of application is where we perform a so-called "swap". $(\lambda x : aA.q)sr$ rewrites in $(\lambda x : aA.qr)s$. From a typing derivation with conclusion $\Gamma,\Delta_1,\Delta_2,\Delta_3 \vdash (\lambda x : aA.q)sr : C$ we can easily extract derivations for the following:

$$\Gamma;\Delta_1, x : aA \vdash q : bD \to E$$
$$\Gamma;\Delta_3 \vdash r : B$$
$$\Gamma;\Delta_2 \vdash s : F$$

    where $B <: D$, $E <: C$ and $A <: F$ and $\Gamma,\Delta_3 <: b$ and $\Gamma,\Delta_2 <: a$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma,\Delta_3 <: b \qquad \Gamma;\Delta_3 \vdash r : B}{\dfrac{\Gamma;\Delta_1, x : aA \vdash q : bD \to E}{\Gamma;\Delta_1,\Delta_3, x : aA \vdash qr : E}\ (\text{T-Arr-E})}{\Gamma;\Delta_1,\Delta_3,\vdash \lambda x : aA.qr : aA \to E}\ (\text{T-Arr-I})}{\Gamma;\Delta_1,\Delta_3,\vdash \lambda x : aA.qr : aF \to C}\ (\text{T-Sub}) \qquad \Gamma,\Delta_2 <: a \quad \Gamma;\Delta_2 \vdash s : F}{\Gamma,\Delta_1,\Delta_2,\Delta_3 \vdash (\lambda x : aA.qr)s : C}\ (\text{T-Arr-E})$$

- All the other cases can be brought back to cases that we have considered.

This concludes the proof. □

Example 2.2. The following example has been proposed by Hofmann [10]. Let f be a variable of type $\blacksquare\mathbf{N} \to \mathbf{N}$. The function $h = \lambda g : \blacksquare(\blacksquare\mathbf{N} \to \mathbf{N}).\lambda x : \blacksquare\mathbf{N}.(f(gx))$ gets type $\blacksquare(\blacksquare\mathbf{N} \to \mathbf{N}) \to \blacksquare\mathbf{N} \to \mathbf{N}$. Thus the function $(\lambda v : \blacksquare(\blacksquare\mathbf{N} \to \mathbf{N}).hv)\mathtt{S}_1$ takes type $\blacksquare\mathbf{N} \to \mathbf{N}$. Let's now fire a β step, by passing the argument $\mathtt{S}_1$ to the function h and we obtain the following term: $\lambda x : \blacksquare\mathbf{N}.(f(\mathtt{S}_1 x))$ It's easy to check that the type has not changed.

2.2. Confluence

In view of the peculiar notion of reduction given in Definition 2.6, let us go back to the counterexample to confluence given in the Introduction. The term $t = (\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))\mathtt{rand}$ cannot be reduced to $t_\oplus\, \mathtt{rand}\, \mathtt{rand}$ anymore, because only numerals can be passed to functions as arguments of base types. The only possibility is reducing t to the sequence

$$(\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))0, (\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))1.$$

Both terms in the sequence can be further reduced to 0. In other words, $t \rightsquigarrow \{0^1\}$.

More generally, the phenomenon of non-convergence of final distributions can no longer happen in RSLR. Technically, this is due to the impossibility of duplicating "probabilistic" terms, i.e., terms containing occurrences of `rand`. In the above example, and in similar cases, we have to evaluate the argument before firing the β-redex — it is therefore not possible to obtain two different distributions. RSLR can also handle correctly the case where `rand` is within an argument t of higher-order type: the only non-normal terms which can be duplicated are the arguments to a recursion, in which reduction cannot take place *by definition*.

Confluence of our system is proved by first showing a strong form of confluence for the single step arrow $\to$, then transferring it to the multistep arrow $\rightsquigarrow$. Showing confluence for $\to$ turns out to be slightly more complicated than expected and is thus split into three separate lemmas.

Lemma 2.5 (Diamond Property, Part I)**.** *Let t be a well-typed term; if $t \to v$ and $t \to z$ (where v and z distinct), then exactly one of the following holds:*

- *$\exists e$ such that $v \to e$ and $z \to e$;*
- *$v \to z$;*
- *$z \to v$.*

Proof. By induction on the structure of the typing derivation for the term t. Some interesting cases:

- If last rule is T-Case, our derivation will have the following shape:

$$\frac{\begin{array}{lll}\Gamma;\Delta_1 \vdash s : N & \Gamma;\Delta_3 \vdash q : A & \\ \Gamma;\Delta_2 \vdash r : A & \Gamma;\Delta_4 \vdash u : A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  We could have reduced one among s, r, q, u or a combination of them. In the first case we prove by applying induction hypothesis and in the latter case we can easily find e

such that $v \to e$ and $z \to e$: it is the term obtained by applying both reductions. Last case is where from one part we reduce the case, selecting a branch and from the other part we reduce one of the subterms. As can be easily seen, we can find a common confluent term.

- If last rule is T-Rec, our derivation will have the following shape:

$$\frac{\begin{array}{cc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & \Gamma_2;\Delta_4 <: \square \\ \Gamma_2,\Gamma_3;\vdash r : \square\mathbf{N} \to \blacksquare B \to B & B \text{ is } \square\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\, s\, r : B}\ (\text{T-Rec})$$

  By definition, we can have reduction only in q or, if q is a value, we can reduce the recursion by unrolling it. In both cases the proof is trivial.
- If last rule is T-Arr-E, our term could have different shapes but the only interesting cases are the following ones. The other cases can be easily brought back to cases that we have already considered.
  - Our derivation will end in the following way:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : aA.r : bC \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: b}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : aA.r)s : B}\ (\text{T-Arr-E})$$

    where $C <: A$ and $b <: a$. We have that $(\lambda x : aA.r)s$ rewrites to $r[x/s]$; if $A = \mathbf{N}$ then s is a value. If we reduce only in s or only in r we can easily prove our thesis by the induction hypothesis. The interesting cases are when we perform the substitution on one hand and on the other hand we make a reduction step on one of the two possible terms s or r. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.r)f$, where $s \to f$. Let e be $r[x/f]$. We have that $(\lambda x : aA.r)f \to e$ and $r[x/s] \to e$. Indeed if A is $\mathbf{N}$, s is a value, no reduction could be made on s. Otherwise, there is at most one occurrence of s in $r[x/s]$ and by executing one reduction step we are able to reach e. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.g)s$, where $r \to g$. As we have shown in the previous case, we are able to find the required terms.
  - The other interesting case is when we perform the so called "swap". $(\lambda x : aA.q)sr$ rewrites in $(\lambda x : aA.qr)s$. If the reduction steps are made only in q or s or r, by applying induction hypothesis we have the thesis. In all the other cases, where we perform one step on subterms and we perform, on the other hand, the swap, it's easy to find a confluent term e.

This concludes the proof. □

Lemma 2.6 (Diamond Property, Part II)**.** *Let t be a well typed term in* RSLR*; if $t \to v_1, v_2$ and $t \to z$ then one of the following sentence is valid:*

- $\exists e_1, e_2$ *s.t.* $v_1 \to e_1$ *and* $v_2 \to e_2$ *and* $z \to e_1, e_2$
- $\forall i. v_i \to z$
- $z \to v_1, v_2$

Proof. By induction on the structure of a typing derivation for the term t.

- t cannot be a constant or a variable. Indeed if t is `rand`, t reduces in $0, 1$ and this contradict our hypothesis.
- If last rule is T-Sub or T-Arr-I, the thesis is easily proved by applying induction hypothesis.
- If last rule is T-Case, our derivation will have the following shape:

$$\frac{\begin{array}{ccc}\Gamma;\Delta_1 \vdash s : N & & \Gamma;\Delta_3 \vdash q : A \\ \Gamma;\Delta_2 \vdash r : A & & \Gamma;\Delta_4 \vdash u : A \qquad A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  If we perform the two reductions on the same subterm we could be in the following case (all the other cases are similar). For example, if t reduces to $\mathtt{case}_A\ s_1\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$ and $\mathtt{case}_A\ s_2\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$ and also to $t \to \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ f$, it is easy to check that the two required terms are $e_1 = \mathtt{case}_A\ s_1\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ f$ and $e_2 = \mathtt{case}_A\ s_2\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ f$. Another possible case is where on one hand we perform a reduction by selecting a branch and on the other case we make a reduction on one branch. As example, $t \to q$ and $r \to r_1, r_2$. This case is trivial.
- If last rule was T-Rec, our derivation will have the following shape:

$$\frac{\begin{array}{ccc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & & \Gamma_2;\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\, s\, r : B}\ (\text{T-Rec})$$

  By definition, we can have reduction only in q. By applying induction hypothesis the thesis is proved.
- If last rule was T-Arr-E. Our term could have different shapes but the only interesting cases are the following ones. The other cases can be easily brought back to cases that we have considered.
  - Our derivation will end in the following way:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : aA.r : bC \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: b}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : aA.r)s : B}\ (\text{T-Arr-E})$$

    where $C <: A$ and $b <: a$. We have that $(\lambda x : aA.r)s$ rewrites in $r[x/s]$; if $A = \mathbf{N}$ then s is a value, otherwise we are able to make the substitution whenever we want. If we reduce only in s or only in r we can easily prove our thesis by applying induction hypothesis. The interesting cases are when we perform the substitution on one hand and on the other hand we make a reduction step on one of the two possible terms s or r. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.r)s_1, (\lambda x : aA.r)s_2$, where $s \to s_1, s_2$. Let e_1 be $r[x/s_1]$ and e_2 be $r[x/s_2]$. We have that $(\lambda x : aA.r)s_1 \to e_1$, $(\lambda x : aA.r)s_2 \to e_2$ and $r[x/s] \to e_1, e_2$. Indeed if A is $\mathbf{N}$ then s is a value (because we are making substitutions) and we cannot have the reductions on s, otherwise there is at least one occurrence of s in $r[x/s]$ and by performing one reduction step on the subterm s we are able to have e_1, e_2. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.r_1)s, (\lambda x : aA.r_2)s$, where $r \to r_1, r_2$. This, again, can be easily managed.

- The other interesting case is when we perform the so called "swap". $(\lambda x : aA.q)sr$ rewrites to $(\lambda x : aA.qr)s$. If the reduction steps are made only on q or s or r, applying induction hypothesis suffices. In all the other cases, where we perform one step on subterms and we perform, on the other hand, the swap, it's easy to find the required term e.

