

Safe Recursion on Notation into a light logic by levels

Luca Roversi

Dipartimento di Informatica - Università di Torino
roversi@di.unito.it

Luca Vercelli

Dipartimento di Matematica - Università di Torino
luca.vercelli@unito.it

Partially supported by MIUR PRIN CONCERTO — protocol number 2007BHXCFH

We embed Safe Recursion on Notation (**SRN**) into Light Affine Logic by Levels (**LALL**), derived from the logic \mathbf{ML}^4 . **LALL** is an intuitionistic deductive system, with a polynomial time cut elimination strategy. The embedding allows to represent every term t of **SRN** as a family of nets $\langle [t]^l \rangle_{l \in \mathbb{N}}$ in **LALL**. Every net $[t]^l$ in the family simulates t on arguments whose bit length is bounded by the integer l . The embedding is based on two crucial features. One is the recursive type in **LALL** that encodes Scott binary numerals, *i.e.* Scott words, as nets. Scott words represent the arguments of t in place of the more standard Church binary numerals. Also, the embedding exploits the “fuzzy” borders of paragraph boxes that **LALL** inherits from \mathbf{ML}^4 to “freely” duplicate the arguments, especially the safe ones, of t . Finally, the type of $[t]^l$ depends on the number of composition and recursion schemes used to define t , namely the structural complexity of t . Moreover, the size of $[t]^l$ is a polynomial in l , whose degree depends on the structural complexity of t . So, this work makes closer both the predicative recursive theoretic principles **SRN** relies on, and the proof theoretic one, called *stratification*, at the base of Light Linear Logic.

1 Introduction

Slightly rephrasing the *incipit* of [6], comparing implicit characterizations of computational complexity classes may provide insights into their nature, while offering concepts and methods for generalizing computational complexity to computing over arbitrary structures and to higher type functions. Here, we relate two implicit characterizations of polynomial time functions (**PTIME**). One is Safe Recursion on Notation (**SRN**) [4], that we take as representative of the characterizations of **PTIME** that restrict the primitive recursion. The other one is Light Affine Linear Logic by Levels (**LALL**), a proof theoretical system we derive from Light Linear Logic by Levels (\mathbf{ML}^4) [3] and from Intuitionistic Light Affine Logic (**ILAL**) [2]. We recall, \mathbf{ML}^4 and **ILAL** are two *Light Logics*, *i.e.* restrictions of Linear Logic that characterize some complexity class, in this case **PTIME**, under the proofs-as-programs analogy. These two logics control the complexity of the algorithms they can express by the technical a notion *Stratification*, which expresses specific structural restrictions on the derivations of \mathbf{ML}^4 and **ILAL**. **SRN**, of which we recall some more aspects in Section 4, provides a *predicative analysis* of primitive recursion. It is the least set that contains the *zero* 0 (considered as a 0-ary function), the *successors* $s_0(;x) = 2x$, $s_1(;x) = 2x + 1$, the *predecessor* $p(;2x + i) = x$, the *projection* $\pi_k^{n;s}(\vec{x}; \vec{y}) = x_k$ if $1 \leq k \leq n$, and y_k if $1 \leq k \leq s$, the *conditional* $B(;y, y_1, y_2) = y_1$ if y is odd, and y_2 otherwise, and which is closed under *safe composition* and *predicative recursion on notation* ((2) and (3) in Figure 1). The work [8] is the first one relating the two different traditions: it defines a map from terms of a strict fragment \mathbf{BC}^- of **SRN** into nets of **ILAL**. The main obstacle to a full representation of **SRN** into **ILAL** is that the duplication of nets in **ILAL**, hence of the safe arguments, is far from being free, as required instead by (3). In fact, [8] also shows that an extension \mathbf{BC}^\pm , polynomial time complete, can be represented inside **ILAL**. However, since the primitives added to \mathbf{BC}^\pm are not in **SRN**, we cannot see \mathbf{BC}^\pm as

relevant to the goal of understanding the possible relation between full **SRN** and the above stratification principle, basic to **ML**⁴ and **ILAL**. Since [8], no extension of the relation between **SRN** and **ILAL** has been produced, to our knowledge. Here, we show to which extent we can avoid that obstacle inside **LALL**. **LALL**, that will be formally defined in Section 2, is an intuitionistic system of nets endowed with: (i) edges labelled by indices, or levels, (ii) unconstrained weakening, to make programming with its nets somewhat more comfortable, (iii) a language of formulæ quotiented by the recursive equivalence $\mathcal{S} = \forall \alpha. (\alpha \multimap (\mathbb{B} \multimap \mathcal{S} \multimap \alpha) \multimap \alpha)$, where \mathbb{B} is the type of booleans, and \mathcal{S} the data type of *Scott words* [1], and (iv) a polynomial time sound cut elimination procedure (Section 3) which does not depend on the types that label the edges of a given net.

SRN embeds into **LALL** by means of the map $\lceil \cdot \rceil$ (Sections 5 and 6.) The map $\lceil \cdot \rceil$ has the same *natural* and *inductive* structure as the one of the map in [8] from **SRN** to **ILAL**. However, $\lceil \cdot \rceil$ takes two arguments: (i) any term t of **SRN**^{n;s}, with normal arity n , the number of arguments to the left of the semicolon, and safe arity s , the number of those ones to the right, and (ii) an integer $l \geq 0$ that bounds the size of every argument of t . Then, $\lceil t \rceil^l$ yields a net that simulates $t(\vec{x}; \vec{y})$ whenever every element in \vec{x}, \vec{y} is at most as long as l (Proposition 12). This suggests that we get to a situation we can summarize by the analogy:

$$\frac{\mathbf{LALL}}{\mathbf{SRN}} = \frac{\mathbf{PTIME}\text{-uniform Boolean Circuits}}{\mathbf{PTIME}\text{ Problems}} \quad (1)$$

What we do reads as follows. Let t be a term of **SRN**^{n;s}. We write $\partial_C(t)$ and $\partial_R(t)$ for the number of composition and recursion schemes, respectively, that are used to build t . That way, $\text{cplx}(t) = \partial_C(t) + \partial_R(t)$ is a naïve measure of the *static* complexity of t . Also, let p_t be the characteristic polynomial of t , whose values bound the length of the output of t . Let $\partial(p_t)$ be its degree. Then, t is represented in **LALL** by a family $\langle \lceil t \rceil^l \rangle_{l \in \mathbb{N}}$ of nets such that:

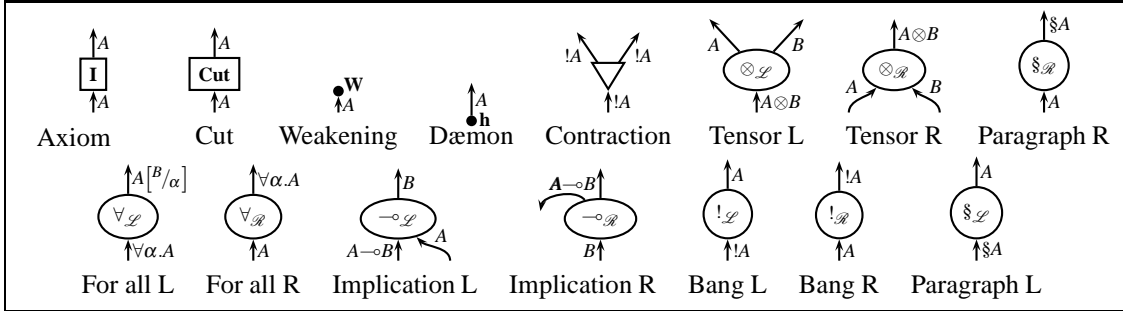
1. The size of every net $\lceil t \rceil^l$ is $O(l^{\partial(p_t) \text{cplx}(t)})$, namely polynomial in l ;
2. If l is at least as great as every bit length $|x_1|, \dots, |x_n|, |y_1|, \dots, |y_s|$ of the arguments, then the application of $\lceil t \rceil^l$ to $\lceil x_1 \rceil^l, \dots, \lceil x_n \rceil^l, \lceil y_1 \rceil^l, \dots, \lceil y_s \rceil^l$ equals $\lceil t(x_1, \dots, x_n; y_1, \dots, y_s) \rceil^l$;
3. Every $\lceil t \rceil^l$ is a map from $(\otimes^n \mathcal{S}) \otimes (\otimes^s (\mathcal{S}^k))$ to \mathcal{S}^k , where k depends on $\text{cplx}(t)$.

