Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Polynomial time in untyped elementary linear logic ☆

Olivier Laurent

Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

ARTICLE INFO

Article history:
Received 12 October 2018
Received in revised form 16 May 2019
Accepted 3 October 2019
Available online xxxx

Keywords:
Implicit computational complexity
Polynomial time computation
Elementary linear logic
Proof nets
Strong normalization

ABSTRACT

We show how to represent polynomial time computation in an untyped version of proof-nets for elementary linear logic. This follows previous work by P. Baillot but which was developed in a typed and affine setting. We describe how these two properties can be adapted.

1. Introduction

Implicit computational complexity tries to find structural characterizations of complexity classes in computational models where no explicit constraint on time or space is given. In this domain, the resource sensitivity of linear logic [1] has shown to be a good property for defining logical systems corresponding to different complexity classes. Let us mention as key examples: polynomial time (PTIME) [2,3], elementary time (ELEMENTARY) [2,4] or polynomial space (PSPACE) [5].

Given a complexity class $\mathcal{C}$, representing it in a logical system L is usually done in two steps. First we prove that normalization or cut-elimination in L can be done in $\mathcal{C}$, this is called *complexity soundness* of L with respect to $\mathcal{C}$. Second we encode decision problems of $\mathcal{C}$ into proofs in L, and we define an encoding of inputs into L in such a way that the normalization of (the representation of) the problem applied to / cut with (the representation of) some input allows us to get the output. This is called *complexity completeness* of L with respect to $\mathcal{C}$.

Among the variants of linear logic related to complexity classes, elementary linear logic (ELL [2]) is one of the most expressive since it is sound and complete for the class ELEMENTARY of elementary problems (time bounded by an arbitrary tower of exponentials). It is an easy-to-define restriction of linear logic which benefits from nice geometric properties like the depth-invariance property of the proof-net syntax called the *stratification property* [4]. Different denotational models have been described [6–8].

However ELEMENTARY can be considered as way too big to be meaningful from a computational point of view. Other logical systems are known to correspond to smaller classes, but one can also wonder whether it is possible to characterize these classes inside ELL. The goal is to take advantage of the good properties of ELL on the one side, and to describe a general framework for different complexity classes on the other side.

☆ This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007), and by the project Elica (ANR-14-CE25-0005), both operated by the French National Research Agency (ANR). This work was also supported by GDRI Linear Logic.

E-mail address: olivier.laurent@ens-lyon.fr.

https://doi.org/10.1016/j.tcs.2019.10.002
0304-3975/

Following a question of J.-Y. Girard asking whether a restriction on the types of elementary linear logic (ELL [2]) could represent exactly polynomial time computation (PTIME), P. Baillot [9] showed that proofs of $!\mathbb{W} \multimap !!\mathbb{B}_a$ (with $\mathbb{W} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$ encoding binary words, and $\mathbb{B}_a = \forall\alpha.\alpha \multimap \alpha \multimap \alpha$ an *affine* encoding of Booleans) in second-order intuitionistic elementary affine logic enriched with type fixpoints exactly represent PTIME decision problems. Rather than defining yet another PTIME encoding in a new logical system, the goal of this line of work is to stress that ELL is not only a very well-behaved logic but also a versatile and expressive system.

Our goal is to recast P. Baillot's work while dealing with an arguably simpler model of proof-nets. Indeed we prove a similar result as [9] in *untyped classical* elementary *linear* logic. This stresses the standard fact that types are useless for the control of complexity in elementary linear logic (in particular no need to deal with fixpoints of types nor second-order quantification in proof-nets). Moreover it shows that linear proofs are enough (the general weakening principle of affine logics is not required). Finally, once general weakening is removed, there is no reason to stick to an intuitionistic world and we can move to a classical one.

1.1. Design choices

We introduce a notion of untyped classical linear proof-nets for elementary linear logic as a computational model: e-nets, and we describe how it can be made sound and complete for PTIME (and then for the whole k-EXPTIME family). Let us discuss the pros and cons of the different choices to be made in the design of e-nets.

- Logical systems are traditionally described with sequential structures: sequent calculus or natural deduction. The introduction of proof-nets with linear logic offers a more parallel model in which cut-elimination is much easier to analyze. This is particularly true for complexity analyses. The only slight defect of proof-nets is to present proofs as graphs and cut-elimination as graph rewriting when most people are more comfortable with terms or trees, and term rewriting.
- In the context of elementary linear logic, complexity bounds are given by the stratification property, without relying on the involved formulas. It is thus folklore that untyped versions of ELL have the same complexity bounds. As a consequence an untyped version is interesting to consider since it is more expressive than typed versions. Any notion of recursive types, quantification, etc. can be represented by type-erasure into the untyped world. On the other side, types provide useful guidelines for building programs in an understandable way, and they also provide safeguards for execution: "well-typed programs can't go wrong". To sum up, the untyped setting makes completeness easier, soundness more difficult and may involve dead-locks in computation.
- The expressive power of a logic depends on the connectives involved. This is in particular the case for the additive connectives in linear logic. Additives give a direct representation of conditionals which allow to choose one branch in a computation while forgetting the others. However additives are difficult to handle in proof-nets. In an affine setting (where general weakening is allowed on any formula), the additive behavior can be encoded by means of multiplicative connectives and second-order quantifiers [10].
  General weakening is not well defined in classical systems (see Lafont's critical pair [11]) if we want reasonable computational interpretations of cut elimination. In intuitionistic settings, there is no problem from the proof-theoretical point of view, but the dynamics of intuitionistic affine proof-nets is rather heavy [12,9]. Since both additives and general weakening generate difficulties in proof nets, we prefer to focus on the simplest possible computational model: multiplicative exponential linear proof-nets. Concretely our e-nets are defined with only five different reduction rules.
  The erasure capability provided by additives and useful in encoding conditionals can sometimes be simulated in a linear setting [13]. However this only applies to some particular formulas and, as in [9] the very precise management of exponentials we need, requires a Scott-style encoding for which erasure cannot be deduced in the linear world. For this reason, we follow [14] where, rather than trying to erase useless parts of the computation, they are gathered in a dedicated garbage part of the encodings.
  To sum up, additives and general weakening would make the logic more expressive but obfuscate the cut-elimination process of proof-nets. Our choice of rejecting both of them will make soundness easier but completeness will require dealing with explicit garbage.
- As already mentioned, dealing with general weakening requires to stick to *intuitionistic* logics. However once we decide to stay linear, there is no reason not to move to a classical system. It makes the system more expressive than its intuitionistic variant. We have less steps to consider in cut-elimination. The impact on complexity soundness and completeness happens to be negligible: soundness is even slightly simpler since we have less cases to consider, and completeness is in fact presented through the intuitionistic bridge to make the input/output behavior more readable.

1.2. Structure of the paper

- Section 2 introduces our computational model: *e-nets*, an untyped version of proof-nets for classical elementary linear logic. A direct proof of strong normalization for e-nets based on a decreasing ordinal measure is presented. As far as we know this is the first such proof for ELL proof-nets in the literature. Confluence is then obtained from strong normalization and local confluence.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

- We explain the specificities of the way computation is represented in e-nets in Section 3. We describe how to encode Booleans and binary words with depth 2 e-nets. The linear setting we use requires us to deal with the presence of some (non erasable) garbage along computations.
- While Section 2 deals with "qualitative" properties of e-nets reduction, we consider "quantitative" aspects in Section 4 by providing complexity bounds on some reduction strategies. The concluding result of this section is a polynomial time upper bound on the decision problems we can represent in e-nets through the depth 2 encoding of Section 3.
- Section 5 is about the converse result. We adapt the encoding of PTIME Turing machines of [15,9] to a non-affine setting. For this we show how second-order existential quantification offers a versatile way of masking and handling garbage. The encoding of Turing machines is presented in a typed setting, using fixpoints and second-order quantification. Note the introduction of fixpoints of types is somehow required as recently showed [16]: otherwise only regular languages would be representable. In a second step, types disappear since our untyped e-nets make translation by type-erasure possible.
- Finally Section 6 follows [9] in extending the results of the previous sections to the characterization of the k-EXPTIME hierarchy in e-nets. This relies on encodings with higher depths.

2. Untyped elementary proof-nets

Through proof-nets, elementary linear logic (ELL) defines a computational model which is sound and complete for elementary time [2,4]. A specificity of this system is that the complexity bound on the cut-elimination procedure only relies on the geometric constraints on the proofs and not at all on the structure/size of the formulas. We make this known fact completely explicit in our approach by defining a notion of *untyped* proof-nets for ELL: e-nets, which will be our computational model.

E-nets are a variant of the nets of [7] in which we remove any label on edges. In such nets, some cuts cannot be reduced because of a mismatch between the nodes above the two premises (this situation never happens in the usual typed setting). Following [17], we will prefer the terminology *clash* rather than *deadlock* [7] for such cuts.

In this first section, we present the e-net model and we prove the main properties of reduction: strong normalization and confluence. The key point is strong normalization. We give a proof based on associating an ordinal measure in ω^{ω^2} to any e-net and on proving reduction to make it decrease in any case. Strong normalization is not required for the rest of the paper but we use it as a way to obtain confluence which is necessary for our representation results.

2.1. E-nets

An *e-net* is a directed acyclic graph with outgoing half-edges (edges may have no target). Vertices are called *nodes*. Incoming edges of a node are called its *premises* and its outgoing edges are its *conclusions*. The outgoing half-edges are called the *conclusions* of the e-net. Each node has an associated *kind* which constrains the number of premises and conclusions:

- *ax*: with no premise and two conclusions,
- *cut*: with two premises and no conclusion,
- $\otimes$, $\parr$ or ?c: with two premises and one conclusion,
- ?w: with no premise and one conclusion,
- ?p or !: with one premise and one conclusion.

The premises of the $\otimes$-nodes and $\parr$-nodes are ordered. The first one is called the *left premise* and the second one is called the *right premise*.

With each !-node is associated a sub-graph with no premise, called its *box*, such that all its conclusions are premises of ?p-nodes (the *auxiliary doors* of the box) or of the !-node under consideration (the *main door* of the box). Each ?p-node is the auxiliary door of exactly one box. Two boxes are either disjoint or included one in the other. Each box has itself to be an e-net. The *depth of a node* is the number of boxes it is contained in. The *depth of the e-net* is the maximal depth of its nodes.

We also require the following Danos–Regnier correctness criterion [18]. A *correctness graph* is an *undirected* graph obtained:

- by replacing the premise of each ?p-node p by an edge connecting p to the main door of the box of p;
- and by erasing one of the two premises of each $\parr$ and ?c-node.

All the correctness graphs of an e-net are required to be *acyclic graphs*.

In the graphical representation of e-nets (see for example Figs. 1 and 4), we always represent edges oriented in the top-down direction so that we do not need to put the orientation of edges on the drawings. In this spirit, we say that a node is *above* another one if there is a directed path from the first one to the second. Half-edges are dangling. $\otimes$-nodes and $\parr$-nodes are drawn with the left premise on the left and the right premise on the right. Each box is drawn as a rectangle with its doors on the bottom line.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

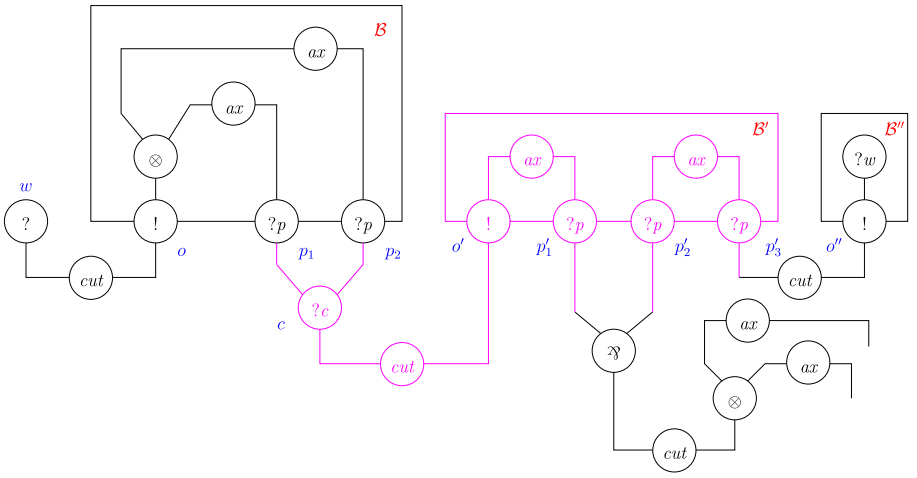


Fig. 1. An e-net.

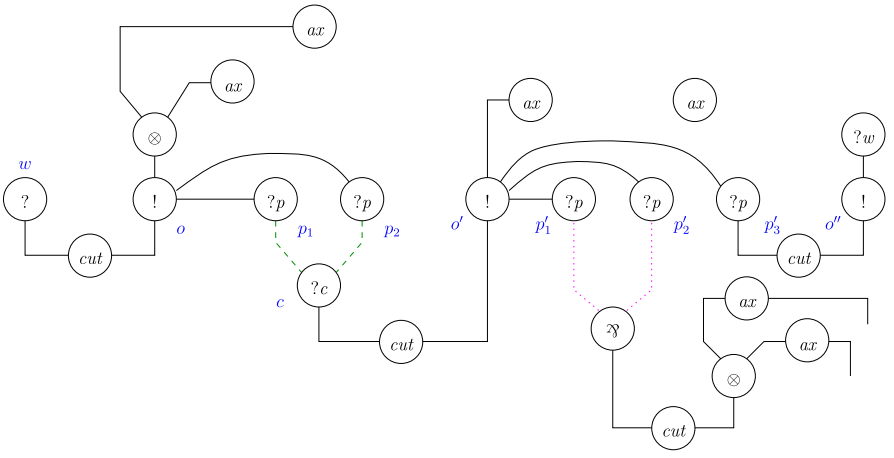


Fig. 2. Correctness graphs.

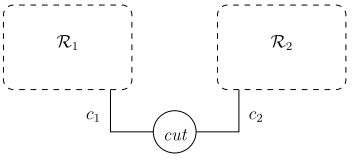


Fig. 3. $\mathcal{R}_1 \,{}_{c_1}\!\bowtie_{c_2} \mathcal{R}_2$.

On Fig. 1, one can find an e-net $\mathcal{R}$. $\mathcal{R}$ has two conclusions (outgoing half-edges) on the *ax*-nodes on the bottom-right corner. $\mathcal{R}$ contains three !-nodes o, o' and o'' whose respective boxes are the contents of the rectangles named $\mathcal{B}$, $\mathcal{B}'$ and $\mathcal{B}''$. The depth of $\mathcal{R}$ is 1. $\mathcal{R}$ has four correctness graphs. They are obtained from the graph of Fig. 2 by selecting only one of the two dashed edges and only one of the two dotted edges. Note that if we look at the *cut*-node connected to a $\parr$-node and to a $\otimes$-node, and if we modify $\mathcal{R}$ by putting a $\otimes$-node on the left and a $\parr$-node on the right, we do not have an e-net anymore since it creates a cycle between the new $\otimes$-node, p'_1, o' and p'_2 (both dotted edges being kept when their target is a $\otimes$-node).

When no distinction is required, ?*w*, ?*c* and ?*p*-nodes are called ?-*nodes*. The *size* of an e-net is its total number of nodes. Its *size at depth d* is the number of nodes at depth d. By convention, if an e-net has depth D, when counting nodes at depth strictly bigger than D, we consider there are 0 nodes at those depths.

