Safe Recursion on Notation into a Light Logic by Levels

This is the author's manuscript

Original Citation:
Safe Recursion on Notation into a Light Logic by Levels / Luca Roversi ; Luca Vercelli. - In: ELECTRONIC PROCEEDINGS IN THEORETICAL COMPUTER SCIENCE. - ISSN 2075-2180. - ELETTRONICO. - 23(2010), pp. 63-77. ((Intervento presentato al convegno International Workshop on Developments in Implicit Computational complExity tenutosi a Paphos, Cyprus nel 27-28th March 2010.

Availability:
This version is available http://hdl.handle.net/2318/131262 since

Published version:
DOI:10.4204/EPTCS.23

Terms of use:
Open Access
Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)
Safe Recursion on Notation into a Light Logic by Levels\(^\star\)

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

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

We embed Safe Recursion on Notation (SRN) into Light Affine Logic by Levels (LALL), derived from the logic 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 \([|t|]_l\) 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, \textit{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 ML\(^4\) to “freely” duplicate the arguments, especially the safe ones, of \(t\). Finally, the type \([|t|]_l\) 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 \textit{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 (ML\(^4\)) [3] and from Intuitionistic Light Affine Logic (ILAL) [2]. We recall, ML\(^4\) and ILAL are two Light Logics, \textit{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 notion \textit{Stratification}, which expresses specific structural restrictions on the derivations of ML\(^4\) and ILAL. SRN, of which we recall some more aspects in Section [4], provides a \textit{predicative analysis} of primitive recursion. It is the least set that contains the \textit{zero} 0 (considered as a \textit{0}-ary function), the \textit{successors} \(s_0(;x) = 2x, s_1(;x) = 2x + 1\), the \textit{predecessor} \(p(2x + i) = x\), the \textit{projection} \(\pi^i_{\leq s}(\overrightarrow{x},\overrightarrow{y}) = x_k\) if \(1 \leq k \leq n\), and \(y_k\) if \(1 \leq k \leq s\), the \textit{conditional} \(B(;y,y_1,y_2) = y_1\) if \(y\) is odd, and \(y_2\) otherwise, and which is closed under \textit{safe composition} and \textit{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 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 [5]. In fact, [8] also shows that an extension BC\(^\pm\), polynomial time complete, can be represented inside ILAL. However, since the primitives added to BC\(^\pm\) are not in SRN, we cannot see BC\(^\pm\) as relevant to the goal of understanding the

\(^\star\)Partially supported by MIUR PRIN CONCERTO — protocol number 2007BHXCDFH

\(\copyright\) L. Roversi & L. Vercelli  
This work is licensed under the  
Creative Commons Attribution License.
possible relation between full SRN and the above stratification principle, basic to ML\textsuperscript{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 \( \mathcal{J} = \forall \alpha . (\alpha \rightarrow (\mathbb{B} \rightarrow \alpha) \rightarrow \alpha) \), where \( \mathbb{B} \) is the type of booleans, and \( \mathcal{J} \) the data type of Scott words [11], 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 \( [\cdot] \cdot \) (Sections 5 and 6). The map \( [\cdot] \cdot \) has the same natural and inductive structure as the one of the map in [8] from SRN to ILAL. However, \( [\cdot] \cdot \) takes two arguments: (i) any term \( t \) of \( \text{SRN}^{\text{pos}} \), 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, \( [t]^l \) yields a net that simulates \( t(\overrightarrow{x}; \overrightarrow{y}) \) whenever every element in \( \overrightarrow{x} \), \( \overrightarrow{y} \) is at most as long as \( l \) (Proposition [12]). This suggests to summarize the situation we move in by:

\[
\frac{\text{LALL}}{\text{SRN}} = \frac{\text{PTIME-uniform Boolean Circuits}}{\text{PTIME Problems}}
\]

We remark, however, that such an analogy should be read as such, and not as a formal correspondence. I.e., we are not at all assuming any classical complexity theoretic perspective like the one in [12], which shows a proofs-as-programs correspondence between Boolean Circuits and nets of Multiplicative Linear Logic.

Instead, what we do reads as follows.

Let \( t \) be a term of \( \text{SRN}^{\text{pos}} \). We write \( \partial_c(t) \) and \( \partial_h(t) \) for the number of composition and recursion schemes, respectively, that are used to build \( t \). That way, \( \text{cmplx}(t) = \partial_c(t) + \partial_h(t) \) is a naive measure of the static complexity of \( t \). Also, let \( p_i \) be the characteristic polynomial of \( t \), whose values bound the length of the output of \( t \). Let \( \partial(p_i) \) be its degree. Then, \( t \) is represented in LALL by a family \( \{[t]^l\}_{l \in \mathbb{N}} \) of nets such that:

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

The first two points suggest the analogy [11]. 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 \( [t]^l \) 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 \( \mathcal{J} \) as a first fundamental choice to write \( [\cdot] \cdot \). The reason is twofold. One reason is a kind of obvious, since \( \mathcal{J} \) supports the representation of successors, predecessor, projection and conditional as constant time operations, unlike \( \mathcal{C} = \forall \alpha . !(\alpha \rightarrow \alpha) \rightarrow !(\alpha \rightarrow \alpha) \rightarrow \mathcal{S}(\alpha \rightarrow \alpha) \), which is generally used to represent Church words in Light Logics. The other reason, instead, brings a certain
\[ \circ [t' u_1 \ldots u_{\ell}, v_1 \ldots v_{\ell}](x; y) = t'(u_1(x); \ldots, u_{\ell}(x); v_1(x; y), \ldots, v_{\ell}(x; y)) \] (2)

\[ \searrow [u_{\ell}, u_0, u_1](0, x; y) = u_{\ell}(x; y) \] (3)

\[ \searrow [u_{\ell}, u_0, u_1](2z + i, x; y) = u_i(z, x; y, \searrow [u_{\ell}, u_0, u_1](z, x; y)) \quad i \in \{0, 1\} \]

Figure 1: **SRN**: predicative recursion on notation and safe composition.

![Proof nets of LALL](https://via.placeholder.com/150)

Figure 2: The nodes in the proof nets of **LALL**.

A second step to get \([\cdot]\) for every \(l\), we exploit what we like to call the **fuzzy** borders of paragraph boxes of **LALL** to write the net \(\nabla S_k^l\). The net \(\nabla S_k^l\) duplicates a Scott word at most as long as \(l\), starting from a premise of type \(S^k\) and concluding with the type \(S^k \otimes S^k\), for any \(k\). We remark that in **ILAL**, where the border of paragraph boxes is “rigid”, we could only write a net, analogous to \(\nabla S_k^l\), concluding with type \(S^k(\nabla \otimes \nabla)\) which would generally impede to get the right type for \([r]^l\). 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 \(\Pi\) is **stratified** if the number of boxes around every node keeps being constant in every net we reach from \(\Pi\) 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.
Figure 3: Constraints on the indexing. The nodes we omit have the same index on all of their incident edges.

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:

\[
F ::= \mathcal{S} \mid \alpha \mid F \otimes F \mid F \rightarrow F \mid \forall \alpha. F \mid \exists F \mid \exists^* F \mid \alpha \in \mathcal{V}
\]

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

\[
S = \forall \alpha. (\alpha \rightarrow (B \rightarrow S \rightarrow \alpha) \rightarrow \alpha)
\]

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

\[
\Pi \supset A_1, \ldots, A_r \vdash C \quad \text{and} \quad \Sigma \supset B_1, \ldots, B_l \vdash D,
\]

denoted as \( \Pi \vdash_{A_1, \ldots, A_r} C \) and \( \Sigma \vdash_{B_1, \ldots, B_l} D \), respectively, with \( r, l \geq 0 \), then all the graphs inductively built from \( \Pi \) and \( \Sigma \) by the rule schemes in Figure 4 are proof structures.

If \( \Pi \vdash_{\Gamma} A \), we say that \( \Pi \) proves the sequent \( \Gamma \vdash A \). The inputs (resp. outputs) of \( \Pi \) are the edges labelled \( \Gamma \) (resp. \( A \)). The set of the nodes of \( \Pi \) is \( V_{\Pi} \), and \( E_{\Pi} \) is the set of edges. The size \( |\Pi| \) of \( \Pi \) is the cardinality of \( V_{\Pi} \). 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 \( \Pi \) is the greatest depth among the nodes of \( \Pi \).

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 \( \Pi \) be a proof structure.

1. An indexing for \( \Pi \) is a function \( I \) from the edges of \( \Pi \) 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 \( \Pi \).

2. A net is a proof structure that admits an indexing.

3. An indexing \( I \) of \( \Pi \) is canonical if \( \Pi \) has an edge \( e \) such that \( I(e) = 0 \), and \( I(e') \geq 0 \) for all edges \( e' \) of \( \Pi \).
As in [3], we can state that every net of LALL admits a unique canonical indexing. The indexing tells that the nodes !\_L and §\_L 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 §-boxes whose border is somewhat fuzzy. We mean that a §-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 §\_R. To distinguish §-boxes from the ! ones we adopt a dotted border.

Let I\_0 be the canonical indexing of \( \Pi \) and \( e \in E_\Pi \). The level of \( e \) is \( l(e) \). It is defined as \( I_0(e) \). The level of \( \Pi \) is \( l(\Pi) \). It is defined as the greatest value assumed by \( I_0 \) on the edges of \( \Pi \). We denote as \( B_\Pi \) the set of the !-boxes in \( \Pi \), and it is naturally in bijection with the set of the !\_R nodes in \( \Pi \). Finally, for every net \( \Pi \), and for \( \dagger \in \{!, \$\} \), \( \dagger^n \Pi \) denotes \( n \) nested \( \dagger \)-boxes around \( \Pi \).

**Cut elimination.** We just recall its steps, which are standard. The linear cut elimination steps annihilate in the natural way a pair of linear nodes (Identity/Cut, \( \rightarrow \_L / \rightarrow \_R, \otimes \_L / \otimes \_R, \$\_L / \$\_R, \forall \_L / \forall \_R \)). The exponential cut elimination steps are of two kinds: \( !\_L / !\_R \) is reduced merging the two involved boxes which can be !-boxes as well as §-boxes with fuzzy borders. Instead, contraction/\( !\_L \) duplicates the whole !-box cut with the contraction, as in ILAL. The garbage collection cut elimination steps involve the weakening or the daemon nodes, cut with any other node. It is always possible to reduce such a cut with the help of some more weakening and daemon nodes, as done in ILAL [2]. The set of cut nodes of \( \Pi \) is \( \text{cuts}(\Pi) \).