This concludes the proof. □

Lemma 2.7 (Diamond Property, Part III)**.** *Let t be a well typed term in* RSLR*; if $t \to v_1, v_2$ and $t \to z_1, z_2$ (v_1, v_2 and z_1, z_2 different) then $\exists e_1, e_2, e_3, e_4$ s.t. $v_1 \to e_1, e_2$ and $v_2 \to e_3, e_4$ and $z_1 \to e_1, e_3$ and $z_2 \to e_2, e_4$.*

Proof. By induction on the structure of a typing derivation for t. Some interesting cases:

- If last rule was (T-Case) our derivation has the following shape:

$$\frac{\begin{array}{ccc}\Gamma;\Delta_1 \vdash s : N & \Gamma;\Delta_3 \vdash q : A & \\ \Gamma;\Delta_2 \vdash r : A & \Gamma;\Delta_4 \vdash u : A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  Also this case is easy to prove. Indeed if the reduction steps are made only on single subterms: s or r or q or u we can prove by using induction hypothesis. Otherwise we are in the case where one reduction step is made on some subterm and the other is made considering a different subterm. Suppose $s \to s_1, s_2$ and $q \to q_1, q_2$. We could have two possible reduction. One is $t \to \mathtt{case}_A\ s_1\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u, \mathtt{case}_A\ s_2\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$ and the other is $t \to \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q_1\ \mathtt{odd}\ u, \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q_2\ \mathtt{odd}\ u$. It is easy to find the common confluent terms: are the ones in which we have performed both $s \to s_1, s_2$ and $q \to q_1, q_2$.
- If last rule was (T-Rec) our derivation will have the following shape:

$$\frac{\begin{array}{ccc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & & \Gamma_2;\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\, s\, r : B}\ (\text{T-Rec})$$

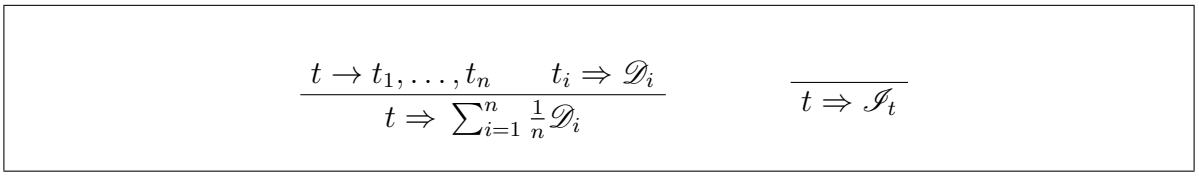
  By definition, we can have reduction only in q. By applying induction hypothesis the thesis is proved.
- If last rule was (T-Arr-E). Our term could have different shapes but all of them are trivial or can be easily brought back to cases that we have considered. Also the case where we consider the so called "swap" and the usual application with a lambda abstraction are not interesting in this lemma. Indeed, we cannot consider the "swap" or the substitution case because the reduction relation gives only one term on the right side of the arrow $\to$.

This concludes the proof. □

It is definitely not trivial to prove confluence for $\rightsquigarrow$. For this purpose we will prove our statement on a different definition of *multistep* arrow. This new definition is laxer than the

standard one. Being able to prove our theorems for multistep arrow allows us to conclude that these theorems hold also for $\rightsquigarrow$.

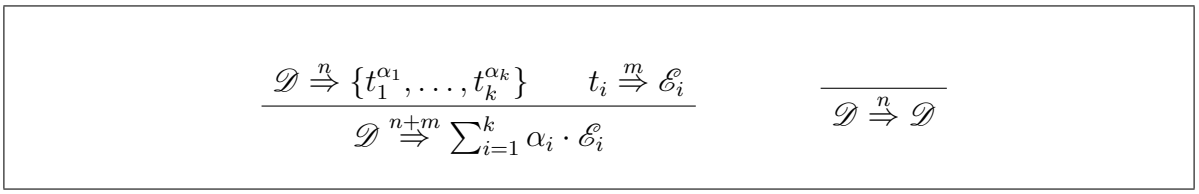
Definition 2.11. The binary relation $\Rightarrow$ is a set of pairs whose first component is a term and whose second component is a distribution on *not-necessarily-normal* terms, namely a function $\mathscr{D} : \Lambda \to [0,1]$ such that $\sum_{t\in\Lambda} \mathscr{D}(t) = 1$. As usual, $\mathscr{I}_t$ is the distribution that maps term t to 1 and any other term to 0. It is easy to check that if $t \rightsquigarrow \mathscr{D}$ then $t \Rightarrow \mathscr{D}$ (but not vice-versa). Formally, rules for $\Rightarrow$ are in Figure 5.



$$\frac{t \to t_1, \ldots, t_n \qquad t_i \Rightarrow \mathscr{D}_i}{t \Rightarrow \sum_{i=1}^{n} \frac{1}{n}\mathscr{D}_i} \qquad\qquad \frac{}{t \Rightarrow \mathscr{I}_t}$$

Figure 5: Generalized Multistep Reduction: Inference Rules

In any formal system, the height $||\pi||$ of a any derivation π is just the height of the derivation, seen as a tree (where by convention leaves have null height). Write $t \overset{n}{\Rightarrow} \mathscr{D}$ if the height $||\pi||$ of the derivation $\pi : t \Rightarrow \mathscr{D}$ is *bounded* by n. We can generalize $\overset{n}{\Rightarrow}$ to a ternary relation on *distributions* by the rules in Figure 6. When $\mathscr{D} \overset{n}{\Rightarrow} \mathscr{E}$ for some n, we simply write $\mathscr{D} \Rightarrow \mathscr{E}$.



$$\frac{\mathscr{D} \overset{n}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{m}{\Rightarrow} \mathscr{E}_i}{\mathscr{D} \overset{n+m}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{E}_i} \qquad\qquad \frac{}{\mathscr{D} \overset{n}{\Rightarrow} \mathscr{D}}$$

Figure 6: Generalized Multistep Reduction on Distributions: Inference Rules

Lemma 2.8. *If $\mathscr{D} \overset{n}{\Rightarrow} \mathscr{E}$ and $m \geq n$, then $\mathscr{D} \overset{m}{\Rightarrow} \mathscr{E}$.*

Proof. A simple induction on the structure of the proof that $\mathscr{D} \overset{n}{\Rightarrow} \mathscr{E}$. □

Lemma 2.9. *If $\mathscr{D} \overset{n}{\Rightarrow} \mathscr{E}$ and $\mathscr{E} \overset{m}{\Rightarrow} \mathscr{P}$, then $\mathscr{D} \overset{n+m}{\Rightarrow} \mathscr{P}$.*

Proof. By induction on the structure of the proof that $\mathscr{E} \overset{m}{\Rightarrow} \mathscr{P}$:

- If $\mathscr{E} = \mathscr{P}$, then $\mathscr{D} \overset{n}{\Rightarrow} \mathscr{E} = \mathscr{P}$ by hypothesis and, by Lemma 2.8, $\mathscr{D} \overset{n+m}{\Rightarrow} \mathscr{P}$;
- If the last inference step in the proof of $\mathscr{E} \overset{m}{\Rightarrow} \mathscr{P}$ looks as follows

$$\frac{\mathscr{E} \overset{x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{E} \overset{x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i}$$

then we can apply the inductive hypothesis and conclude that $\mathscr{D} \overset{n+x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\}$, from which one easily gets the thesis (since $m = x + y$):

$$\frac{\mathscr{D} \overset{n+x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{E} \overset{n+x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i}.$$

This concludes the proof. □

Lemma 2.10. *$\mathscr{D} \overset{0}{\Rightarrow} \mathscr{E}$, then $\mathscr{D} = \mathscr{E}$.*

Proof. By an easy induction on the structure of a proof that $\mathscr{D} \overset{0}{\Rightarrow} \mathscr{E}$. □

Lemma 2.11. *If $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$, then there is $\mathscr{P}$ such that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P} \overset{n}{\Rightarrow} \mathscr{E}$.*

Proof. An induction on the structure of a proof that $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$:

- If $\mathscr{D} = \mathscr{E}$, then of course $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{D} \overset{n}{\Rightarrow} \mathscr{D} = \mathscr{E}$.
- If the last inference step in the proof of $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$ looks as follows

$$\frac{\mathscr{D} \overset{x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{D} \overset{x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i = \mathscr{E}}$$

  and $x \geq 1$, then there is z such that $x = z + 1$. We can apply the induction hypothesis, obtaining a distribution $\mathscr{J}$ such that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{J} \overset{z}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\}$. Moreover, we can easily prove that $\mathscr{J} \overset{z+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i$:

$$\frac{\mathscr{J} \overset{z}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{J} \overset{z+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i}$$

- If the last inference step in the proof of $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$ looks as follows

$$\frac{\mathscr{D} \overset{x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{D} \overset{x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i = \mathscr{E}}$$

  and $x = 0$, then by Lemma 2.8 we can conclude that $\mathscr{D} = \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\}$. For every $1 \leq i \leq n$ there is a sequence $s_{i,1}, \ldots, s_{i,m_i}$ such that:
  - either $m_i = 1$, $t_i = s_{i,1}$ and $\mathscr{L}_i = \{t_i^1\}$;
  - or $t_i \to s_{i,1}, \ldots, s_{i,m_i}$, for every $1 \leq j \leq m_i$ there is $\mathscr{J}_j^i$ such that $s_{i,j} \overset{z}{\Rightarrow} \mathscr{J}_j^i$, where $y = z + 1$ and $\mathscr{L}_i = \sum_{j=1}^{m_i} \frac{1}{m_i} \cdot \mathscr{J}_j^i$.

  The required distribution $\mathscr{P}$, then, is just

$$\{s_{1,1}^{\frac{\alpha_1}{m_1}}, \ldots, s_{1,m_1}^{\frac{\alpha_1}{m_1}}, \ldots, s_{k,1}^{\frac{\alpha_k}{m_k}}, \ldots, s_{k,m_k}^{\frac{\alpha_k}{m_k}}\}.$$

  The fact $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P} \overset{n}{\Rightarrow} \mathscr{E}$ can be easily derived, since

$$\mathscr{E} = \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i = \sum_{i=1}^{k} \alpha_i \cdot \sum_{j=1}^{m_i} \frac{1}{m_i} \cdot \mathscr{J}_j^i = \sum_{i=1}^{k} \sum_{j=1}^{m_i} \frac{\alpha_i}{m_i} \cdot \mathscr{J}_j^i.$$

This concludes the proof. □

Lemma 2.12. *If $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{E}$ and $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P}$, then there is $\mathscr{L}$ such that $\mathscr{E} \overset{1}{\Rightarrow} \mathscr{L}$ and $\mathscr{P} \overset{1}{\Rightarrow} \mathscr{L}$.*

Proof. By induction on the structure of the proofs that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{E}$ and $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P}$. The only interesting case is the one in which $\mathscr{E} = \sum_{i=1}^{n} \alpha_i \cdot \mathscr{L}_i$ and $\mathscr{P} = \sum_{j=1}^{m} \beta_j \cdot \mathscr{J}_j$:

$$\begin{aligned}
&\mathscr{D} = \{t_1^{\alpha_1}, \ldots, t_n^{\alpha_n}\};\\
&\mathscr{D} = \{s_1^{\beta_1}, \ldots, s_m^{\beta_m}\};\\
&\quad \forall i \in \{1, \ldots, n\}. t_i \overset{1}{\Rightarrow} \mathscr{L}_i;\\
&\quad \forall j \in \{1, \ldots, m\}. s_j \overset{1}{\Rightarrow} \mathscr{J}_j.
\end{aligned}$$

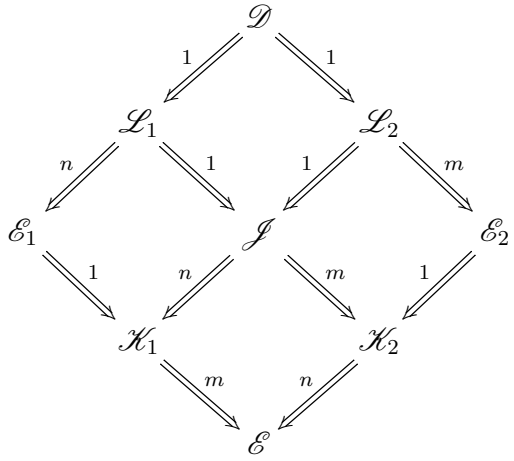
Without losing generality, we can assume that $n = m$, $t_i = s_i$ and $\alpha_i = \beta_i$. Moreover, $\mathscr{E}$ can be written as follows

$$\{r_{1,1}^{\frac{\alpha_1}{m_1}}, \ldots, r_{1,m_1}^{\frac{\alpha_1}{m_1}}, \ldots, r_{n,1}^{\frac{\alpha_n}{m_n}}, \ldots, r_{n,m_n}^{\frac{\alpha_n}{m_n}}\},$$

where either $m_i = 1$ and $r_{i,1} = t_i$ or $t_i \to r_{i,1}, \ldots, r_{i,m_i}$. Similarly for $\mathscr{P}$. By exploiting Lemma 2.5, Lemma 2.6 and Lemma 2.7, the distribution $\mathscr{L}$ can be easily defined. □

