Polynomial Time in Untyped Elementary Linear Logic

Olivier . Laurent @ens-lyon.fr

April 19, 2013

Abstract

We show how to represent polynomial time computation in an untyped version of proof-nets of elementary linear logic.

Following a question of J.-Y. Girard asking whether a restriction on the types of elementary linear logic (ELL [Gir98]) could represent exactly polynomial time computation (PTIME), P. Baillot [Bai11] shown that proofs of $!\mathbb{W} \multimap !!\mathbb{B}_{\mathtt{a}}$ (with $\mathbb{W} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$ and $\mathbb{B}_{\mathtt{a}} = \forall\alpha.\alpha \multimap \alpha \multimap \alpha$) in second-order intuitionistic elementary affine logic enriched with type fix-points exactly represent PTIME problems. We prove a similar result in untyped classical elementary linear logic. This stresses the standard fact that types are useless for the control of complexity in elementary linear logic. Moreover it shows that linear proofs are enough (the general weakening principle of affine logics is not required).

1 Untyped Elementary Proof-Nets

1.1 E-Nets

An *e-net* is a directed acyclic graph with loose edges. Incoming edges of a node are called its *premises* and its outgoing edges are its *conclusions*. The loose edges are called the *conclusions* of the e-net. Each node has an associated kind:

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

With each !-node is associated a sub-graph with no premise, called its *box*, such that all its conclusions are premises of $?p$-node (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 hat 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 [DR89]. A *correctness graph* is the undirected graph obtained by selecting the content of a box (or all the nodes at depth 0) as an undirected graph, by putting edges between each !-node and the $?p$-nodes which

are auxiliary doors of its box (and the content of the box itself is erased), 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.

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.

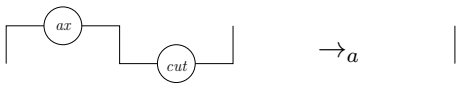
By convention, if an e-net has depth D, when counting nodes at depth strictly smaller than 0 or strictly bigger than D, we consider there are 0 nodes at those depths.

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 union of $\mathcal{R}_1$ and $\mathcal{R}_2$, a *cut*-node with premises c_1 and c_2 (when c_1 or c_2 are immediate to infer from the context (the e-net has a unique conclusion for example), we will sometimes omit them).

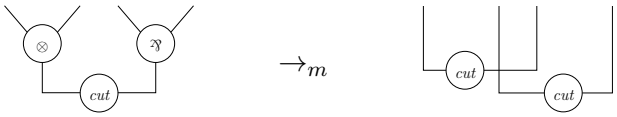
1.2 Reduction

Reduction occurs around *cut*-nodes. 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:

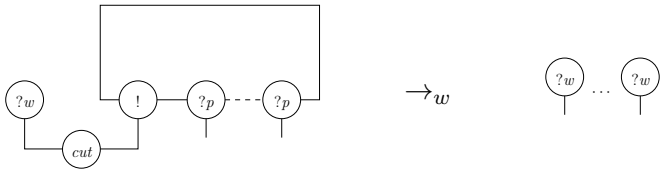
- a-step:



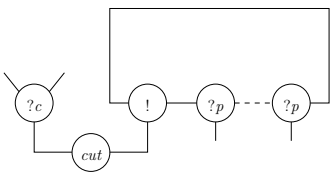
- m-step:

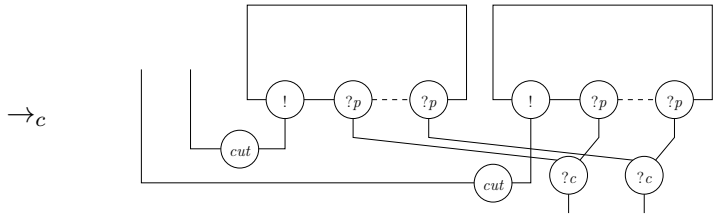


- w-step:

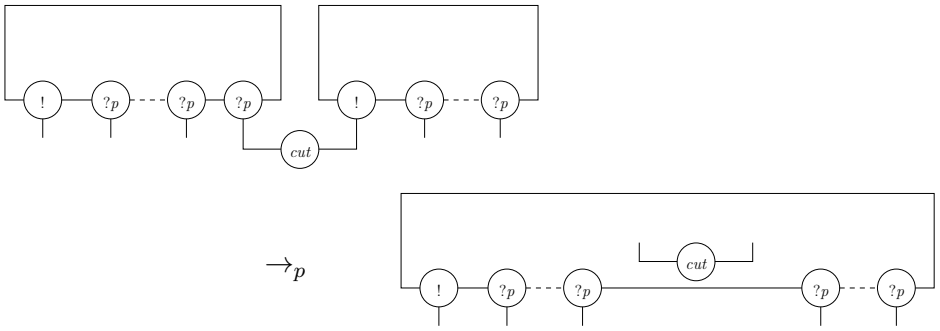


- c-step:





- p-step:



A reducible *cut*-node is called a *redex*. Non reducible cuts are called *clashes*.

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*.

For any $d \in \mathbb{N}$, if s is the size of $\mathcal{R}$ at depth d, m is the number of multiplicative redexes at depth d, e is the number of exponential redexes at depth d and c is the number of clashes at depth d, we have $m + e + c \leq s$.

1.3 Main Properties

We focus here on *qualitative* properties. *Quantitative* properties concerning the size of e-nets, the reduction lengths, ... will be studied in Section 3.

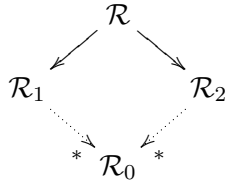
Fact 1 (Non Increasing Depth): If $\mathcal{R}$ reduces to $\mathcal{R}'$, either they have the same depth or the reduction is a w-step and the depth of $\mathcal{R}'$ may be strictly smaller than the depth of $\mathcal{R}$.

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.

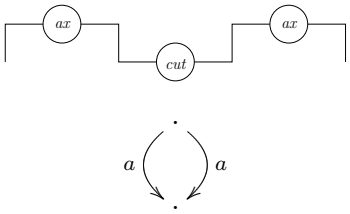
Fact 4 (Persistence of Clashes): If c is a clash in the e-net $\mathcal{R}$ which reduces to $\mathcal{R}'$, either c persists in $\mathcal{R}'$ or the reduction is a w-step and c belongs to the erased box.

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

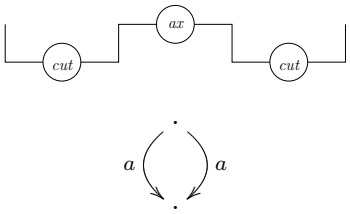


PROOF: We have the following critical pairs to consider:

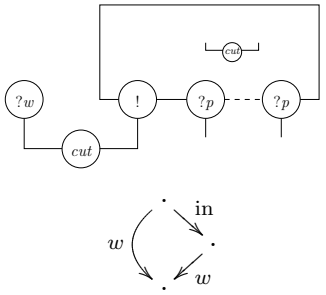
- a/a (shared cut)



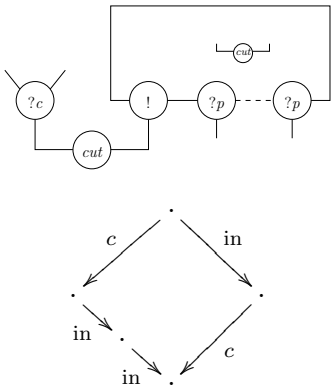
- a/a (shared ax)