If c_1 is a conclusion of an e-net $\mathcal{R}_1$ and c_2 is a conclusion of an e-net $\mathcal{R}_2$, the e-net $\mathcal{R}_1 \,{}_{c_1}\!\bowtie_{c_2} \mathcal{R}_2$ is obtained by adding, to the disjoint union of $\mathcal{R}_1$ and $\mathcal{R}_2$, a *cut*-node with premises c_1 and c_2 (see Fig. 3). When c_1 or c_2 are immediate to infer from the context, we will sometimes omit them. This is the case in particular when an e-net has a unique conclusion.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

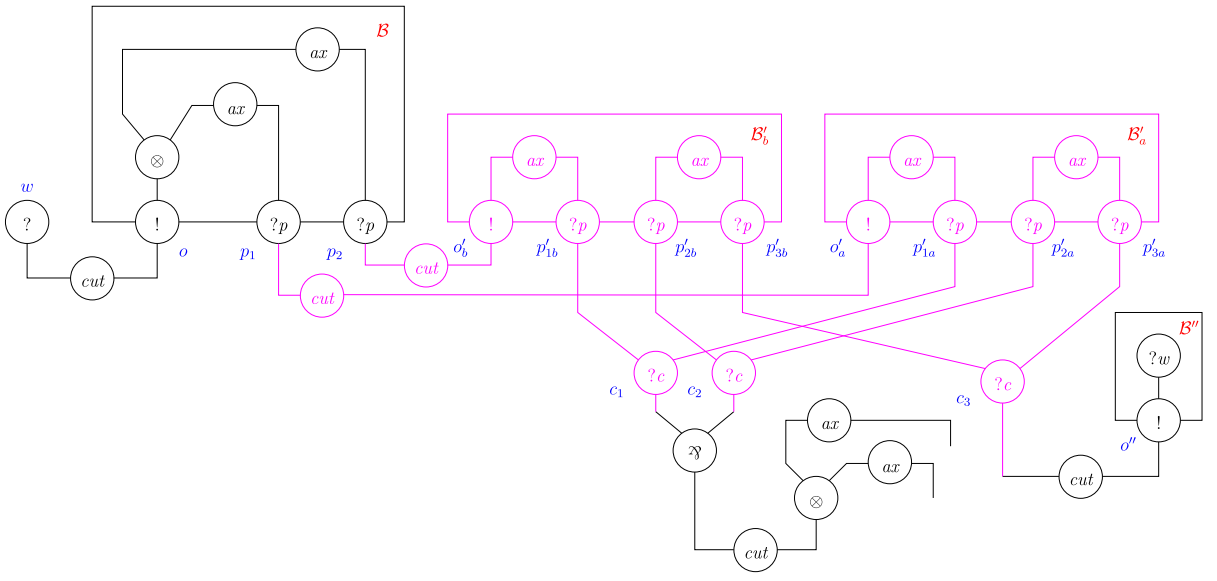
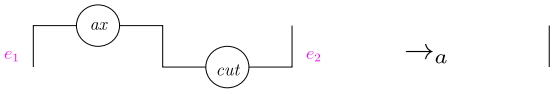


Fig. 4. A one-step reduct of Fig. 1.

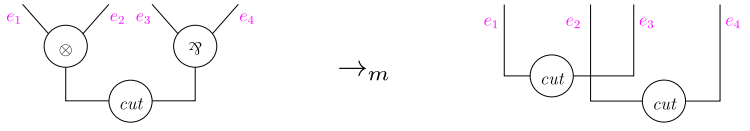
2.2. Reduction

The dynamics of e-nets is given by local rewritings of some sub-graphs. Such a rewrite step turns a given sub-graph into a new one which has the same "interface" (number of incoming and outgoing edges). Reduction occurs around *cut*-nodes only. However due to the untyped nature of e-nets, not all cuts can be reduced. A cut can be reduced if one of the following five reduction rules can be fired:

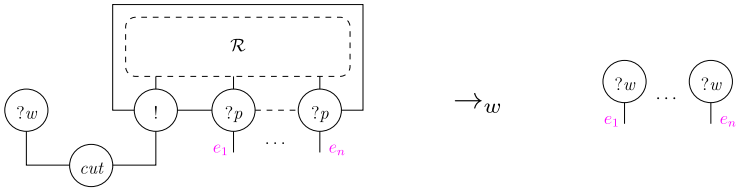
- (*a*-step) It applies if the premise of a *cut*-node is conclusion of an *ax*-node:



  The other conclusion e_1 of the *ax*-node and the other premise e_2 of the *cut*-node cannot be the same edge, otherwise we would have a cycle in the correctness graphs. The reduction connects, through a single edge, the source of e_2 and the target of e_1 (if any, otherwise we get an half-edge).
- (*m*-step) It applies if the premises of a *cut*-node are conclusions of a $\otimes$-node and of a $\parr$-node respectively:



- (*w*-step) It applies if the premises of a *cut*-node are conclusions of a *w*-node and of an !-node respectively:



  The box of the !-node is erased.
- (*c*-step) It applies if the premises of a *cut*-node are conclusions of a *c*-node and of an !-node respectively:

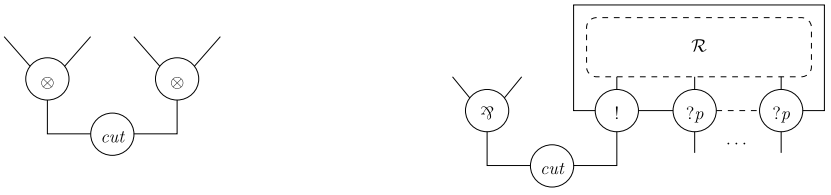
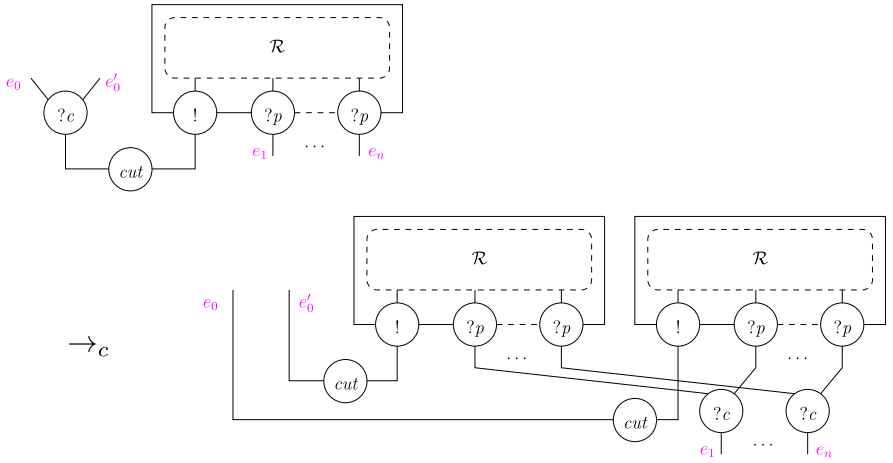
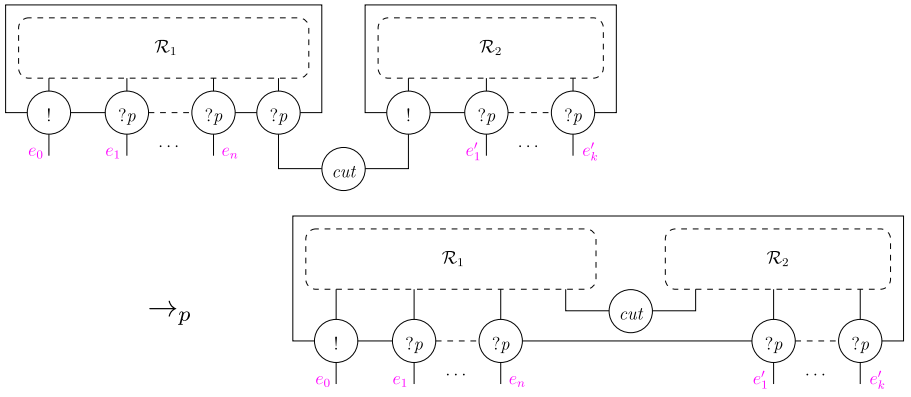


Fig. 5. Two examples of clashes.



The box of the !-node is duplicated.

- (p-step) It applies if the premises of a *cut*-node are conclusions of a p-node and of an !-node respectively:



The two boxes are “merged”.

Following [19], one can check that acyclicity of the correctness graphs is preserved under these reduction steps.

The e-net of Fig. 4 is obtained by applying a c-step to the e-net of Fig. 1. The box $\mathcal{B}'$ is duplicated into $\mathcal{B}'_a$ and $\mathcal{B}'_b$.

A reducible *cut*-node is called a *redex*. Non reducible cuts are called *clashes* (see Fig. 5). An e-net is called *normal* if it contains no redex. It is in *multiplicative normal form* (at depth d) if it contains no redex for the a-step or the m-step (at depth d). Redexes of the a and m-steps are called *multiplicative redexes*, the other redexes are the *exponential redexes*. A reduction step involving a multiplicative (resp. exponential) redex is called a *multiplicative* (resp. *exponential*) *step*.

2.3. Main properties

We focus here on *qualitative* properties of e-nets and reduction. *Quantitative* properties concerning the size of e-nets, the reduction lengths, etc., will be studied in Section 4.

We start with some simple facts about cuts and reduction.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

Fact 1 *(Non increasing depth). If $\mathcal{R}$ reduces to $\mathcal{R}'$ in one step, either they have the same depth or the reduction is a w-step and the depth of $\mathcal{R}'$ may be strictly smaller than the depth of $\mathcal{R}$.* □

Fact 2 *(Reduction at bigger depth). If $\mathcal{R}$ reduces to $\mathcal{R}'$ by a step applied at depth d, the part of $\mathcal{R}$ with depth strictly smaller than d is not modified.* □

Fact 3 *(Exponential residues). If $\mathcal{R}$ reduces to $\mathcal{R}'$ by an exponential step at depth d, the generated cuts are never multiplicative redexes at depth d, and no multiplicative node (ax, $\otimes$ or $\parr$) is modified or created at depth d.* □

Fact 4 *(Residual ?-nodes). If $\mathcal{R}$ reduces to $\mathcal{R}'$ by a step at depth d, if $\mathcal{R}$ contains a ?-node at depth d which is above a conclusion of $\mathcal{R}$, then $\mathcal{R}'$ also contains such a node.* □

Fact 5 *(Persistence of clashes). If c is a clash in the e-net $\mathcal{R}$ which reduces to $\mathcal{R}'$ by a step at depth d, either c persists in $\mathcal{R}'$ or the reduction is a w-step and c belongs to the erased box (thus c is at depth at least $d+1$ in $\mathcal{R}$).* □

We now move to a proof of the strong normalization property for e-nets. In contrast with many logical systems, we are here in an untyped setting so that we cannot rely on types to prove normalization. It relies on the geometric structure of e-nets, the key idea of [2].

Definition 1 *(Reachability).* An !-node o is *immediately reachable* from a ?-node n if:

- the conclusion of o is the premise of a *cut*-node c;
- n is above c, and the directed path from n to c crosses ?c-nodes only.

This is extended transitively, by saying that o_k is *reachable* from n_1 if there is a sequence $n_1o_1n_2o_2\dots n_ko_k$ (called a *reachability path*) where o_i is immediately reachable from n_i $(1 \le i \le k)$ and n_{i+1} is an auxiliary door of the box of o_i $(1 \le i \le k-1)$.

If o is immediately reachable from n, they are above two different premises of the same *cut*-node. All the nodes of a reachability path are at the same depth.

Thanks to the correctness criterion, in a reachability path $n_1o_1n_2o_2\dots n_ko_k$, o_i is not reachable from n_j if $i < j$ (otherwise one would obtain a cycle in a correctness graph), leading to:

Fact 6 *(Decreasing reachability). In a reachability path $n_1o_1n_2o_2\dots n_ko_k$, the number of reachability paths starting from n_i is strictly bigger than the number of reachability paths starting from n_j if $i < j$.* □

In the e-net of Fig. 1, o is immediately reachable from w, o' is immediately reachable from p_1, p_2 and c, and o'' is immediately reachable from p'_3. The reachability paths are thus wo, wop_1o', wop_2o', $wop_1o'p'_3o''$, $wop_2o'p'_3o''$ (5 paths starting from w), p_1o', $p_1o'p'_3o''$ (2 paths starting from p_1), p_2o', $p_2o'p'_3o''$ (2 paths starting from p_2), co', $co'p'_3o''$ (2 paths starting from c) and p'_3o'' (1 path starting from p'_3).

Note that a given !-node might be reachable from a given ?-node through more than one reachability path: wop_1o' and wop_2o'. The intuition here is that, through reductions, the node w will eventually act on (copies of) the boxes whose main door is reachable from w. In this example w will act on the box $\mathcal{B}$ of o, then on two copies of the box $\mathcal{B}'$ of o', and after that on copies of $\mathcal{B}''$. This can be understood as a simplified and more qualitative version of [20].

Theorem 1 *(Strong normalization). The reduction of e-nets is strongly normalizing: there is no infinite sequence of reductions.*

Proof. We associate with each e-net $\mathcal{R}$ of depth at most D a tuple $|\mathcal{R}|_{\mathrm{sn}} = (|\mathcal{R}|^D_{\mathrm{sn}}, \dots, |\mathcal{R}|^0_{\mathrm{sn}})$, where for each $0 \le d \le D$, $|\mathcal{R}|^d_{\mathrm{sn}}$ is a pair (μ_d, m_d):

- μ_d is the multiset containing, for each ?-node at depth d, the number of reachability paths starting from it (we ignore the ?-nodes with no reachability path starting from them);
- and m_d is the number of *ax*, $\otimes$ and $\parr$-nodes at depth d in $\mathcal{R}$.

If the depth of $\mathcal{R}$ is strictly smaller than d, then $|\mathcal{R}|^d_{\mathrm{sn}} = ([\,], 0)$. We need this only to compare the tuples associated with $\mathcal{R}$ and with its reduct in a w-step since the depth may strictly decrease. Otherwise the depth is not modified (Fact 1).

To each e-net $\mathcal{R}$ we can also associate an ordinal $|\mathcal{R}|^{\mathrm{ord}}_{\mathrm{sn}} \in \omega^{(\omega+1)\cdot(D+1)} = \omega^{\omega\cdot(D+1)+1} \in \omega^{\omega^2}$ since $|\mathcal{R}|_{\mathrm{sn}}$ is order-isomorphic to such an ordinal when:

- $\mathbb{N}$ is endowed with its usual order, leading to the ordinal ω;

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

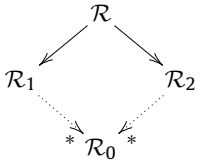
- multisets over $\mathbb{N}$ are endowed with the multiset-ordering, leading to the ordinal ω^ω;
- pairs (μ_d, m_d) are ordered with a lexicographic order with biggest weight on the right-most component, leading (up to order-isomorphism) to elements of $\omega^\omega \cdot \omega$;
- and similarly for tuples of $D+1$ elements $(|\mathcal{R}|^D_{\mathrm{sn}}, \dots, |\mathcal{R}|^0_{\mathrm{sn}})$, leading (up to order-isomorphism) to elements of $\underbrace{(\omega^\omega \cdot \omega) \cdots (\omega^\omega \cdot \omega)}_{D+1 \text{ times}} = \omega^{(\omega+1)\cdot(D+1)}$.