Theorem 2.13. *If $\mathscr{D} \Rightarrow \mathscr{E}$ and $\mathscr{D} \Rightarrow \mathscr{P}$, then there is $\mathscr{L}$ such that $\mathscr{E} \Rightarrow \mathscr{L}$ and $\mathscr{P} \Rightarrow \mathscr{L}$.*

Proof. We will actually prove a strengthening of the theorem above, namely the following: If $\mathscr{D} \overset{n}{\Rightarrow} \mathscr{E}_1$ and $\mathscr{D} \overset{m}{\Rightarrow} \mathscr{E}_2$, then there is $\mathscr{P}$ such that $\mathscr{E}_1 \overset{m}{\Rightarrow} \mathscr{P}$ and $\mathscr{E}_2 \overset{n}{\Rightarrow} \mathscr{P}$. This goes by a very simple induction, whose only interesting case is the one where $n, m \geq 1$. In that case, Lemma 2.11 guarantees that there are $\mathscr{L}_1, \mathscr{L}_2$ such that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{L}_1 \overset{n}{\Rightarrow} \mathscr{E}_1$ and $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{L}_2 \overset{m}{\Rightarrow} \mathscr{E}_2$. By Lemma 2.12, one can build a distribution $\mathscr{J}$ such that $\mathscr{L}_1 \overset{1}{\Rightarrow} \mathscr{J}_1$ and $\mathscr{L}_2 \overset{1}{\Rightarrow} \mathscr{J}_2$. By the induction hypothesis (applied twice) one obtains $\mathscr{K}_1, \mathscr{K}_2$ such that $\mathscr{J} \overset{n}{\Rightarrow} \mathscr{K}_1$, $\mathscr{E}_1 \overset{1}{\Rightarrow} \mathscr{K}_1$, $\mathscr{J} \overset{m}{\Rightarrow} \mathscr{K}_2$, $\mathscr{E}_2 \overset{1}{\Rightarrow} \mathscr{K}_2$. Finally, another application of the induction hypothesis gives us a distribution $\mathscr{P}$ such that $\mathscr{K}_1 \overset{m}{\Rightarrow} \mathscr{P}$ and $\mathscr{K}_2 \overset{n}{\Rightarrow} \mathscr{P}$. Lemma 2.9 is now enough to get the thesis. Summing up, the structure of the proof is the classic one:



□

Corollary 2.14 (Multistep Confluence)**.** *If $t \rightsquigarrow \mathscr{D}$ and $t \rightsquigarrow \mathscr{E}$ then $\mathscr{D} = \mathscr{E}$.*

Proof. An easy consequence of Theorem 2.13: if $t \rightsquigarrow \mathscr{D}$ and $t \rightsquigarrow \mathscr{E}$, then $\{t^1\} \Rightarrow \mathscr{D}$ and $\{t^1\} \Rightarrow \mathscr{E}$. As a consequence, $\mathscr{D} \Rightarrow \mathscr{P}$ and $\mathscr{E} \Rightarrow \mathscr{P}$, but $\mathscr{D} = \mathscr{P} = \mathscr{E}$, because all the terms to which $\mathscr{D}$ and $\mathscr{E}$ attribute a nonnull probability are normal forms. □

Example 2.3. Consider again the term $t = (\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))\mathtt{rand}$, where $t_\oplus$ is a term computing $\oplus$ on natural numbers seen as booleans (0 stands for "false" and everything else stands for "true"):

$$
\begin{aligned}
t_\oplus &= \lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus;\\
s_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 0\ \mathtt{even}\ 1\ \mathtt{odd}\ 1;\\
r_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 1\ \mathtt{even}\ 0\ \mathtt{odd}\ 0.
\end{aligned}
$$

In order to simplify reading, let us define:

$$
\begin{aligned}
f &= (t_\oplus xx);\\
g_0 &= (\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ 0\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus);\\
g_1 &= (\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ 1\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus);\\
h_0 &= \mathtt{case}_{\mathbf{N}}\ 0\ \mathtt{zero}\ 0\ \mathtt{even}\ 1\ \mathtt{odd}\ 1;\\
h_1 &= \mathtt{case}_{\mathbf{N}}\ 1\ \mathtt{zero}\ 1\ \mathtt{even}\ 0\ \mathtt{odd}\ 0.
\end{aligned}
$$

We can now give the following derivation tree:

$$
\dfrac{(\lambda x : \blacksquare\mathbf{N}.f)\mathtt{rand} \to (\lambda x : \blacksquare\mathbf{N}.f)0, (\lambda x : \blacksquare\mathbf{N}.f)1 \qquad \pi : (\lambda x : \blacksquare\mathbf{N}.f)0 \rightsquigarrow \{0^1\} \qquad \rho : (\lambda x : \blacksquare\mathbf{N}.f)1 \rightsquigarrow \{0^1\}}{(\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))\mathtt{rand} \rightsquigarrow \{0^1\}}
$$

where π and ρ are as follows:

$$
\pi :\ \dfrac{(\lambda x : \blacksquare\mathbf{N}.f)0 \to t_\oplus 00 \qquad \dfrac{t_\oplus 0\,0 \to g_0 0 \qquad \dfrac{g_0 0 \to s_\oplus 0 \qquad \dfrac{s_\oplus 0 \to h_0 \qquad \dfrac{h_0 \to 0 \qquad 0 \rightsquigarrow \{0^1\}}{h_0 \rightsquigarrow \{0^1\}}}{s_\oplus 0 \rightsquigarrow \{0^1\}}}{g_0 0 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus)0\,0 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.f)0 \rightsquigarrow \{0^1\}}
$$

$$
\rho :\ \dfrac{(\lambda x : \blacksquare\mathbf{N}.f)1 \to t_\oplus 11 \qquad \dfrac{t_\oplus 1\,1 \to g_1 1 \qquad \dfrac{g_1 1 \to r_\oplus 1 \qquad \dfrac{r_\oplus 1 \to h_1 \qquad \dfrac{h_1 \to 0 \qquad 0 \rightsquigarrow \{0^1\}}{h_1 \rightsquigarrow \{0^1\}}}{r_\oplus 1 \rightsquigarrow \{0^1\}}}{g_1 1 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus)1\,1 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.f)1 \rightsquigarrow \{0^1\}}
$$

3. Probabilistic Polytime Soundness

The most difficult (and interesting!) result about RSLR is definitely polytime soundness: every (instance of) a first-order term can be reduced to a numeral in a polynomial number of steps by a probabilistic Turing machine. Polytime soundness can be proved, following [2], by showing that:

- Any explicit term of base type can be reduced to its normal form with very low time complexity;
- Any term (non necessarily of base type) can be put in explicit form in polynomial time.

By gluing these two results together, we obtain what we need, namely an effective and efficient procedure to compute the normal forms of terms. Formally, two notions of evaluation for terms correspond to the two steps defined above:

- On the one hand, we need a ternary relation $\Downarrow_{\mathsf{nf}}$ between closed terms of type $\mathbf{N}$, probabilities and numerals. Intuitively, $t \Downarrow_{\mathsf{nf}}^{\alpha} n$ holds when t is explicit and rewrites to n with probability α. The inference rules for $\Downarrow_{\mathsf{nf}}$ are defined in Figure 7;
- On the other hand, we need a ternary relation $\Downarrow_{\mathsf{rf}}$ between terms of non modal type, probabilities and terms. We can derive $t \Downarrow_{\mathsf{rf}}^{\alpha} s$ only if t can be transformed into s with probability α consistently with the reduction relation. The inference rules for $\Downarrow_{\mathsf{rf}}$ are in Figure 8.

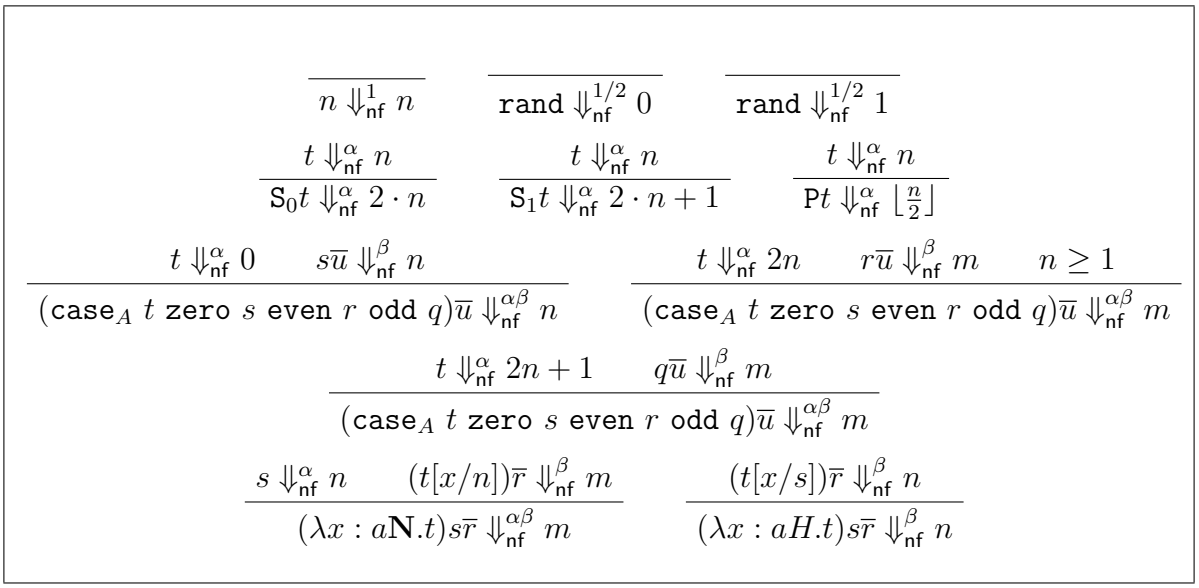


Figure 7: The Relation $\Downarrow_{\mathsf{nf}}$: Inference Rules

Moreover, a third ternary relation $\Downarrow$ between closed terms of type $\mathbf{N}$, probabilities and numerals can be defined by the rule below:

$$\frac{t \Downarrow_{\mathsf{rf}}^{\alpha} s \quad s \Downarrow_{\mathsf{nf}}^{\beta} n}{t \Downarrow^{\alpha\beta} n}$$

$$\frac{}{c \Downarrow_{\mathsf{rf}}^{1} c} \qquad \frac{t \Downarrow_{\mathsf{rf}}^{\alpha} v}{\mathtt{S}_0 t \Downarrow_{\mathsf{rf}}^{\alpha} \mathtt{S}_0 v} \qquad \frac{t \Downarrow_{\mathsf{rf}}^{\alpha} v}{\mathtt{S}_1 t \Downarrow_{\mathsf{rf}}^{\alpha} \mathtt{S}_1 v} \qquad \frac{t \Downarrow_{\mathsf{rf}}^{\alpha} v}{\mathtt{P} t \Downarrow_{\mathsf{rf}}^{\alpha} \mathtt{P} v}$$

