Hierarchical Digging is EXPTIME

Brian F. Redmond
Department of Computing, Mathematics and Statistical Sciences
Grande Prairie Regional College
Alberta, Canada
bredmond@gprc.ab.ca

We show that Lafont's system of Soft Linear Logic together with a hierarchical version of digging captures the complexity class EXPTIME.

1 Introduction

Lafont's Soft Linear Logic (SLL_2) is a system of second-order intuitionistic linear logic with modified exponentials that captures polynomial time computations [1]. In other words, any SLL_2 proof net reduces to a unique normal form in a polynomial number of steps, and conversely, any polynomial time Turing machine algorithm can be encoded in the system.

The exponential rules in SLL_2 have a particularly simple form: soft promotion (i.e. monoidal-functoriality) and multiplexing of rank n, $n \in \mathbb{N}$. These rules enforce a predicative control on the duplication of resources, and in fact the polynomial time bound on reduction is easy to prove. However, programming in SLL_2 is more of a challenge. SLL_2 has a strict programming discipline whereby programs are required to be *generic* (i.e. do not use multiplexing) and data is required to be *homogeneous* (i.e. all multiplexors have the same rank). In this paper, we shall maintain the genericity of programs but we shall relax slightly the homogeneity restriction on data.

In the conclusion of Lafont's paper, it was suggested that the introduction of a hierarchy of modalities $!_0, !_1, !_2, \ldots$ with a hierarchical version of digging would lead to a system for elementary recursive computation. The purpose of this note is to investigate this claim.

2 Soft Linear Logic with Hierarchical Digging

The setting for this paper is Lafont's Soft Linear Logic [1] extended with hierarchical digging. Formulas are given by the following grammar:

$$A,B,C \quad ::= \quad \alpha \mid A \,\&\, A \mid \mathbf{1} \mid A \otimes A \mid A \multimap A \mid !_s A \mid \forall \alpha . A$$

where α is an atomic type variable and $s \in \mathbb{N}$. We shall use the abbreviation $!_s^n$ to denote the iteration $!_s \cdots !_s$ ($n \geq 1$ times). We also introduce the notation $!_{s,r} = !_s !_{s-1} \cdots !_r$ whenever $s > r$. Finally, the notation $A^{(n)}$ is short for $A, \ldots, A$ ($n \geq 0$ times). Sequents are written intuitionistically in the form $\Gamma \vdash A$ where Γ is a finite (possibly empty) list of formulas and A is a single formula. Also, $!_s \Gamma = !_s A_1, \ldots, !_s A_k$ if $\Gamma = A_1, \ldots, A_k$. The rules are as follows:

- Identity, exchange and cut:

$$\frac{}{A \vdash A}\ (id) \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, B, A \vdash C}\ (ex) \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}\ (cut)$$

Submitted to:
DICE-FOPARA 2019

- Multiplicatives:

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \frac{}{\vdash \mathbf{1}} \qquad \frac{\Gamma \vdash C}{\Gamma, \mathbf{1} \vdash C}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}$$

- Additives:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B} \qquad \frac{\Gamma, A \vdash C}{\Gamma, A \,\&\, B \vdash C} \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \,\&\, B \vdash C}$$

- Exponential rules *soft promotion* (at level s), *multiplexing* (of rank $n \geq 0$), and *digging* (of rank $n \geq 1$ and level $s+1$):

$$\frac{\Gamma \vdash A}{!_s \Gamma \vdash !_s A} \qquad \frac{\Gamma, A^{(n)} \vdash C}{\Gamma, !_0 A \vdash C} \qquad \frac{\Gamma, !_s^n A \vdash C}{\Gamma, !_{s+1} A \vdash C}$$

- Quantification rules:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha . A} \; \alpha \notin FTV(\Gamma) \qquad \frac{\Gamma, A[B/\alpha] \vdash C}{\Gamma, \forall \alpha . A \vdash C}$$

Proofs in this paper are presented using the sequent calculus formulation above, but are implicitly identified with their corresponding proof nets as in [1]. Furthermore, only *external* reduction is assumed. Proof nets can be interpreted in second-order lambda calculus with explicit pairs. It is straightforward to show that SLL_2 with hierarchical digging satisfies cut-elimination.