The first two points suggest the analogy (1). Specifically, the first point expresses a uniformity condition on the nets in the family, since it states that their dimension are bounded only by the length of the inputs. The second point says that $\lceil t \rceil^l$ soundly simulates t on *every* input of length *smaller or equal* to l . Finally, the third point is a natural property we can expect as soon as we try to compositionally and naturally represent first order algebraic terms, that operate on a given domain, into a higher order language. It is a static description of the behavior of t in terms of types of **LALL**, a kind of information we cannot have by, for example, representing **SRN** as circuit families.

We see the use of \mathcal{S} as a first fundamental choice to write $\lceil \cdot \rceil$. The reason is twofold. One reason is a kind of obvious, since \mathcal{S} supports the representation of successors, predecessor, projection and conditional as constant time operations, unlike $\mathcal{C} = \forall \alpha. !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \mathcal{S}(\alpha \multimap \alpha)$, which is generally used to represent *Church words* in Light Logics. The other reason, instead, brings a certain degree of novelty with it because we exploit a crucial property of **LALL**, and of Light Logics, *which had hardly been used so far*. The crucial property is that the polynomial time cut elimination of **LALL** depends only on the structure of any given net Π , while the logical complexity of the formulæ in Π does not affect it. So, we are free to add fixpoints formulæ, like \mathcal{S} is, which adds a huge expressivity to the logic.

$$\circ[t' u_1 \dots u_n v_1 \dots v_{s'}](\vec{x}; \vec{y}) = t'(u_1(\vec{x}); \dots, u_n(\vec{x}); v_1(\vec{x}; \vec{y}), \dots, v_{s'}(\vec{x}; \vec{y})) \quad (2)$$

$$\begin{cases} r[u_\varepsilon, u_0, u_1](0, \vec{x}; \vec{y}) = u_\varepsilon(\vec{x}; \vec{y}) \\ r[u_\varepsilon, u_0, u_1](2z + i, \vec{x}; \vec{y}) = u_i(z, \vec{x}; \vec{y}, r[u_\varepsilon, u_0, u_1](z, \vec{x}; \vec{y})) \end{cases} \quad i \in \{0, 1\} \quad (3)$$

Figure 1: **SRN**: predicative recursion on notation and safe composition.Figure 2: The nodes in the proof nets of **LALL**.

A second step to get $[\cdot]^l$, for every l , we exploit what we like to call the *fuzzy* borders of paragraph boxes of **LALL** to write the net ∇S_l^k . The net ∇S_l^k duplicates a Scott word at most as long as l , starting from a premise of type $\S^k \mathcal{S}$ and concluding with the type $\S^k \mathcal{S} \otimes \S^k \mathcal{S}$, for any k . We remark that in **ILAL**, where the border of paragraph boxes is “rigid”, we could only write a net, analogous to ∇S_l^k , concluding with type $\S^k(\mathcal{S} \otimes \mathcal{S})$ which would generally impede to get the right type for $[t]^l$. By the way, this is why [8] shows how to embed **BC**⁻ but not **SRN** into **ILAL**. Indeed, in **BC**⁻, composition and safe recursion schemes allow *linear* safe arguments only, i.e. the safe arguments are never duplicated.

To conclude, we recall what *stratification* means. It is a structural property underpinning the **PTIME**-sound cut elimination of Girard’s Light Linear Logic (**LLL**) [5], and its variants **ILAL**, **ML**⁴, and **LALL**. A net Π is *stratified* if the number of boxes around every node keeps being constant in every net we reach from Π by cut elimination. This work should be a step further towards studying how the *stratification* is compatible with the predicative analysis of **PTIME**-sound computations that **SRN** embodies.

Acknowledgments. We want to thank the anonymous referees whose questions helped us to better address the points subject of this work.

2 Light Affine Logic by Levels (LALL)

The language of formulæ. First, for any fixed countable set \mathcal{V} of propositional variables, the set \mathcal{F} of formulæ is generated by the following grammar:

$$F ::= \mathcal{S} \mid \alpha \mid F \otimes F \mid F \multimap F \mid \forall \alpha. F \mid !F \mid \S F \quad \alpha \in \mathcal{V}$$

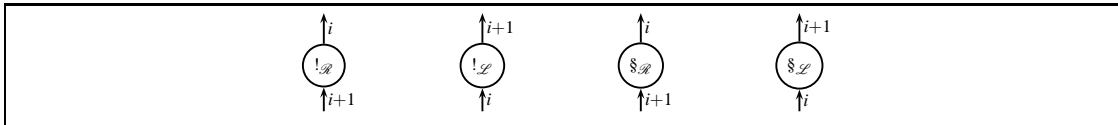


Figure 3: Constraints on the indexing. The nodes we omit have the same index on all of their incident edges.

where \mathcal{S} is a propositional constant. Second, we define the quotient $\mathcal{F}_{\mathcal{S}}$ of \mathcal{F} by assuming:

$$\mathcal{S} = \forall \alpha. (\alpha \multimap (\mathbb{B} \multimap \mathcal{S} \multimap \alpha) \multimap \alpha) \quad (4)$$

among the elements of \mathcal{F} . Namely, (4) says that \mathcal{S} represents Scott words [1]. The formulæ we shall effectively use are the equivalence classes in $\mathcal{F}_{\mathcal{S}}$. Every time we label an edge of a net of **LALL** by \mathcal{S} , we can also label that edge by any “unfolding” of \mathcal{S} that obeys (4). A $[B/\alpha]$ is the substitution of every free occurrence of α in A with B .

Proof structures and nets. **LALL** is a language of nets. Nets will be defined as particular proof structures. Given the nodes in Figure 2, we say that *an Axiom node and a Daemon nodes are proof structures*. Moreover, given two proof structures Π and Σ :



denoted as $\Pi \triangleright A_1, \dots, A_r \vdash C$ and $\Sigma \triangleright B_1, \dots, B_l \vdash D$, respectively, with $r, l \geq 0$, then all the graphs inductively built from Π and Σ by the rule schemes in Figure 4 are proof structures.

If $\Pi \triangleright \Gamma \vdash A$, we say that Π *proves* the sequent $\Gamma \vdash A$. The *inputs* (resp. *outputs*) of Π are the edges labelled Γ (resp. A). The set of the nodes of Π is V_{Π} , and E_{Π} is the set of edges. The *size* $|\Pi|$ of Π is the cardinality of V_{Π} . The *depth* $\partial(x)$ of a node or edge $x \in V_{\Pi} \cup E_{\Pi}$ is the number of nested $!$ -boxes containing x . The *depth* $\partial(\Pi)$ of Π is the greatest depth among the nodes of Π .