$$\frac{\begin{array}{lll} t \Downarrow_{\mathsf{rf}}^{\alpha} v & r \Downarrow_{\mathsf{rf}}^{\gamma} e & \\ s \Downarrow_{\mathsf{rf}}^{\beta} z & q \Downarrow_{\mathsf{rf}}^{\delta} f & \forall u_i \in \overline{u}, u_i \Downarrow_{\mathsf{rf}}^{\epsilon_i} g_i \end{array}}{(\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q)\overline{u} \Downarrow_{\mathsf{rf}}^{\alpha\beta\gamma\delta\prod_i \epsilon_i} (\mathtt{case}_A\ v\ \mathtt{zero}\ z\ \mathtt{even}\ e\ \mathtt{odd}\ f)\overline{g}}$$

$$\frac{\begin{array}{lll} t \Downarrow_{\mathsf{rf}}^{\alpha} v & n > 0 & s \Downarrow_{\mathsf{rf}}^{\gamma} z \\ v \Downarrow_{\mathsf{nf}}^{\beta} n & \forall q_i \in \overline{q}, q_i \Downarrow_{\mathsf{rf}}^{\delta_i} f_i & r\lfloor \frac{n}{2^0} \rfloor \Downarrow_{\mathsf{rf}}^{\gamma_0} r_0 \ \ldots \ r\lfloor \frac{n}{2^{|n|-1}} \rfloor \Downarrow_{\mathsf{rf}}^{\gamma_{|n|-1}} r_{|n|-1} \end{array}}{(\mathtt{recursion}_A\ t\, s\, r)\overline{q} \Downarrow_{\mathsf{rf}}^{\alpha\beta\gamma(\prod_j \gamma_j)(\prod_i \delta_i)} r_0(\ldots(r_{(|n|-1)}z)\ldots)\overline{f}}$$

$$\frac{\begin{array}{ll} t \Downarrow_{\mathsf{rf}}^{\alpha} v & s \Downarrow_{\mathsf{rf}}^{\gamma} z \\ v \Downarrow_{\mathsf{nf}}^{\beta} 0 & \forall q_i \in \overline{q}, q_i \Downarrow_{\mathsf{rf}}^{\delta_i} f_i \end{array}}{(\mathtt{recursion}_A\ t\, s\, r)\overline{q} \Downarrow_{\mathsf{rf}}^{\alpha\beta\gamma(\prod_i \delta_i)} z\overline{f}}$$

$$\frac{\begin{array}{ll} s \Downarrow_{\mathsf{rf}}^{\alpha} z & \\ z \Downarrow_{\mathsf{nf}}^{\gamma} n & (t[x/n])\overline{r} \Downarrow_{\mathsf{rf}}^{\beta} u \end{array}}{(\lambda x : \Box\mathbf{N}.t)s\overline{r} \Downarrow_{\mathsf{rf}}^{\alpha\gamma\beta} u} \qquad \frac{\begin{array}{ll} s \Downarrow_{\mathsf{rf}}^{\alpha} z & \\ z \Downarrow_{\mathsf{nf}}^{\gamma} n & t\overline{r} \Downarrow_{\mathsf{rf}}^{\beta} u \end{array}}{(\lambda x : \blacksquare\mathbf{N}.t)s\overline{r} \Downarrow_{\mathsf{rf}}^{\alpha\gamma\beta} (\lambda x : \blacksquare\mathbf{N}.u)n}$$

$$\frac{(t[x/s])\overline{r} \Downarrow_{\mathsf{rf}}^{\beta} u}{(\lambda x : aH.t)s\overline{r} \Downarrow_{\mathsf{rf}}^{\beta} u} \qquad \frac{t \Downarrow_{\mathsf{rf}}^{\beta} u}{\lambda x : aA.t \Downarrow_{\mathsf{rf}}^{\beta} \lambda x : aA.u} \qquad \frac{t_j \Downarrow_{\mathsf{rf}}^{\alpha_j} s_j}{x\overline{t} \Downarrow_{\mathsf{rf}}^{\prod_i \alpha_i} x\overline{s}}$$

Figure 8: The Relation $\Downarrow_{\mathsf{rf}}$: Inference Rules

A peculiarity of the just introduced relations with respect to similar ones is the following: whenever a statement in the form $t \Downarrow_{\mathsf{nf}}^{\alpha} s$ is an immediate premise of another statement $r \Downarrow_{\mathsf{nf}}^{\beta} q$, then t needs to be structurally smaller than r, provided all numerals are assumed to have the same internal structure. A similar but weaker statement holds for $\Downarrow_{\mathsf{rf}}$. This relies on the peculiarities of RSLR, and in particular on the fact that variables of higher-order types can appear free at most once in terms, and that terms of base type cannot be passed to functions without having been completely evaluated. In other words, the just described operational semantics is structural in a very strong sense, and this allows to prove properties about it by induction on the structure of *terms*, as we will experience in a moment.

Before starting to study the combinatorial properties of $\Downarrow_{\mathsf{rf}}$ and $\Downarrow_{\mathsf{nf}}$, it is necessary to show that, at least, $\Downarrow$ is adequate as a way to evaluate lambda terms. In the following, the size $|\pi|$ of any derivation π (for any formal system) is simply the number of distinct rule occurrences in π.

Theorem 3.1 (Adequacy)**.** *For every term t such that $\vdash t : \mathbf{N}$, the following two conditions are equivalent:*

1. *There are j distinct derivations $\pi_1 : t \Downarrow^{\alpha_1} n_1, \ldots, \pi_j : t \Downarrow^{\alpha_j} n_j$ such that $\sum_{i=1}^{j} \alpha_i = 1$;*
2. *$t \rightsquigarrow \mathscr{D}$, where for every m, $\mathscr{D}(m) = \sum_{n_i = m} \alpha_i$.*

Proof. Implication 1. $\Rightarrow$ 2. can be proved by an induction on $\sum_{k=1}^{j} |\pi_k|$, appropriately enriching the thesis with similar statements for $\Downarrow_{\mathsf{rf}}$ and $\Downarrow_{\mathsf{nf}}$. About the converse, just observe that, *some* derivations like the ones required in Condition 1. need to exist. On $\Downarrow_{\mathsf{nf}}$, this can be formally proved by induction on $|t|_{\mathsf{w}}$, where $|\cdot|_{\mathsf{w}}$ is defined as follows: $|x|_{\mathsf{w}} = 1$, $|ts|_{\mathsf{w}} = |t|_{\mathsf{w}} + |s|_{\mathsf{w}}$, $|\lambda x : aA.t|_{\mathsf{w}} = |t|_{\mathsf{w}} + 1$, $|\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q|_{\mathsf{w}} = |t|_{\mathsf{w}} + |s|_{\mathsf{w}} + |r|_{\mathsf{w}} + |q|_{\mathsf{w}} + 1$, $|\mathtt{recursion}_A\ t\, s\, r|_{\mathsf{w}} = |t|_{\mathsf{w}} + |s|_{\mathsf{w}} + |r|_{\mathsf{w}} + 1$, $|n|_{\mathsf{w}} = 1$, $|\mathtt{S}_0|_{\mathsf{w}} = |\mathtt{S}_1|_{\mathsf{w}} = |\mathtt{P}|_{\mathsf{w}} = |\mathtt{rand}|_{\mathsf{w}} = 1$. On $\Downarrow_{\mathsf{rf}}$, the same can be proved by induction on a lexicographic order taking into account the recursion depth of t and $|t|_{\mathsf{w}}$. Thanks to multistep confluence, we can conclude. □

It's now time to analyse how big derivations for $\Downarrow_{\mathsf{nf}}$ and $\Downarrow_{\mathsf{rf}}$ can be with respect to the size of the underlying term. Let us start with $\Downarrow_{\mathsf{nf}}$ and prove that, since it can only be applied to explicit terms, the sizes of derivations must be very small:

Proposition 3.2. *Suppose that $\vdash t : \mathbf{N}$, where t is explicit. Then for every $\pi : t \Downarrow_{\mathsf{nf}}^{\alpha} m$ it holds that:*

1. *$|\pi| \leq |t|$;*
2. *If $s \in \pi$, then $|s| \leq 2 \cdot |t|^2$.*

Proof. Given any term t, $|t|_{\mathsf{w}}$ and $|t|_{\mathsf{n}}$ are defined, respectively, as the size of t where every numeral counts for 1 and the maximum size of the numerals that occur in t. For a formal definition of $|\cdot|_{\mathsf{w}}$, see the proof of Theorem 3.1. On the other hand, $|\cdot|_{\mathsf{n}}$ is defined as follows: $|x|_{\mathsf{n}} = 1$, $|ts|_{\mathsf{n}} = \max\{|t|_{\mathsf{n}}, |s|_{\mathsf{n}}\}$, $|\lambda x : aA.t|_{\mathsf{n}} = |t|_{\mathsf{n}}$, $|\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q|_{\mathsf{n}} = \max\{|t|_{\mathsf{n}}, |s|_{\mathsf{n}}, |r|_{\mathsf{n}}, |q|_{\mathsf{n}}\}$, $|\mathtt{recursion}_A\ t\, s\, r|_{\mathsf{n}} = \max\{|t|_{\mathsf{n}}, |s|_{\mathsf{n}}, |r|_{\mathsf{n}}\}$, $|n|_{\mathsf{n}} = \lfloor \log_2(n) \rfloor + 1$,

and $|\mathtt{S}_0|_{\mathsf{n}} = |\mathtt{S}_1|_{\mathsf{n}} = |\mathtt{P}|_{\mathsf{n}} = |\mathtt{rand}|_{\mathsf{n}} = 1$. It holds that $|t| \leq |t|_{\mathsf{w}} \cdot |t|_{\mathsf{n}}$. It can be proved by structural induction on term t. We prove the following strengthening of the statements above by induction on $|t|_{\mathsf{w}}$:

1. $|\pi| \leq |t|_{\mathsf{w}}$;
2. If $s \in \pi$, then $|s|_{\mathsf{w}} \leq |t|_{\mathsf{w}}$ and $|s|_{\mathsf{n}} \leq |t|_{\mathsf{n}} + |t|_{\mathsf{w}}$;

First we prove that this is indeed a strengthening of the thesis. From the first case of the strengthening, we can deduce the first case of the main thesis. Notice indeed that $|t|_{\mathsf{w}} \leq |t|$. Regarding the latter point, notice that $|s| \leq |s|_{\mathsf{w}} \cdot |s|_{\mathsf{n}} \leq |t|_{\mathsf{w}} \cdot (|t|_{\mathsf{n}} + |t|_{\mathsf{w}}) \leq |t|^2 + |t| \leq 2 \cdot |t|^2$. Some interesting cases:

- Suppose t is `rand`. We could have two derivations:

$$\frac{}{\mathtt{rand} \Downarrow_{\mathsf{nf}}^{1/2} 0} \qquad \frac{}{\mathtt{rand} \Downarrow_{\mathsf{nf}}^{1/2} 1}$$

  The thesis is easily proved.
- Suppose t is $\mathtt{S}_i s$. Depending on $\mathtt{S}_i$ we could have two different derivations:

$$\frac{\rho : s \Downarrow_{\mathsf{nf}}^{\alpha} n}{\mathtt{S}_0 s \Downarrow_{\mathsf{nf}}^{\alpha} 2 \cdot n} \qquad \frac{\rho : s \Downarrow_{\mathsf{nf}}^{\alpha} n}{\mathtt{S}_1 s \Downarrow_{\mathsf{nf}}^{\alpha} 2 \cdot n + 1}$$