**Proposition 2 (Cut-elimination)** Every LALL net reduces to a cut-free net.

A direct proof would be very long; anyway, such a proof directly follows from the proof of the namesake propositions in ILAL and ML. Please notice that the presence of fixpoints (i.e. the recursive type \( \mathcal{S} \)) does not affect the proof in any way, because the cut-elimination independently by the formulae.

---

Figure 4: Inductive rule schemes to build proof structures of LALL. (*) \( \alpha \) does not occur free in \( A_1, \ldots, A_r \). (**) A !-box, which has at most a single assumption.
labelling the edges of a net. This is not true in full Linear Logic.

3 Polynomial time soundness of LALL

We adapt [3] to prove the cut elimination PTIME-soundness in presence of unconstrained Weakening, which we introduce for easy of programming since it is handy to erase nets structure. Let us fix a proof net \( \Pi \) to reduce. We define an ordering over cuts(\( \Pi \)) that determines which cuts to reduce first.

A graph theoretic path in any proof net \( \Pi \) is exponential if it contains a, possibly empty, sequence of consecutive contractions and stops at a !\( \neq \) node.

Let \( B, C \in B_\Pi \). Let \( B \prec_1^t C \) 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 \( \prec_1^t \).

One can show that \( \preceq_L \) is a partial order, upward arborescent: for every \( C \) there is at most one \( B \) such that \( B \prec_1^t C \).

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 demon, 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 \( \Pi \) is canonical whenever smaller cuts relatively to \( \leq \) are eliminated before higher ones.

**Theorem 4 (Polynomial bound for LALL)** Let \( \Pi \) 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 \( \text{SRN} \) whose terms have normal arity \( n \), and safe arity \( s \). If not otherwise stated, \( t^m = t_1, \ldots, t_m \) will always denote sequences of \( m \geq 0 \) terms of \( \text{SRN} \). Moreover, we write \( |t^m| \leq l \), 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 \( \overrightarrow{x} = x_1, \ldots, x_n \), \( \overrightarrow{y} = y_1, \ldots, y_s \):

\[
|t(\overrightarrow{x}; \overrightarrow{y})| \leq p_t(|x_1|, \ldots, |x_n|) + \max \{|y_1|, \ldots, |y_s|\}
\]

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 \( \partial_h(t) \) 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 \( tb_t(l) = p_t(l, \ldots, l) + l \).

**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 \( \overrightarrow{x}, \overrightarrow{y} \) such that \( |\overrightarrow{x}|, |\overrightarrow{y}| \leq l \), we have \( |t(\overrightarrow{x}; \overrightarrow{y})| \leq tb_t(l) \).
\[x^A \in V \Rightarrow x \in A^A_V\]
\[m \geq 1, M \in \Lambda^A_V, x_i \in \Lambda^A_{V_i}, \ldots, x_m \in \Lambda^A_{V_m} \Rightarrow (\lambda \otimes^m_{i=1} x^A_i.M) \in \Lambda^A_{V_1 \otimes \cdots \otimes A_m \mapsto A}\]
\[M \in \Lambda^A_{V_i} \setminus N \in \Lambda^A_{V_i}, U \cup W \subseteq V, U \cap W = \emptyset \Rightarrow (MN) \in \Lambda^A_{V_i}\]
\[m \geq 2, \left(1 \leq i \neq j \leq m \Rightarrow N_i \in \Lambda^A_{V_j}\right), W_i \cap W_j = \emptyset, W_i, W_j \subseteq V \Rightarrow (\otimes^m_{i=1} N_i) \in \Lambda^A_{V_1 \otimes \cdots \otimes A_m}\]
\[M \in \Lambda^A_{V_i} \Rightarrow \Lambda \alpha.M \in \Lambda^A_{V \downarrow \alpha.A} \quad \text{(8)}\]
\[M \in \Lambda^A_{V \downarrow \alpha.A} \Rightarrow M\{z \downarrow \alpha\} \in \Lambda^A_{V_i}\]  

**Definition 7 (The Net Bounding Function nb.(·))** Let \( t \) in \( \text{SRN}^{n,s} \) and \( l \geq 0 \). We define \( \text{nb}(\cdot) \), that takes \( t \) and \( l \) as arguments, as \( \text{nb}_t(l) = \text{th}_1(t, \ldots, \text{th}_l(\ldots)) \), with \( \partial_t(l) + \partial_\ell(l) \) occurrences of \( \text{th}_t(\cdot) \).

**Fact 8 (nb.(·) is a Polynomial)** For every fixed \( t \) in \( \text{SRN}^{n,s} \), \( \text{nb}_t(l) \) is a polynomial in the free variable \( l \), whose degree is \( \partial_t(l) \).