Every $!$ -box simultaneously introduces one Bang R node and at most one Bang L node, recording this by the box border as in Figure 4.

Definition 1 (Indexing and Nets, adapted from [3]) *Let Π be a proof structure.*

1. *An indexing for Π is a function I from the edges of Π to \mathbb{Z} that satisfies the constraints in Figure 3 and such that $I(e) = I(e')$, for every pair e, e' of inputs and output of Π .*
2. *A net is a proof structure that admits an indexing.*
3. *An indexing I of Π is canonical if Π has an edge e such that $I(e) = 0$, and $I(e') \geq 0$ for all edges e' of Π .*

As in [3], we can state that every net of **LALL** admits a unique canonical indexing.

The indexing tells that the nodes $!\mathcal{S}$ and \mathcal{S} are not *dereliction* nodes. Remember that the dereliction rule of Linear Logic is inherently not stratified, because the cut-elimination is presence of a dereliction node may also “open” boxes. Instead, these nodes can be considered as auxiliary ports of \mathcal{S} -boxes whose border is somewhat *fuzzy*. We mean that a \mathcal{S} -box need not be contained in or disjoint from another box. Instead, it can “overlap” a $!$ -box, and it can have more than one conclusion \mathcal{S} . To distinguish \mathcal{S} -boxes from the $!$ ones we adopt a dotted border.

Let I_0 be the canonical indexing of Π and $e \in E_{\Pi}$. The *level* of e is $l(e)$. It is defined as $I_0(e)$. The *level* of Π is $l(\Pi)$. It is defined as the greatest value assumed by I_0 on the edges of Π . We denote as B_{Π}

Let $B, C \in B_\Pi$. Let $B \prec_1^L C$ if the roots of B and C lie at the same level, and the root of B is in cut with an exponential path that enters an auxiliary port of C . \preceq^L is the reflexive and transitive closure of \prec_1^L . One can show that \preceq^L is a partial order, *upward arborescent*: for every C there is at most one B such that $B \prec_1^L C$.

Let $c, c' \in \text{cuts}(\Pi)$. We write $c \leq c'$ iff one of the following conditions holds. (i) c' is connected to a weakening or a dæmon, and c is not. (ii) The condition (i) is false but $l(c) < l(c')$ holds. (iii) The conditions (i) and (ii) are false, so $l(c) = l(c')$. In this case, $c \leq c'$ iff: (a) either c' is connected to a contraction, and c is not, or (b) c, c' are connected to a contraction on one side, to the boxes B, B' , respectively, on the other, and $B \preceq^L B'$.

Definition 3 (Canonical normalization) *A sequence of normalization steps that starts from a given proof net Π is canonical whenever smaller cuts relatively to \leq are eliminated before higher ones.*

Theorem 4 (Polynomial bound for LALL) *Let Π be an LALL proof net of size s , level l , and depth d . Then, every canonical reduction is at most $(l+1)s^{(d+2)^l}$ steps long.*

The proof strategy coincides with the one in [3], with the following adaptation: the reduction of the garbage collection steps is always delayed till the end.

4 Preliminary notions about SRN

We recall from Section 1 that $\text{SRN}^{n;s}$ is the subset of SRN whose terms have normal arity n , and safe arity s . If not otherwise stated, $\vec{t}^m = t_1, \dots, t_m$ will always denote sequences of $m \geq 0$ terms of SRN . Moreover, we write $|\vec{t}^m| \leq l$, for some $l > 0$, meaning that the size of every term t_i is not greater than l . Now, from [4], we recall that, for every t in $\text{SRN}^{n;s}$, and $\vec{x} = x_1, \dots, x_n$, $\vec{y} = y_1, \dots, y_s$:

$$|t(\vec{x}; \vec{y})| \leq p_t(|x_1|, \dots, |x_n|) + \max\{|y_1|, \dots, |y_s|\} \quad (5)$$

where p_t is the *characteristic polynomial* of t which is non-decreasing and depends on t . We notice that if u is a subterm of t , then $\partial(p_u) \leq \partial(p_t)$. At last, we define the *composition degree* $\partial_C(t)$ and the *recursion degree* $\partial_R(t)$ of t , as the functions that count resp. the number of safe composition and recursion schemes inside t .

Definition 5 (The Term Bounding Function $tb(\cdot)$) *Let t in $\text{SRN}^{n;s}$ and $l \geq 0$. We define $tb(\cdot)$, that takes t and l as arguments, as $tb_t(l) = p_t(l, \dots, l) + l$.*

Fact 6 ($tb(\cdot)$ Bounds the Output Length of $t \in \text{SRN}$) *For every t in $\text{SRN}^{n;s}$, $l \geq 0$, and sequences \vec{x}, \vec{y} such that $|\vec{x}|, |\vec{y}| \leq l$, we have $|t(\vec{x}; \vec{y})| \leq tb_t(l)$.*

Definition 7 (The Net Bounding Function $nb(\cdot)$) *Let t in $\text{SRN}^{n;s}$ and $l \geq 0$. We define $nb(\cdot)$, that takes t and l as arguments, as $nb_t(l) = tb_t(tb_t(\dots tb_t(l) \dots))$, with $\partial_R(t) + \partial_C(t)$ occurrences of $tb_t(\cdot)$.*

Fact 8 ($nb(\cdot)$ is a Polynomial) *For every fixed t in $\text{SRN}^{n;s}$, $nb_t(l)$ is a polynomial in the free variable l , whose degree is $\partial(p_t)^{\partial_C(t) + \partial_R(t)}$.*

5 Preliminary useful nets in LALL

We introduce a first set of nets useful to define the embedding from SRN to LALL . However, whenever neither boxes, nor contractions are used in a given net Π , whose conclusion has type A , to save space, we

$$\begin{aligned}
& x^A \in V \Rightarrow x \in \Lambda_V^A \\
& m \geq 1, M \in \Lambda_V^A, x_1 \in \Lambda_V^{A_1}, \dots, x_m \in \Lambda_V^{A_m} \Rightarrow (\lambda \otimes_{i=1}^m x_i^{A_i}.M) \in \Lambda_V^{A_1 \otimes \dots \otimes A_m \multimap A} \quad (6) \\
& M \in \Lambda_U^{A' \multimap A}, N \in \Lambda_W^{A'}, U \cup W \subseteq V, U \cap W = \emptyset \Rightarrow (MN) \in \Lambda_V^A \\
& m \geq 2, (1 \leq i \neq j \leq m \Rightarrow N_i \in \Lambda_{W_i}^{A_i}, \\
& \quad W_i \cap W_j = \emptyset, W_i \subseteq V) \Rightarrow (\otimes_{i=1}^m N_i) \in \Lambda_V^{A_1 \otimes \dots \otimes A_m} \quad (7) \\
& M \in \Lambda_V^A \Rightarrow \Lambda \alpha.M \in \Lambda_V^{\forall \alpha.A} \quad (8) \\
& M \in \Lambda_V^{\forall \alpha.A} \Rightarrow M\{A'\} \in \Lambda_V^{A[A'/\alpha]} \quad (9)
\end{aligned}$$

Figure 5: Typed II order affine λ -terms.

represent Π by means of a λ -term. The term belongs to the set Λ_V^A of polymorphic typed *affine* λ -terms with variables in V , patterns, tuples, and type $A \in \mathcal{F}_{\mathcal{S}}$. Figure 5 defines Λ_V^A . (6) introduces λ -abstractions on a tuple pattern, while (7) introduces tuples. The application is left-associative. We shall drop useless parenthesis to avoid cluttering the terms. For any A and V , the terms in Λ_V^A rewrite under the standard β -reduction, extended with the following two rules: (i) $(\lambda \otimes_{i=1}^m x_i.M)(\otimes_{i=1}^m N_i) \rightarrow_{\beta} M [^{N_1/x_1} \dots ^{N_m/x_m}]$, where $M [^{N_1/x_1} \dots ^{N_m/x_m}]$ stands for the simultaneous substitution of N_i for x_i , with $1 \leq i \leq m$, and (ii) $(\Lambda \alpha.M)\{B\} \rightarrow_{\beta} M [^{B/\alpha}]$.