  Suppose we are in the case where $\mathtt{S}_i = \mathtt{S}_0$. Then, for every $r \in \pi$,

$$\begin{aligned} |\pi| &= |\rho| + 1 \leq |s|_{\mathsf{w}} + 1 = |t|_{\mathsf{w}}; \\ |r|_{\mathsf{w}} &\leq |s|_{\mathsf{w}} \leq |t|_{\mathsf{w}}; \\ |r|_{\mathsf{n}} &\leq |s|_{\mathsf{n}} + |s|_{\mathsf{w}} + 1 = |s|_{\mathsf{n}} + |t|_{\mathsf{w}} \ = |t|_{\mathsf{n}} + |t|_{\mathsf{w}}. \end{aligned}$$

  The case where $\mathtt{S}_i = \mathtt{S}_1$ is proved in the same way.
- Suppose t is $\mathtt{P}s$.

$$\frac{\rho : s \Downarrow_{\mathsf{nf}}^{\alpha} 0}{\mathtt{P}s \Downarrow_{\mathsf{nf}}^{\alpha} 0} \qquad \frac{\rho : s \Downarrow_{\mathsf{nf}}^{\alpha} n \qquad n \geq 1}{\mathtt{P}s \Downarrow_{\mathsf{nf}}^{\alpha} \lfloor \frac{n}{2} \rfloor}$$

  We focus on case where $n > 1$, the other case is similar. For every $r \in \pi$ we have

$$\begin{aligned} |\pi| &= |\rho| + 1 \leq |s|_{\mathsf{w}} + 1 = |t|_{\mathsf{w}}; \\ |r|_{\mathsf{w}} &\leq |s|_{\mathsf{w}} \leq |t|_{\mathsf{w}}; \\ |r|_{\mathsf{n}} &\leq |s|_{\mathsf{n}} + |s|_{\mathsf{w}} + 1 = |s|_{\mathsf{n}} + |t|_{\mathsf{w}} = |t|_{\mathsf{n}} + |t|_{\mathsf{w}}. \end{aligned}$$

- Suppose t is n.

$$\frac{}{n \Downarrow_{\mathsf{nf}}^{1} n}$$

  By knowing $|\pi| = 1$, $|n|_{\mathsf{w}} = 1$ and $|n|_{\mathsf{n}} = |n|$, the proof is trivial.
- Suppose that t is $(\lambda y : a\mathbf{N}.s)r\overline{q}$. All derivations π for t are in the following form:

$$\frac{\rho : r \Downarrow_{\mathsf{nf}}^{\alpha} o \qquad \mu : (s[y/o])\overline{q} \Downarrow_{\mathsf{nf}}^{\beta} m}{t \Downarrow_{\mathsf{nf}}^{\alpha\beta} m}$$

Then, for every $u \in \pi$,

$$\begin{aligned}
|\pi| &\leq |\rho| + |\mu| + 1 \leq |r|_{\mathsf{w}} + |s[y/o]\overline{q}|_{\mathsf{w}} + 1 \\
&= |r|_{\mathsf{w}} + |s\overline{q}|_{\mathsf{w}} + 1 \leq |t|_{\mathsf{w}}; \\
|u|_{\mathsf{n}} &\leq \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{n}} + |s[y/o]\overline{q}|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{n}} + |s\overline{q}|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, \max\{|s\overline{q}|_{\mathsf{n}}, |o|\} + |s\overline{q}|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s\overline{q}|_{\mathsf{n}} + |s\overline{q}|_{\mathsf{w}}, |o| + |s\overline{q}|_{\mathsf{w}}\} \\
&\leq \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s\overline{q}|_{\mathsf{n}} + |s\overline{q}|_{\mathsf{w}}, |r|_{\mathsf{n}} + |r|_{\mathsf{w}} + |s\overline{q}|_{\mathsf{w}}\} \\
&\leq \max\{|r|_{\mathsf{n}}, |s\overline{q}|_{\mathsf{n}}\} + |r|_{\mathsf{w}} + |s\overline{q}|_{\mathsf{w}} \\
&\leq \max\{|r|_{\mathsf{n}}, |s\overline{q}|_{\mathsf{n}}\} + |t|_{\mathsf{w}} \\
&= |t|_{\mathsf{n}} + |t|_{\mathsf{w}}; \\
|u|_{\mathsf{w}} &\leq \max\{|r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{w}}, |t|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{w}}, |s\overline{q}|_{\mathsf{w}}, |t|_{\mathsf{w}}\} \leq |t|_{\mathsf{w}}.
\end{aligned}$$

If $u \in \pi$, then either $u \in \rho$ or $u \in \mu$ or simply $u = t$. This, together with the induction hypothesis, implies $|u|_{\mathsf{w}} \leq \max\{|r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{w}}, |t|_{\mathsf{w}}\}$. Notice that $|s\overline{q}|_{\mathsf{w}} = |s[y/o]\overline{q}|_{\mathsf{n}}$ holds because any occurrence of y in s counts for 1, but also o itself counts for 1 (see the definition of $|\cdot|_{\mathsf{w}}$ above). More generally, duplication of *numerals* for a variable in t does not make $|t|_{\mathsf{w}}$ bigger.

- Suppose t is $(\lambda y : aH.s)r\overline{q}$. Without loosing generality we can say that it derives from the following derivation:

$$\frac{\rho : (s[y/r])\overline{q} \Downarrow^{\beta}_{\mathsf{nf}} n}{(\lambda y : aH.s)r\overline{q} \Downarrow^{\beta}_{\mathsf{nf}} n}$$

  For the reason that y has type H we can be sure that it appears at most once in s. So, $|s[y/r]| \leq |sr|$ and, moreover, $|s[y/r]\overline{q}|_{\mathsf{w}} \leq |sr\overline{q}|_{\mathsf{w}}$ and $|s[y/r]\overline{q}|_{\mathsf{n}} \leq |sr\overline{q}|_{\mathsf{n}}$. We have, for all $u \in \rho$:

$$\begin{aligned}
|\pi| = |\rho| + 1 &\leq |s[y/r]\overline{q}|_{\mathsf{w}} + 1 \leq |t|_{\mathsf{w}}; \\
|u|_{\mathsf{w}} &\leq |s[y/r]\overline{q}|_{\mathsf{w}} \leq |sr\overline{q}|_{\mathsf{w}} \leq |t|_{\mathsf{w}}; \\
|u|_{\mathsf{n}} &\leq |s[y/r]\overline{q}|_{\mathsf{n}} + |s[y/r]\overline{q}|_{\mathsf{w}} \leq |sr\overline{q}|_{\mathsf{n}} + |sr\overline{q}|_{\mathsf{w}} \leq |t|_{\mathsf{n}} + |t|_{\mathsf{w}};
\end{aligned}$$

  and this means that the same inequalities hold for every $u \in \pi$.

- Suppose t is $\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$. We could have three possible derivations:

$$\frac{\rho : s \Downarrow^{\alpha}_{\mathsf{nf}} 0 \qquad \mu : r\overline{v} \Downarrow^{\beta}_{\mathsf{nf}} n}{(\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)\overline{v} \Downarrow^{\alpha\beta}_{\mathsf{nf}} n}$$

$$\frac{\rho : s \Downarrow^{\alpha}_{\mathsf{nf}} 2n \qquad \mu : q\overline{v} \Downarrow^{\beta}_{\mathsf{nf}} m \qquad n \geq 1}{(\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)\overline{v} \Downarrow^{\alpha\beta}_{\mathsf{nf}} m}$$

$$\frac{\rho : s \Downarrow_{\mathsf{nf}}^{\alpha} 2n+1 \qquad \mu : u\overline{v} \Downarrow_{\mathsf{nf}}^{\beta} m}{(\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)\overline{v} \Downarrow_{\mathsf{nf}}^{\alpha\beta} m}$$

we will focus on the case where the value of s is odd. All the other cases are similar. For all $z \in \pi$ we have:

$$\begin{aligned}
|\pi| &\leq |\rho| + |\mu| + 1 \\
&\leq |s|_{\mathsf{w}} + |u\overline{v}|_{\mathsf{w}} + 1 \leq |t|_{\mathsf{w}}; \\
|z|_{\mathsf{w}} &\leq |s|_{\mathsf{w}} + |r|_{\mathsf{w}} + |q|_{\mathsf{w}} + |u\overline{v}|_{\mathsf{w}} \leq |t|_{\mathsf{w}}; \\
|z|_{\mathsf{n}} &= \max\{|s|_{\mathsf{n}} + |s|_{\mathsf{w}}, |u\overline{v}|_{\mathsf{n}} + |u\overline{v}|_{\mathsf{w}}, |r|_{\mathsf{n}}, |q|_{\mathsf{n}}\} \\
&\leq \max\{|s|_{\mathsf{n}}, |u\overline{v}|_{\mathsf{n}} | r|_{\mathsf{n}}, |q|_{\mathsf{n}}\} + |s|_{\mathsf{w}} + |u\overline{v}|_{\mathsf{w}} \\
&\leq |t|_{\mathsf{w}} + |t|_{\mathsf{n}}.
\end{aligned}$$

This concludes the proof. □

As opposed to $\Downarrow_{\mathsf{nf}}$, $\Downarrow_{\mathsf{rf}}$ unrolls instances of primitive recursion, and thus cannot have the very simple combinatorial behaviour of $\Downarrow_{\mathsf{nf}}$. Fortunately, however, everything stays under control:

Proposition 3.3. *Suppose that $x_1 : \Box\mathbf{N}, \ldots, x_i : \Box\mathbf{N} \vdash t : A$, where A is $\Box$-free type. Then there are polynomials p_t and q_t such that for every $n_1, \ldots, n_i$ and for every $\pi : t[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} s$ it holds that:*

1. *$|\pi| \leq p_t(\sum_i |n_i|)$;*
2. *If $s \in \pi$, then $|s| \leq q_t(\sum_i |n_i|)$.*

Proof. The following strengthening of the result can be proved by induction on the structure of a type derivation μ for t: if $x_1 : \Box\mathbf{N}, \ldots, x_i : \Box\mathbf{N}, y_1 : \blacksquare A_1, \ldots, y_j : \blacksquare A_j \vdash t : A$, where A is positively $\Box$-free and $A_1, \ldots, A_j$ are negatively $\Box$-free. Then there are polynomials p_t and q_t such that for every $n_1, \ldots, n_i$ and for every $\pi : t[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} s$ it holds that:

1. $|\pi| \leq p_t(\sum_i |n_i|)$;
2. If $s \in \pi$, then $|s| \leq q_t(\sum_i |n_i|)$.

In defining positively and negatively $\Box$-free types, let us proceed by induction on types:

- $\mathbf{N}$ is both positively and negatively $\Box$-free;
- $\Box A \to B$ is *not* positively $\Box$-free, and is negatively $\Box$-free whenever A is positively $\Box$-free and B is negatively $\Box$-free;
- $C = \blacksquare A \to B$ is positively $\Box$-free if A is negatively and B is positively $\Box$-free. C is negatively $\Box$-free if A is positively $\Box$-free and B is negatively $\Box$-free.