## 5 Preliminary useful nets in LALL

We introduce a first set of nets useful to define the embedding from \( \text{SRN} \) to \( \text{LALL} \). However, whenever neither boxes, nor contractions are used in a given net \( I \), whose conclusion has type \( A \), to save space, we represent \( I \) by means of a \( \lambda \)-term. The term belongs to the set \( \Lambda^A_V \) of polymorphic typed affine \( \lambda \)-terms with variables in \( V \), patterns, tuples, and type \( A \in V \). Figure 5 defines \( \Lambda^A_V \). (6) introduces \( \lambda \)-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 \( \Lambda^A_V \) rewrite under the standard \( \beta \)-reduction, extended with the following two rules: (i) \( (\lambda \otimes^m_{i=1} x_i.M)(\otimes^m_{i=1} N_i) \mapsto_\beta M[N_i/x_i \ldots N_m/x_m] \), where \( M[N_i/x_i \ldots N_m/x_m] \) stands for the simultaneous substitution of \( N_i \) for \( x_i \), with \( 1 \leq i \leq m \), and (ii) \( (\Lambda \alpha.M)\{B\} \mapsto_\beta M[B/\alpha] \).

**Booleans.** The type of booleans is \( \mathbb{B} = \forall \gamma.\gamma \rightarrow \gamma \rightarrow \gamma \) whose representative nets are:
\[
\begin{align*}
F &= \Lambda \gamma.\lambda x.\gamma x.\gamma \triangleright \mathbb{B} \\
T &= \Lambda \gamma.\lambda x.\gamma x.\gamma \triangleright \mathbb{B} \\
\forall \mathbb{B}[b] &= b \{ \mathbb{B} \otimes \mathbb{B} \} \{ T \otimes T \} \{ F \otimes F \} \triangleright \mathbb{B} \triangleright \mathbb{B} \otimes \mathbb{B}
\end{align*}
\]

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

**Church words or, simply, words.** The type of words is \( \mathcal{C} = \forall \alpha.!(\alpha \rightarrow \alpha) \rightarrow !(\alpha \rightarrow \alpha) \rightarrow \exists \alpha (\alpha \rightarrow \alpha) \). 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 \( \epsilon \mathcal{C} \). If \( w \) is a natural number in binary notation, \( \overline{w} \) is its usual representation by a net. Figures 6(b) and 6(d) introduce nets that invert the bits inside any \( \overline{w} \), plugged by Cut into the dangling input of \( \text{RevC}[w] \).

**Scott words.** Intuitively, the type \( \mathcal{F} \) of Scott words describes a tuple of booleans. On Scott words
Moreover, a Scott word is in fact a stack of bits, the least significant bit being on the top of the stack. We have the following nets:

\[
\begin{align*}
\epsilon \mathcal{S} &= \Lambda \alpha. \lambda x^\alpha. y^\beta \leftarrow \alpha. x \triangleright \not\mathcal{J} \\
\text{Succ}_0[s] &= \Lambda \alpha. \lambda x^\alpha. y^\beta \leftarrow \alpha. yFs \triangleright \not\mathcal{J} \triangleright \not\mathcal{J} \\
\text{Succ}_1[s] &= \Lambda \alpha. \lambda x^\alpha. y^\beta \leftarrow \alpha. yTs \triangleright \not\mathcal{J} \triangleright \not\mathcal{J} \\
\text{Pred}[s] &= s \{ \not\mathcal{J} \} \epsilon \mathcal{S} (\lambda b^B. b \{ \not\mathcal{J} \} yx) \triangleright \not\mathcal{J}, \not\mathcal{J} \triangleright \not\mathcal{J} \\
\text{Cond}[s, x, y] &= \text{Prep}[s] \{ \not\mathcal{J} \} \epsilon \mathcal{S} (\lambda b^B. b \{ \not\mathcal{J} \} yx) \triangleright \not\mathcal{J}, \not\mathcal{J} \triangleright \not\mathcal{J} \\
\text{Prep}[s] &= s \{ \not\mathcal{J} \} \text{Succ}_0[\epsilon \mathcal{S}] (\lambda x^\not\mathcal{J}. \text{Succ}_0[\epsilon \mathcal{S}]) (\lambda x^\not\mathcal{J}. \text{Succ}_1[\epsilon \mathcal{S}]) w \triangleright \not\mathcal{J} \triangleright \not\mathcal{J}
\end{align*}
\]

We remark that \(\text{Succ}_0[s]\) adds to \(s\) the least significant bit \(T\), which stands for the digit 0, and \(\text{Succ}_1[s]\) adds \(F\), instead, which stands for 1. \(\text{Pred}[s]\) 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, \(\text{Cond}[s, x, y]\) branches a computation, depending on the value of \(s\). It yields \(x\) if the least significant bit of \(s\) is 0, or if \(s = \epsilon \mathcal{S}\), while it yields \(y\) if the least significant bit of \(s\) is 1. The preprocessing avoids to return \(\epsilon \mathcal{S}\): if \(s = \epsilon \mathcal{S}\), then \(s\) becomes \(\text{Succ}_0[\epsilon \mathcal{S}]\). Also, the three assumptions of type \(\not\mathcal{J}\) in \(\not\mathcal{J}, \not\mathcal{J}, \not\mathcal{J} \triangleright \not\mathcal{J}\) specify the type of \(s, x, \) and \(y\), respectively.
Fact 10 (Relation between naturals, Scott words, and words-as-terms) Every sequence \((d_1, \ldots, d_l)\) with \(d_1, \ldots, d_l \in \{0,1\}\) and \(l \geq 0\), identifies uniquely a number \(n = 2^{l-1} \cdot d_1 + \cdots + 2^0 \cdot d_l \in \mathbb{N}\). So, both the term of \(\text{SRN} \ s_{d_1} ; \ldots ; s_{d_l} ; (0) \ldots\) and the Scott word \([n]\) identify \(n\), too. We say that the sequence, as well as the Scott number and the \(\text{SRN}\) term, represent the same \(n\).