We prove that any reduction step makes $|\mathcal{R}|^{\mathrm{ord}}_{\mathrm{sn}}$ decrease, and thus reduction always terminates. According to Fact 2, if reduction occurs at depth d, it is enough to focus on $(|\mathcal{R}|^i_{\mathrm{sn}})_{d \le i}$. We prove for each possible reduction step at depth d that (the ordinal associated with) $|\mathcal{R}|^d_{\mathrm{sn}}$ strictly decreases.

- a-step: No $\otimes$ or $\parr$-node is created and one ax-node is erased at depth d, thus m_d decreases.
- m-step: The ax-nodes are not modified and one $\otimes$-node and one $\parr$-node are erased at depth d, thus m_d decreases.
- w-step: The ax, $\otimes$ and $\parr$-nodes at depth d are not modified, thus the value of m_d does not change. Concerning μ_d, the element corresponding to the $?w$-node above the cut in the redex disappears, and the created $?w$-nodes give elements which are the same as those of the $?p$-nodes they replace, thus μ_d decreases.
- c-step: The ax, $\otimes$ and $\parr$-nodes at depth d are not modified, thus the value of m_d does not change. Concerning μ_d, let us first recall (Fact 6) that the values associated with the $?p$-nodes of the box above the cut in the redex are strictly smaller than the value associated with the $?c$-node above the cut. This last value is erased after reduction, and the (smaller) value associated with each $?p$ is replaced by three copies of it (two from $?p$-nodes and one from a $?c$-node), thus μ_d decreases.
- p-step: The ax, $\otimes$ and $\parr$-nodes at depth d are not modified, thus the value of m_d does not change. Concerning μ_d, the element corresponding to the $?p$-node above the cut in the redex disappears and the others do not increase, thus μ_d decreases. □

If we look at the e-net $\mathcal{R}$ of Fig. 1 and its one-step reduct $\mathcal{R}'$ in Fig. 4, we have: $|\mathcal{R}|_{\mathrm{sn}} = (([\,], 5), ([1,2,2,2,5], 4)) \simeq \omega^{\omega\cdot 2} \cdot 4 + \omega^{\omega+6} + \omega^{\omega+3} \cdot 3 + \omega^{\omega+2} + \omega^\omega \cdot 5 = |\mathcal{R}|^{\mathrm{ord}}_{\mathrm{sn}} > |\mathcal{R}'|^{\mathrm{ord}}_{\mathrm{sn}} = \omega^{\omega\cdot 2} \cdot 4 + \omega^{\omega+6} + \omega^{\omega+3} \cdot 2 + \omega^{\omega+2} \cdot 3 + \omega^\omega \cdot 7 \simeq (([\,], 7), ([1,1,1,2,2,5], 4)) = |\mathcal{R}'|_{\mathrm{sn}}$. Indeed the reachability paths in $\mathcal{R}'$ are: wo, $wop_1o'_a$, $wop_2o'_b$, $wop_1o'_ap'_{3a}o''$, $wop_2o'_bp'_{3b}o''$, $p_1o'_a$, $p_1o'_ap'_{3a}o''$, $p_2o'_b$, $p_2o'_bp'_{3b}o''$, $p'_{3a}o''$, $p'_{3b}o''$, c_3o'', thus $w \mapsto 5$, $p_1 \mapsto 2$, $p_2 \mapsto 2$, $p'_{3a} \mapsto 1$, $p'_{3b} \mapsto 1$ and $c_3 \mapsto 1$.

Proposition 1 *(Local confluence). The reduction of e-nets is locally confluent:*

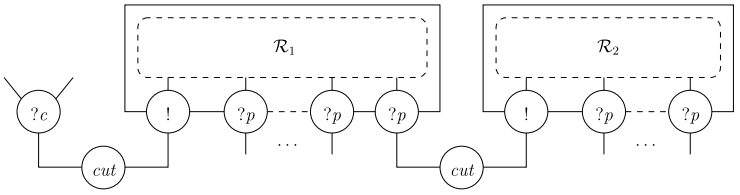


Proof. The proof is the same as for typed proof-nets [19] by considering all possible critical pairs. There are two main categories of such pairs:

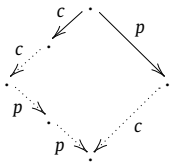
- If one of the redexes is inside a box involved in the other redex, this outermost redex must be an exponential redex. Depending whether it reduces by a w-step, c-step or p-step, the innermost redex has then disappeared or must be reduced twice, or just once (respectively).
- The two redexes may have a non-trivial overlap, which can be:
  - an ax-node shared by two redexes of the a-step, in which case the two reductions lead to the same result;
  - a cut-node shared by two redexes of the a-step, in which case the two reductions lead to the same result;
  - a whole box shared by the redexes of a p-step and of another exponential step.

This last family is the most interesting one with four cases. First the redexes of two p-steps involving two auxiliary doors of a given box, second a p-step on an auxiliary door of a box and a w-step, c-step or p-step on its main door. We consider here the c-step vs. p-step case which is the trickiest:

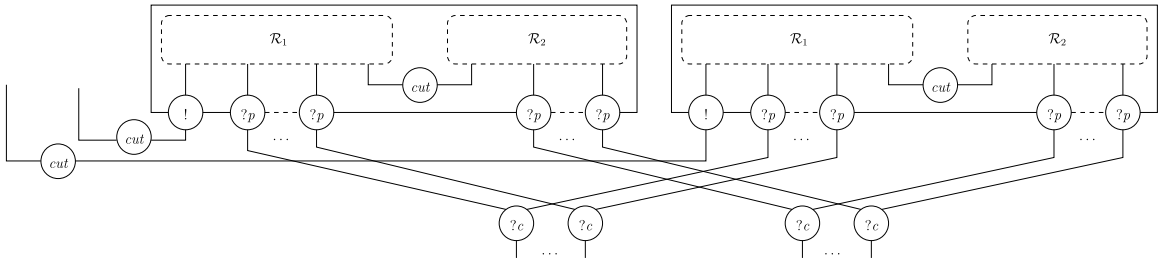
Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002



we obtain the following reduction diagram for closing the critical pair:

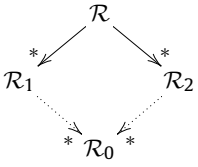


leading to the following final e-net:



□

Proposition 2 *(Confluence). The reduction of e-nets is confluent:*



Proof. By Theorem 1 and Proposition 1 with Newman's Lemma [21]. □

In particular, thanks to Theorem 1 and Proposition 2, an e-net as a unique normal form.

2.4. Closed reduction

For studying the representation of Turing machines, it is enough to focus on some particular reduction strategies for e-nets. We define here the key ingredients of the strategy we are going to focus on.

Definition 2 *(Closed redex).* An exponential redex is called *closed* if the box with main door involved has no auxiliary door (such a box is called a *closed box*).

Lemma 1 *(Closed redex). Let $\mathcal{R}$ be an e-net with no multiplicative redex and no clash at depth* 0 *and with at least one exponential redex at depth* 0*. If there is no* ?*-node at depth* 0 *which is above a conclusion (thus they are all above cuts), then $\mathcal{R}$ contains a closed exponential redex at depth* 0*.*

There are many proofs of this lemma in the literature. We give here a simple one based on paths and correctness (rather than the usual approach via sequentialization).

Proof. In an e-net with no multiplicative redex and no clash at depth 0, one can extend the notion of !-node reachable from a ?-node by allowing to cross any kind of node while going down from the ?-nodes (not just ?*c*-nodes as before). Thanks to the correctness criterion again, the reachability relation does not contain cycles. Starting from the exponential redex and following a reachability path, one can find a cut !-node such that none of the auxiliary doors of its box can reach

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

an !-node. The box of such a maximal !-node cannot have any ?p-node otherwise they would be above a conclusion (or contradict maximality). The cut below this !-node is then a closed redex. □

A sequence of reduction steps is called a *stubborn closed exponential reduction* if it starts with the reduction of a closed redex at depth d and then only reduces redexes which are at depth d and were generated by previous steps in the sequence. Moreover it is required to be a maximal such sequence. One easily checks that all the reduced redexes are closed. *Iterated stubborn closed exponential reduction* at depth d of an e-net $\mathcal{R}$ consists in iteratively choosing a closed redex at depth d in $\mathcal{R}$ and reducing it by a stubborn closed exponential reduction, until there is no closed redex at depth d in $\mathcal{R}$ anymore.

While [9] considers the reduction of *special cuts* (exponential redexes such that the associated box has no auxiliary door above a cut), we prefer to focus here on closed cuts (associated box with no auxiliary door at all) since they are slightly simpler to handle and general enough in the study of the representation of functions and decision problems on binary words.

Lemma 2. *Let $\mathcal{R}$ be an e-net with no cut at depth strictly smaller than d, with no multiplicative redex at depth d, but with at least one cut at depth d. If there is a reduction sequence from $\mathcal{R}$ to an e-net $\mathcal{R}'$ with no cut at depth d and no ?-node at depth d which is above a conclusion, then $\mathcal{R}$ contains a closed exponential redex at depth d and iterated stubborn closed exponential reduction of $\mathcal{R}$ at depth d leads to an e-net with no cut at depth d.*

Proof. Let us consider the content $\mathcal{R}_1$ of a box at depth d in $\mathcal{R}$ which contains a cut c at depth d (it exists by assumption). $\mathcal{R}_1$ does not contain any multiplicative redex at depth 0. $\mathcal{R}_1$ does not contain any clash at depth 0 by Fact 5 since $\mathcal{R}'$ does not contain any clash at depth d. The cut c is thus an exponential redex. By Facts 4 and 5 and Lemma 1, $\mathcal{R}_1$ contains a closed exponential redex at depth 0, that is $\mathcal{R}$ contains a closed exponential redex at depth d.

Let $\mathcal{R}_0$ be obtained by iterated stubborn closed exponential reduction of $\mathcal{R}$ at depth d. By applying Proposition 2 to $\mathcal{R}_0$ and $\mathcal{R}'$, we obtain a common reduct $\mathcal{R}'_0$. By Fact 2, $\mathcal{R}'_0$ does not contain any cut at depth d or smaller (it is a reduct of $\mathcal{R}$ and $\mathcal{R}'$). Similarly $\mathcal{R}'_0$ does not contain ?-nodes at depth d which are above conclusions. As a consequence, if $\mathcal{R}_0$ contains a cut at depth d, it cannot be a multiplicative redex (Fact 3 applied to $\mathcal{R}$) nor a clash (contrapositive of Fact 5 applied to $\mathcal{R}'_0$). So if $\mathcal{R}_0$ contains a cut at depth d, by Lemma 1 (using the contrapositive of Fact 4 applied to $\mathcal{R}'_0$), it contains a closed exponential redex, contradicting the definition of iterated stubborn closed exponential reduction. □

3. Representation of decision problems

In order to measure the expressive power of the computational model given by e-nets, we are going to define a representation of (some) decision problems in e-nets. Not all problems can be represented since our e-nets model is not Turing complete: it comes with intrinsic bounds on computation time.

We denote by $\mathbf{B}$ (for Booleans) the two values set $\{0, 1\}$ and by $\mathbf{W}$ (for words) the set $\mathbf{B}^*$ of finite binary words. Given a function $\mathcal{P} : \mathbf{W} \to \mathbf{B}$ (*i.e.* a decision problem), we want to define an e-net and a way to give it an element w of $\mathbf{W}$ as an argument so that computation (*i.e.* reduction) in e-nets gives us a way to recover the value of $\mathcal{P}(w) \in \mathbf{B}$.

3.1. Booleans

Let us start with this read-back question. The standard way of representing $\mathbf{B}$ in another data-type is to choose two objects, one for reflecting the value 0 and the other one for reflecting the value 1.

Our computational model is not powerful enough for this approach. Due to the linear flavor of e-nets, we cannot erase arbitrary sub-nets through reduction. As a consequence the computations will accumulate some useless results: the *garbage*. We are not able to ensure that all computations with result 1 will lead to the same e-net: we have no control on their garbage parts. This idea of dealing with garbage in linear computations directly comes from [14].

For this reason, we consider a set of e-nets associated with each Boolean value. The families of e-nets representing 0 and 1 are given on Fig. 6 (0 on the left and 1 on the right) where $\mathcal{G}$ is the parameter of the family which can be any e-net.

We can check that the presence of $\mathcal{G}$ does not blur the representation:

Fact 7 *(Unambiguous Boolean representation). There is no e-net representing both* 0 *and* 1*, and if $\mathcal{R}_b$ represents b then all its reducts represent b as well.* □

Lemma 3 *(Reduction to a Boolean). If $\mathcal{R}$ is an e-net which reduces to some $\mathcal{R}_b$ representing b, and if $\mathcal{R}'$ is a reduct of $\mathcal{R}$ then $\mathcal{R}'$ also reduces to a representation of b.*

Proof. By Proposition 2, there exists some $\mathcal{R}_0$ which is a reduct of both $\mathcal{R}_b$ and $\mathcal{R}'$. According to Fact 7, $\mathcal{R}_0$ is a representation of b. □

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

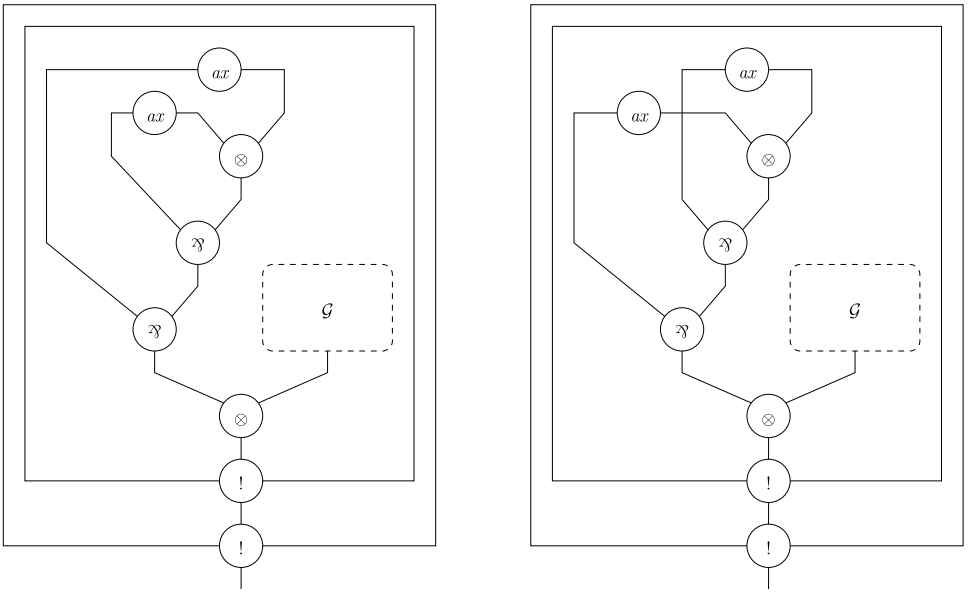
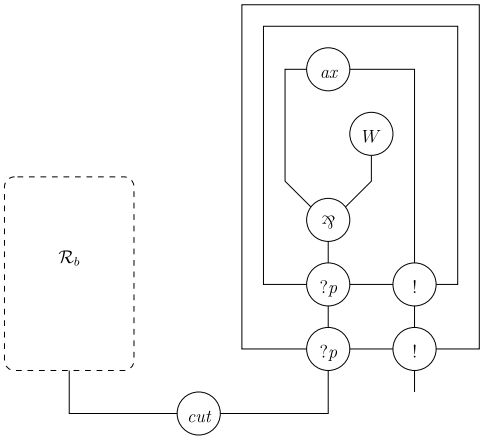


Fig. 6. Representation of Booleans (0 and 1 respectively).