Please observe that if A is positively $\Box$-free and $B <: A$, then B is positively $\Box$-free. Conversely, if A is negatively $\Box$-free and $A <: B$, then B is negatively $\Box$-free. This can be easily proved by induction on the structure of A. We are ready to start the proof, now. Let us consider some cases, depending on the shape of μ

- If the only typing rule in μ is (T-Const-Aff), then $t = c$, $p_t(x) = 1$ and $q_t(x) = 1$. The thesis is proved.
- If the last rule was (T-Var-Aff), then $t = x$, $p_t(x) = 1$ and $q_t(x) = x$. The thesis is proved
- If the last rule was (T-Arr-I), then $t = \lambda x : \blacksquare A.s$. Notice that the aspect is $\blacksquare$ because the type of our term has to be positively $\Box$-free. So, we have the following derivation:

$$\frac{\rho : s[\overline{x}/\overline{n}] \Downarrow^{\beta}_{\mathsf{rf}} v}{\lambda x : aA.s[\overline{x}/\overline{n}] \Downarrow^{\beta}_{\mathsf{rf}} \lambda x : aA.v}$$

  If the type of t is positively $\Box$-free, then also the type of s is positively $\Box$-free. We can apply induction hypothesis. Define p_t and q_t as:

$$p_t(x) = p_s(x) + 1;$$
$$q_t(x) = q_s(x) + 1.$$

  Indeed, we have:

$$|\pi| = |\rho| + 1 \leq p_s(\sum_i |n_i|) + 1.$$

- If last rule was (T-Sub) then we have a typing derivation that ends in the following way:

$$\frac{\Gamma \vdash t : A \qquad A <: B}{\Gamma \vdash t : B}$$

  we can apply induction hypothesis on $t : A$ because if B is positively $\Box$-free, then also A will be positively $\Box$-free. Define $p_{t:B}(x) = p_{t:A}(x)$ and $q_{t:B}(x) = q_{t:A}(x)$.
- If the last rule was (T-Case), suppose $t = (\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)$. The constraints on the typing rule (T-Case) ensure us that the induction hypothesis can be applied to s, r, q, u. The definition of $\Downarrow_{\mathsf{rf}}$ tells us that any derivation of $t[\overline{x}/\overline{n}]$ must have the following shape:

$$\frac{\begin{array}{cc}\rho : s[\overline{x}/\overline{n}] \Downarrow^{\alpha}_{\mathsf{rf}} z & \nu : q[\overline{x}/\overline{n}] \Downarrow^{\gamma}_{\mathsf{rf}} f \\ \mu : r[\overline{x}/\overline{n}] \Downarrow^{\beta}_{\mathsf{rf}} e & \sigma : u[\overline{x}/\overline{n}] \Downarrow^{\delta}_{\mathsf{rf}} g\end{array}}{t[\overline{x}/\overline{n}] \Downarrow^{\alpha\beta\gamma\delta}_{\mathsf{rf}} (\mathtt{case}_A\ z\ \mathtt{zero}\ e\ \mathtt{even}\ f\ \mathtt{odd}\ g)}$$

  Let us now define p_t and q_t as follows:

$$p_t(x) = p_s(x) + p_r(x) + p_q(x) + p_u(x) + 1;$$
$$q_t(x) = q_s(x) + q_r(x) + q_q(x) + q_u(x) + 1.$$

  We have:

$$\begin{aligned}|\pi| &\leq |\rho| + |\mu| + |\nu| + |\sigma| + 1 \\ &\leq p_s(\sum_i |n_i|) + p_r(\sum_i |n_i|) + p_q(\sum_i |n_i|) + p_u(\sum_i |n_i|) + 1 \\ &= p_t(\sum_i |n_i|).\end{aligned}$$

  Similarly, if $z \in \pi$, it is easy to prove that $|z| \leq q_z(\sum_i |n_i|)$.

- If the last rule was (T-Rec),m We consider the most interesting case, where the first term computes to a value greater than 0. Suppose $t = (\texttt{recursion}_A\ s\, r\, q)$. By looking at the typing rule (figure 4) for (T-Rec) we are sure to be able to apply induction hypothesis on s, r, q. Definition of $\Downarrow_{\mathsf{rf}}$ ensure also that any derivation for $t[\overline{x}/\overline{n}]$ must have the following shape:

$$\frac{\begin{array}{c}\rho : s[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} z \quad \mu : z[\overline{x}/\overline{n}] \Downarrow_{\mathsf{nf}}^{\beta} n \\ \nu : r[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\gamma} e \\ \varrho_0 : qy[\overline{x}, y/\overline{n}, \lfloor \frac{n}{2^0} \rfloor] \Downarrow_{\mathsf{rf}}^{\gamma_0} q_0 \\ \cdots \\ \varrho_{|n|-1} : qy[\overline{x}, y/\overline{n}, \lfloor \frac{n}{2^{|n|-1}} \rfloor] \Downarrow_{\mathsf{rf}}^{\gamma_{|n|-1}} q_{|n|-1}\end{array}}{(\texttt{recursion}_A\ s\, r\, q)[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha\beta\gamma(\prod_j \gamma_j)} q_0(\ldots(q_{(|n|-1)}e)\ldots)}$$

  Notice that we are able to apply $\Downarrow_{\mathsf{nf}}$ on term z because, by definition, s has only free variables of type $\Box\mathbf{N}$ (see figure 4). So, we are sure that z is a closed term of type $\mathbf{N}$ and we are able to apply the $\Downarrow_{\mathsf{nf}}$ algorithm. Let define p_t and q_t as follows:

$$\begin{aligned} p_t(x) &= p_s(x) + 2 \cdot q_s(x) + p_r(x) + 2 \cdot q_s(x)^2 \cdot p_q(x + 2 \cdot q_s(x)^2) + 1; \\ q_t(x) &= q_s(x) + q_r(x) + 2 \cdot q_s(x)^2 + q_q(x + 2 \cdot q_s(x)^2). \end{aligned}$$

  Notice that $|z|$ is bounded by $q_s(x)$. Notice that by applying theorem 3.2 on μ (z has no free variables) we have that every $v \in \mu$ is such that$v \leq 2 \cdot |z|^2$. We have:

$$\begin{aligned} |\pi| &\leq |\rho| + |\mu| + |\nu| + \sum_i (|\varrho_i|) + 1 \\ &\leq p_s(\sum_i |n_i|) + 2 \cdot |z| + p_r(\sum_i |n_i|) + |n| \cdot p_{qy}(\sum_i |n_i| + |n|) + 1 \\ &\leq p_s(\sum_i |n_i|) + 2 \cdot q_s(\sum_i |n_i|) + p_r(\sum_i |n_i|) + \\ &\quad + 2 \cdot q_s(\sum_i |n_i|)^2 \cdot p_{qy}(\sum_i |n_i| + 2 \cdot q_s(\sum_i |n_i|)^2) + 1. \end{aligned}$$

  Similarly, for every $w \in \pi$:

$$\begin{aligned} |w| &\leq q_s(\sum_i |n_i|) + 2 \cdot q_s(\sum_i |n_i|)^2 + q_r(\sum_i |n_i|) + q_{qy}(\sum_i |n_i| + |n|) \\ &\leq q_s(\sum_i |n_i|) + 2 \cdot q_z(\sum_i |n_i|)^2 + q_r(\sum_i |n_i|) + q_{qy}(\sum_i |n_i| + 2 \cdot q_s(\sum_i |n_i|)^2). \end{aligned}$$

- In this and the following cases the last rule is (T-Arr-E). Let's first consider $t = x\overline{s}$. In this case, obviously, the free variable x has type $\blacksquare A_i$ $(1 \leq i \leq j)$. By definition x is negatively $\Box$-free. This means that every term in $\overline{s}$ has a type that is positively $\Box$-free. By knowing that the type of x is negatively $\Box$-free, we conclude that the type of our

term t is $\Box$-free (because is both negatively and positively $\Box$-free at the same time). Definition of $\Downarrow_{\mathsf{rf}}$ ensures us that the derivation will have the following shape:

$$\frac{\rho_i : s_j[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha_j} r_j}{x\overline{s}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\prod_i \alpha_i} x\overline{r}}$$

We define p_t and q_t as:

$$p_t(x) = \sum_j p_{s_j}(x) + 1;$$
$$q_t(x) = \sum_j q_{s_j}(x) + 1.$$

Indeed we have

$$|\pi| \leq \sum_j |\rho_j| + 1 \leq \sum_j \{p_{s_j}(\sum_i |n_i|)\} + 1.$$

Similarly, if $z \in \pi$, it is easy to prove that $|z| \leq q_z(\sum_i |n_i|)$.

- If $t = \mathtt{S}_0 s$, then s have type $\mathbf{N}$ in the context Γ. The derivation π has the following form

$$\frac{\rho : s[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} z}{\mathtt{S}_0 s[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} \mathtt{S}_0 z}$$

Define $p_t(x) = p_s(x) + 1$ and $q_t(x) = q_s(x) + 1$. One can easily check that, by induction hypothesis

$$|\pi| \leq |\rho| + 1 \leq p_s(\sum_i |n_i|) + 1 = p_t(\sum_i |n_i|).$$

Analogously, if $r \in \pi$ then

$$|s| \leq q_s(\sum_i |n_i|) + 1 \leq q_t(\sum_i |n_i|).$$

- If $t = \mathtt{S}_1 s$ or $t = \mathtt{P}s$, then we can proceed exactly as in the previous case.
- Cases where we have on the left side a case or a recursion with some arguments are trivial, since they can be brought back to cases that we have considered.
- If t is $(\lambda x : \Box\mathbf{N}.s)r\overline{q}$, then we have the following derivation:

$$\frac{\begin{array}{ll}\rho : r[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} e & \\ \mu : e[\overline{x}/\overline{n}] \Downarrow_{\mathsf{nf}}^{\gamma} n & \nu : (s[x/n])\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} v\end{array}}{(\lambda x : \Box\mathbf{N}.s)r\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha\gamma\beta} v}$$

By hypothesis t is positively $\Box$-free and so also r (whose type is $\mathbf{N}$) and $s\overline{q}$ are positively $\Box$-free. So, we are sure that we are able to use induction hypothesis. Let p_t and q_t be:

$$p_t(x) = p_r(x) + 2 \cdot q_r(x) + p_{s\overline{q}}(x + 2 \cdot q_r(x)^2) + 1;$$
$$q_t(x) = q_{s\overline{q}}(x + 2 \cdot q_r(x)^2) + q_r(x) + 2 \cdot q_r(x)^2 + 1.$$

We have:

$$\begin{aligned}|\pi| &= |\rho| + |\mu| + |\nu| + 1\\ &\leq p_r(\sum_i |n_i|) + 2 \cdot |e| + p_{s\overline{q}}(\sum_i |n_i| + |n|) + 1\\ &\leq p_r(\sum_i |n_i|) + 2 \cdot q_r(\sum_i |n_i|) + p_{s\overline{q}}(\sum_i |n_i| + 2 \cdot q_r(\sum_i |n_i|)^2) + 1.\end{aligned}$$

By construction, remember that s has no free variables of type $\blacksquare\mathbf{N}$. By Theorem 3.2 (z has no free variables) we have $v \in \mu$ is such that $|v| \leq 2 \cdot |e|^2$. By applying induction hypothesis we have that every $v \in \rho$ is such that $|v| \leq q_r(\sum_i |n_i|)$, every $v \in \nu$ is such that

$$\begin{aligned}|v| &\leq q_{s\overline{q}}(\sum_i |n_i| + |n|) \leq q_{s\overline{q}}(\sum_i |n_i| + 2 \cdot |e|^2)\\ &\leq q_{s\overline{q}}(\sum_i |n_i| + 2 \cdot q_r(\sum_i |n_i|)^2).\end{aligned}$$

We can prove the second point of our thesis by setting $q_t(\sum_i |n_i|)$ as $q_{s\overline{q}}(\sum_i |n_i| + 2 \cdot q_r(\sum_i |n_i|)^2) + q_r(\sum_i |n_i|) + 2 \cdot q_r(\sum_i |n_i|)^2 + 1$.