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

Scott words to words. For any \(l \geq 0\), \(\text{StoC}_l [s] \supset \mathcal{I} \vdash \varepsilon \) is inductively defined on \(l\):

\[
\begin{align*}
\text{StoC}_0 [s] &= \varepsilon \mathcal{C} \\
\text{StoC}_l [s] &= s \{ \varepsilon \} \mathcal{C} (\lambda x^2 \cdot x . (\lambda z^\mathcal{I} . \text{SuccC}_0 [z] ) (\lambda z^\mathcal{I} . \text{SuccC}_1 [z] )) \text{StoC}_{l-1} [s]
\end{align*}
\]

The net \(\text{StoC}_l [s]\) normalizes to the word \(\overline{\varpi}\), for any Scott word at most as long as \(l\).

Duplicating Scott words. For any \(l \geq 0\), the net \(\text{VS}_l [s]\) is inductively defined on \(l\):

\[
\begin{align*}
\text{VS}_0 [s] &= \varepsilon \mathcal{S} \otimes \varepsilon \mathcal{S} \\
\text{VS}_l [s] &= s \{ \mathcal{I} \otimes \mathcal{I} \} (\varepsilon \mathcal{S} \otimes \varepsilon \mathcal{S} ) (\lambda b^\mathcal{I} \cdot b \{ \mathcal{I}^2 \rightarrow \mathcal{I}^3 \} (\lambda x^\mathcal{I} \cdot y^\mathcal{I} \cdot \text{SuccS}_0 [x] \otimes \text{SuccS}_1 [y] ) ) \\
&\quad (\lambda x^\mathcal{I} \cdot y^\mathcal{I} \cdot \text{SuccS}_1 [x] \otimes \text{SuccS}_1 [y] ) \text{VS}_{l-1} [s]
\end{align*}
\]

such that \(\text{VS}_l [s] \supset \mathcal{I} \vdash \mathcal{I}^2\), where \(\mathcal{I}^2 = \mathcal{I} \otimes \mathcal{I}\). The net \(\text{VS}_l [s]\) builds two copies of any Scott word at most as long as \(l\). The generalization \(\text{VS}^*_l [s] \supset \mathcal{I} \vdash \mathcal{I}^k \) of \(\text{VS}_l [s]\) duplicates a given Scott word at most as long as \(l\) which lies inside \(k \geq 0\) paragraph boxes. Specifically, \(\text{VS}^0 [s]\) is \(\text{VS}_l [s]\), while \(\text{VS}^*_l [s]\) is in Figure 7 with \(k > 0\), which is the only net that exploits the fuzzy borders of paragraph boxes. By induction on \(k\), \(|\text{VS}^*_l [s]| = 19 + 89l + 3k \in O(l)\).

Coercing Scott words. For any \(k, l \geq 0\), we define \(\text{CoerS}^*_l [s] \supset \mathcal{I} \vdash \mathcal{I}^k \mathcal{S}\) by cases on \(k\), and by induction on \(l\). If \(k = 0\), then \(\text{CoerS}^0 [s]\) is the node Axiom. Otherwise, the net is in Figure 8 where, for \(i \in \{0,1\}\), \(\lambda s . \mathcal{I}^k (\text{SuccS}_i [s] ) \supset \mathcal{I}^k \rightarrow \mathcal{I}^k \mathcal{S}\) denotes the net that we build by: (i) enclosing \(\text{SuccS}_i [s]\) into \(k\) paragraph boxes to get \(\mathcal{I}^k (\text{SuccS}_i [s] ) \supset \mathcal{I}^k \rightarrow \mathcal{I}^k \Rightarrow\), and (ii) adding an Implication \(\mathcal{R}\) to \(\mathcal{I}^k (\text{SuccS}_i [s] )\) so to close it and get its type \(\mathcal{I}^k \rightarrow \mathcal{I}^k \mathcal{S}\). The net \(\text{CoerS}^*_l [s]\) reconstructs a given Scott word at most as long as \(l\) into an identical Scott word inside \(k\) paragraph boxes. We can show that \(|\text{CoerS}^*_l [s]| = 43l + 3kl \in O(l)\).