Booleans. The type of booleans is $\mathbb{B} = \forall \gamma. \gamma \multimap \gamma \multimap \gamma$ whose representative nets are:

$$\begin{aligned}
\mathbb{F} &= \Lambda \gamma. \lambda x^{\gamma} \lambda y^{\gamma}. x \triangleright \vdash \mathbb{B} && \text{(True)} \\
\mathbb{T} &= \Lambda \gamma. \lambda x^{\gamma} \lambda y^{\gamma}. y \triangleright \vdash \mathbb{B} && \text{(False)} \\
\mathbb{V}B[b] &= b \{ \mathbb{B} \otimes \mathbb{B} \} (\mathbb{T} \otimes \mathbb{T}) (\mathbb{F} \otimes \mathbb{F}) \triangleright \mathbb{B} \vdash \mathbb{B} \otimes \mathbb{B} && \text{(Duplication)}
\end{aligned}$$

The net $\mathbb{V}B[b]$ duplicates any boolean we may plug into b by a Cut node.

Church words or, simply, words. The type of words is $\mathcal{C} = \forall \alpha. !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$. Figure 6(a) introduces the successor $\text{SuccC}_0[w]$, where w identifies the lowermost dangling edge. It should be trivial to recover $\text{SuccC}_1[w]$ from $\text{SuccC}_0[w]$. Figure 6(c) introduces $\varepsilon\mathcal{C}$. If w is a natural number in binary notation, \bar{w} is its usual representation by a net. Figures 6(b) and 6(d) introduce nets that invert the bits inside any \bar{w} , plugged by Cut into the dangling input of $\text{RevC}[w]$.

Scott words. Intuitively, the type \mathcal{S} of Scott words describes a tuple of booleans. On Scott words we have the following nets:

$$\begin{aligned}
\varepsilon\mathcal{S} &= \Lambda \alpha. \lambda x^{\alpha} \lambda y^{\mathbb{B} \multimap \mathcal{S} \multimap \alpha}. x \triangleright \vdash \mathcal{S} && \text{(Empty Scott word)} \\
\text{SuccS}_0[s] &= \Lambda \alpha. \lambda x^{\alpha} \lambda y^{\mathbb{B} \multimap \mathcal{S} \multimap \alpha}. y \mathbb{F} s \triangleright \mathcal{S} \vdash \mathcal{S} && \text{(Successor zero)} \\
\text{SuccS}_1[s] &= \Lambda \alpha. \lambda x^{\alpha} \lambda y^{\mathbb{B} \multimap \mathcal{S} \multimap \alpha}. y \mathbb{T} s \triangleright \mathcal{S} \vdash \mathcal{S} && \text{(Successor one)} \\
\text{PredS}[s] &= s \{ \mathcal{S} \} \varepsilon\mathcal{S} (\lambda b^{\mathbb{B}} w^{\mathcal{S}}. w) \triangleright \mathcal{S} \vdash \mathcal{S} && \text{(Predecessor)} \\
\text{CondS}[s, x, y] &= \text{PrepS}[s] \{ \mathcal{S} \} \varepsilon\mathcal{S} (\lambda b^{\mathbb{B}}. b \{ \mathcal{S} \} yx) \triangleright \mathcal{S}, \mathcal{S}, \mathcal{S} \vdash \mathcal{S} && \text{(Conditional)} \\
\text{PrepS}[s] &= s \{ \mathcal{S} \} \text{SuccS}_0[\varepsilon\mathcal{S}] (\lambda b^{\mathbb{B}} \lambda w^{\mathcal{S}}. b \{ \mathcal{S} \multimap \mathcal{S} \} \\
& \quad (\lambda x^{\mathcal{S}}. \text{SuccS}_0[x]) (\lambda x^{\mathcal{S}}. \text{SuccS}_1[x]) w) \triangleright \mathcal{S} \vdash \mathcal{S} && \text{(Preprocessing)}
\end{aligned}$$

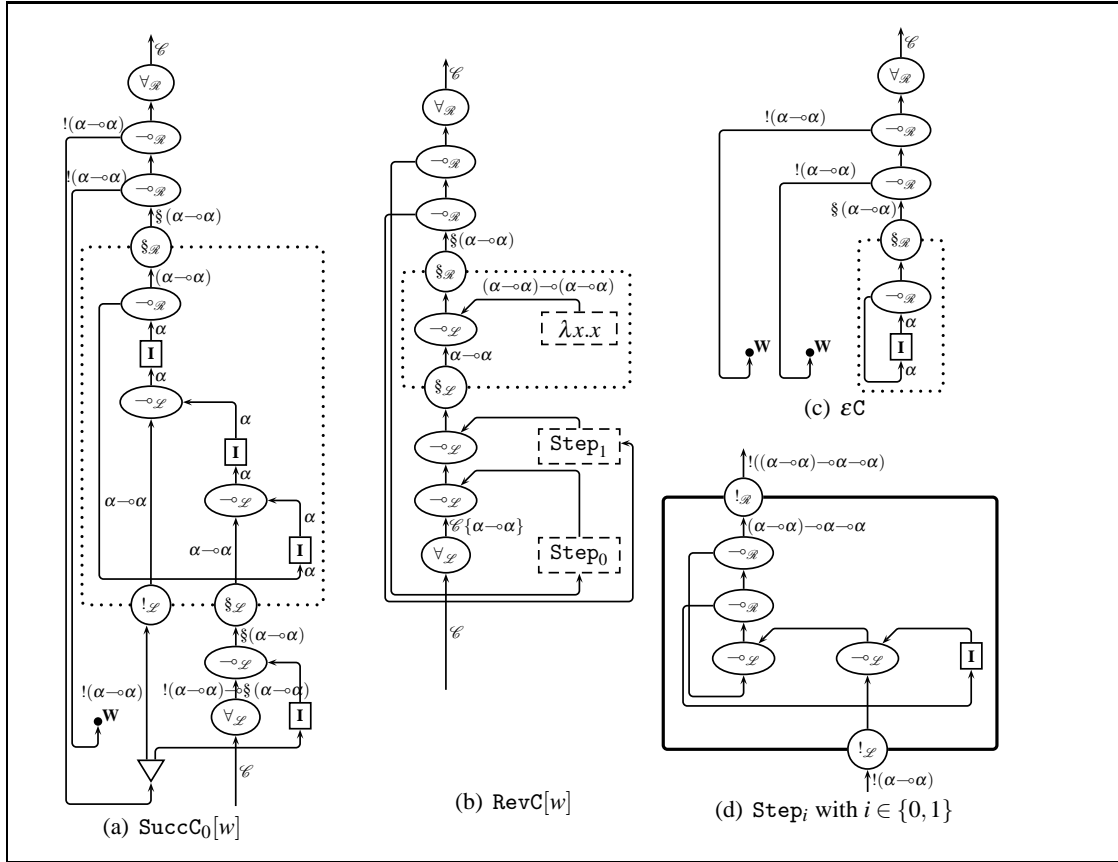


Figure 6: (Church) Words.