Theorem 1. *SLL_2 with hierarchical digging satisfies cut-elimination.*

Proof. The three non-standard reduction steps in the cut-elimination procedure are as follows. The first two already appear in SLL_2.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A}}{!_s\Gamma \vdash !_s A} \quad \dfrac{\dfrac{\vdots}{\Delta, A \vdash B}}{!_s\Delta, !_s A \vdash !_s B}}{!_s\Gamma, !_s\Delta \vdash !_s B} \quad \rightsquigarrow \quad \dfrac{\dfrac{\vdots}{\Gamma \vdash A} \quad \dfrac{\vdots}{\Delta, A \vdash B}}{\dfrac{\Gamma, \Delta \vdash B}{!_s\Gamma, !_s\Delta \vdash !_s B}}$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A}}{!_0\Gamma \vdash !_0 A} \quad \dfrac{\dfrac{\vdots}{\Delta, A^{(n)} \vdash B}}{\Delta, !_0 A \vdash B}}{!_0\Gamma, \Delta \vdash B} \quad \rightsquigarrow \quad \dfrac{\dfrac{\vdots}{\Gamma \vdash A} \;\cdots\; \dfrac{\vdots}{\Gamma \vdash A} \quad \dfrac{\vdots}{\Delta, A^{(n)} \vdash B}}{\dfrac{\Gamma^{(n)}, \Delta \vdash B}{!_0\Gamma, \Delta \vdash B}}$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A}}{!_{s+1}\Gamma \vdash !_{s+1} A} \quad \dfrac{\dfrac{\vdots}{\Delta, !_s^n A \vdash B}}{\Delta, !_{s+1} A \vdash B}}{!_{s+1}\Gamma, \Delta \vdash B} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A}}{!_s^n\Gamma \vdash !_s^n A} \quad \dfrac{\vdots}{\Delta, !_s^n A \vdash B}}{\dfrac{!_s^n\Gamma, \Delta \vdash B}{!_{s+1}\Gamma, \Delta \vdash B}}$$

In the last reduction, note that soft promotion is applied successively $n \geq 1$ times and digging is used for each of the formulas in $!_s^n\Gamma$. □

The following are the important measures on proof nets in this setting.

- The **level** of a proof is the maximum level of digging or soft promotion that appears in the proof, and 0 otherwise. A level 0 proof is just an SLL_2 proof.
- The **degree** of a proof is the maximum nesting of exponential boxes (of any type).
- If u is a proof of level s, let n_0 denote to maximum rank of multiplexing in the proof, and for each $i \in \{1,\ldots,s\}$, let n_i denote the maximum rank of digging at level i, and 0 otherwise. The **rank** of a proof is then defined to be $n = \max\{n_0,\ldots,n_s,1\}$. A proof is called **homogeneous** if all multiplexing and digging rules have the same rank. If a proof does not contain any multiplexing or digging rules, then it is called **generic**.
- The **weight** of a proof u at level s is an exponential expression $W_u(X,Y_1,\ldots,Y_s)$ defined inductively as follows. The weight of an atomic cell is constant 1 if it corresponds to a right logical rule and constant 0 if it corresponds to a left logical rule. The weight of a box is given by the following formulas, where $s \geq 0$:

$$W_{u\&v} = W_u + W_v + 1 \qquad W_{\forall\alpha.u} = W_u + 1$$

$$W_{!_s u} = X^{Y_1\cdots Y_s} W_u + (s+1)\sum_{i=0}^{Y_1\cdots Y_s-1} X^i$$

Note that if $s = 0$, this gives $W_{!_0} = XW_u + 1$ as expected. The following theorem gives the complexity of (external) reduction.