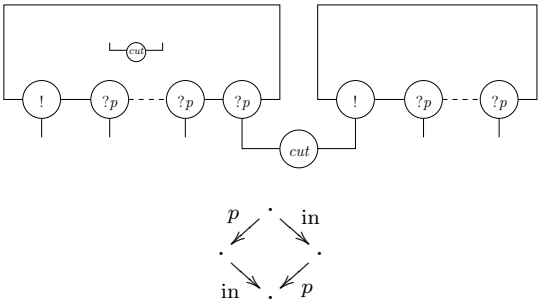
- w/in



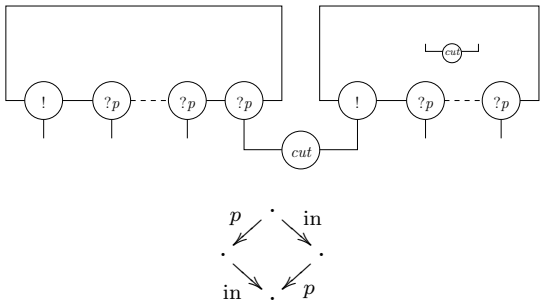
- c/in



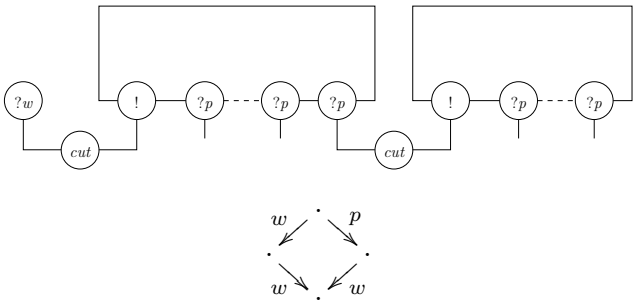
- p/in (left side)



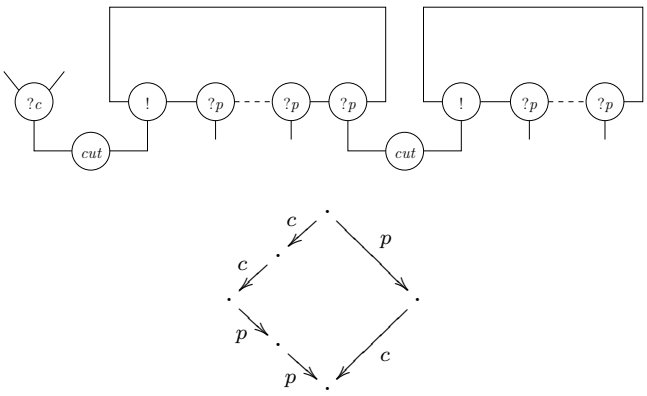
- p/in (right side)



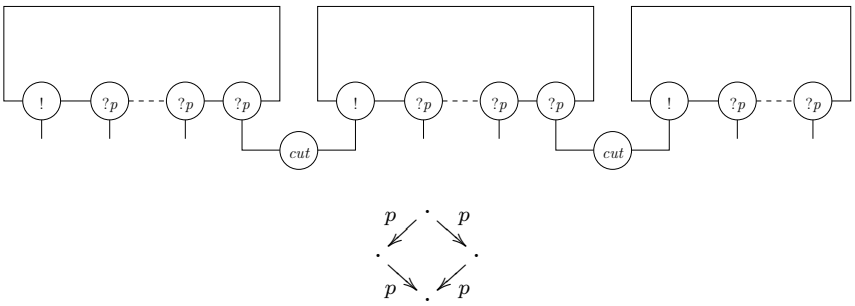
- w/p



- c/p



- p/p



□

An !-node o is *immediately reachable* from a ?-node n if by going down from n (*i.e.* by following the orientation of edges), one crosses only $?c$-nodes and one reaches a cut whose

other premise is the conclusion of o. By transitivity, o_k is *reachable* from n_1 if there is a sequence $n_1 o_1 n_2 o_2 \dots n_k o_k$ (called a *reachability path*) where o_i is immediately reachable from n_i ($1 \leq i \leq k$) and n_{i+1} is an auxiliary door of the box of o_i ($1 \leq i \leq k-1$). Thanks to the correctness criterion, in such a reachability path $n_1 o_1 n_2 o_2 \dots n_k o_k$, o_i is not reachable from n_j if $i < j$ (otherwise one would obtain a cycle in a correctness graph). In particular the number reachability paths starting from n_i is strictly bigger than the number of reachability paths starting from n_j if $i < j$.