We remark that $\text{SuccS}_0[s]$ adds to s the least significant bit T , which stands for the digit 0, and $\text{SuccS}_1[s]$ adds F , instead, which stands for 1. $\text{PredS}[s]$ shifts s to its right deleting the *least significant* bit. So:

Remark 9 A Scott word is in fact a stack of bits, the least significant bit being on the top of the stack.

Moreover, $\text{CondS}[s, x, y]$ branches a computation, depending on the value of s . It yields x if the least significant bit of s is 0, or if $s = \epsilon S$, while it yields y if the least significant bit of s is 1. The preprocessing avoids to return ϵS : if $s = \epsilon S$, then s becomes $\text{SuccS}_0[\epsilon S]$. Also, the three assumptions of type \mathcal{S} in $\mathcal{S}, \mathcal{S}, \mathcal{S} \vdash \mathcal{S}$ specify the type of s, x , and y , respectively.

Fact 10 (Relation between naturals, Scott words, and words-as-terms) Every sequence (d_1, \dots, d_l) with $d_1, \dots, d_l \in \{0, 1\}$ and $l \geq 0$, identifies uniquely a number $n = 2^{l-1} \cdot d_l + \dots + 2^0 \cdot d_1 \in \mathbb{N}$. So, both the term of **SRN** $\mathfrak{s}_{d_l}(\dots \mathfrak{s}_{d_1}(\dots))$ and the Scott word $[n]$ identify n , too. We say that the sequence, as well as the Scott number and the **SRN** term, represent n .

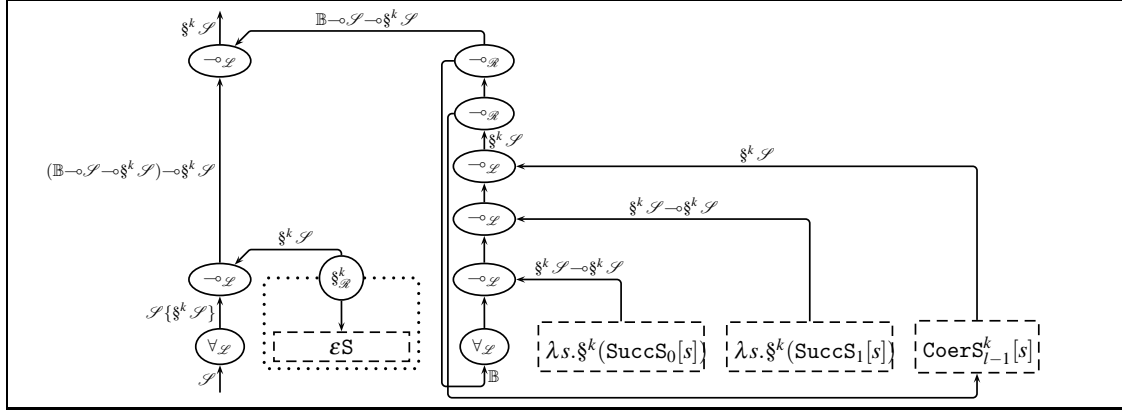
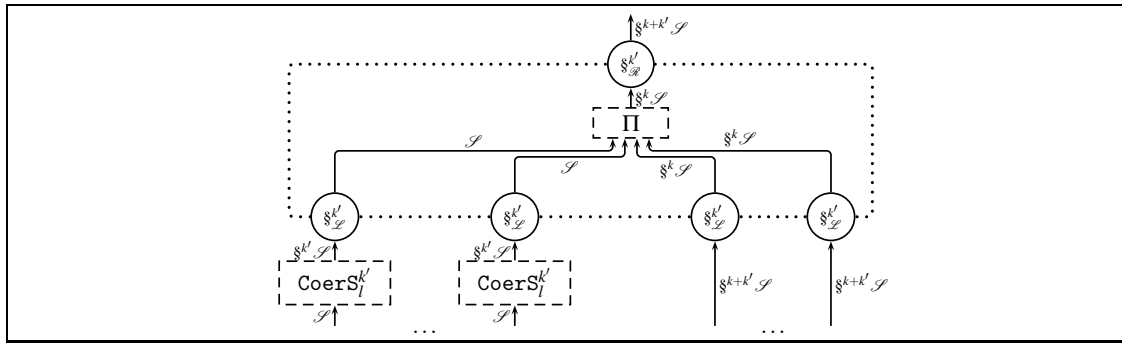
We underline that an infinite number of sequences, and of terms, represent the same n .

Scott words to words. For any $l \geq 0$, $\text{StoC}_l[s] \triangleright \mathcal{S} \vdash \mathcal{C}$ is inductively defined on l :

$$\text{StoC}_0[s] = \epsilon C$$

$$\text{StoC}_l[s] = s \{ \mathcal{C} \} \epsilon C (\lambda x^{\mathbb{B}} y^{\mathcal{S}} . x (\lambda z^{\mathcal{C}} . \text{SuccC}_0[z]) (\lambda z^{\mathcal{C}} . \text{SuccC}_1[z]) \text{StoC}_{l-1}[s])$$

The net $\text{StoC}_l[s]$ normalizes to the word \bar{w} , for any Scott word *at most as long as* l .

Figure 8: The coerce net $\text{CoerS}_l^k[s]$ on Scott words.Figure 9: The *Lifting* $\text{Lift}_{k'}[\Pi]$ of a proof net Π .

6 The embedding $\lceil \cdot \rceil$ from SRN to LALL

The goal is to compositionally embed **SRN** into **LALL**, with a map as much analogous as possible to the natural, and inductively defined one from **BC⁻** into **ILAL** [8]. For any fixed n and s , the map $\lceil \cdot \rceil$ takes a term t of **SRN** ^{$n;s$} as first and $l \geq 0$ as second argument, and yields a net $\lceil t \rceil^l \triangleright \overrightarrow{\mathcal{S}}^n, \mathcal{S}^k \mathcal{S} \vdash \mathcal{S}^k \mathcal{S}$, for some k . We define the map inductively on the first argument.