So if we can ensure that an e-net $\mathcal{R}$ reduces to some $\mathcal{R}_b$, it is sensible to say that $\mathcal{R}$ *computes the value* b.

Remark 1. In an affine setting (see [9] for example), one can extract a unique representation of Booleans by building the following e-net from any representation $\mathcal{R}_b$ of a Boolean b:



where W is the general weakening node (used to erase the garbage part $\mathcal{G}$ in $\mathcal{R}_b$, but remember the reduction steps for such a W-node are not so easy to handle).

This shows that our setting is more general than the affine one. Indeed our encodings apply in the affine case and we can extract standard representations of Booleans at any point by introducing the appropriate cut as above.

3.2. Binary words

In order to be able to give an element w of $\mathbf{W}$ as an argument to a given e-net, we want to inject $\mathbf{W}$ into the set of e-nets. Let $w \in \mathbf{W}$ be a binary word, its *representation* is the cut-free e-net $\overline{w}$ of Fig. 7 where the number of ?p-nodes is the length of w, and the ? part represents two trees of ?c-nodes and ?w-nodes leading to the appropriate arities (a canonical choice is to use a ?w-node for arity 0 only, to use ?c-nodes for arities 2 and above, and to associate the tree of ?c-nodes to the left) in such a way that if the ith letter of w is 0, the ith ?p-node is connected to the first (left-most) tree, and if the ith letter of w is 1, the ith ?p-node is connected to the second (right-most) tree.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

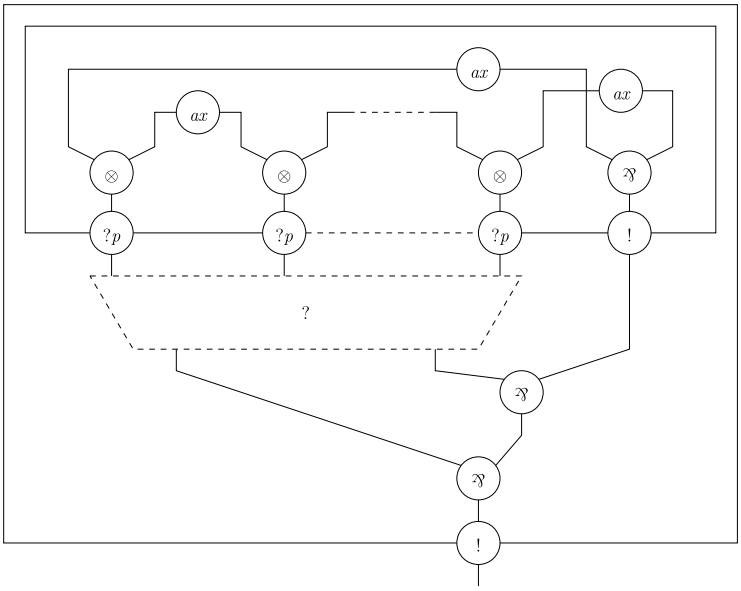


Fig. 7. Representation of binary words.

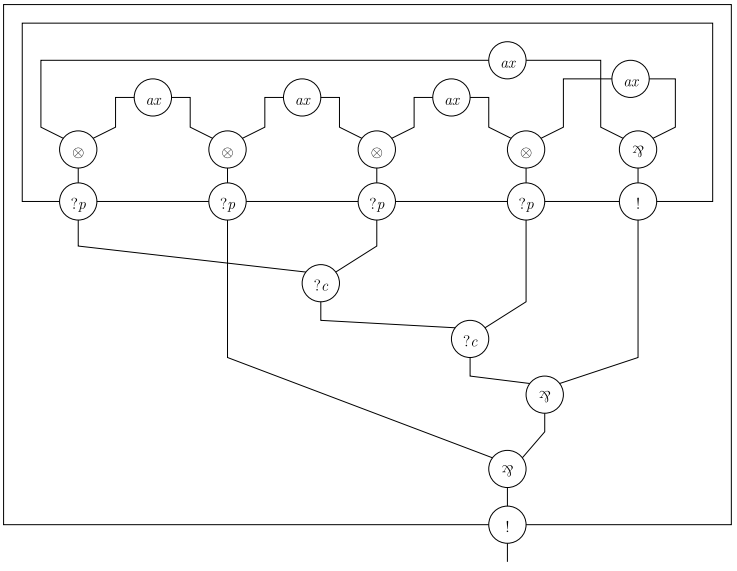


Fig. 8. Representation of 1011.

For example, the representation of 1011 is on Fig. 8.

Since binary words will be used as inputs, we can have perfect control on the way they are represented without having to consider families of representations as for Booleans which arrive as outputs.

3.3. Decision problems

The missing ingredients are now: how to associate an e-net with a decision problem and how to evaluate it on a given argument in **W**. This last part is provided by the following definition:

Definition 3 *(Representation of a decision problem).* Let $\mathcal{P} : \mathbf{W} \to \mathbf{B}$ be a decision problem on binary words, the e-net $\mathcal{R}$ *represents* $\mathcal{P}$ if:

- $\mathcal{R}$ is an e-net with two conclusions ι and o,
- for any $w \in \mathbf{W}$, $\mathcal{R} \,_{\iota}\!\bowtie \overline{w}$ reduces to a representation of $\mathcal{P}(w)$.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

Section 5 will deal with the construction of $\mathcal{R}$ from $\mathcal{P}$ when $\mathcal{P}$ is a PTIME decision problem. But let assume we have built such an e-net $\mathcal{R}$ representing the decision problem $\mathcal{P}$. According to Definition 3, in order to compute the value of $\mathcal{P}(w)$ for some given $w \in \mathbf{W}$, it suffices:

- to construct the e-net $\overline{w}$ (see Section 3.2),
- to build the e-net $\mathcal{R} \,{}_{\iota}\!\bowtie \overline{w}$ by adding a *cut*-node to the disjoint union of $\overline{w}$ and $\mathcal{R}$,
- and to reduce the obtained e-net until we reach a representation $\mathcal{R}_b$ of a Boolean b (see Section 3.1).

We conclude that $\mathcal{P}(w) = b$.

Let us make a few comments on this process. First Lemma 3 tells us that, no matter in which way we reduce $\mathcal{R} \,{}_{\iota}\!\bowtie \overline{w}$, we will always get the same b. In particular some reduction sequences may be longer to compute than others. When complexity comes into the picture, choosing appropriate reduction strategies will be important (this is studied in Section 4). Second when some $\mathcal{R}_b$ is reached, it takes constant time to figure out which b is represented: start from the unique conclusion of $\mathcal{R}_b$ and check whether the *ax*-nodes reached by the following two paths are the same or not:

(1) go up through the !-node above the conclusion, go up again through the !-node above, go left through the $\otimes$-node, go right through the $\parr$ node, and go left through the $\parr$-node above,
(2) go up through the !-node above the conclusion, go up again through the !-node above, go left through the $\otimes$-node, go right through the $\parr$ node, go right through the $\parr$-node above, and go left through the $\otimes$-node.

If we reach the same *ax*-node, $b = 0$, otherwise $b = 1$.

Lemma 4 *(Normal form representation). If $\mathcal{R}$ represents $\mathcal{P}$ and $\mathcal{R}'$ is a reduct of $\mathcal{R}$ (in particular its normal form) then $\mathcal{R}'$ represents $\mathcal{P}$ as well.*

Proof. First, the number of conclusions of an e-net is not modified by reduction. Second, if $\mathcal{R} \,{}_{\iota}\!\bowtie \overline{w}$ reduces to some $\mathcal{R}_{\mathcal{P}(w)}$ representing $\mathcal{P}(w)$, then $\mathcal{R}' \,{}_{\iota}\!\bowtie \overline{w}$ reduces to a representation of $\mathcal{P}(w)$ as well (Lemma 3). □

4. Complexity bounds

We study here quantitative bounds on the reduction of e-nets. The goal is to prove a PTIME upper bound on the complexity of representable decision problems (Theorem 2). For that purpose, we are going to study reduction strategies with a PTIME upper bound, but also powerful enough to reach representations of Booleans from the representation of PTIME problems applied to some input.

In order to manipulate in a compact manner the size informations about e-nets we need to monitor along reductions, we first introduce a notion of weight and weight matrix.

Definition 4 *(Weight at depth d).* If $\mathcal{R}$ is an e-net, its *weight at depth d* is the triple $w_d(\mathcal{R}) = (s, m, e) \in \mathbb{N}^3$ where:

- s is the size of $\mathcal{R}$ at depth d in $\mathcal{R}$;
- m is the number of multiplicative redexes at depth d in $\mathcal{R}$;
- e is the number of exponential redexes at depth d in $\mathcal{R}$.

Definition 5 *(Weight matrix).* Let $\mathcal{R}$ be an e-net, a *weight matrix $\mathcal{M}$ at depth d* of $\mathcal{R}$ is a 3×3 matrix with coefficients in $\mathbb{N} \cup \{\infty\}$:

$$\begin{pmatrix} s_0 & m_0 & e_0 \\ s_1 & m_1 & e_1 \\ s_+ & m_+ & e_+ \end{pmatrix}$$

such that, according to the componentwise order:

$$\begin{aligned} w_d(\mathcal{R}) &\leq (s_0, m_0, e_0) \\ w_{d+1}(\mathcal{R}) &\leq (s_1, m_1, e_1) \\ w_{d'}(\mathcal{R}) &\leq (s_+, m_+, e_+) \qquad \text{for any } d' > d+1. \end{aligned}$$

If $\mathcal{M}$ is a weight matrix of $\mathcal{R}$ at depth d, we use the notation $\mathcal{M} \rhd_d \mathcal{R}$.

Remember that sizes, numbers of nodes, etc., are defined to be 0 at depths not occurring in $\mathcal{R}$ (see Section 2.1). The value ∞ is used for coefficients we do not want to (or cannot) monitor. If $\mathcal{M} \rhd_d \mathcal{R}$ and if $\mathcal{M}'$ is obtained from $\mathcal{M}$ by replacing some coefficients with ∞, then we also have $\mathcal{M}' \rhd_d \mathcal{R}$.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

Lemma 5 *(Weight matrix shift).*

$$\text{If } \begin{pmatrix} s_0 & m_0 & e_0 \\ s_1 & m_1 & e_1 \\ s_+ & m_+ & e_+ \end{pmatrix} \triangleright_d \mathcal{R} \text{ then } \begin{pmatrix} s_1 & m_1 & e_1 \\ s_+ & m_+ & e_+ \\ s_+ & m_+ & e_+ \end{pmatrix} \triangleright_{d+1} \mathcal{R}.$$

Proof. This is a direct consequence of Definition 5. □

We now study how weight matrices can be used to bound the evolutions of an e-net along some particular sequences of reductions. They sum up key quantitative measures on e-nets allowing us to get upper bounds on the lengths of some reduction sequences and on the sizes of the generated e-nets (see the proof of Theorem 2 in particular).

Lemma 6 *(Iterated multiplicative weight reduction). If $\mathcal{R}$ reduces to $\mathcal{R}'$ by a sequence of multiplicative reduction steps at depth d then:*

$$\begin{pmatrix} s_0 & \infty & \infty \\ s_1 & \infty & \infty \\ s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R} \quad \Longrightarrow \quad \begin{pmatrix} s_0 & \infty & \infty \\ s_1 & \infty & \infty \\ s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}'$$

Moreover the number of reduction steps is at most s_0.

Proof. The size at depth d strictly decreases at each multiplicative step, and nodes (and thus sizes) at depths strictly bigger than d are not modified. □

Lemma 7 *(Stubborn closed exponential weight reduction). If $\mathcal{R}$, with at most k ?-nodes at depth d, reduces to $\mathcal{R}'$ by stubborn closed exponential reduction at depth d,*

$$\begin{pmatrix} s_0 & m_0 & e_0 \\ s_1 & m_1 & \infty \\ s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R} \quad \Longrightarrow \quad \begin{pmatrix} k+s_0 & m_0 & e_0-1 \\ (k+1)s_1 & (k+1)m_1+k & \infty \\ (k+1)s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}'$$

Moreover each reduction step from $\mathcal{R}$ to $\mathcal{R}'$ makes the number of ?-nodes at depth d strictly decrease, and the length of the reduction is bounded by k.

Proof.

- A closed reduction step at depth d removes a ?-node at depth d (and no such node is duplicated or added), so the total length of the reduction cannot be strictly bigger than k.
- Through a closed reduction step at depth d, the size at depth d increases at most by 1.
- An exponential reduction cannot generate multiplicative redexes at depth d (Fact 3).
- By maximality of the reduction sequence in a stubborn closed exponential reduction, an exponential redex is removed.
- There are at most k steps and each one adds at most one copy of the box concerned by the reduction sequence.
- For the same reason, one can move from m_1 multiplicative redexes at depth $d+1$ to $(k+1)m_1$ multiplicative redexes at depth $d+1$, but also each copy of the box may come with a new multiplicative redex at depth $d+1$. □

Lemma 8 *(Iterated stubborn closed exponential weight reduction). If $\mathcal{R}$ containing a closed exponential redex reduces to $\mathcal{R}'$ by iterated stubborn closed exponential reduction at depth d,*

$$\begin{pmatrix} s_0 & m_0 & e_0 \\ s_1 & \infty & \infty \\ s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R} \quad \Longrightarrow \quad \begin{pmatrix} 2s_0 & m_0 & e_0 \\ s_0^{e_0}s_1 & \infty & \infty \\ s_0^{e_0}s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}'$$

and the number of reduction steps is bounded by s_0.

Proof. Each time we apply a stubborn closed exponential reduction step from $\mathcal{R}_i$ to $\mathcal{R}_{i+1}$, Lemma 7 gives:

$$\begin{pmatrix} s_0 & m_0 & e_0 \\ s_1 & \infty & \infty \\ s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}_i \quad \Longrightarrow \quad \begin{pmatrix} k_i+s_0 & m_0 & e_0-1 \\ (k_i+1)s_1 & \infty & \infty \\ (k_i+1)s_+ & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}_{i+1}$$

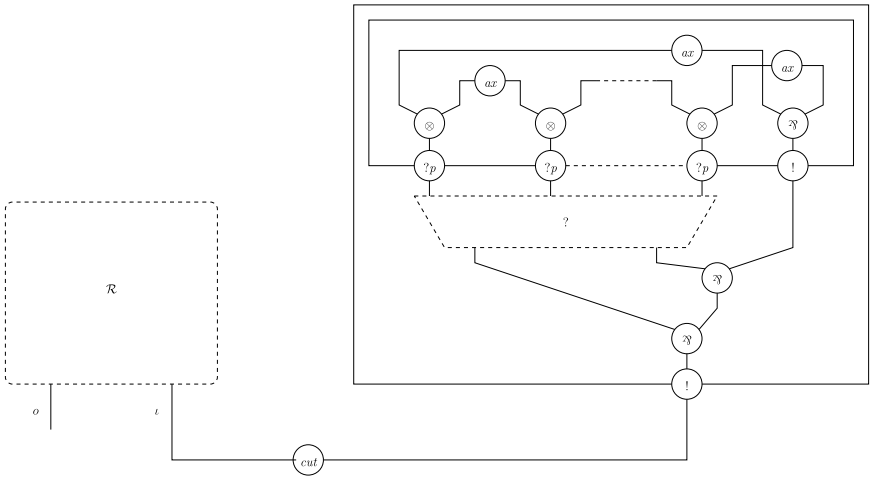
so we obtain the result by iterating at most e_0 times with $\sum_i k_i < s_0$. □

Theorem 2 *(PTIME soundness). If the e-net $\mathcal{R}$ represents the decision problem $\mathcal{P}$ then $\mathcal{P}$ is in* PTIME.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

Proof. By Lemma 4, it is enough to prove that any decision problem represented by a normal e-net is in PTIME. We thus now assume $\mathcal{R}$ to be a normal form.