Lifting. Let \(\Pi \supset \mathcal{I}^n, \mathcal{I}^s \vdash \mathcal{I}^k \mathcal{S}\) for some \(n, s, k \geq 0\). For every \(k' \geq 0\) we can build \(\text{Lift}_{k'} [\Pi] \supset \mathcal{I}^n, \mathcal{I}^{s+k'} \vdash \mathcal{I}^{s+k'} \mathcal{S}\) by: (i) enclosing \(\Pi\) into \(k'\) paragraph boxes, getting \(\Pi'\), and (ii) plugging the...
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 $\llbracket - \rrbracket$ takes a term $t$ of SRN$^{n,s}$ as first and $l \geq 0$ as second argument, and yields a net $[t]^l \triangleright \rhd^\to_n l^k \ni \vdash \impliedby s^k$, for some $k$. We define the map inductively on the first argument.

The base cases of $\llbracket - \rrbracket$. Some of them are straightforward:

- $\llbracket [0] \rrbracket = \epsilon \ni \vdash \impliedby [s_0] = \text{SuccS}[s] \ni \vdash \impliedby \quad (i \in \{0,1\})$
- $\llbracket [p] \rrbracket = \text{PredS}[s] \ni \vdash \impliedby \quad [B] = \text{CondS}[s,x,y] \ni \vdash \impliedby , \impliedby , \impliedby$
where, $s, x, y$ denote the inputs of the nets they appear into. Concerning the projection, $[\pi_i^{ms}]^l$ 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:

$$[\pi_i^{ms}]^l \vdash \varphi, s_i^k, \varphi \vdash s_i^k \varphi$$

$$[v_i] \vdash \varphi, s^k \varphi \vdash s^k \varphi \quad \text{for composition, we set} \quad n \vdash \varphi \vdash s^k \varphi$$

By letting $\kappa = \max\{k', k_0, k_1, k_2\}$, we get:

$$\text{Lift}_{(k-k')}[[r']^{zh}(l)] \oplus \varphi, s^{k'} \varphi \vdash s^{k'} \varphi$$

$$\text{Lift}_{(k-k')}[[v_i]^{zh}] \oplus \varphi, s^{k} \varphi \vdash s^{k} \varphi$$

By letting $k = \max\{k', k_0, k_1, k_2\}$, we get:

$$\text{Lift}_{(k-k')}[[r']^{zh}(l)] \oplus \varphi, s^{k'} \varphi \vdash s^{k'} \varphi$$

$$\text{Lift}_{(k-k')}[[v_i]^{zh}] \oplus \varphi, s^{k} \varphi \vdash s^{k} \varphi$$

By letting $k = \max\{k', k_0, k_1, k_2\}$, we get:

$$\text{Lift}_{(k-k')}[[r']^{zh}(l)] \oplus \varphi, s^{k'} \varphi \vdash s^{k'} \varphi$$

$$\text{Lift}_{(k-k')}[[v_i]^{zh}] \oplus \varphi, s^{k} \varphi \vdash s^{k} \varphi$$

The case of $[\cdot]$ on the composition. We now focus on $t = o[t', u_1, \ldots, u_m, v_1, \ldots, v_l]$ such that $t'$ be in $\text{SRN}^{m,s}$. Without loss of generality, we show how to build $[t']^l$ by assuming $m = n = s = 1$, and $r = 2$. By induction we have:

$$[t']^{zh}(l) \oplus \varphi, s^{k'} \varphi \vdash s^{k'} \varphi$$

$$[v_i]^{zh} \oplus \varphi, s^{k} \varphi \vdash s^{k} \varphi$$

$$[u_i]^{zh} \oplus \varphi, s^{k} \varphi \vdash s^{k} \varphi \quad (i \in \{1, 2\})$$

Next, if we build $\Pi'$ in Figure 10 then $[t']^l$ is $\nabla_1^{2k}[\nabla_1^0[\nabla_1^0[\Pi']]]$. The two occurrences of $\nabla_1^0$ contract three “normal” inputs. One is from $[u_1]^{zh}$. The other two are from $[v_1]^{zh}$, $[v_2]^{zh}$. The occurrence of $\nabla_1^{2k}$ contracts the single “safe” input $[v_1]^{zh}$ and $[v_2]^{zh}$. We insist remarking the existence of $[t']^l$ for any $m, n, r, s$. One can count: $|[t']^l| \leq |\text{Lift}_{(k-k')}[[r']^{zh}(l)]| + \sum_i m \cdot |\text{Lift}_{(k-k')}[[u_i]^{zh}]| + \sum_j n \cdot |\text{Lift}_{(k-k')}[[v_j]^{zh}]| + k(1 + n + s' + s'') + s'n|\text{CoerS}^{zh}[s]|$. So, it follows $|[t']^l| \in O((|r'|^{zh}(l)) + \sum_i m \cdot |u_i|^l) + \sum_j n \cdot |v_j|^l + k(l + l)).$

The case of $[\cdot]$ on the recursion. Let $t = \tau[u_e, u_0, u_1]$ with $u_e \in \text{SRN}^{m,s}, u_0, u_1 \in \text{SRN}^{n+1:s+1}$. 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:

$$[u_e]^{zh} \oplus \varphi, s^{k} \varphi \vdash s^{k} \varphi$$

$$[u_i]^{zh} \oplus \varphi, s^{k} \varphi \vdash s^{k} \varphi \quad (i \in \{0, 1\})$$

By letting $k = \max\{k_0, k_1, k_2\}$, and using Lift $[\cdot]$ in analogy to the translation of the composition, $[t']^l$ is in Figure II. The Scott word that drives the recursion unfolding, becomes a word, and, then, it is necessary to reverse it by $\text{RevC}$. Otherwise we would unfold the iteration according to a wrong bit order, as implied by Remark 9. Moreover, (i) $\Pi$ 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 \( \bigotimes \) and \( \bigotimes \) are obvious trees of Tensor R and L nodes. Finally, we can prove \(| |t| | \in O \left( | |u_2| | + | |u_0| |d_0(l)| + | |v_1| |d_1(l)| + t_b(l) \right)\).