Proposition 2 (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 an element $|\mathcal{R}|_{\mathrm{sn}} = (|\mathcal{R}|^D_{\mathrm{sn}}, \dots, |\mathcal{R}|^0_{\mathrm{sn}})$ of the ordinal:

$$\underbrace{(\omega^\omega \cdot \omega) \cdots (\omega^\omega \cdot \omega)}_{D+1 \text{ times}}$$

(thus $\omega^{(\omega+1)\cdot(D+1)}$). Remember that in products of ordinals, the element with bigger weight is the right-most one, and that ω^α is (isomorphic to) the set of finite multisets of elements of α. For each $0 \leq d \leq D$, $|\mathcal{R}|^d_{\mathrm{sn}}$ is a pair (μ_d, m_d) where:

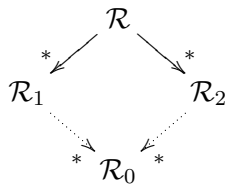
- μ_d is the multiset of the numbers of reachability paths starting from each ?-node at depth d (note that a given !-node might be reachable from the same ?-node through two different reachability paths),
- and m_d is the number of *ax*, $\otimes$ and $\parr$-nodes at depth d in $\mathcal{R}$.

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 values 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)).

We prove that any reduction step makes $|\mathcal{R}|_{\mathrm{sn}}$ decrease, and thus reduction always terminates. According to Fact 2, if reduction occurs at depth d, it is enough to focus on $(|\mathcal{R}|^i_{\mathrm{sn}})_{d \leq i}$. We prove for each possible reduction step at depth d that $|\mathcal{R}|^d_{\mathrm{sn}}$ 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 m_d as well. 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 m_d as well. Concerning μ_d, let us first recall that the value 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 m_d as well. 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. □

Proposition 3 (Confluence)
The reduction of e-nets is confluent:



PROOF: By Propositions 1 and 2 with Newman's Lemma [Ter03]. □

1.4 Closed Reduction

An exponential redex is called *closed* if the box involved has no auxiliary door.

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 (hereditarily) above a conclusion (thus they are all hereditarily above cuts), then $\mathcal{R}$ contains a closed exponential redex at depth 0.

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. Thanks to the correctness criterion again, the reachability relation does not contain cycles. Starting from the exponential redex, one can consider an !-node such that none of the auxiliary doors of its box can reach 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 and to generate no clash. One easily checks that all the reduced redexes are then 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 in $\mathcal{R}$ anymore.

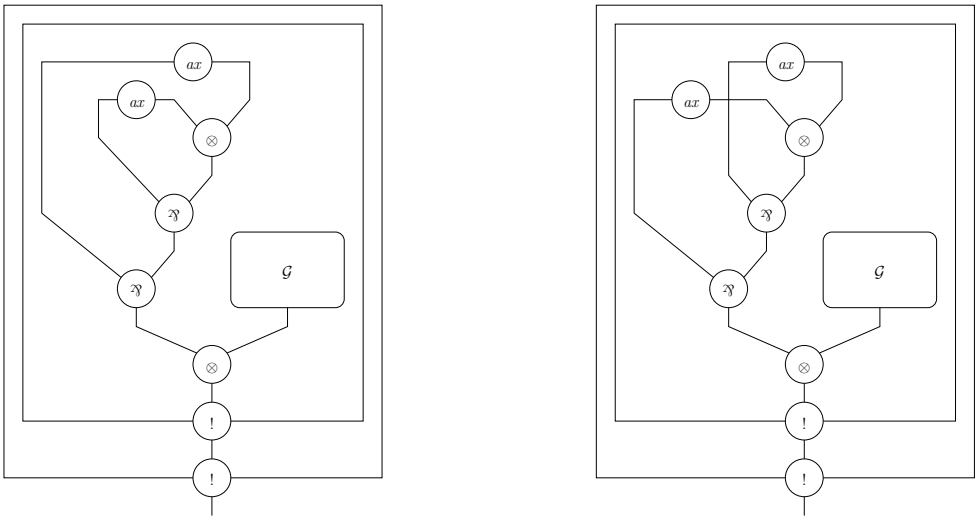
2 Representation of Problems

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.