If $w \in \mathbf{W}$ of length l, we note $\mathcal{R}_w = \mathcal{R} \,{}_\iota\!\bowtie \overline{w}$:



Let S be 3 plus the size of $\mathcal{R}$, and $N = S + 2l$, we have:

$$\begin{pmatrix} S & 0 & 1 \\ N & 0 & 0 \\ N & 0 & 0 \end{pmatrix} \triangleright_0 \mathcal{R}_w$$

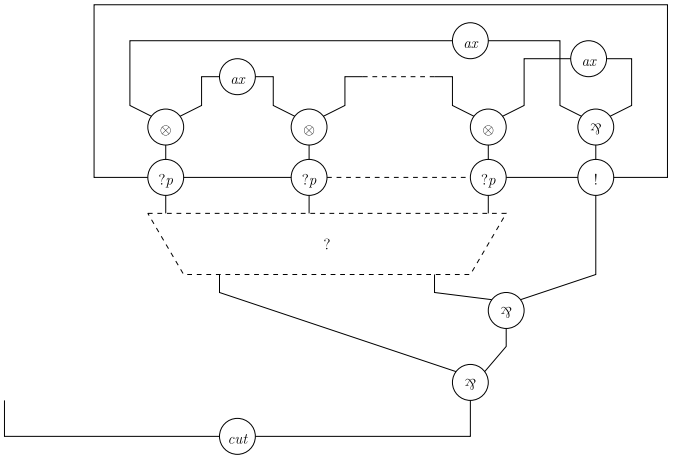
Indeed, since $\mathcal{R}$ and $\overline{w}$ are both in normal form, the only redex of $\mathcal{R} \,{}_\iota\!\bowtie \overline{w}$ is the cut introduced between $\mathcal{R}$ and $\overline{w}$ which is at depth 0 and is an exponential redex (otherwise it would be a clash or an a-step redex and reduction could not lead to the representation of a Boolean).

Concerning sizes, S is an immediate bound at depth 0. The size of $\overline{w}$ at depth 1 is at most $2l+5$. The size of $\overline{w}$ at depth 2 is $2l+2$. The nodes of $\mathcal{R}$ above ι and o cannot be the same (otherwise it is an ax-node and reduction could not lead to the representation of a Boolean), so that $\mathcal{R}$ contains at least 2 nodes at depth 0 and thus at most $S-5$ nodes at depth 1 (and higher).

$\mathcal{R}_w$ contains a unique exponential redex which is a closed redex, so we can apply stubborn closed exponential reduction to it and obtain $\mathcal{R}'_w$ with no redex at depth 0 (using Fact 3). By Lemma 7 (with the number of ?-nodes at depth 0 bounded by $S-1$), we have:

$$\begin{pmatrix} 2S & 0 & 0 \\ SN & S & \infty \\ SN & \infty & \infty \end{pmatrix} \triangleright_0 \mathcal{R}'_w \qquad \text{and} \qquad \begin{pmatrix} SN & S & 0 \\ SN & \infty & \infty \\ SN & \infty & \infty \end{pmatrix} \triangleright_1 \mathcal{R}'_w$$

by Lemma 5 and since, due to the definition of $\overline{w}$, the cuts generated at depth 1 cannot be exponential redexes because they are all of the shape:



Let $\mathcal{R}''_w$ be the multiplicative normal form of $\mathcal{R}'_w$ at depth 1, by Lemma 6, we have:

$$\begin{pmatrix} SN & \infty & \infty \\ SN & \infty & \infty \\ SN & \infty & \infty \end{pmatrix} \triangleright_1 \mathcal{R}''_w \qquad \text{but also} \qquad \begin{pmatrix} SN & 0 & 3S \\ SN & \infty & \infty \\ SN & \infty & \infty \end{pmatrix} \triangleright_1 \mathcal{R}''_w$$

since $\mathcal{R}''_w$ contains no multiplicative redex at depth 1, and by definition of $\overline{w}$, each multiplicative redex at depth 1 in $\mathcal{R}'_w$ generates at most 3 exponential redexes at depth 1 in $\mathcal{R}''_w$: one for each premise of the $\parr$-nodes in the picture above.

By Lemma 3, we can apply Lemma 2 to $\mathcal{R}''_w$ and get an e-net $\mathcal{R}'''_w$ (with no cut at depth 1) after iterated stubborn closed exponential reduction at depth 1 in $\mathcal{R}''_w$. By Lemma 8, we have:

$$\begin{pmatrix} 2SN & 0 & 0 \\ (SN)^{3S+1} & \infty & \infty \\ (SN)^{3S+1} & \infty & \infty \end{pmatrix} \triangleright_1 \mathcal{R}'''_w \qquad \text{and} \qquad \begin{pmatrix} (SN)^{3S+1} & \infty & \infty \\ (SN)^{3S+1} & \infty & \infty \\ (SN)^{3S+1} & \infty & \infty \end{pmatrix} \triangleright_2 \mathcal{R}'''_w$$

by Lemma 5.

Let $\mathcal{R}^0_w$ be the multiplicative normal form of $\mathcal{R}'''_w$ at depth 2, by Lemma 6, we have:

$$\begin{pmatrix} (SN)^{3S+1} & 0 & \infty \\ (SN)^{3S+1} & \infty & \infty \\ (SN)^{3S+1} & \infty & \infty \end{pmatrix} \triangleright_2 \mathcal{R}^0_w$$

By Lemma 3 and Fact 3, $\mathcal{R}^0_w$ represents b.

The depth cannot increase during reduction (Fact 1) thus the depth of $\mathcal{R}^0_w$ is bounded by $S < SN$. This implies the size of $\mathcal{R}^0_w$ to be bounded by $(SN)^{3S+2}$ as well as the size of all the e-nets met during reduction.

The length of the reduction sequence is obtained through the bounds:

$$\mathcal{R}_w \xrightarrow{S \text{ (Lemma 7)}} \mathcal{R}'_w \xrightarrow{SN \text{ (Lemma 6)}} \mathcal{R}''_w \xrightarrow{SN \text{ (Lemma 8)}} \mathcal{R}'''_w \xrightarrow{(SN)^{3S+1} \text{ (Lemma 6)}} \mathcal{R}^0_w$$

thus it is bounded by $(SN)^{3S+2}$ as well:

$$S + SN + SN + (SN)^{3S+1} \leq 4(SN)^{3S+1} \leq (SN)^{3S+2}$$

As already mentioned, a constant time procedure then easily allows us to read the Boolean value b from (its representation) $\mathcal{R}^0_w$.

Both the size bound and the time bound are thus polynomial in the length l of the input. Such a reduction sequence can then be implemented by a PTIME Turing machine [2]. □

5. Completeness of the representation

We want to simulate PTIME Turing machines with e-nets. We will adapt the encoding of Turing machines from [15,9] to a non-affine setting. We really follow the same pattern while carefully introducing additional garbage collection at the appropriate places to deal with the strictly linear case.

It would be possible to give a direct encoding into e-nets but understanding how simulation works could be difficult. Since the encoding is compatible with a notion of recursive types with second-order quantification, we first give such a typed encoding. Types help understanding the encoding in a modular way, and can be erased without impacting the final computation.

Similarly our target is a classical system, but intuitionistic presentations may look more natural to the readers more comfortable with functional programming than with cut-elimination in one-sided systems for classical logics. Similarly, the input/output behavior of computation appears more clearly in intuitionistic frameworks.

Since defining a notion of intuitionistic typed nets would be tedious, we focus here on a intuitionistic sequent calculus. We can then reach our e-nets target by moving all formulas to the right side of $\vdash$ by means of (linear) negation (thus trivially embedding the intuitionistic system into a one-sided classical one), by translating rules into nodes (thus building graphs from proof trees), and by erasing types. The point of this detour is to make things easier to read with no consequence on the final objects we extract. We simply take advantage of the fact that the target of the encoding lies in a well structured subset of e-nets.

The logic we use is IELL^2_μ, a typed intuitionistic linear sequent calculus with second-order quantification and recursive types. Formulas of IELL^2_μ are given by:

$$A ::= \alpha \mid A \otimes A \mid A \multimap A \mid !A \mid \forall\alpha.A \mid \exists\alpha.A \mid \mu\alpha.A$$

The associated rules are:

$$\frac{}{A \vdash A}\, ax \qquad \frac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\, cut$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \otimes R \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \otimes L$$

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

$$\frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\, !C \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B}\, !W \qquad \frac{\Gamma \vdash A}{!\Gamma \vdash !A}\, !$$

$$\frac{\Gamma \vdash A \qquad \alpha \notin \Gamma}{\Gamma \vdash \forall \alpha . A} \forall R \qquad \frac{\Gamma, A[{}^{B}/_{\alpha}] \vdash C}{\Gamma, \forall \alpha . A \vdash C} \forall L$$

$$\frac{\Gamma \vdash A[{}^{B}/_{\alpha}]}{\Gamma \vdash \exists \alpha . A} \exists R \qquad \frac{\Gamma, A \vdash C \qquad \alpha \notin \Gamma, C}{\Gamma, \exists \alpha . A \vdash C} \exists L$$

$$\frac{\Gamma \vdash A[{}^{\mu\beta.A}/_{\beta}]}{\Gamma \vdash \mu\beta . A} \mu R \qquad \frac{\Gamma, A[{}^{\mu\beta.A}/_{\beta}] \vdash B}{\Gamma, \mu\beta . A \vdash B} \mu L$$

The presence of second order quantification and fixpoints in formulas makes IELL^2_μ a very powerful typed system allowing for the possibility of rich encodings.

In order to encode Turing machines, we first define representations of some data types: types are represented by appropriate formulas of IELL^2_μ and associated values, constructors or operations by sequent calculus derivations.

- **Booleans.** The formula for representing Booleans in our *linear* framework is: $\mathbb{B}_\ell = \forall \alpha . \alpha \multimap \alpha \multimap (\alpha \otimes \alpha)$.
  The proofs $\mathtt{0}_\ell$ and $\mathtt{1}_\ell$ (representing the two Boolean values) are obtained from the two possible choices of the splitting of the occurrences of α in:

$$\mathtt{0}_\ell, \mathtt{1}_\ell \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\, ax \qquad \dfrac{}{\alpha \vdash \alpha}\, ax}{\alpha, \alpha \vdash \alpha \otimes \alpha} \otimes R}{\vdash \alpha \multimap \alpha \multimap (\alpha \otimes \alpha)} \multimap R}{\vdash \mathbb{B}_\ell} \forall R$$

  These are the only two cut-free proofs of the formula $\mathbb{B}_\ell$.
- **Garbage.** The non-affine setting we use requires to introduce some garbage collection mechanism. For this we introduce the generic garbage type $\exists \gamma . \gamma$. It can be used in particular to build the following family of proofs collecting arbitrary garbage into $\exists \gamma . \gamma$:

$$\mathtt{gc} \quad = \quad \dfrac{\dfrac{\dfrac{}{A_1 \vdash A_1}\, ax \qquad \cdots \qquad \dfrac{}{A_n \vdash A_n}\, ax}{A_1, \ldots, A_n \vdash A_1 \otimes \cdots \otimes A_n} \otimes R}{A_1, \ldots, A_n \vdash \exists \gamma . \gamma} \exists R$$

- **Enumerated data types.** For $s \in \mathbb{N}$, a finite enumerated data-type containing s elements (but also with room for some garbage) can be represented by: $\mathbb{E}_s = \forall \alpha . \alpha \multimap \cdots \multimap \alpha \multimap (\alpha \otimes \exists \gamma . \gamma)$ (with s arrows).
  The proofs $\mathtt{state}_k$ $(1 \leq k \leq s)$ are obtained from the s possible choices of the selection of an occurrence of α in the left top-most axiom rule:

$$\mathtt{state}_k \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\, ax \qquad \dfrac{\mathtt{gc}}{\alpha, \ldots, \alpha \vdash \exists \gamma . \gamma}}{\alpha, \ldots, \alpha \vdash \alpha \otimes \exists \gamma . \gamma} \otimes R}{\vdash \alpha \multimap \cdots \multimap \alpha \multimap (\alpha \otimes \exists \gamma . \gamma)} \multimap R}{\vdash \mathbb{E}_s} \forall R$$

  We can do case distinction on the elements of $\mathbb{E}_s$ through the following derivable rule which produces garbage (if $\Gamma = B_1, \ldots, B_p$, we use the notation $\Gamma \multimap A$ for $B_1 \multimap \cdots \multimap B_p \multimap A$):

$$\frac{\Gamma\vdash A \quad \cdots \quad \Gamma\vdash A}{\mathbb{E}_s,\Gamma\vdash A\otimes\exists\gamma.\gamma}\,\mathbb{E}L \quad\equiv$$

$$\dfrac{\dfrac{\Gamma\vdash A}{\vdash\Gamma\multimap A}\multimap R \quad\cdots\quad \dfrac{\Gamma\vdash A}{\vdash\Gamma\multimap A}\multimap R \qquad \dfrac{\dfrac{\cdots \quad \dfrac{}{A\vdash A}\,ax}{\Gamma\multimap A,\Gamma\vdash A}\multimap L \qquad \dfrac{}{\exists\gamma.\gamma\vdash\exists\gamma.\gamma}\,ax}{\dfrac{\Gamma\multimap A,\exists\gamma.\gamma,\Gamma\vdash A\otimes\exists\gamma.\gamma}{(\Gamma\multimap A)\otimes\exists\gamma.\gamma,\Gamma\vdash A\otimes\exists\gamma.\gamma}\otimes L}\otimes R}{\dfrac{(\Gamma\multimap A)\multimap\cdots\multimap(\Gamma\multimap A)\multimap((\Gamma\multimap A)\otimes\exists\gamma.\gamma),\Gamma\vdash A\otimes\exists\gamma.\gamma}{\mathbb{E}_s,\Gamma\vdash A\otimes\exists\gamma.\gamma}\,\forall L}\multimap L$$

By introducing a cut between $\mathtt{state}_k$ and $\mathbb{E}L$ (and by eliminating it), one can recover the kth proof $\Gamma\vdash A$.