- If t is $(\lambda x : \blacksquare\mathbf{N}.s)r\overline{q}$, then we have the following derivation:

$$\frac{\begin{array}{c}\rho : r[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} e\\ \mu : e[\overline{x}/\overline{n}] \Downarrow_{\mathsf{nf}}^{\gamma} n \qquad \nu : s\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} u\end{array}}{(\lambda x : \blacksquare\mathbf{N}.s)r\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha\gamma\beta} (\lambda x : \blacksquare\mathbf{N}.u)n}$$

By hypothesis we have t that is positively $\Box$-free. So, also r and e (whose type is $\mathbf{N}$) and $s\overline{q}$ are positively $\Box$-free. We define p_t and q_t as:

$$\begin{aligned}p_t(x) &= p_r(x) + 2 \cdot q_r(x) + p_{s\overline{q}}(x) + 1;\\ q_t(x) &= q_r(x) + 2 \cdot q_r(x)^2 + q_{s\overline{q}}(x) + 1.\end{aligned}$$

We have:

$$\begin{aligned}|\pi| &= |\rho| + |\mu| + |\nu| + 1\\ &\leq p_r(\sum_i |n_i|) + 2 \cdot q_r(\sum_i |n_i|) + p_{s\overline{q}}(\sum_i |n_i|) + 1.\end{aligned}$$

Similarly, if $z \in \pi$, it is easy to prove that $|z| \leq q_t(\sum_i |n_i|)$.

- If t is $(\lambda x : aH.s)r\overline{q}$, then we have the following derivation:

$$\frac{\rho : (s[x/r])\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} v}{(\lambda x : aH.s)r\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} v}$$

By hypothesis we have t that is positively $\Box$-free. So, also $s\overline{q}$ is positively $\Box$-free. r has an higher-order type H and so we are sure that $|(s[x/r])\overline{q}| < |(\lambda x : aH.s)r\overline{q}|$. Define p_t and q_t as:

$$p_t(x) = p_{(s[x/r])\overline{q}}(x) + 1;$$
$$q_t(x) = q_{(s[x/r])\overline{q}}(x) + 1.$$

By applying induction hypothesis we have:

$$|\pi| = |\rho| + 1 \leq p_{(s[x/r])\overline{q}}(\sum_i |n_i|) + 1.$$

By induction, we are able to prove the second point of our thesis.
This concludes the proof. $\Box$

Following the definition of $\Downarrow$, it is quite easy to obtain, given a first order term t, of arity k, a probabilistic Turing machine that, when receiving on input (an encoding of) $n_1 \ldots n_k$, produces in output m with probability equal to $\mathscr{D}(m)$, where $\mathscr{D}$ is the (unique!) distribution such that $t \rightsquigarrow \mathscr{D}$. Indeed, $\Downarrow_{\mathsf{rf}}$ and $\Downarrow_{\mathsf{nf}}$ are designed as algorithms. Moreover, the obtained Turing machine works in polynomial time, due to propositions 3.2 and 3.3. Formally:

Theorem 3.4 (Soundness)**.** *Suppose t is a first order term of arity k. Then there is a probabilistic Turing machine M_t running in polynomial time such that M_t on input $n_1 \ldots n_k$ returns m with probability exactly $\mathscr{D}(m)$, where $\mathscr{D}$ is a probability distribution such that $tn_1 \ldots n_k \rightsquigarrow \mathscr{D}$.*

Proof. By propositions 3.2 and 3.3. $\Box$

4. Probabilistic Polytime Completeness

In the previous section, we proved that the behaviour of any RSLR first-order term can be somehow simulated by a probabilistic polytime Turing machine. What about the converse? In this section, we prove that any probabilistic polynomial time Turing machine (PPTM in the following) can be encoded in RSLR. PPTMs are here seen as one-tape Turing machines which are capable at any step during the computation of "tossing a fair coin", and proceeding in two different ways depending on the outcome of the tossing.

To facilitate the encoding, we extend our system with pairs. All the proofs in previous sections remain valid. Types are extended by the following production:

$$A \ ::= \ A \otimes A;$$

Terms now contain two binary constructs $\langle t, s\rangle$ and $\mathtt{let}_A\ t\ \mathtt{be}\ \langle x, y\rangle\ \mathtt{in}\ r$

$$\frac{\Gamma; \Delta_1 \vdash t : A \qquad \Gamma; \Delta_2 \vdash s : B}{\Gamma; \Delta_1, \Delta_2 \vdash \langle t, s\rangle : A \otimes B}$$

$$\frac{\Gamma;\Delta_1 \vdash t : A \otimes B \qquad \Gamma; x : aA, y : aB, \Delta_2 \vdash s : C \qquad \Gamma, \Delta_1 <: a}{\Gamma;\Delta_1,\Delta_2 \vdash \mathtt{let}_C\ t\ \mathtt{be}\ \langle x, y\rangle\ \mathtt{in}\ s : C}$$

We will never make use of the `let` construct, except through projections, which can be easily defined and which have the expected types:

$$\frac{\Gamma \vdash t : A \otimes B}{\Gamma \vdash \pi_1(t) : A} \qquad \frac{\Gamma \vdash t : A \otimes B}{\Gamma \vdash \pi_2(t) : B}$$

As syntactic sugar, we will use $\langle t_1 \ldots, t_i\rangle$ (where $i \geq 1$) for the term

$$\langle t_1, \langle t_2, \ldots \langle t_{i-1}, t_i\rangle \ldots\rangle\rangle.$$

For every $n \geq 1$ and every $1 \leq i \leq n$, we can easily build a term π_i^n which extracts the i-th component from tuples of n elements: this can be done by composing $\pi_1(\cdot)$ and $\pi_2(\cdot)$. With a slight abuse on notation, we sometime write π_i for π_i^n.

4.1. Unary Natural Numbers and Polynomials

Natural numbers are represented in binary in RSLR. In other words, the basic operations allowed on them are $\mathtt{S}_0$, $\mathtt{S}_1$ and $\mathtt{P}$, which correspond to appending a binary digit to the right of the number (seen as a binary string) or stripping the rightmost such digit. This is even clearer if we consider the length $|n|$ of a numeral n, which is only logarithmic in n.

Sometimes, however, it is more convenient to work in unary notation. Given a natural number i, its *unary encoding* is simply the numeral that, written in binary notation, is 1^i. Given a natural number i we will refer to its unary encoding $\underline{i}$. The type in which unary natural numbers will be written, is just $\mathbf{N}$, but for reasons of clarity we will use the symbol $\mathbf{U}$ instead.

From any numeral n, we can extract the unary encoding of its length:

$$\mathsf{encode} = \lambda t : \square\mathbf{N}.\mathtt{recursion}_{\mathbf{U}}\ t\, 0\, (\lambda x : \square\mathbf{U}.\lambda y : \blacksquare\mathbf{U}.\mathtt{S}_1 y) : \square\mathbf{N} \to \mathbf{U}$$

Predecessor and successor functions are simply $\mathtt{P}$ and $\mathtt{S}_1$. We need to show how to express polynomials, and in order to do so we define the operators $\mathsf{add} : \square\mathbf{U} \to \blacksquare\mathbf{U} \to \mathbf{U}$ and $\mathsf{mult} : \square\mathbf{U} \to \square\mathbf{U} \to \mathbf{U}$. We define add as

$$\begin{aligned}\mathsf{add} =& \lambda x : \square\mathbf{U}.\lambda y : \blacksquare\mathbf{U}.\\ & \mathtt{recursion}_{\mathbf{U}}\ x\, y\, (\lambda x : \square\mathbf{U}.\lambda y : \blacksquare\mathbf{U}.\mathtt{S}_1 y) : \square\mathbf{U} \to \blacksquare\mathbf{U} \to \mathbf{U}.\end{aligned}$$

Similarly, we define mult as

$$\begin{aligned}\mathsf{mult} =& \lambda x : \square\mathbf{U}.\lambda y : \square\mathbf{U}.\\ & \mathtt{recursion}_{\mathbf{U}}\ (\mathtt{P}x)\, y\, (\lambda x : \square\mathbf{U}.\lambda z : \blacksquare\mathbf{U}.\mathsf{add} y z) : \square\mathbf{U} \to \square\mathbf{U} \to \mathbf{U}.\end{aligned}$$

The following is quite easy:

Lemma 4.1. *Every polynomial of one variable with natural coefficients can be encoded as a term of type* $\square\mathbf{U} \to \mathbf{U}$.

Proof. Simply, turn add into a term of type $\square\mathbf{U} \to \square\mathbf{U} \to \mathbf{U}$ by way of subtyping and then compose add and mult has much as needed to encode the polynomial at hand. □

4.2. Finite Sets

Any finite, linearly ordered set $F = (|F|, \sqsubseteq_F)$ can be naturally encoded as an "initial segment" of $\mathbf{N}$: if $|F| = \{a_0, \ldots, a_i\}$ where $a_i \sqsubseteq_F a_j$ whenever $i \leq j$, then a_i is encoded simply by the natural number whose binary representation is 10^i. For reasons of clarity, we will denote $\mathbf{N}$ as $\mathbf{F}_F$. We can do some case analysis on an element of $\mathbf{F}_F$ by the combinator

$$\mathsf{switch}_A^F : \blacksquare\mathbf{F}_F \to \underbrace{\blacksquare A \to \ldots \to \blacksquare A}_{i \text{ times}} \to \blacksquare A \to A,$$

where A is a $\square$-free type and i is the cardinality of $|F|$. The term above can be defined by induction on i:

- If $i = 0$, then it is simply $\lambda x : \blacksquare\mathbf{F}_F.\lambda y : \blacksquare A.y$.
- If $i \geq 1$, then it is the following:

$$\begin{array}{l}\lambda x : \blacksquare\mathbf{F}_F.\lambda y_0 : \blacksquare A. \ldots \lambda y_i : \blacksquare A. \\ \quad \mathtt{case}_A\ x\ \mathtt{zero}(\lambda h : \blacksquare A.h) \\ \qquad \mathtt{even}\ (\lambda h : \blacksquare A.\mathsf{switch}_A^E(\mathtt{P}x)y_1 \ldots y_i h) \\ \qquad \mathtt{odd}\ (\lambda h : \blacksquare A.y_0)\end{array}$$

where E is the subset of F of those elements with strictly positive indices.

4.3. Strings

Suppose $\Sigma = \{a_0, \ldots, a_i\}$ is a finite alphabet. Elements of Σ can be encoded following the just described scheme, but how about strings in Σ^*? We can proceed somehow similarly: the string $a_{j_1} \ldots a_{j_k}$ can be encoded as the natural number

$$10^{j_1}10^{j_2} \ldots 10^{j_k}.$$

Whenever we want to emphasize that a natural number is used as a string, we write $\mathbf{S}_\Sigma$ instead of $\mathbf{N}$. It is easy to build a term $\mathsf{append}_\Sigma : \blacksquare(\mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma) \to \mathbf{S}_\Sigma$ which appends the second argument to the first argument. Similarly, one can define a term $\mathsf{tail}_\Sigma : \blacksquare\mathbf{S}_\Sigma \to \mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma$ which strips off the rightmost character a from the argument string and returns a together with the rest of the string; if the string is empty, a_0 is returned, by convention.