2.1 Booleans

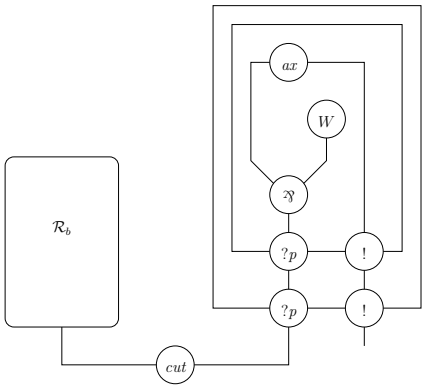
We do not define a unique representation of booleans, but a set of e-nets associated with each boolean value. This will be justified by the fact that our linear (non affine) representation of computation will require some garbage part of the computation to be collected in the boolean results.

The families of e-nets representing 0 and 1 are respectively (for any e-net $\mathcal{G}$):



Fact 5 (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.

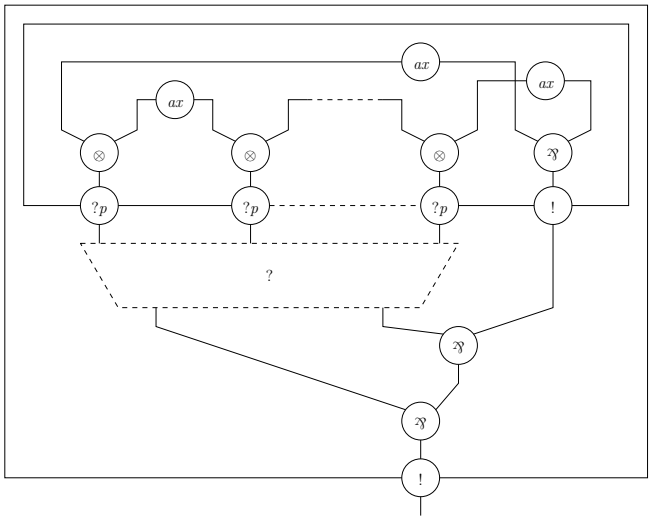
Remark: In the affine setting of [Bai11] for example, one can extract a unique representation of booleans by building the following e-net from some representation $\mathcal{R}_b$ of a boolean b:



where W is the general weakening node (used to erase the garbage part $\mathcal{G}$ in $\mathcal{R}_b$).

2.2 Binary Words

Let $w \in \mathbf{W}$ be a binary word, its *representation* is the cut-free e-net $\overline{w}$ built as follows:



where 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.

2.3 Problems

Let $\mathcal{P} : \mathbf{W} \to \mathbf{B}$ be a problem on binary words, the e-net $\mathcal{R}$ *represents* $\mathcal{P}$ if:

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

3 Correctness of the Complexity Bound

We here give quantitative bounds on the reduction of e-nets.

Definition 1 (Weight Matrix)
Let $\mathcal{R}$ be an e-net of depth D, if $0 \leq d \leq D$, a *weight matrix* $\mathcal{M}$ *at depth* d of $\mathcal{R}$ is a 4×4 matrix with coefficients in $\mathbb{N} \cup \{\infty\}$:

$$\begin{pmatrix} s_- & m_- & e_- & c_- \\ s_0 & m_0 & e_0 & c_0 \\ s_1 & m_1 & e_1 & c_1 \\ s_+ & m_+ & e_+ & c_+ \end{pmatrix}$$

such that:

- For any depth $d' < d$, the size of $\mathcal{R}$ at depth d' is smaller or equal to s_-, the number of multiplicative redexes at depth d' is smaller or equal to m_-, the number of exponential redexes at depth d' is smaller or equal to e_-, and the number of clashes at depth d' is smaller or equal to c_-.
- For $i \in \{0, 1\}$, the size of $\mathcal{R}$ at depth $d + i$ is smaller or equal to s_i, the number of multiplicative redexes at depth $d + i$ is smaller or equal to m_i, the number of exponential redexes at depth $d + i$ is smaller or equal to e_i, and the number of clashes at depth $d + i$ is smaller or equal to c_i.