- **Booleans with garbage.** We can extend the representation of Booleans with garbage by: $\mathbb{B}=\forall\alpha.\alpha\multimap\alpha\multimap(\alpha\otimes\exists\gamma.\gamma)$ (note we have $\mathbb{B}=\mathbb{E}_2$).
  $\mathtt{0}$ and $\mathtt{1}$ are obtained from the two possible choices of the splitting of the occurrences of α in:

$$\mathtt{0,1} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{}{\alpha\vdash\alpha}\,ax \quad \dfrac{\dfrac{}{\alpha\vdash\alpha}\,ax}{\alpha\vdash\exists\gamma.\gamma}\,\exists R}{\alpha,\alpha\vdash\alpha\otimes\exists\gamma.\gamma}\otimes R}{\vdash\alpha\multimap\alpha\multimap(\alpha\otimes\exists\gamma.\gamma)}\multimap R}{\vdash\mathbb{B}}\,\forall R$$

They simply are $\mathtt{state}_1$ and $\mathtt{state}_2$.
The use of garbage in the type $\mathbb{B}$ allows us to collect garbage into $\mathbb{B}$ through:

$$\mathtt{gc}_{\mathbb{B}} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha\vdash\alpha}\,ax \quad \dfrac{}{\alpha\vdash\alpha}\,ax \qquad \dfrac{\dfrac{\mathtt{gc}}{\exists\gamma.\gamma,A_1,\ldots,A_n\vdash\exists\gamma.\gamma}}{\alpha,\exists\gamma.\gamma,A_1,\ldots,A_n\vdash\alpha\otimes\exists\gamma.\gamma}\otimes R}{\alpha\otimes\exists\gamma.\gamma,A_1,\ldots,A_n\vdash\alpha\otimes\exists\gamma.\gamma}\otimes L}{\alpha\multimap\alpha\multimap(\alpha\otimes\exists\gamma.\gamma),A_1,\ldots,A_n,\alpha,\alpha\vdash\alpha\otimes\exists\gamma.\gamma}\multimap L}{\mathbb{B},A_1,\ldots,A_n,\alpha,\alpha\vdash\alpha\otimes\exists\gamma.\gamma}\,\forall L}{\mathbb{B},A_1,\ldots,A_n\vdash\alpha\multimap\alpha\multimap(\alpha\otimes\exists\gamma.\gamma)}\multimap R}{\mathbb{B},A_1,\ldots,A_n\vdash\mathbb{B}}\,\forall R$$

Conversely, garbage can be extracted from an element of $\mathbb{B}$ to obtain a garbage-free Boolean in $\mathbb{B}_\ell$:

$$\mathtt{b2bl} \;=\; \dfrac{\overset{\mathtt{0}_\ell}{\vdash\mathbb{B}_\ell} \qquad \overset{\mathtt{1}_\ell}{\vdash\mathbb{B}_\ell}}{\mathbb{B}\vdash\mathbb{B}_\ell\otimes\exists\gamma.\gamma}\,\mathbb{E}L$$

- **Church numerals.** We represent Church numerals with the formula: $\mathbb{C}=\forall\alpha.!(\alpha\multimap\alpha)\multimap!(\alpha\multimap\alpha)$.
  For example, the Church numeral representation of 2 is:

$$\mathtt{2} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha\vdash\alpha}\,ax \quad \dfrac{\dfrac{}{\alpha\vdash\alpha}\,ax \quad \dfrac{}{\alpha\vdash\alpha}\,ax}{\alpha\multimap\alpha,\alpha\vdash\alpha}\multimap L}{\alpha\multimap\alpha,\alpha\multimap\alpha,\alpha\vdash\alpha}\multimap L}{\alpha\multimap\alpha,\alpha\multimap\alpha\vdash\alpha\multimap\alpha}\multimap R}{!(\alpha\multimap\alpha),!(\alpha\multimap\alpha)\vdash!(\alpha\multimap\alpha)}\,!}{!(\alpha\multimap\alpha)\vdash!(\alpha\multimap\alpha)}\,!C}{\vdash!(\alpha\multimap\alpha)\multimap!(\alpha\multimap\alpha)}\multimap R}{\vdash\mathbb{C}}\,\forall R$$

Let P be a polynomial, it can be represented as a proof of $!\mathbb{C}\vdash!\mathbb{C}$, meaning that if $\mathtt{n}$ represents the Church numeral n, the following proof reduces to the proof associated with $P(n)$ followed by a promotion rule ($!$):

$$\dfrac{\dfrac{\overset{\mathtt{n}}{\vdash\mathbb{C}}}{\vdash!\mathbb{C}}\,! \qquad \overset{\mathtt{P}}{!\mathbb{C}\vdash!\mathbb{C}}}{\vdash!\mathbb{C}}\,cut \quad\to^*\quad \dfrac{\overset{\mathtt{P(n)}}{\vdash\mathbb{C}}}{\vdash!\mathbb{C}}\,!$$

The key ingredients for representing P are the representations of the addition and multiplication functions in $!\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}$ obtained by adding a promotion rule (!) to:

$$\texttt{add} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\,ax \quad \dfrac{\dfrac{}{\alpha \vdash \alpha}\,ax \quad \dfrac{}{\alpha \vdash \alpha}\,ax}{\alpha, \alpha \multimap \alpha \vdash \alpha}\multimap L}{\alpha \multimap \alpha, \alpha \multimap \alpha, \alpha \vdash \alpha}\multimap L}{\alpha \multimap \alpha, \alpha \multimap \alpha \vdash \alpha \multimap \alpha}\multimap R}{!(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,! \quad \dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax}{!(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap L}{!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap L}{!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,!C}{!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,\forall L}{\mathbb{C}, \mathbb{C}, !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap R}{\mathbb{C}, \mathbb{C} \vdash !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)}\,\forall R}{\mathbb{C}, \mathbb{C} \vdash \mathbb{C}}$$

and

$$\texttt{mul} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax \quad \dfrac{\dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax \quad \dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax}{!(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap L}{!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap L}{\mathbb{C}, \mathbb{C}, !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,\forall L}{\mathbb{C}, \mathbb{C} \vdash !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)}\multimap R}{\mathbb{C}, \mathbb{C} \vdash \mathbb{C}}\,\forall R$$

For example the polynomial $X^3 + 2X = X \times X \times X + X + X$ can be represented by:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{mul}}{\mathbb{C}, \mathbb{C} \vdash \mathbb{C}}}{!\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}}\,! \quad \dfrac{\dfrac{\texttt{mul}}{\mathbb{C}, \mathbb{C} \vdash \mathbb{C}}}{!\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}}\,!}{!\mathbb{C}, !\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}}\,cut \quad \dfrac{\dfrac{\texttt{add}}{\mathbb{C}, \mathbb{C} \vdash \mathbb{C}}}{!\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}}\,!}{!\mathbb{C}, !\mathbb{C}, !\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}}\,cut \quad \dfrac{\dfrac{\texttt{add}}{\mathbb{C}, \mathbb{C} \vdash \mathbb{C}}}{!\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}}\,!}{!\mathbb{C}, !\mathbb{C}, !\mathbb{C}, !\mathbb{C}, !\mathbb{C} \vdash !\mathbb{C}}\,cut}{!\mathbb{C} \vdash !\mathbb{C}}\,!C$$

- **Church binary words.** Similarly, we represent (Church-style) binary words with the formula: $\mathbb{W} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$.
  The two operations of pushing an element (0 or 1) on top of a word are $\texttt{cons}_0^{\mathbb{W}}$ and $\texttt{cons}_1^{\mathbb{W}}$ given by:

$$\texttt{cons}_0^{\mathbb{W}}, \texttt{cons}_1^{\mathbb{W}} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\,ax \quad \dfrac{\dfrac{}{\alpha \vdash \alpha}\,ax \quad \dfrac{}{\alpha \vdash \alpha}\,ax}{\alpha, \alpha \multimap \alpha \vdash \alpha}\multimap L}{\alpha \multimap \alpha, \alpha \multimap \alpha, \alpha \vdash \alpha}\multimap L}{\alpha \multimap \alpha, \alpha \multimap \alpha \vdash \alpha \multimap \alpha}\multimap R}{!(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,! \quad \dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax}{!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap L}{!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap L}{\mathbb{W}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,\forall L}{\mathbb{W}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,!C}{\mathbb{W} \vdash !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)}\multimap R}{\mathbb{W} \vdash \mathbb{W}}\,\forall R$$

  depending on which $!(\alpha \multimap \alpha)$ is duplicated in the $!C$-rule. The empty word is represented by:

$$\mathtt{nil}^{\mathbb{W}} \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\,ax}{\vdash \alpha \multimap \alpha}\multimap R}{\vdash !(\alpha \multimap \alpha)}\,!}{!(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,!W}{\vdash !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)}\multimap R}{\vdash \mathbb{W}}\,\forall R$$

Given two step functions and a base case on a type A, it is possible to define an iteration taking an argument of type $\mathbb{W}$ and building $!A$. This comes as a derivable rule:

$$\dfrac{A \vdash A \quad A \vdash A \quad \vdash A}{\mathbb{W} \vdash !A}\,\mathtt{iter}^{\mathbb{W}} \quad \equiv$$

$$\dfrac{\dfrac{\dfrac{A \vdash A}{\vdash A \multimap A}\multimap R}{\vdash !(A \multimap A)}\,! \quad \dfrac{\dfrac{A \vdash A}{\vdash A \multimap A}\multimap R}{\vdash !(A \multimap A)}\,! \quad \dfrac{\vdash A \quad \dfrac{}{A \vdash A}\,ax}{A \multimap A \vdash A}\multimap L}{!(A \multimap A)\vdash !A}\,!}{!(A \multimap A) \multimap !(A \multimap A) \multimap !(A \multimap A) \vdash !A}\multimap L}{\mathbb{W} \vdash !A}\,\forall L$$

The length of a binary word can be obtained as an element of $\mathbb{C}$ through the proof `length` of $\mathbb{W} \vdash \mathbb{C}$:

$$\mathtt{length} \quad =$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax \quad \dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax \quad \dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax}{!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\multimap L}{\mathbb{W}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,\forall L}{\mathbb{W}, !(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,!C}{\mathbb{W} \vdash !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)}\multimap R}{\mathbb{W} \vdash \mathbb{C}}\,\forall R$$

We could try to use iteration of the successor function but in this case the conclusion would be $\mathbb{W} \vdash !\mathbb{C}$ instead.

- **Scott binary words.** We also need a Scott-style representation of binary words (or stacks) using a recursive type definition and incorporating garbage. This is done with the formula (which uses all the three binders allowed in formulas):

$$\mathbb{S} = \mu\beta.\forall\alpha.(\beta \multimap \alpha) \multimap (\beta \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)$$

which intuitively satisfies $\mathbb{S} = \forall\alpha.(\mathbb{S} \multimap \alpha) \multimap (\mathbb{S} \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)$.
The two operations of pushing an element (0 or 1) on top of a word $\mathtt{cons}_0^{\mathbb{S}}$ and $\mathtt{cons}_1^{\mathbb{S}}$ are obtained from the two possible choices of the splitting of the occurrences of $\mathbb{S} \multimap \alpha$ in:

$$\mathtt{cons}_0^{\mathbb{S}}, \mathtt{cons}_1^{\mathbb{S}} \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{S} \vdash \mathbb{S}}\,ax \quad \dfrac{}{\alpha \vdash \alpha}\,ax}{\mathbb{S}, \mathbb{S} \multimap \alpha \vdash \alpha}\multimap L \quad \dfrac{\mathtt{gc}}{\mathbb{S} \multimap \alpha, \alpha \vdash \exists\gamma.\gamma}}{\mathbb{S}, \mathbb{S} \multimap \alpha, \mathbb{S} \multimap \alpha, \alpha \vdash \alpha \otimes \exists\gamma.\gamma}\otimes R}{\mathbb{S} \vdash (\mathbb{S} \multimap \alpha) \multimap (\mathbb{S} \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)}\multimap R}{\mathbb{S} \vdash \forall\alpha.(\mathbb{S} \multimap \alpha) \multimap (\mathbb{S} \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)}\,\forall R}{\mathbb{S} \vdash \mathbb{S}}\,\mu R$$

and the empty word is given by:

$$\mathtt{nil}^{\mathbb{S}} \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\,ax \quad \dfrac{\mathtt{gc}}{\mathbb{S} \multimap \alpha, \mathbb{S} \multimap \alpha \vdash \exists\gamma.\gamma}}{\mathbb{S} \multimap \alpha, \mathbb{S} \multimap \alpha, \alpha \vdash \alpha \otimes \exists\gamma.\gamma}\otimes R}{\vdash (\mathbb{S} \multimap \alpha) \multimap (\mathbb{S} \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)}\multimap R}{\vdash \forall\alpha.(\mathbb{S} \multimap \alpha) \multimap (\mathbb{S} \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)}\,\forall R}{\vdash \mathbb{S}}\,\mu R$$

Separating a word into its head (first element in $\mathbb{B}$) and tail (other elements in $\mathbb{S}$) is obtained with the `pop` proof of $\mathbb{S} \vdash \mathbb{B} \otimes \mathbb{S}$:

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

$$\mathtt{pop} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{\mathtt{0}}{\vdash \mathbb{B}} \qquad \dfrac{}{\mathbb{S}\vdash\mathbb{S}}\,ax}{\mathbb{S}\vdash \mathbb{B}\otimes\mathbb{S}}\,\otimes R}{\vdash \mathbb{S}\multimap \mathbb{B}\otimes\mathbb{S}}\,\multimap R \qquad \dfrac{\dfrac{\overset{\mathtt{1}}{\vdash \mathbb{B}} \qquad \dfrac{}{\mathbb{S}\vdash\mathbb{S}}\,ax}{\mathbb{S}\vdash \mathbb{B}\otimes\mathbb{S}}\,\otimes R}{\vdash \mathbb{S}\multimap \mathbb{B}\otimes\mathbb{S}}\,\multimap R \qquad \dfrac{\overset{\mathtt{0}}{\vdash \mathbb{B}} \qquad \overset{\mathtt{nil}^{\mathbb{S}}}{\vdash \mathbb{S}}}{\vdash \mathbb{B}\otimes\mathbb{S}}\,\otimes R \qquad \dfrac{\dfrac{\overset{\mathtt{gc}_{\mathbb{B}}}{\mathbb{B},\exists\gamma.\gamma\vdash\mathbb{B}} \qquad \dfrac{}{\mathbb{S}\vdash\mathbb{S}}\,ax}{\mathbb{B},\mathbb{S},\exists\gamma.\gamma\vdash\mathbb{B}\otimes\mathbb{S}}\,\otimes R}{(\mathbb{B}\otimes\mathbb{S})\otimes\exists\gamma.\gamma\vdash\mathbb{B}\otimes\mathbb{S}}\,\otimes L}{(\mathbb{S}\multimap\mathbb{B}\otimes\mathbb{S})\multimap(\mathbb{S}\multimap\mathbb{B}\otimes\mathbb{S})\multimap(\mathbb{B}\otimes\mathbb{S})\multimap((\mathbb{B}\otimes\mathbb{S})\otimes\exists\gamma.\gamma)\vdash\mathbb{B}\otimes\mathbb{S}}\,\multimap L}{\forall\alpha.(\mathbb{S}\multimap\alpha)\multimap(\mathbb{S}\multimap\alpha)\multimap\alpha\multimap(\alpha\otimes\exists\gamma.\gamma)\vdash\mathbb{B}\otimes\mathbb{S}}\,\forall L}{\mathbb{S}\vdash\mathbb{B}\otimes\mathbb{S}}\,\mu L$$

In case of an empty input, it answers `0` and an empty tail.