Theorem 2. *A proof net u of level s and rank n reduces to a unique normal form in at most $W(n,\ldots,n)$ steps.*

Proof. We shall omit the first two cases above as they are the same as in SLL_2. We shall prove the third by comparing weights before and after reduction.

$$\begin{aligned}
W_{!_s^n u} + W_v &= X^{Y_1\cdots Y_s} W_{!_s^{n-1} u} + (s+1)\sum_{i=0}^{Y_1\cdots Y_s-1} X^i + W_v \\
&= X^{Y_1\cdots Y_s}\left(X^{Y_1\cdots Y_s} W_{!_s^{n-2} u} + (s+1)\sum_{i=0}^{Y_1\cdots Y_s-1} X^i\right) + (s+1)\sum_{i=0}^{Y_1\cdots Y_s-1} X^i + W_v \\
&= X^{Y_1\cdots Y_s 2} W_{!_s^{n-2} u} + (s+1)\sum_{i=Y_1\cdots Y_s}^{Y_1\cdots Y_s 2-1} X^i + (s+1)\sum_{i=0}^{Y_1\cdots Y_s-1} X^i + W_v \\
&= X^{Y_1\cdots Y_s 2} W_{!_s^{n-2} u} + (s+1)\sum_{i=0}^{Y_1\cdots Y_s 2-1} X^i + W_v \\
&\vdots \\
&= X^{Y_1\cdots Y_s n} W_u + (s+1)\sum_{i=0}^{Y_1\cdots Y_s n-1} X^i + W_v \\
&< X^{Y_1\cdots Y_s Y_{s+1}} W_u + (s+2)\sum_{i=0}^{Y_1\cdots Y_s Y_{s+1}-1} X^i + W_v \\
&= W_{!_{s+1} u} + W_v
\end{aligned}$$

□

By induction on the structure of u, it is not difficult to show that if u is a proof net of degree d and level s, then $W_u(n,\ldots,n) \leq kn^{dn^s}$ where $k = W_u(1,\ldots,1)$. Hence, in general, reduction requires more than exponential time. As we shall see in the next section, however, the level s of data remains fixed, and so reduction is in exponential time as expected.

3 Encoding Exponential Time

Data types in this system have a somewhat peculiar form. For example, the data type(s) of natural numbers is defined as follows:

$$\begin{array}{rcl} \mathbf{N}_s & = & \forall\alpha.!_s(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha \\ \mathbf{N}_{s+1/2} & = & \forall\alpha.!_{s+1}(!_s\alpha \multimap \alpha) \multimap !_{s+1}\alpha \multimap \alpha \end{array}$$

where $s \in \mathbb{N}$. Nets of the above types translate to the usual Church numerals. For example, the following is a level 1 non-homogeneous[1] proof for the number 3, which has type $\mathbf{N}_{1/2}$, rank 3, and degree 3:

$$\begin{array}{c}
\hline
\alpha \vdash \alpha \\ \hline
!_0\alpha \vdash !_0\alpha \qquad \overline{\alpha \vdash \alpha} \\ \hline
!_0\alpha, !_0\alpha \multimap \alpha \vdash \alpha \\ \hline
!_0!_0\alpha, !_0(!_0\alpha \multimap \alpha) \vdash !_0\alpha \qquad \overline{\alpha \vdash \alpha} \\ \hline
!_0!_0\alpha, !_0(!_0\alpha \multimap \alpha), !_0\alpha \multimap \alpha \vdash \alpha \\ \hline
!_0!_0!_0\alpha, !_0!_0(!_0\alpha \multimap \alpha), !_0(!_0\alpha \multimap \alpha) \vdash !_0\alpha \qquad \overline{\alpha \vdash \alpha} \\ \hline
!_0!_0!_0\alpha, !_0!_0(!_0\alpha \multimap \alpha), !_0(!_0\alpha \multimap \alpha), !_0\alpha \multimap \alpha \vdash \alpha \\ \hline
!_0!_0!_0\alpha, !_0!_0(!_0\alpha \multimap \alpha)^{(3)} \vdash \alpha \\ \hline
!_0!_0!_0\alpha, !_0!_0!_0(!_0\alpha \multimap \alpha) \vdash \alpha \\ \hline
!_1\alpha, !_1(!_0\alpha \multimap \alpha) \vdash \alpha \\ \hline
\vdash !_1(!_0\alpha \multimap \alpha) \multimap !_1\alpha \multimap \alpha \\ \hline
\vdash \forall\alpha.!_1(!_0\alpha \multimap \alpha) \multimap !_1\alpha \multimap \alpha
\end{array}$$