- For any depth $d' > d+1$, the size of $\mathcal{R}$ at depth d' is smaller or equal to s_+, the number of multiplicative redexes at depth d' is smaller or equal to m_+, the number of exponential redexes at depth d' is smaller or equal to e_+, and the number of clashes at depth d' is smaller or equal to c_+.

(Remind that sizes, numbers of nodes, ... are defined to be 0 at depths not occurring in $\mathcal{R}$, see Section 1.1)

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

Lemma 2 (Iterated Multiplicative Weight Reduction)
If $\mathcal{R}$ reduces to $\mathcal{R}'$ by a sequence of multiplicative reduction steps at depth d then:

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

Moreover the number of reduction steps is at most s_0.

Proof:

- Nothing is modified at depth strictly smaller than d (Fact 2).
- The size at depth d can only decrease.
- The nodes (and thus the sizes) at depths strictly bigger than d are not modified. □

Lemma 3 (Stubborn Closed Exponential Weight Reduction)
Assuming $\mathcal{R}$ reduces to $\mathcal{R}'$ by stubborn closed exponential reduction at depth d,

$$\begin{pmatrix} \infty & 0 & 0 & 0 \\ s_0 & 0 & e_0 & \infty \\ s_1 & m_1 & \infty & \infty \\ s_+ & \infty & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R} \quad \Longrightarrow \quad \begin{pmatrix} \infty & 0 & 0 & 0 \\ s_0 & 0 & e_0 - 1 & \infty \\ s_0 s_1 & s_0(m_1+1) & \infty & \infty \\ s_0 s_+ & \infty & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}'$$

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 s_0.

Proof:

- A closed reduction step at depth d removes a ?-node at depth d, so the total length of the reduction cannot be bigger than s_0.
- Nothing is modified at depth strictly smaller than d (Fact 2).
- Eventually all copies of the box under consideration in the reduction sequence are pushed at bigger depth, and some ?-nodes have been erased so the sizes at depth d after reduction is bounded by s_0.
- 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 cut is removed.
- There are at most s_0 steps and each one adds at most one copy (and at bigger depth) of the box concerned by the reduction sequence.
- For the same reason, one can move from m_1 multiplicative cuts at depth $d+1$ to $s_0 m_1$ multiplicative cuts at depth $d+1$, but also each copy of the box may come with a new multiplicative cut at depth $d+1$. □

Lemma 4 (Iterated Stubborn Closed Exponential Weight Reduction)
Assuming $\mathcal{R}$ reduces to $\mathcal{R}'$ with no exponential redex at depth d by iterated stubborn closed exponential reduction at depth d,

$$\begin{pmatrix} \infty & 0 & 0 & 0 \\ s_0 & 0 & e_0 & \infty \\ s_1 & \infty & \infty & \infty \\ s_+ & \infty & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R} \quad \Longrightarrow \quad \begin{pmatrix} \infty & 0 & 0 & 0 \\ \infty & 0 & 0 & \infty \\ s_0^{e_0} s_1 & \infty & \infty & \infty \\ s_0^{e_0} s_+ & \infty & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}'$$

and the number of reduction steps is bounded by s_0.

PROOF: Each time we apply a stubborn closed exponential reduction step from $\mathcal{R}_1$ to $\mathcal{R}_2$, Lemma 3 gives:

$$\begin{pmatrix} \infty & 0 & 0 & 0 \\ s_0 & 0 & e_0 & \infty \\ s_1 & \infty & \infty & \infty \\ s_+ & \infty & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R} \quad \Longrightarrow \quad \begin{pmatrix} \infty & 0 & 0 & 0 \\ s_0 & 0 & e_0 - 1 & \infty \\ s_0 s_1 & \infty & \infty & \infty \\ s_0 s_+ & \infty & \infty & \infty \end{pmatrix} \triangleright_d \mathcal{R}'$$

so we obtain the result by iterating e_0 times.