We also define a function $\mathsf{NtoS}_\Sigma : \square\mathbf{N} \to \mathbf{S}_\Sigma$ that takes a natural number and produce in output an encoding of the corresponding string in Σ^* (where i_0 and i_1 are the indices of 0 and 1 in Σ):

$$\begin{array}{l}\mathsf{NtoS}_\Sigma = \lambda x : \square\mathbf{N}.\mathtt{recursion}_{\mathbf{S}_\Sigma}\ x\ 1 \\ \quad \lambda x : \blacksquare\mathbf{N}.\lambda y : \blacksquare\mathbf{S}. \\ \qquad \mathtt{case}_{\mathbf{N}}\ x\ \mathtt{zero}\ \mathsf{append}_\Sigma\langle y, i_0\rangle \\ \qquad\quad \mathtt{even}\ \mathsf{append}_\Sigma\langle y, i_1\rangle \\ \qquad\quad \mathtt{odd}\ \mathsf{append}_\Sigma\langle y, i_1\rangle : \square\mathbf{N} \to \mathbf{S}.\end{array}$$

Similarly, one can write a term $\mathsf{StoN}_\Sigma : \square\mathbf{S}_\Sigma \to \mathbf{N}$.

4.4. Probabilistic Turing Machines

Let M be a probabilistic Turing machine $M = (Q, q_0, F, \Sigma, \sqcup, \delta)$, where Q is the finite set of states of the machine; q_0 is the initial state; F is the set of final states of M; Σ is the finite alphabet of the tape; $\sqcup \in \Sigma$ is the symbol for empty string; $\delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\})$ is the transition function of M. For each pair $(q, s) \in Q \times \Sigma$, there are exactly two triples (r_1, t_1, d_1) and (r_2, t_2, d_2) such that $((q, s), (r_1, t_1, d_1)) \in \delta$ and $((q, s), (r_2, t_2, d_2)) \in \delta$. Configurations of M can be encoded as follows:

$$\langle t_{left}, t, t_{right}, s\rangle : \mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma \otimes \mathbf{S}_\Sigma \otimes \mathbf{F}_Q,$$

where t_{left} represents the left part of the main tape, t is the symbol read from the head of M, t_{right} the right part of the main tape; s is the state of our Turing Machine. Let $\mathbf{C}_M$ be a shortcut for $\mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma \otimes \mathbf{S}_\Sigma \otimes \mathbf{F}_Q$. Let t_C be the term encoding the configuration C.

Suppose that M on input x runs in time bounded by a polynomial $p : \mathbb{N} \to \mathbb{N}$. Then we can proceed as follows:

- encode the polynomial p by using function $\mathsf{encode}, \mathsf{add}, \mathsf{mult}, \mathsf{dec}$ so that at the end we will have a function $\underline{p} : \Box\mathbf{N} \to \mathbf{U}$;
- write a term $\underline{\delta} : \blacksquare\mathbf{C}_M \to \mathbf{C}_M$ which mimicks δ;
- write a term $\mathsf{init}_M : \blacksquare\mathbf{S}_\Sigma \to \mathbf{C}_M$ which returns the initial configuration for M corresponding to the input string.

The term t_M of type $\Box\mathbf{N} \to \mathbf{N}$ which has exactly the same behavior as M is the following:

$$\lambda x : \Box\mathbf{N}.\mathsf{StoN}_\Sigma(\mathtt{recursion}_{\mathbf{C}_M}\ (\underline{p}\ x)\,(\mathsf{init}_M\ (\mathsf{NtoS}_\Sigma(x)))\,(\lambda y : \blacksquare\mathbf{N}.\lambda z : \blacksquare\mathbf{C}_M.\underline{\delta}\ z)).$$

We then get a faithful encoding of PPTM into RSLR, which will be useful in the forthcoming section:

Theorem 4.2. *Suppose M is a probabilistic Turing machine running in polynomial time such that for every n, $\mathscr{D}_n$ is the distribution of possible results obtained by running M on input n. Then there is a first order term t_M such that for every n, tn evaluates to $\mathscr{D}_n$.*

Proof. Proving that t_M does what it is supposed to do can be done as follows:

- First of all, one can show that $\underline{p}\ n$ evaluates to $\mathscr{I}_m$, where m is the unary encoding of the length of $p(n)$;
- Similarly, one can show that init_M correctly computes an initial configuration for M when fed with an input string;
- Finally, $\underline{\delta}$ can be showed to evaluate to $\{t_D^{\frac{1}{2}}, t_E^{\frac{1}{2}}\}$ whenever fed with t_C and whenever C is a configuration which evolves in one step to D and E.

We elide the proof here, since all the steps above are quite simple anyway. □

5. Relations with Complexity Classes

The last two sections established a precise correspondence between RSLR and probabilistic polynomial time Turing machines. But how about probabilistic complexity *classes*,

like **BPP** or **PP**? They are defined on top of probabilistic Turing machines, imposing constraints on the probability of error: in the case of **PP**, the error probability can be anywhere near $\frac{1}{2}$, but not equal to it, while in **BPP** it can be non-negligibly smaller than $\frac{1}{2}$. There are two ways RSLR can be put in correspondence with probabilistic complexity classes, and these are explained in the following two sections.

5.1. Leaving the Error Probability Explicit

Of course, one possibility consists in leaving bounds on the error probability explicit *in the very definition* of what an RSLR term represents:

Definition 5.1 (Recognising a Language with Error ε)**.** A first-order term t of arity 1 recognizes a language $L \subseteq \mathbb{N}$ with probability less than ε if, and only if, both:
- $x \in L$ and $tx \rightsquigarrow \mathscr{D}$ implies $\mathscr{D}(0) > 1 - \varepsilon$;
- $x \notin L$ and $tx \rightsquigarrow \mathscr{D}$ implies $\sum_{s>0} \mathscr{D}(s) > 1 - \varepsilon$.

So, 0 encodes an accepting state of tx and $s > 0$ encodes a reject state of tx. Theorem 3.4, together with Theorem 4.2 allows us to conclude that:

Theorem 5.1 ($\frac{1}{2}$-Completeness for **PP**)**.** *The set of languages which can be recognized with error ε in* RSLR *for some $0 < \varepsilon \leq 1/2$ equals* **PP**.

But, interestingly, we can go beyond and capture a more interesting complexity class:

Theorem 5.2 ($\frac{1}{2}$-Completeness for **BPP**)**.** *The set of languages which can be recognized with error ε in* RSLR *for some $0 < \varepsilon < 1/2$ equals* **BPP**.

Observe how ε can be even equal to $\frac{1}{2}$ in Theorem 5.1, while it cannot in Theorem 5.2. This is the main difference between **PP** and **BPP**: in the first class, the error probability can very fast approach $\frac{1}{2}$ when the size of the input grows, while in the second it cannot.

The notion of recognizing a language with an error ε allows to capture complexity classes in RSLR, but it has an obvious drawback: the error probability remains *explicit* and *external* to the system; in other words, RSLR does not characterize *one* complexity class but many, depending on the allowed values for ε. Moreover, given an RSLR term t and an error ε, determining whether t recognizes *any* function with error ε is undecidable. As a consequence, theorems 5.1 and 5.2 do not suggest an enumeration of all languages in either **PP** or **BPP**. This in contrast to what happens with other ICC systems, e.g. SLR, in which all terms (of certain types) compute a function in **FP** (and, *viceversa*, all functions in **FP** are computed this way). As we have already mentioned in the Introduction, this discrepancy between **FP** and **BPP** has a name: the former is a *syntactic* class, while the latter is a *semantic* class (see [1]).

5.2. Getting Rid of the Error Probability

One may wonder whether a more implicit notion of representation can be somehow introduced, and which complexity class corresponds to RSLR this way. One possibility is taking representability by majority:

Definition 5.2 (Representability-by-Majority)**.** Let t be a first-order term of arity 1. Then t is said to *represent-by-majority* a language $L \subseteq \mathbb{N}$ iff:

1. If $n \in L$ and $tn \rightsquigarrow \mathscr{D}$, then $\mathscr{D}(0) \geq \sum_{m>0} \mathscr{D}(m)$;
2. If $n \notin L$ and $tn \rightsquigarrow \mathscr{D}$, then $\sum_{m>0} \mathscr{D}(m) > \mathscr{D}(0)$.

There is a striking difference between Definition 5.2 and Definition 5.1: the latter is asymmetric, while the first is symmetric.

Please observe that any RSLR first order term t represents-by-majority *a* language, namely the language defined from t by Definition 5.2. It is well known that **PP** can be defined by majority itself [1], stipulating that the error probability should be *at most* $\frac{1}{2}$ when handling strings in the language and *strictly smaller than* $\frac{1}{2}$ when handling strings not in the language. As a consequence:

Theorem 5.3 (Completeness-by-Majority for **PP**)**.** *The set of languages which can be represented-by-majority in* RSLR *equals* **PP**.

In other words, RSLR can indeed be considered as a tool to enumerate all functions in a complexity class, namely **PP**. It comes with no surprise, since the latter is a syntactic class.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity — A Modern Approach*. Cambridge University Press, 2009.

[2] S.J. Bellantoni, K.H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104(1-3):17–30, 2000.

[3] Stephen Bellantoni. Predicative recursion and the polytime hierarchy. In P. Clote and J.B. Remmel, editors, *Feasible Mathematics II*, pages 15–29. Birkhauser, 1995.

[4] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[5] Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Recursion schemata for NC^k. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, Proceedings*, volume 5213 of *LNCS*, pages 49–63, 2008.

[6] Ugo Dal Lago, Simone Martini, and Davide Sangiorgi. Light logics and higher-order processes. In Sibylle B. Fröschle and Frank D. Valencia, editors, *17th International Workshop on Expressiveness in Concurrency, Proceedings*, volume 41 of *EPTCS*, 2010.

[7] Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. Quantum implicit computational complexity. *Theoretical Computer Science*, 411(2):377–409, 2010.

[8] Lance Fortnow and Rahul Santhanam. Hierarchy theorems for probabilistic polynomial time. In *45th Annual IEEE Symposium on Foundations of Computer Science, Proceedings*, pages 316–324. IEEE Computer Society, 2004.

[9] John Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6(4):675–695, 1977.

[10] Martin Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic, 11th International Workshop, Proceedings*, volume 1414 of *LNCS*, pages 275–294, 1997.

[11] Neil D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theoretical Computer Science*, 228:151–174, 1999.

[12] Daniel Leivant. Stratified functional programs and computational complexity. In *Principles of Programming Languages, 20th International Symposium, Proceedings*, pages 325–333. ACM, 1993.

[13] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: Substitution and poly-space. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 9th International Workshop, Proceedings*, volume 933 of *LNCS*, pages 486–500. 1995.

[14] John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Foundations of Computer Science, 39th Annual Symposium, Proceedings*, pages 725–733. IEEE Computer Society, 1998.

[15] Helmut Schwichtenberg and Steven Bellantoni. Feasible computation with higher types. In *Proof and System-Reliability*, pages 399–415. Kluwer Academic Publisher, 2001.

[16] Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. *Mathematical Structures in Computer Science*, 20(5):951–975, 2010.