**Definition 11 (Representing a term by a net)** Let \( t \) be in \( \text{SRN}^{n,s} \), \( l \in \mathbb{N} \), and \( \Pi \triangleright \mathcal{T}^n \left( \frac{s}{k} \right)^k \triangleright \mathcal{T}^s \), for some \( k \in \mathbb{N} \). Then, \( \Pi \) \( k \)-simulates \( t \) with \( l \)-bounded inputs if, for every pair of vectors of natural numbers \( \overline{x^n}, \overline{y^s} \), such that \( \overline{x^n}, \overline{y^s} \leq l \), the net we get by plugging \( [x_1]^l, \ldots, [x_n]^l, [y_1]^l, \ldots, [y_s]^l \) into the inputs of \( \Pi \), in the natural way, normalizes to \( \frac{s}{k} \left( \overline{x^n} ; \overline{y^s} \right) \), whenever \( z \) is the result of \( t \left( \overline{x^n} ; \overline{y^s} \right) \).

**Proposition 12 (SRN embeds into LALL)** Let \( l \geq 0 \), and \( t \in \text{SRN}^{n,s} \). Then, \( \frac{t}{l} \) \( k \)-simulates \( t \) with \( l \)-bounded inputs. Moreover, (i) \( k \leq \partial_R(t) \cdot 2^\partial_L(t) \), and (ii) \( | |t| | \in O \left( \left( \frac{s}{k} \right)^k \left( \overline{y^s} \right)^l \right) \), namely a polynomial in \( l \).

The statement holds by induction on \( t \), using the definition of \( [\cdot] \), the size of every net that \( [\cdot] \) generates, the definitions of \( nb_s(\cdot) \), and \( tb_t(\cdot) \), 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 \( \text{SRN} \) encodes, and the proof theoretical stratification, that regulates the complexity of some Light Logic that characterize \( \text{PTIME} \), can be improved, provided that (i) the stratification we find in Light Linear Logic and \( \text{ILAL} \) relaxes to boxes with “fuzzy” border, as in \( \text{ML}^\delta \) or \( \text{LALL} \), and (ii) we move to a representation of words alternative to the standard one, able both to represent the basic functions of \( \text{SRN} \) 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 \( \textbf{SRN} \) maps to a family \( \langle [t]^l \rangle_{l \in \mathbb{N}} \) of nets in \( \textbf{LALL} \), where \([t]^l\) simulates \( t \) with inputs at 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 \( \textbf{SRN} \) but not in \( \textbf{BC}^- \):

\[
\begin{align*}
g(0; y, z) &= z \\
g(s_i w; y, z) &= h(w; y, z, g(w; y, z)) ,
\end{align*}
\]

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

Admittedly, the representation of a term of \( \textbf{SRN} \) 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 \( \textbf{LALL} \) represents every function with finite domain. Beware, however, that not every algorithm is in \( \textbf{LALL} \). For example, in analogy with \cite{7}, we show an algorithm \( \text{exp} \) that cannot exist as a net of \( \textbf{LALL} \) because it calculates a non-polynomial time function. \( \text{exp} \) will be defined using the traditional non-predicative recursive schemes, so that it is not a program of \( \textbf{SRN} \). \( \text{exp} \) is defined as follows. We know that the two programs \( \text{concat}(x; y) \), which concatenates two strings of bits, and \( \text{double}(x) = \text{concat}(x; \pi_1^0(x)) \) belong to \( \textbf{SRN} \). Then, we can define the recursive function \( \text{exp} \):

\[
\text{exp}(0; \gamma, \delta) = \pi_1^1(0; \gamma, \delta) \quad \text{exp}(s_i(x; \gamma, \delta)) = \text{double}(\text{exp}(x; \gamma, \delta)) \quad (i \in \{0, 1\})
\]

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

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

\[
\begin{align*}
\text{gt}(0, y) &= \text{False} \\
\text{gt}(s_i x, y) &= \text{if} (y = 0) \text{ then } \text{True} \text{ else } \text{gt}(x, \text{pred}(y)).
\end{align*}
\]