In this system there is a generic proof for the exponential function:

$$\begin{array}{c}
\vdots \\ \hline
l_0 : \mathbf{N}_0, \vdash l_0 : !_0A \multimap A \qquad \overline{y : !_1(\alpha \multimap \alpha) \vdash y : !_1A} \quad \overline{z : A \vdash z : \alpha \multimap \alpha} \\ \hline
l_0 : !_1\mathbf{N}_0 \vdash l_0 : !_1(!_0A \multimap A) \qquad y : !_1(\alpha \multimap \alpha), g : !_1A \multimap A \vdash gy : \alpha \multimap \alpha \\ \hline
y : !_1(\alpha \multimap \alpha), l_0 : !_1\mathbf{N}_0, l_1 : !_1(!_0A \multimap A) \multimap !_1A \multimap A \vdash l_1 l_0 y : \alpha \multimap \alpha \\ \hline
y : !_1(\alpha \multimap \alpha), l_0 : !_1\mathbf{N}_0, l_1 : \mathbf{N}_{1/2} \vdash l_1 l_0 y : \alpha \multimap \alpha \\ \hline
l_0 : !_1\mathbf{N}_0, l_1 : \mathbf{N}_{1/2} \vdash \lambda y.l_1 l_0 y : !_1(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha \\ \hline
l_0 : !_1\mathbf{N}_0, l_1 : \mathbf{N}_{1/2} \vdash \lambda y.l_1 l_0 y : \mathbf{N}_1
\end{array}$$

where $A = \alpha \multimap \alpha$. Indeed, up to η-equivalence this leads to the lambda term $\lambda l_0 l_1.l_1 l_0$. Also note that this is a generic proof! The above proof generalizes by induction on $s \geq 1$ to give a hierarchy of

[1] Homogeneity can be maintained for data at level 0, but it seems that higher level data generally require additional rank 1 instances of both multiplexing and digging.

exponential maps:

$$\begin{array}{rcl} l_0 : !_1\mathbf{N}_0, l_1 : \mathbf{N}_{1/2} & \vdash & l_1 l_0 : \mathbf{N}_1 \\ l_0 : !_2!_1\mathbf{N}_0, l_1 : !_2\mathbf{N}_{1/2}, l_2 : \mathbf{N}_{3/2} & \vdash & l_2(l_1 l_0) : \mathbf{N}_2 \\ l_0 : !_3!_2!_1\mathbf{N}_0, l_1 : !_3!_2\mathbf{N}_{1/2}, l_2 : !_3\mathbf{N}_{3/2}, l_3 : \mathbf{N}_{5/2} & \vdash & l_3(l_2(l_1 l_0)) : \mathbf{N}_3 \\ & & \vdots \end{array}$$

where terms are written up to η-equivalence for clarity. For each $s \geq 1$, these terms compute the exponential function $l_0^{l_1 l_2 \cdots l_s}$.

Other data types that are required in our encoding include binary strings, Turing machines (with k states) and booleans:

$$\begin{array}{rcl} \mathbf{S}_s & = & \forall\alpha.!_s((\alpha \multimap \alpha) \,\&\, (\alpha \multimap \alpha)) \multimap \alpha \multimap \alpha \\ \mathbf{S}_{s+1/2} & = & \forall\alpha.!_{s+1}((!_s\alpha \multimap \alpha) \,\&\, (!_s\alpha \multimap \alpha)) \multimap !_{s+1}\alpha \multimap \alpha \\ \mathbf{M}_s & = & \forall\alpha.!_s((\alpha \multimap \alpha) \,\&\, (\alpha \multimap \alpha)) \multimap \mathbf{F}_k \otimes \mathbf{F}_3 \otimes (\alpha \multimap \alpha)^2 \\ \mathbf{B} & = & \forall\alpha.(\alpha \,\&\, \alpha) \multimap \alpha \end{array}$$

where $s \in \mathbb{N}$ and $\mathbf{F}_k = \mathbf{1} \oplus \cdots \oplus \mathbf{1}$ (k times) is a finite type (see [1]).