- **From Church to Scott.** A translation from the Church-style representation of binary words $\mathbb{W}$ into their Scott-style representation $\mathbb{S}$ can be obtained by iteration:

$$\mathtt{w2s} \;=\; \dfrac{\overset{\mathtt{cons}_0^{\mathbb{S}}}{\mathbb{S}\vdash\mathbb{S}} \qquad \overset{\mathtt{cons}_1^{\mathbb{S}}}{\mathbb{S}\vdash\mathbb{S}} \qquad \overset{\mathtt{nil}^{\mathbb{S}}}{\vdash\mathbb{S}}}{\mathbb{W}\vdash !\mathbb{S}}\,\mathtt{iter}^{\mathbb{W}}$$

- **Configurations.** The representation of the execution of Turing machines with two symbols and s states is based on the following formula for configurations: $\text{Config} = \mathbb{S}\otimes\mathbb{B}\otimes\mathbb{S}\otimes\mathbb{E}_s$. The first occurrence of $\mathbb{S}$ represents the left part of the tape (in reverse order), $\mathbb{B}$ represents the value of the current cell, the second occurrence of $\mathbb{S}$ represents the right part of the tape, and $\mathbb{E}_s$ represents the current state.
- **Initial configuration.** Assuming that 1 is the initial state, and given the representation in $\mathbb{S}$ of a binary word, one can build the initial configuration by a cut with:

$$\mathtt{init} = \dfrac{\overset{\mathtt{nil}^{\mathbb{S}}}{\vdash\mathbb{S}} \qquad \overset{\mathtt{pop}}{\mathbb{S}\vdash\mathbb{B}\otimes\mathbb{S}} \qquad \overset{\mathtt{state}_1}{\vdash\mathbb{E}_s}}{\mathbb{S}\vdash\text{Config}}\,\otimes R$$

- **Transition function.** A transition of the machine is computed through the proof `step` of $\text{Config}\vdash\text{Config}$:

$$\mathtt{step} \;=\; \dfrac{\dfrac{\dfrac{\cdots \qquad \dfrac{\overset{\mathtt{transit}_0^k}{\mathbb{S},\mathbb{S}\vdash\text{Config}} \qquad \overset{\mathtt{transit}_1^k}{\mathbb{S},\mathbb{S}\vdash\text{Config}}}{\mathbb{S},\mathbb{B},\mathbb{S}\vdash\text{Config}\otimes\exists\gamma.\gamma}\,\mathbb{E}L \qquad \cdots}{\mathbb{S},\mathbb{B},\mathbb{S},\mathbb{E}_s\vdash\text{Config}\otimes\exists\gamma.\gamma\otimes\exists\gamma.\gamma}\,\mathbb{E}L}{\text{Config}\vdash\text{Config}\otimes\exists\gamma.\gamma\otimes\exists\gamma.\gamma}\,\otimes L \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{S}\vdash\mathbb{S}}\,ax \qquad \overset{\mathtt{gc}_{\mathbb{B}}}{\mathbb{B},\exists\gamma.\gamma,\exists\gamma.\gamma\vdash\mathbb{B}} \qquad \dfrac{}{\mathbb{S}\otimes\mathbb{E}_s\vdash\mathbb{S}\otimes\mathbb{E}_s}\,ax}{\mathbb{S},\mathbb{B},\mathbb{S}\otimes\mathbb{E}_s,\exists\gamma.\gamma,\exists\gamma.\gamma\vdash\text{Config}}\,\otimes R}{\text{Config},\exists\gamma.\gamma,\exists\gamma.\gamma\vdash\text{Config}}\,\otimes L}{\text{Config}\otimes\exists\gamma.\gamma\otimes\exists\gamma.\gamma\vdash\text{Config}}\,\otimes L}{\text{Config}\vdash\text{Config}}\,cut$$

The proofs $\mathtt{transit}_i^k$ encode the transition table of the Turing machine:
  - if, in state k, while reading value i, the machine goes to state k', writes i', and moves its head to the right, then:

$$\mathtt{transit}_i^k \;=\; \dfrac{\overset{\mathtt{cons}_{i'}^{\mathbb{S}}}{\mathbb{S}\vdash\mathbb{S}} \qquad \overset{\mathtt{pop}}{\mathbb{S}\vdash\mathbb{B}\otimes\mathbb{S}} \qquad \overset{\mathtt{state}_{k'}}{\vdash\mathbb{E}_s}}{\mathbb{S},\mathbb{S}\vdash\text{Config}}\,\otimes R$$

  - if, in state k, while reading value i, the machine goes to state k', writes i', and moves its head to the left, then:

$$\mathtt{transit}_i^k \;=\; \dfrac{\dfrac{\overset{\mathtt{pop}}{\mathbb{S}\vdash\mathbb{B}\otimes\mathbb{S}} \qquad \dfrac{\dfrac{\dfrac{}{\mathbb{S}\vdash\mathbb{S}}\,ax \qquad \dfrac{}{\mathbb{B}\vdash\mathbb{B}}\,ax}{\mathbb{B},\mathbb{S}\vdash\mathbb{S}\otimes\mathbb{B}}\,\otimes R}{\mathbb{B}\otimes\mathbb{S}\vdash\mathbb{S}\otimes\mathbb{B}}\,\otimes L}{\mathbb{S}\vdash\mathbb{S}\otimes\mathbb{B}}\,cut \qquad \overset{\mathtt{cons}_{i'}^{\mathbb{S}}}{\mathbb{S}\vdash\mathbb{S}} \qquad \overset{\mathtt{state}_{k'}}{\vdash\mathbb{E}_s}}{\mathbb{S},\mathbb{S}\vdash\text{Config}}\,\otimes R$$

- **Execution.** Given a number n represented in $\mathbb{C}$ and a binary word w in $!\mathbb{S}$ (using a promotion if necessary), we can run the machine for n steps from the initial state associated with w by using cuts with:

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

$$\texttt{run} \quad = \quad \dfrac{\dfrac{\dfrac{\overset{\texttt{step}}{\textsc{Config} \vdash \textsc{Config}}}{\vdash \textsc{Config} \multimap \textsc{Config}}\multimap R}{\vdash !(\textsc{Config} \multimap \textsc{Config})}\,! \qquad \dfrac{\dfrac{\overset{\texttt{init}}{\mathbb{S} \vdash \textsc{Config}} \qquad \dfrac{}{\textsc{Config} \vdash \textsc{Config}}\,ax}{\textsc{Config} \multimap \textsc{Config}, \mathbb{S} \vdash \textsc{Config}}\multimap L}{!(\textsc{Config} \multimap \textsc{Config}), !\mathbb{S} \vdash !\textsc{Config}}\,!}{\dfrac{!(\textsc{Config} \multimap \textsc{Config}) \multimap !(\textsc{Config} \multimap \textsc{Config}), !\mathbb{S} \vdash !\textsc{Config}}{\mathbb{C}, !\mathbb{S} \vdash !\textsc{Config}}\forall L}\multimap L$$

This is iteration over $\mathbb{C}$.

- **Acceptance.** Assuming the state s is the accepting one, the acceptance of a configuration is tested through the proof:

$$\texttt{accept} \quad = \quad \dfrac{\dfrac{\dfrac{\overset{\texttt{0}}{\vdash \mathbb{B}} \quad \cdots \quad \overset{\texttt{0}}{\vdash \mathbb{B}} \quad \overset{\texttt{1}}{\vdash \mathbb{B}}}{\mathbb{E}_s \vdash \mathbb{B} \otimes \exists \gamma.\gamma}\,\mathbb{E}L \qquad \dfrac{\overset{\texttt{gc}_{\mathbb{B}}}{\mathbb{S}, \mathbb{B}, \mathbb{S}, \mathbb{B}, \exists \gamma.\gamma \vdash \mathbb{B}}}{\mathbb{S}, \mathbb{B}, \mathbb{S}, \mathbb{B} \otimes \exists \gamma.\gamma \vdash \mathbb{B}}\otimes L}{\mathbb{S}, \mathbb{B}, \mathbb{S}, \mathbb{E}_s \vdash \mathbb{B}}\,cut}{\textsc{Config} \vdash \mathbb{B}}\otimes L$$

- **Whole computation.** The result of the evaluation of the PTIME Turing machine on an input represented in $!\mathbb{W}$ is finally obtained by a cut with:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{\texttt{length}}{\mathbb{W} \vdash \mathbb{C}}}{!\mathbb{W} \vdash !\mathbb{C}}\,! \qquad \overset{\texttt{P}}{!\mathbb{C} \vdash !\mathbb{C}}}{!\mathbb{W} \vdash !\mathbb{C}}\,cut \qquad \dfrac{\dfrac{\overset{\texttt{w2s}}{\mathbb{W} \vdash !\mathbb{S}} \qquad \overset{\texttt{run}}{\mathbb{C}, !\mathbb{S} \vdash !\textsc{Config}}}{\mathbb{C}, \mathbb{W} \vdash !\textsc{Config}}\,cut}{!\mathbb{C}, !\mathbb{W} \vdash !!\textsc{Config}}\,! \qquad \dfrac{\dfrac{\overset{\texttt{accept}}{\textsc{Config} \vdash \mathbb{B}}}{!\textsc{Config} \vdash !\mathbb{B}}\,!}{!!\textsc{Config} \vdash !!\mathbb{B}}\,! \qquad \dfrac{\dfrac{\overset{\texttt{b2bl}}{\mathbb{B} \vdash \mathbb{B}_\ell \otimes \exists \gamma.\gamma}}{!\mathbb{B} \vdash !(\mathbb{B}_\ell \otimes \exists \gamma.\gamma)}\,!}{!!\mathbb{B} \vdash !!(\mathbb{B}_\ell \otimes \exists \gamma.\gamma)}\,!}{!\mathbb{W}, !\mathbb{W} \vdash !!(\mathbb{B}_\ell \otimes \exists \gamma.\gamma)}\,cut}{!\mathbb{W} \vdash !!(\mathbb{B}_\ell \otimes \exists \gamma.\gamma)}\,!C$$

The input is used once to compute (through P) the required number of execution steps and another time to turn it into its Scott-style representation used to build the initial state of the machine.

If we erase garbage, we find back the same encoding as in [9], thus cut elimination simulates the evaluation of the encoded Turing machine.

Theorem 3 (PTIME *completeness*). *If $\mathcal{P}$ is a* PTIME *decision problem, there exists an e-net which represents $\mathcal{P}$.*

Proof. As described above, we can associate with a PTIME Turing machine computing $\mathcal{P}$, a sequent calculus proof of $!\mathbb{W} \vdash !!(\mathbb{B}_\ell \otimes \exists \gamma.\gamma)$ in IELL^2_μ.

We can turn this sequent calculus proof into an e-net. Nodes are associated with rules through the following mapping:

$$\dfrac{}{A \vdash A}\,ax \quad \mapsto \quad ax\text{-node}$$

$$\dfrac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\,cut \quad \mapsto \quad cut\text{-node}$$

$$\dfrac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\otimes R \quad \mapsto \quad \otimes\text{-node}$$

$$\dfrac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\otimes L \quad \mapsto \quad \parr\text{-node}$$

$$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\multimap R \quad \mapsto \quad \parr\text{-node}$$

$$\dfrac{\Gamma \vdash A \qquad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\multimap L \quad \mapsto \quad \otimes\text{-node}$$

$$\dfrac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\,!C \quad \mapsto \quad ?c\text{-node}$$

$$\frac{\Gamma \vdash B}{\Gamma, !A \vdash B}\, !W \quad \mapsto \quad ?w\text{-node}$$

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A}\, ! \quad \mapsto \quad !\text{-node and } ?p\text{-nodes defining a box}$$

Rules for $\forall$, $\exists$ and μ are simply ignored in the translation.

One can check (by induction on the proof) the generated e-net to satisfy the correctness criterion.

This transforms an IELL^2_μ proof of $!\mathbb{W}$ representing a binary word w into its e-net representation $\overline{w}$ (and conversely, the e-net $\overline{w}$ can be turned into a proof of $!\mathbb{W}$). Similarly any proof of $!!(\mathbb{B}_\ell \otimes \exists\gamma.\gamma)$ is turned into an e-net representing a Boolean. This means we obtain an e-net which represents $\mathcal{P}$ since the e-net reduction simulates the sequent calculus cut-elimination. □

6. Beyond PTIME

In [9], not only PTIME is considered but a whole exponential hierarchy. It is possible to get similar results here.

We define the tower of exponentials 2^x_k of base 2 and height k by:

$$2^x_0 = x \qquad\qquad 2^x_{k+1} = 2^{(2^x_k)}$$

Lemma 9. *If $x \geq 3$, $(2^x_k)^{2^x_k+1} \leq 2^{x^2}_{k+1}$.*

Proof. By induction on k. For $k = 0$, we have $x^{x+1} \leq 2^{x^2}$. Then we have the following deduction sequence:

$$\begin{aligned}
2^n + 1 &\leq n^n && \text{if } n \geq 3\\
n(2^n + 1) &\leq n^{n+1} && \text{if } n \geq 3\\
2^x_k(2^x_{k+1} + 1) &\leq (2^x_k)^{2^x_k+1} && \text{choose } n = 2^x_k\\
2^x_k(2^x_{k+1} + 1) &\leq 2^{x^2}_{k+1} && \text{induction hypothesis}\\
(2^x_{k+1})^{2^x_{k+1}+1} &\leq 2^{x^2}_{k+2}
\end{aligned}$$

□

The complexity class k-EXPTIME is the class of decision problems computable on a deterministic Turing machine in time $O\left(2^{P(n)}_k\right)$ (for some polynomial P) for words of size n. In particular 0-EXPTIME = PTIME, 1-EXPTIME is the class of exponential time problems, and ELEMENTARY $= \bigcup_{k\in\mathbb{N}} k$-EXPTIME.

6.1. Representations

Let us generalize the notions of representations of Section 3.

Definition 6 *(k-representation of Booleans).* An e-net *k-represents* the Boolean 0 (resp. 1) if it has the shape presented on the left (resp. on the right) of Fig. 9, with exactly $k+2$!-nodes below the $\otimes$-node.

In particular the notion of Boolean representation of Section 3.1 corresponds to 0-representation here.

Definition 7 *(k-representation of decision problems).* Let $\mathcal{P} : \mathbf{W} \to \mathbf{B}$ be a decision problem on binary words, the e-net $\mathcal{R}$ *k-represents* $\mathcal{P}$ if:

- $\mathcal{R}$ is an e-net with two conclusions ι and o,
- for any $w \in \mathbf{W}$, $\mathcal{R} \,{}_\iota\!\bowtie \overline{w}$ reduces to a k-representation of $\mathcal{P}(w)$.

This leads to the following generalization of Theorems 2 and 3:

Theorem 4 *(k-EXPTIME representation). $\mathcal{P}$ is a k-EXPTIME decision problem if and only if there exists an e-net which k-represents $\mathcal{P}$.*

The proof is explained in the following two sections. We go faster than for the PTIME case since the arguments are extremely similar.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

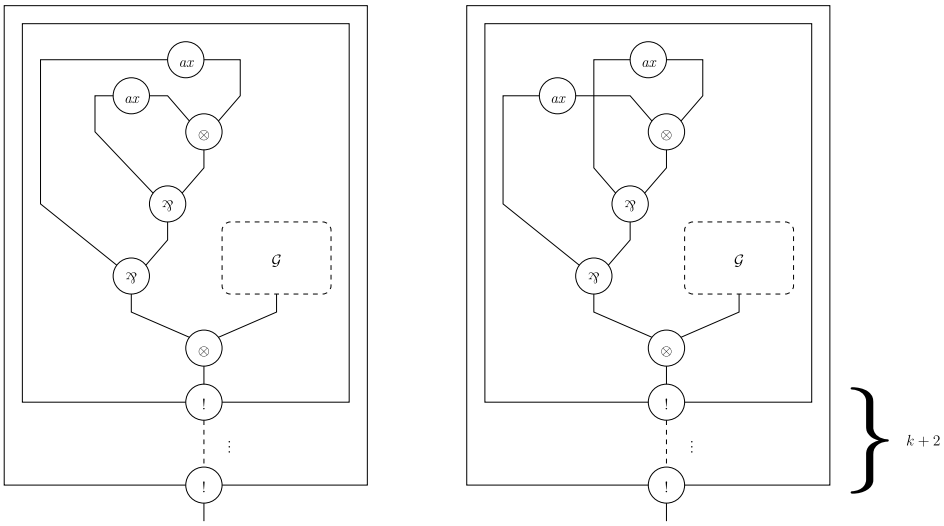


Fig. 9. k-representation of Booleans (0 and 1 respectively).