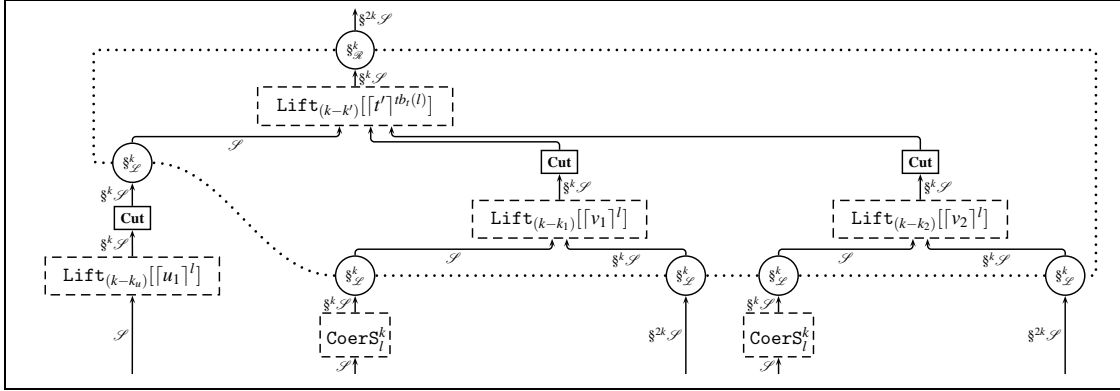
The base cases of $\lceil \cdot \rceil$. Some of them are straightforward:

$$\begin{aligned} \lceil 0 \rceil^l &= \varepsilon S \triangleright \vdash \mathcal{S} & \lceil s_i \rceil^l &= \text{SuccS}_i[s] \triangleright \mathcal{S} \vdash \mathcal{S} & (i \in \{0, 1\}) \\ \lceil p \rceil^l &= \text{PredS}[s] \triangleright \mathcal{S} \vdash \mathcal{S} & \lceil B \rceil^l &= \text{CondS}[s, x, y] \triangleright \mathcal{S}, \mathcal{S}, \mathcal{S} \vdash \mathcal{S} \end{aligned}$$

where, s, x, y denote the inputs of the nets they appear into. Concerning the projection, $\lceil \pi_i^{n;s} \rceil^l$ is an Axiom that connects the i -th input to the conclusion, erasing all the other inputs by Weakening. An example with $1 \leq i \leq n$, and, notice, $k = 0$ is:



The case of $\lceil \cdot \rceil$ on the composition. We now focus on $t = \circ[t', u_1, \dots, u_m, v_1, \dots, v_r]$ such that t' be in **SRN** ^{$m;r$} . Without loss of generality, we show how to build $\lceil t \rceil^l$ by assuming $m = n = s = 1$, and $r = 2$.

Figure 10: The (partial) translation of $\circ[t', u_1, v_1, v_2]$ with missing contractions.

By induction we have:

$$\begin{array}{ll} [t']^{tb_r(l)} \triangleright \mathcal{S}, \mathfrak{S}^k \mathcal{S}, \mathfrak{S}^k \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} & [u_1]^l \triangleright \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} \\ [v_i]^l \triangleright \mathcal{S}, \mathfrak{S}^k \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} & (i \in \{1, 2\}) \end{array}$$

By letting $k = \max\{k', k_u, k_1, k_2\}$, we get:

$$\begin{array}{ll} \text{Lift}_{(k-k')}[[t']^{tb_r(l)}] \triangleright \mathcal{S}, \mathfrak{S}^k \mathcal{S}, \mathfrak{S}^k \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} & \text{Lift}_{(k-k_u)}[[u_1]^l] \triangleright \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} \\ \text{Lift}_{(k-k_i)}[[v_i]^l] \triangleright \mathcal{S}, \mathfrak{S}^k \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} & (i \in \{1, 2\}) \end{array}$$

Next, if we build Π' in Figure 10, then $[t]^l$ is $\nabla_l^{2k}[\nabla_l^0[\nabla_l^0[\Pi']]]$. The two occurrences of ∇_l^0 contract three “normal” inputs. One is from $[u_1]^l$. The other two are from $[v_1]^l, [v_2]^l$. The occurrence of ∇_l^{2k} contracts the single “safe” input of $[v_1]^l$ and $[v_2]^l$. We insist remarking the existence of $[t]^l$ for any m, n, r, s . One can count: $||[t]^l|| \leq |\text{Lift}_{(k-k')}[[t']^{tb_r(l)}]| + \sum_i^m |\text{Lift}_{(k-k_i)}[[u_i]^l]| + \sum_j^r |\text{Lift}_{(k-k_j)}[[u_j]^l]| + k(1+n+s'n+s's) + s'n|\text{CoerS}_l^k[s]|$. So, it follows $||[t]^l|| \in O(|[t']^{tb_r(l)}| + \sum_i^m |[u_i]^l| + \sum_j^r |[u_j]^l| + tb_r(l) + l)$.

The case of $[\cdot]$ on the recursion. Let $t = r[u_\varepsilon, u_0, u_1]$ with $u_\varepsilon \in \mathbf{SRN}^{n;s}, u_0, u_1 \in \mathbf{SRN}^{n+1;s+1}$. As for composition, we set $n = s = 1$ which is general enough to show the key point of the embedding. In the course of the iteration unfolding that $[t]^l$ carries out, the safe argument gets duplicated, so we must contract it. By induction:

$$[u_\varepsilon]^l \triangleright \mathcal{S}, \mathfrak{S}^k \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} \quad [u_i]^{tb_r(l)} \triangleright \mathcal{S}, \mathcal{S}, \mathfrak{S}^k \mathcal{S}, \mathfrak{S}^k \mathcal{S} \vdash \mathfrak{S}^k \mathcal{S} \quad (i \in \{0, 1\})$$

By letting $k = \max\{k_\varepsilon, k_0, k_1\}$, and using $\text{Lift}[\cdot]$ in analogy to the translation of the composition, $[t]^l$ is in Figure 11. The Scott word that drives the recursion unfolding, becomes a word, and, then, it is necessary to reverse it by $\text{RevC}[w]$. Otherwise we would unfold the iteration according to a wrong bit order, as implied by Remark 9. Moreover, (i) Π projects the rightmost $n + s + 1$ -th element of type A it gets in input and which contains the result, and (ii) the two nets $\boxed{\otimes_{\mathcal{R}}}, \boxed{\otimes_{\mathcal{L}}}$ are obvious trees of Tensor R and L nodes. Finally, we can prove $||[t]^l|| \in O(|[u_\varepsilon]^l| + |[u_0]^{tb_r(l)}| + |[u_1]^{tb_r(l)}| + tb_r(l))$.

Definition 11 (Representing a term by a net) Let t be in $\mathbf{SRN}^{n;s}$, $l \in \mathbb{N}$, and $\Pi \triangleright \overrightarrow{\mathcal{S}}^n (\mathfrak{S}^k \mathcal{S})^s \vdash \mathfrak{S}^k \mathcal{S}$, for some $k \in \mathbb{N}$. Then, Π k -simulates t with l -bounded inputs if, for every pair of vectors of natural numbers \vec{x}^n, \vec{y}^s , such that $|\vec{x}^n|, |\vec{y}^s| \leq l$, the net we get by plugging $[x_1]^l, \dots, [x_n]^l, \mathfrak{S}^k [y_1]^l, \dots, \mathfrak{S}^k [y_s]^l$ into the inputs of Π , in the natural way, normalizes to $\mathfrak{S}^k [z]^l$, whenever z is the result of $t(\vec{x}^n; \vec{y}^s)$.