Concerning the number of steps, by Lemma 3, each reduction step makes the number of ?-nodes at depth d decrease so the total number of reduction steps is bounded by the initial size at depth d. □

Theorem 1 (PTIME Correctness)
If the e-net $\mathcal{R}$ represents the problem $\mathcal{P}$ then $\mathcal{P}$ is in PTIME.

PROOF: If $w \in \mathbf{W}$ of length l, we note $\mathcal{R}_w = \overline{w} \bowtie_\iota \mathcal{R}$. Since $\mathcal{R}_w$ normalizes to the representation $\mathcal{R}_b$ of a boolean, it reduces to an e-net with no cut at depths 0 and 1 and exactly one !-node (and no other node) at these depths. By Facts 2 and 4, no clash will appear at depth 0 and 1 during reduction.

Let S be the size of $\mathcal{R}$ plus 1 and $N = S + 4l + 4$, we have:

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

$\mathcal{R}_w$ contains a unique exponential cut which is a closed redex, so we can apply stubborn closed exponential reduction to it and obtain $\mathcal{R}'_w$ with no exponential redex (using that no clash is generated at depth 0 and Fact 3). By Lemma 3, we have:

$$\begin{pmatrix} \infty & 0 & 0 & 0 \\ \infty & 0 & 0 & \infty \\ SN & S & \infty & \infty \\ SN & \infty & \infty & \infty \end{pmatrix} \triangleright_0 \mathcal{R}'_w$$

and also

$$\begin{pmatrix} \infty & 0 & 0 & 0 \\ SN & S & 0 & \infty \\ SN & \infty & \infty & \infty \\ SN & \infty & \infty & \infty \end{pmatrix} \triangleright_1 \mathcal{R}'_w$$

since no clash is generated at depth 0 and due to the definition of $\overline{w}$, the cuts generated at depth 1 cannot be exponential redexes.

Let $\mathcal{R}''_w$ be the multiplicative normal form of $\mathcal{R}'_w$ at depth 1, by Lemma 2, we have:

$$\begin{pmatrix} \infty & 0 & 0 & 0 \\ SN & \infty & \infty & \infty \\ SN & \infty & \infty & \infty \\ SN & \infty & \infty & \infty \end{pmatrix} \triangleright_1 \mathcal{R}''_w$$

but also more precisely

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

since $\mathcal{R}''_w$ contains no multiplicative cut at depth 1, no clash is generated at depth 1, and by definition of $\overline{w}$, each multiplicative cut at depth 1 in $\mathcal{R}'_w$ generates at most 3 exponential cuts at depth 1 in $\mathcal{R}''_w$.

Since $\mathcal{R}_b$ contains only an !-node at depths 0 and 1, by Fact 2, $\mathcal{R}''_w$ contains no ?-node at depth 0 and no ?-node at depth 1 above a conclusion (otherwise they would persist along reduction). By Lemma 1 applied to the depth 1 part of $\mathcal{R}''_w$, one can apply iterated stubborn closed exponential reduction at depth 1 and obtain $\mathcal{R}'''_w$. By Lemma 4, we have:

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

and also

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

since no clash is generated at depth 1.

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

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

and finally:

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

since $\mathcal{R}^0_w$ contains no multiplicative cut at depth 2 and by using Fact 2 ($\mathcal{R}_b$ contains a unique node at depths 0 and 1).

The depth cannot increase during reduction (Fact 1) thus the depth of $\mathcal{R}^0_w$ is bounded by $N \leq SN$. 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 meet during reduction.