6.2. Completeness

On the completeness side (Theorem 3), the only missing ingredients are the possibility of representing any function $x \mapsto 2_k^{P(x)}$ as a proof with conclusion $!\mathbb{C} \vdash !^{k+1}\mathbb{C}$, and the existence of a coercion proof with conclusion $\mathbb{W} \vdash !^k\mathbb{W}$ which acts as the identity on its argument (see [9]).

Let us consider the following proof:

$$\texttt{exp} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha)}\,ax}{\vdash !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)}\multimap R}{\vdash !\mathbb{C}}\forall R \qquad \dfrac{}{\mathbb{C} \vdash \mathbb{C}}\,ax}{\mathbb{C} \multimap \mathbb{C} \vdash \mathbb{C}}\multimap L}{!(\mathbb{C} \multimap \mathbb{C}) \vdash !\mathbb{C}}\,!}{!(\mathbb{C} \multimap \mathbb{C}) \multimap !(\mathbb{C} \multimap \mathbb{C}) \vdash !\mathbb{C}}\multimap L}{\mathbb{C} \vdash !\mathbb{C}}\forall L$$

$$\dfrac{\dfrac{}{\vdash \mathbb{C}}\,\texttt{2} \qquad \dfrac{}{\mathbb{C}, \mathbb{C} \vdash \mathbb{C}}\,\texttt{mul}}{\mathbb{C} \vdash \mathbb{C}}\,cut \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathbb{C} \multimap \mathbb{C}}\multimap R}{\vdash !(\mathbb{C} \multimap \mathbb{C})}\,!}{!(\mathbb{C} \multimap \mathbb{C}) \vdash !\mathbb{C}}\,!}{!(\mathbb{C} \multimap \mathbb{C}) \multimap !(\mathbb{C} \multimap \mathbb{C}) \vdash !\mathbb{C}}\multimap L$$

The function $x \mapsto 2_0^x = x$ is represented by $\dfrac{}{!\mathbb{C} \vdash !\mathbb{C}}\,ax$, and assuming by induction that we have a representation of $x \mapsto 2_k^x$ with conclusion $!\mathbb{C} \vdash !^{k+1}\mathbb{C}$, we represent $x \mapsto 2_{k+1}^x = 2^{(2_k^x)}$ by:

$$\dfrac{\dfrac{}{!\mathbb{C} \vdash !^{k+1}\mathbb{C}}\,2_k^x \qquad \dfrac{\dfrac{}{\mathbb{C} \vdash !\mathbb{C}}\,\texttt{exp}}{!^{k+1}\mathbb{C} \vdash !^{k+2}\mathbb{C}}\,!}{!\mathbb{C} \vdash !^{k+2}\mathbb{C}}\,cut$$

$x \mapsto 2_k^{P(x)}$ is then obtained by composing (through a cut) the representation of P in $!\mathbb{C} \vdash !\mathbb{C}$ with this representation of $x \mapsto 2_k^x$.

A coercion (identity function) from $\mathbb{W}$ to $!\mathbb{W}$ can be obtained by:

$$\texttt{coer} \;=\; \dfrac{\dfrac{}{\mathbb{W} \vdash \mathbb{W}}\,\texttt{cons}_0^{\mathbb{W}} \qquad \dfrac{}{\mathbb{W} \vdash \mathbb{W}}\,\texttt{cons}_1^{\mathbb{W}} \qquad \dfrac{}{\vdash \mathbb{W}}\,\texttt{nil}^{\mathbb{W}}}{\mathbb{W} \vdash !\mathbb{W}}\,\texttt{iter}^{\mathbb{W}}$$

from which we can deduce a proof $\texttt{coer}^k$ of $\mathbb{W} \vdash !^k\mathbb{W}$ for any k by using the induction step:

$$\texttt{coer}^{k+1} \;=\; \dfrac{\dfrac{}{\mathbb{W} \vdash !\mathbb{W}}\,\texttt{coer} \qquad \dfrac{\dfrac{}{\mathbb{W} \vdash !^k\mathbb{W}}\,\texttt{coer}^k}{!\mathbb{W} \vdash !^{k+1}\mathbb{W}}\,!}{\mathbb{W} \vdash !^{k+1}\mathbb{W}}\,cut$$

The execution of a k-EXPTIME Turing machine is then simulated by:

$$
\dfrac{
\dfrac{
\dfrac{\overset{\texttt{length}}{\mathbb{W}\vdash\mathbb{C}}}{!\mathbb{W}\vdash !\mathbb{C}}\,!
\qquad
\overset{2_k^{P(x)}}{!\mathbb{C}\vdash !^{k+1}\mathbb{C}}
}{!\mathbb{W}\vdash !^{k+1}\mathbb{C}}\,cut
\qquad
\dfrac{\overset{\texttt{coer}^k}{\mathbb{W}\vdash !^k\mathbb{W}}}{!\mathbb{W}\vdash !^{k+1}\mathbb{W}}\,!
\qquad
\dfrac{
\dfrac{
\overset{\texttt{w2s}}{\mathbb{W}\vdash !\mathbb{S}}
\quad
\overset{\texttt{run}}{\mathbb{C}, !\mathbb{S}\vdash !\textsc{Config}}
\quad
\overset{\texttt{accept}}{\dfrac{\textsc{Config}\vdash\mathbb{B}}{!\textsc{Config}\vdash !\mathbb{B}}}\,!
\quad
\overset{\texttt{b2bl}}{\dfrac{\mathbb{B}\vdash\mathbb{B}_\ell\otimes\exists\gamma.\gamma}{!\mathbb{B}\vdash !(\mathbb{B}_\ell\otimes\exists\gamma.\gamma)}}\,!
}{\mathbb{C}, \mathbb{W}\vdash !(\mathbb{B}_\ell\otimes\exists\gamma.\gamma)}\,cut
}{!^{k+1}\mathbb{C}, !^{k+1}\mathbb{W}\vdash !^{k+2}(\mathbb{B}_\ell\otimes\exists\gamma.\gamma)}\,!
}{
\dfrac{!\mathbb{W}, !\mathbb{W}\vdash !^{k+2}(\mathbb{B}_\ell\otimes\exists\gamma.\gamma)}{!\mathbb{W}\vdash !^{k+2}(\mathbb{B}_\ell\otimes\exists\gamma.\gamma)}\,!C
}\,cut
$$

6.3. *Soundness*

On the soundness side, we follow the same pattern as for Theorem 2.

Let $\mathcal{R}$ be a normal e-net k-representing the problem $\mathcal{P}$ and let w be a binary word. We study the reduction of the e-net $\mathcal{R}_w = \mathcal{R}\ {}_\iota\!\bowtie \overline{w}$. The analysis at depth 0 and 1 is exactly the same for any k. Following the first part of the proof of Theorem 2, we get:

$$\begin{pmatrix} (SN)^{3S+1} & 0 & \infty \\ (SN)^{3S+1} & \infty & \infty \\ (SN)^{3S+1} & \infty & \infty \end{pmatrix} \triangleright_2 \mathcal{R}_w^0$$

with $P_0(l) = (SN)^{3S+1}$ polynomial in l. Moreover $\mathcal{R}_w^0$ has only redexes at depth 2 (which are all exponential), and it reduces to the k-representation of a Boolean.

Let us assume $k \geq 1$ and that we have an e-net $\mathcal{R}_w^n$ $(0 \leq n \leq k-1)$ with:

$$\begin{pmatrix} 2_n^{P_n(l)} & 0 & \infty \\ 2_n^{P_n(l)} & \infty & \infty \\ 2_n^{P_n(l)} & \infty & \infty \end{pmatrix} \triangleright_{n+2} \mathcal{R}_w^n$$

for some polynomial $P_n(l)$, with $\mathcal{R}_w^n$ having redexes at depth $n+2$ only (thus all exponential) and which reduces to the k-representation of a Boolean.

We can apply Lemma 2 and by iterated stubborn closed exponential reduction at depth $n+2$ we get some $\mathcal{R}'^n_w$ with by Lemmas 8, 9 and 5:

$$\begin{pmatrix} \left(2_n^{P_n(l)}\right)^{2_n^{P_n(l)}+1} & \infty & \infty \\ \left(2_n^{P_n(l)}\right)^{2_n^{P_n(l)}+1} & \infty & \infty \\ \left(2_n^{P_n(l)}\right)^{2_n^{P_n(l)}+1} & \infty & \infty \end{pmatrix} \triangleright_{n+2} \mathcal{R}'^n_w \quad \text{and} \quad \begin{pmatrix} 2_{n+1}^{P_{n+1}(l)} & \infty & \infty \\ 2_{n+1}^{P_{n+1}(l)} & \infty & \infty \\ 2_{n+1}^{P_{n+1}(l)} & \infty & \infty \end{pmatrix} \triangleright_{n+3} \mathcal{R}'^n_w$$

for $P_{n+1}(l) = P_n(l)^2$, with $\mathcal{R}'^n_w$ having cuts at depth $n+3$ only.

Let $\mathcal{R}_w^{n+1}$ be its multiplicative normal form at depth $n+3$, we have:

$$\begin{pmatrix} 2_{n+1}^{P_{n+1}(l)} & 0 & \infty \\ 2_{n+1}^{P_{n+1}(l)} & \infty & \infty \\ 2_{n+1}^{P_{n+1}(l)} & \infty & \infty \end{pmatrix} \triangleright_{n+3} \mathcal{R}_w^{n+1}$$

By induction we reach:

$$\begin{pmatrix} 2_k^{P_k(l)} & 0 & \infty \\ 2_k^{P_k(l)} & \infty & \infty \\ 2_k^{P_k(l)} & \infty & \infty \end{pmatrix} \triangleright_{k+2} \mathcal{R}_w^k$$

for some polynomial $P_k(l)$.

The sizes of all e-nets from $\mathcal{R}_w$ to $\mathcal{R}_w^k$ are then bounded by some $2_k^{P(l)}$ with P a polynomial in l.

Concerning the length of the reduction sequence, we have:

$$\mathcal{R}_w \xrightarrow{S\ (\text{Lemma } 7)} \mathcal{R}'_w \xrightarrow{SN\ (\text{Lemma } 6)} \mathcal{R}''_w \xrightarrow{SN\ (\text{Lemma } 8)} \mathcal{R}'''_w \xrightarrow{(SN)^{3S+1}\ (\text{Lemma } 6)} \mathcal{R}_w^0$$

$$\cdots\quad \mathcal{R}_w^n \xrightarrow{2_n^{P_n(l)}\ (\text{Lemma } 8)} \mathcal{R}'^n_w \xrightarrow{2_{n+1}^{P_{n+1}(l)}\ (\text{Lemma } 6)} \mathcal{R}_w^{n+1} \quad\cdots\qquad (0 \leq n \leq k-1)$$

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002

whose total length is bounded by $2_k^{P(l)}$ with P a polynomial in l.

We conclude as in Theorem 2 that such a reduction sequence with bounded length and bounded sizes can then be implemented by a k-EXPTIME Turing machine.

7. Conclusion

We have presented the computational model of e-nets: untyped proofs-nets for classical elementary linear logic. We have shown how they characterize PTIME computation (Theorems 2 and 3), and more generally k-EXPTIME computation (Theorem 4). The main novelties are the absence of types and the non-affine framework. We have used types as an intermediary tool to make the representation of PTIME Turing machines more understandable, but it is also possible to directly build the e-net simulating the execution of a given machine, without any reference to types. The restricted erasure power of the model requires to do some garbage management along the simulation, but it has no direct impact on the reading of the final result.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to thank Patrick Baillot for various discussions during the development of this work, and the anonymous referees for their suggestions for improving the presentation.

References

[1] J.-Y. Girard, Linear logic, Theor. Comput. Sci. 50 (1987) 1–102.
[2] J.-Y. Girard, Light linear logic, Inf. Comput. 143 (2) (1998) 175–204.
[3] Y. Lafont, Soft linear logic and polynomial time, Theor. Comput. Sci. 318 (1–2) (2004) 163–180.
[4] V. Danos, J.-B. Joinet, Linear logic and elementary time, Inf. Comput. 183 (1) (2003) 123–137.
[5] M. Gaboardi, J.-Y. Marion, S. Ronchi Della Rocca, An implicit characterization of PSPACE, ACM Trans. Comput. Log. 13 (2) (2012) 18:1–18:36.
[6] P. Baillot, Stratified coherent spaces: a denotational semantics for light linear logic, Theor. Comput. Sci. 318 (1–2) (2004) 29–55.
[7] O. Laurent, L. Tortora de Falco, Obsessional cliques: a semantic characterization of bounded time complexity, in: Proceedings of the Twenty-First Annual Symposium on Logic in Computer Science, IEEE, IEEE Computer Society Press, Seattle, 2006, pp. 179–188.
[8] O. Laurent, On the categorical semantics of elementary linear logic, Theory Appl. Categ. 22 (10) (2009) 269–301.
[9] P. Baillot, On the expressivity of elementary linear logic: characterizing Ptime and an exponential time hierarchy, Inf. Comput. 241 (2015) 3–31.
[10] A. Asperti, Light affine logic, in: Proceedings of the Thirteenth Annual Symposium on Logic in Computer Science, IEEE, IEEE Computer Society Press, Indianapolis, 1998, pp. 300–308.
[11] J.-Y. Girard, Y. Lafont, P. Taylor, Proofs and Types, Cambridge Tracts in Theoretical Computer Science, vol. 7, Cambridge University Press, 1989.
[12] A. Asperti, L. Roversi, Intuitionistic light affine logic, ACM Trans. Comput. Log. 3 (1) (2002) 1–39.
[13] H. Mairson, K. Terui, On the computational complexity of cut-elimination in linear logic, in: Proceedings of the Eighth Italian Conference on Theoretical Computer Science (ICTCS), in: Lecture Notes in Computer Science, vol. 2841, Springer, 2003, pp. 23–36.
[14] K. Terui, Proof nets and Boolean circuits, in: Proceedings of the Nineteenth Annual Symposium on Logic in Computer Science, IEEE, IEEE Computer Society Press, Turku, 2004, pp. 182–191.
[15] U. Dal Lago, P. Baillot, On light logics, uniform encodings and polynomial time, Math. Struct. Comput. Sci. 16 (4) (2006) 713–733.
[16] L.T.D. Nguyen, Around finite second-order coherence spaces, Available at http://arxiv.org/abs/1902.00196, 2019.
[17] D. de Carvalho, M. Pagani, L. Tortora de Falco, A semantic measure of the execution time in linear logic, Theor. Comput. Sci. 412 (20) (2011) 1884–1902.
[18] V. Danos, L. Regnier, The structure of multiplicatives, Arch. Math. Log. 28 (1989) 181–203.
[19] V. Danos, La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul), Thèse de doctorat, Université Paris VII, 1990.
[20] U. Dal Lago, Context semantics, linear logic and computational complexity, ACM Trans. Comput. Log. 10 (4) (2009) 25:1–25:32.
[21] Terese, Term Rewriting Systems, Cambridge Tracts in Theoretical Computer Science, vol. 55, Cambridge University Press, 2003.

Please cite this article in press as: O. Laurent, Polynomial time in untyped elementary linear logic, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.10.002