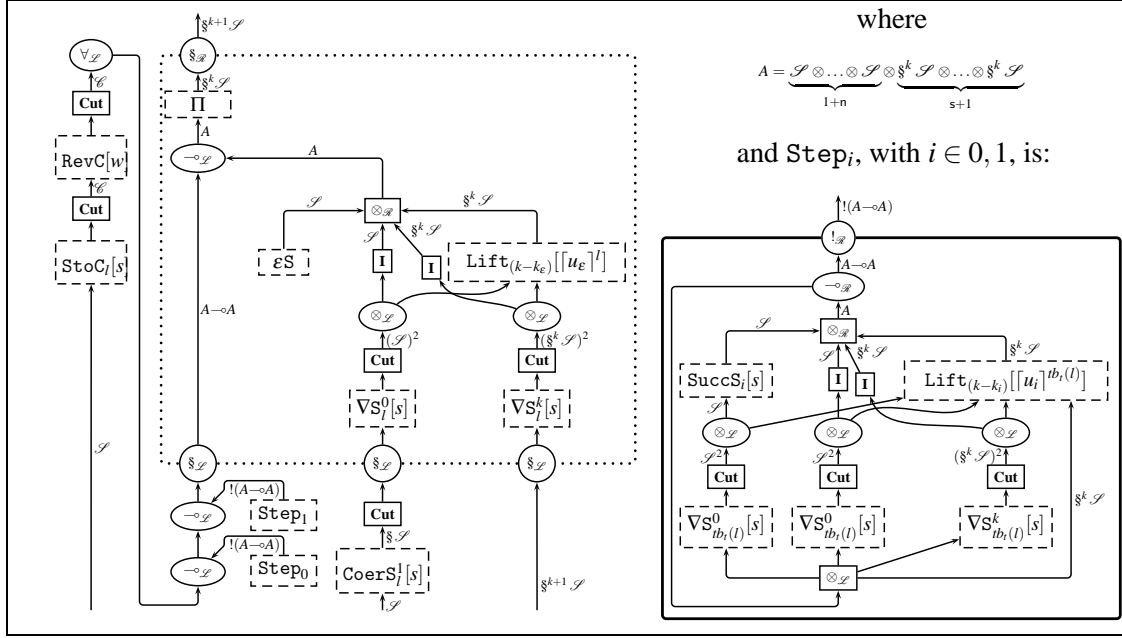


Figure 11: Safe recursion.

Proposition 12 (SRN embeds into LALL) *Let $l \geq 0$, and $t \in \mathbf{SRN}^{n;s}$. Then, $[t]^l$ k -simulates t with l -bounded inputs. Moreover, (i) $k \leq \partial_R(t) \cdot 2^{\partial_C(t)}$, and (ii) $|[t]^l|$ is $O\left(l^{\partial(p_r)(\partial_C(t) + \partial_R(t))}\right)$, namely a polynomial in l .*

The statement holds by induction on t , using the definition of $[\cdot]^l$, the size of every net that $[\cdot]^l$ generates, the definitions of $nb_i(\cdot)$, and $tb_i(\cdot)$, together with Facts 8, 6, and 10.

7 Conclusions and further works

We have shown that the compatibility between the *predicative analysis* over recursive functions that **SRN** encodes, and the proof theoretical *stratification*, that regulates the complexity of some Light Logic that characterize **PTIME**, can be improved, provided that (i) the stratification we find in Light Linear Logic and **ILAL** relaxes to boxes with “fuzzy” border, as in **ML**⁴ or **LALL**, and (ii) we move to a representation of words *alternative to the standard one*, able both to represent the basic functions of **SRN** in constant time, and to exploit the independence of the cut elimination complexity from the logical complexity of the formulæ in a net.

As a consequence, every term t of **SRN** maps to a family $\langle [t]^l \rangle_{l \in \mathbb{N}}$ of nets in **LALL**, where $[t]^l$ simulates t with inputs *as most as long as* l . The number of paragraph modalities in the type of the conclusion of $[t]^l$ depends on the structural complexity of t . The size of $[t]^l$ is a polynomial in l whose degree depends on the degree of the characteristic polynomial of t and on the structural complexity of t .

As an example, the following program, which returns y if $w \neq 0$ contains a digit ‘0’ that is not the lowermost digit, and z otherwise, is in **SRN** but not in **BC**⁻:

$$\begin{cases} g(0; y, z) & = z \\ g(s_i w; y, z) & = h(w; y, z, g(w; y, z)) \end{cases}, \quad (*)$$

where $h(w; y, z, t) = \text{cond}(; w, y, t)$. The embedding we propose gives the family of nets that implement it in **LALL**, while, it is worth remarking, it is unknown how to represent $g(w; y, z)$ inside **ILAL**.

Admittedly, the representation of a term of **SRN** by a family of nets, rather than as a unique net, is not standard. For example, one might be tempted to observe that every function with finite domain is the initial fragment of some polynomial time function, so **LALL** represents *every function with finite domain*. Beware, however, that *not every algorithm* is in **LALL**. For example, in analogy with [7], we show an algorithm exp that cannot exist as a net of **LALL** because it calculates a non-polytime function. exp will be defined using the traditional *non-predicative* recursive schemes, so that it is not a program of **SRN**. exp is defined as follows. We know that the two programs $\text{concat}(x; y)$, which concatenates two strings of bits, and $\text{double}(x;) = \text{concat}(x; \pi_1^{1,0}(x;))$ belong to **SRN**. Then, we can define the recursive function exp :

$$\text{exp}(0;) = s_1(; \pi_1^{1,0}(0;)) \quad \text{exp}(s_i(; x);) = \text{double}(\text{exp}(x;);) \quad (i \in \{0, 1\})$$

The program exp is *not* in **SRN** because of the position of the argument that drives the unfolding. *This reflects into LALL*, where $\lceil \text{concat} \rceil^l : \mathcal{S}, \mathcal{S} \vdash \mathcal{S}$ and $\lceil \text{double} \rceil^l : \mathcal{S} \vdash \mathcal{S}$ exist, but $\lceil \text{double} \rceil^l$ cannot be iterated because of the form of its type. So, $\lceil \text{exp} \rceil^l$ cannot be defined as a net of **LALL** using the constructions of this paper.

We conclude by an example about how the approach “**SRN** as family-of-proofs” we present here can be rewarding. We consider the following program:

$$\begin{cases} \text{gt}(0, y) & = \text{False} \\ \text{gt}(s_i x, y) & = \text{if } (y = 0) \text{ then True else gt}(x, \text{pred}(y)). \end{cases}$$

The program gt is such that $\text{gt}(x, y) = \text{True}$ iff $|x| > |y|$. It has a recursive definition more liberal than the primitive recursion scheme, as the recursive call of gt applies a function on the parameter y that does not drive the unfolding. Namely, gt incorporates a *double iteration*. Certainly, gt cannot exist in **SRN** in the form here above. Instead, **LALL** admits to represent gt as follows:

$$\begin{aligned} \text{gt}^{\mathcal{C} \rightarrow \mathcal{S} \rightarrow \mathbb{B}} &= \lambda x^{\mathcal{C}} . \lambda y^{\mathcal{S}} . \pi_2(x \{ \mathcal{S} \otimes \mathbb{B} \} (\text{step})(y \otimes \mathbb{F})) \\ \text{step}^{\mathcal{S} \otimes \mathbb{B} \rightarrow \mathcal{S} \otimes \mathbb{B}} &= \lambda s^{\mathcal{S}} \otimes b^{\mathbb{B}} . b \{ \mathcal{S} \otimes \mathbb{B} \} (0 \otimes \mathbb{T}) \left(s(0 \otimes \mathbb{T}) (\lambda x^{\mathbb{B}} . \lambda y^{\mathcal{S}} . y \otimes \mathbb{F}) \right) \end{aligned}$$

where $\mathbb{B}, \mathbb{T}, \mathbb{F}$ are at page 7. The existence of gt in **LALL** implies the existence of a family $\langle \text{ord}_l \rangle_{l \in \mathbb{N}}$ of nets. Every ord_l takes two Scott words with *at most l bits*, and gives them back sorted according to their length. The definition of every ord_l is a net of **LALL** that we compactly write as a λ -term:

$$\begin{aligned} \text{ord}_l^{\mathcal{S} \otimes \mathcal{S} \rightarrow \mathcal{S} \otimes \mathcal{S}} &= \lambda x^{\mathcal{S}} \otimes y^{\mathcal{S}} . (\lambda x_1^{\mathcal{S}} \otimes x_2^{\mathcal{S}} . \lambda y_1^{\mathcal{S}} \otimes y_2^{\mathcal{S}} . \text{BtoB}(\text{gt}(\text{StoC}_l[x_1])y_1)(y_2 \otimes x_2)) (\nabla S_l x) (\nabla S_l y) \\ \text{BtoB}^{\mathbb{B} \rightarrow \mathbb{B}} &= \lambda x^{\mathbb{B}} . x \{ \mathbb{B} \} (\Lambda \gamma . \lambda w^\gamma \otimes z^\gamma . w \otimes z) (\Lambda \gamma . \lambda w^\gamma \otimes z^\gamma . z \otimes w) \end{aligned}$$

where $\mathbf{B} = \forall \gamma . (\gamma \otimes \gamma) \rightarrow (\gamma \otimes \gamma)$ is a linear version of the booleans, $\text{StoC}_l^{\mathcal{S} \rightarrow \mathcal{C}}$ is at page 8, and the *safe duplication* $\nabla S_l^{\mathcal{S} \rightarrow \mathcal{S} \otimes \mathcal{S}}$ is at page 9.

Given ord_l , we can write a family of insertion sorts that sort lists of Scott Words as much long as l . Figure 12 describes one element of the family. We warn the reader about the syntax we use. It does not perfectly adhere to the one in Figure 5, but nets would consume too much space. The effort to move from the terms in Figure 12 to the nets of **LALL** they represent should be a reasonably simple exercise. We observe that putTop_l is a linear algorithm that manipulates only the *head* of a given list. Instead,

$$\begin{aligned}
L(X) &= \forall \alpha.!(X \multimap \alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) && \text{is the type of lists,} \\
\emptyset^{L(X)} &= \lambda c^{!(X \multimap \alpha \multimap \alpha)}. \lambda z^\alpha. z \\
\text{sort}_l^{L(\mathcal{S}) \multimap \S L(\mathcal{S})} &= \lambda t^{L(\mathcal{S})}. t \{L(\mathcal{S})\} (\text{insert}_l) \emptyset^{L(\mathcal{S})} \\
\text{insert}_l^{\mathcal{S} \multimap L(\mathcal{S}) \multimap L(\mathcal{S})} &= \lambda n^\mathcal{S}. \lambda t^{L(\mathcal{S})}. \lambda c^{!(\mathcal{S} \multimap \alpha \multimap \alpha)}. \lambda z^\alpha. \\
&\quad (\lambda x^\mathcal{S} \otimes y^\alpha. cxy) ((t \text{putTop}[c]_l) (\text{CoerS}_l^1[n] \otimes z)) \\
\text{putTop}[c]_l^{\mathcal{S} \multimap \mathcal{S} \otimes \alpha \multimap \mathcal{S} \otimes \alpha} &= \lambda a^\mathcal{S}. \lambda b^\mathcal{S} \otimes t^\alpha. (\lambda i^\mathcal{S} \otimes \lambda j^\mathcal{S}. i \otimes cjt) (\text{ord}_l a \otimes b)
\end{aligned}$$

Figure 12: Insertion sort for Scott Words no longer than l .

insert_l takes a number and a sorted list, and puts the number at the correct position of the list, so to preserve the sorting. While performing an iteration, insert_l does *not* add any paragraph \S in front of the type of the output. The reason is that it exploits the general scheme that allows to write a perfectly linear *predecessor* on Church numerals in the λ -calculus [9]. Finally, sort_l , iterates insert_l in the usual way, thus adding a \S in front of its output type.

Future lines of research. We must say that the representation of **SRN** as a family of nets of **LALL** that we present in this work has been an alternative approach to the standard one, which would explore the relations between **SRN** and stratification by mapping a single term of **SRN** into a single net. That standard approach has been developed in [11, 10, 12]. Those works make some progress as compared to [8]. This means that they identify a Light Logic that strictly contains **ILAL**, and which allows to represent a strict extension of **BC**⁻. However, the whole **SRN** still escapes any full representation inside a Light Logic. So, it has been natural to look for an alternative approach; and this brought us to this work.

Naturally enough, future work is about “integrating” both *by level technology* and *multimodality*. Multimodality is in the framework **MS** developed in the previously cited works [11, 10, 12]. The conjecture is that the two technologies together may lead to a more refined proof theoretical representation of the principles underpinning the definition of **SRN**, and of the predicative analysis it encodes, possibly increasing the set of algorithms that we can represent inside Light Logics.

References

- [1] Martìn Abadi, Luca Cardelli & Gordon D. Plotkin (1993): *Types for Scott numerals*. Available at <http://lucacardelli.name/Papers/Notes/scott2.pdf>.
- [2] Andrea Asperti & Luca Roversi (2002): *Intuitionistic Light Affine Logic*. *ACM Trans. Comput. Log.* 3(1), pp. 137–175. Available at <http://doi.acm.org/10.1145/504077.504081>.
- [3] Patrick Baillot & Damiano Mazza (2010): *Linear Logic by Levels and Bounded Time Complexity*. *Theor. Comp. Sci.* 411(2), pp. 470–503.
- [4] Stephen Bellantoni & Stephen A. Cook (1992): *A New Recursion-Theoretic Characterization of the Polytime Functions*. *Computational Complexity* 2, pp. 97–110.
- [5] J.-Y. Girard (1998): *Light Linear Logic*. *Inf. Comput.* 143(2), pp. 175–204.
- [6] D. Leivant (1994): *A Foundational Delineation of Poly-time*. *I.&C.* 110(2), pp. 391–420.

- [7] Daniel Leivant (1993): *Stratified Functional Programs and Computational Complexity*. In: *POPL*. pp. 325–333.
- [8] Andrzej S. Murawski & C.-H. Luke Ong (2004): *On an interpretation of safe recursion in light affine logic*. *Theor. Comput. Sci.* 318(1-2), pp. 197–223.
- [9] L. Roversi (1999): *A P-Time Completeness Proof for Light Logics*. In: *Ninth Annual Conference of the EACSL (CSL'99), LNCS 1683*. Springer-Verlag, Madrid (Spain), pp. 469 – 483.
- [10] L. Roversi & L. Vercelli (2009): *A structural and local criterion for polynomial time computations*. Available at <http://www.di.unito.it/~rover/RESEARCH/PUBLICATIONS/2009-FOPARA/RoversiVercelli2009FOPARA.pdf>. Accepted for the publication in the post-proceedings of the Workshop FOPARA09 (FM2009 affiliated).
- [11] L. Roversi & L. Vercelli (2009): *Some Complexity and Expressiveness Results on Multimodal and Stratified Proof Nets*. In: *Proceedings of TYPES'08*. pp. 306 – 322. Available at http://dx.doi.org/10.1007/978-3-642-02444-3_19.
- [12] Luca Vercelli: *On the Complexity of Stratified Logics*. Ph.D. thesis, Scuola di Dottorato in Scienze e Alta Tecnologia — Università di Torino — Italy. Available at <http://arxiv.org/abs/1002.3453v1>.