Our encoding of exponential time algorithms is similar to Lafont's. We start by building a machine with a sufficiently long tape to handle the entire computation, and then iterate the transition function an exponential number of times. In the following theorem, we do not consider constant factors in our exponential bound for simplicity. However, a more precise statement can be given by following Lafont's accounting techniques (see Theorem 6 in [1]).

Theorem 3. *If a predicate on boolean strings is computable by a Turing machine in time and space bounded by n^{n^s} (for some fixed $s > 0$), then there is a generic proof for the sequent*

$$!_{s,0}\mathbf{S}_0^{(2)}, !_{s,1}\mathbf{S}_{1/2}^{(2)}, !_{s,2}\mathbf{S}_{3/2}^{(2)}, \ldots, \mathbf{S}_{s-1/2}^{(2)}, \mathbf{S}_0 \vdash \mathbf{B}$$

which corresponds to this predicate.

Proof. (Sketch) There is a generic proof for the sequent $!_{s,0}\mathbf{S}_0, !_{s,1}\mathbf{S}_{1/2}, !_{s,2}\mathbf{S}_{3/2}, \ldots, \mathbf{S}_{s-1/2} \vdash \mathbf{M}_s$ which constructs a Turing machine with tape of length n^{n^s}, filled with 0s, and with the head at the beginning of the tape. There is a generic proof for the sequent $\mathbf{S}_0, \mathbf{M}_s \vdash \mathbf{M}_s$ which writes a string on the tape of the machine (assuming the tape bounds the length of the string) and puts the machine in the starting configuration. There is a generic proof for the sequent $\mathbf{M}_s \vdash \mathbf{M}_s$ which encodes the transition function of the machine; it is identical to Lafont's, but lifted to level s. There is a generic proof for the sequent $\mathbf{N}_s, \mathbf{M}_s \vdash \mathbf{M}_s$ which iterates the transition function a given number of times on a given starting configuration. There is a generic proof for the sequent $!_{s,0}\mathbf{S}_0, !_{s,1}\mathbf{S}_{1/2}, !_{s,2}\mathbf{S}_{3/2}, \ldots, \mathbf{S}_{s-1/2} \vdash \mathbf{N}_s$ which constructs a natural number of size n^{n^s}, bounding the total number of steps in the computation. Finally, there is a generic proof for the sequent $\mathbf{M}_s \vdash \mathbf{B}$ which determines whether the machine is in a accepting state. Putting this all together, we get a generic sequent of the form $!_{s,0}\mathbf{S}_0^{(2)}, !_{s,1}\mathbf{S}_{1/2}^{(2)}, !_{s,2}\mathbf{S}_{3/2}^{(2)}, \ldots, \mathbf{S}_{s-1/2}^{(2)}, \mathbf{S}_0 \vdash \mathbf{B}$ corresponding to the predicate. □

4 Conclusion

In this brief note, we have introduced a new system of linear logic that captures EXPTIME, adding to the growing list of the so-called light linear logics. Although we did not prove it here, it is interesting to note that hierarchical digging can be used to give a compressed-rank representation of data, and that generic proofs for sequents of the form $\mathbf{S}_s^{(d)} \vdash \mathbf{B}$ correspond to polynomial time algorithms (for all $s \geq 0$). In the direction of increasing complexity on the other hand, it would be simple to add yet another modality that bounds the hierarchy $!_0, !_1, !_2, \ldots$. This would certainly increase the complexity of reduction. However, what is not clear (to the author at least) is how to encode boolean strings and natural numbers in such a system so that generic proofs correspond to higher complexity (i.e. elementary) algorithms as in Lafont's original conjecture.

References

[1] Yves Lafont (2004): *Soft linear logic and polynomial time*. *Theor. Comput. Sci.* 318(1-2), pp. 163–180, doi:10.1016/j.tcs.2003.10.018. Available at `http://dx.doi.org/10.1016/j.tcs.2003.10.018`.