The program \( \text{gt} \) 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 \( \text{gt} \) applies a function on the parameter \( y \) that does not drive the unfolding. Namely, \( \text{gt} \) incorporates a double iteration. Certainly, \( \text{gt} \) cannot exist in \( \textbf{SRN} \) in the form here above. Instead, \( \textbf{LALL} \) admits to represent \( \text{gt} \) as follows:

\[
\begin{align*}
\text{gt}_{\mathcal{F} \rightarrow \mathcal{F}}^0 &= \lambda x \mathcal{F}. \lambda y \mathcal{F}. \pi_2(x \{\mathcal{F} \otimes \mathcal{B}\}(\text{step})(y \otimes \mathcal{T})) \\
\text{step}_{\mathcal{F} \otimes \mathcal{B} \rightarrow \mathcal{F} \otimes \mathcal{B}} &= \lambda s \mathcal{F}. b. \mathcal{B}(s(0 \otimes \mathcal{T})(\lambda x \mathcal{B}. \lambda y \mathcal{F}. y \otimes \mathcal{T}))
\end{align*}
\]

where \( \mathcal{B}, \mathcal{T}, \mathcal{F} \) are at page \texttt{69} The existence of \( \text{gt} \) in \( \textbf{LALL} \) implies the existence of a family \( \langle \text{ord}_l \rangle_{l \in \mathbb{N}} \) of nets. Every \( \text{ord}_l \) takes two Scott words with at most \( l \) bits, and gives them back sorted according to their length. The definition of every \( \text{ord}_l \) is a net of \( \textbf{LALL} \) that we compactly write as a \( \lambda \)-term:

\[
\begin{align*}
\text{ord}_l^0 \mathcal{F} \rightarrow \mathcal{F} \rightarrow \mathcal{F} &= \lambda x^l \mathcal{F}. y^l. (\lambda x_1^l \mathcal{F}. x_2^l \mathcal{F}. (\lambda y_1^l \mathcal{F}. y_2^l \mathcal{F}. \text{BtoB}(\text{gt} (\text{Stoc}l(0))))(y_2 \otimes x_2))((\text{S}l(x))(\text{S}l(y))) \\
\text{BtoB}^0 \mathcal{B} &= \lambda x \mathcal{B}. (\lambda y. (\lambda w^l \mathcal{F}. z^l \mathcal{F}. (\lambda z \mathcal{F}. w^l \mathcal{F}. z \mathcal{F})) (\lambda y. (\lambda w^l \mathcal{F}. z^l \mathcal{F}. z \mathcal{F})))
\end{align*}
\]
Safe Recursion on Notation into a Light Logic

\[ L(X) = \forall \alpha.(X \rightarrow \alpha \rightarrow \alpha) \rightarrow \text{§}(\alpha \rightarrow \alpha) \] is the type of lists,

\[
\varrho^{L(X)} = \lambda c^{(X \rightarrow \alpha \rightarrow \alpha)} \cdot \lambda z^\alpha \cdot z
\]

\[
\text{sort}_{\lambda}^{L(S) \rightarrow \text{§}L(S)} = \lambda t^{L(S)}. (\text{insert}_{\lambda})(\varrho^{L(S)})
\]

\[
\text{insert}_{\lambda}^{S \rightarrow L(S) \rightarrow L(S)} = \lambda n^{(S \rightarrow \alpha \rightarrow \alpha)}. \lambda c^{(S \rightarrow \alpha \rightarrow \alpha)}. \lambda x^{\alpha}. (\lambda y^\alpha. c x y)((t \text{putTop}[\lambda][c])(\text{ord} l a \otimes b))
\]

\[
\text{putTop}[\lambda][c]^{S \rightarrow \alpha \otimes \alpha \otimes \alpha} = \lambda a^\alpha \cdot \lambda b^\alpha \otimes t^\alpha. (\lambda i^\alpha \otimes \lambda j^\alpha \otimes i \otimes c j t)(\text{ord} l a \otimes b)
\]

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

where \( B = \forall \gamma.(\gamma \otimes \gamma) \rightarrow (\gamma \otimes \gamma) \) is a linear version of the booleans, \( \text{StoC}_{\lambda}^{\gamma \rightarrow \gamma} \) is at page 71 and the safe duplication \( \nabla_{\lambda}^{\gamma \rightarrow \gamma \otimes \gamma} \) is at page 71.

Given \( \text{ord} l \), 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 \( \text{LALL} \) they represent should be a reasonably simple exercise. We observe that \( \text{putTop}[\lambda] \) is a linear algorithm that manipulates only the head of a given list. Instead, \( \text{insert}_{\lambda} \) 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, \( \text{insert}_{\lambda} \) does not add any paragraph § 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 \( \lambda \)-calculus [9]. Finally, \( \text{sort}_{\lambda} \), iterates \( \text{insert}_{\lambda} \) in the usual way, thus adding a § in front of its output type.

Future lines of research. We must say that the representation of \( \text{SRN} \) as a family of nets of \( \text{LALL} \) that we present in this work has been an alternative approach to the standard one, which would explore the relations between \( \text{SRN} \) and stratification by mapping a single term of \( \text{SRN} \) into a single net. That standard approach has been developed in [11, 10, 13]. Those works make some progress as compared to [8]. This means that they identify a Light Logic that strictly contains \( \text{ILAL} \), and which allows to represent a strict extension of \( \text{BC}^- \). However, the whole \( \text{SRN} \) 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 \( \text{MS} \) developed in the previously cited works [11, 10, 13]. The conjecture is that the two technologies together may lead to a more refined proof theoretical representation of the principles underpinning the definition of \( \text{SRN} \), and of the predicative analysis it encodes, possibly increasing the set of algorithms that we can represent inside Light Logics.

References