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

$$\mathcal{R}_w \xrightarrow{S \text{ (Lemma 3)}} \mathcal{R}'_w \xrightarrow{SN \text{ (Lemma 2)}} \mathcal{R}''_w \xrightarrow{SN \text{ (Lemma 4)}} \mathcal{R}'''_w \xrightarrow{(SN)^{3S+1} \text{ (Lemma 2)}} \mathcal{R}^0_w$$

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

Both the size bound and the time bound are thus polynomial in the length l of the input. Such a reduction can then be implemented by a PTIME Turing machine [Gir98].

To conclude, notice that $\mathcal{R}^0_w$ represents b since it has common reducts with $\mathcal{R}_b$ (Fact 5 and Proposition 3), no cut at depths 0 and 1 and no multiplicative cut at depth 2. A constant time procedure then easily allows us to read the boolean value b from (its representation) $\mathcal{R}^0_w$. □

4 Completeness of the Representation

We want to represent PTIME Turing machines with e-nets. We will adapt the encoding of Turing machines from [DLB06, Bai11] to a non affine setting. In order to help the reader, we will first use a typed intuitionistic linear sequent calculus (with second order quantification and recursive types) as a target of the encodings. E-nets can then be extracted from these sequent calculus proofs.

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

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

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

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

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

- The linear type for booleans is given by $\mathbb{B}_1 = \forall\alpha.\alpha \multimap \alpha \multimap (\alpha \otimes \alpha)$.

  The proofs 0_1 and 1_1 are obtained from the two possible choices of the splitting of the occurrences of α in:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha} \quad \dfrac{}{\alpha \vdash \alpha}}{\alpha, \alpha \vdash \alpha \otimes \alpha}}{\vdash \alpha \multimap \alpha \multimap (\alpha \otimes \alpha)}}{\vdash \mathbb{B}_1}$$

- The non affine setting we use requires to introduce some garbage collection through the generic garbage type $\exists\gamma.\gamma$. In particular we have the following family of proofs:

$$\mathsf{gc} = \dfrac{\dfrac{\dfrac{}{A_1 \vdash A_1} \quad \cdots \quad \dfrac{}{A_n \vdash A_n}}{A_1, \ldots, A_n \vdash A_1 \otimes \cdots \otimes A_n}}{A_1, \ldots, A_n \vdash \exists\gamma.\gamma}$$

- For $s \in \mathbb{N}$, finite enumerated data-types containing s elements (with garbage) are 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:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha} \quad \dfrac{\mathtt{gc}}{\alpha, \ldots, \alpha \vdash \exists\gamma.\gamma}}{\alpha, \ldots, \alpha \vdash \alpha \otimes \exists\gamma.\gamma}}{\vdash \alpha \multimap \cdots \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)}}{\vdash \mathbb{E}_s}$$

  For case distinction on the elements of $\mathbb{E}_s$, we have the following derivable rule (which produces garbage):

$$\dfrac{\Gamma_1 \vdash A \quad \cdots \quad \Gamma_s \vdash A}{\mathbb{E}_s, \Gamma_1, \ldots, \Gamma_s \vdash A \otimes \exists\gamma.\gamma} \quad \equiv \quad \dfrac{\dfrac{\Gamma_1 \vdash A \quad \cdots \quad \Gamma_s \vdash A \quad \overline{A \otimes \exists\gamma.\gamma \vdash A \otimes \exists\gamma.\gamma}}{A \multimap \cdots \multimap A \multimap (A \otimes \exists\gamma.\gamma), \Gamma_1, \ldots, \Gamma_s \vdash A \otimes \exists\gamma.\gamma}}{\mathbb{E}_s, \Gamma_1, \ldots, \Gamma_s \vdash A \otimes \exists\gamma.\gamma}$$

- The type for booleans with garbage is $\mathbb{B} = \forall\alpha.\alpha \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)$ (note we have $\mathbb{B} = \mathbb{E}_2$).

  `0` and `1` are obtained from the two possible choices of the splitting of the occurrences of α in:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha} \quad \dfrac{\overline{\alpha \vdash \alpha}}{\alpha \vdash \exists\gamma.\gamma}}{\alpha, \alpha \vdash \alpha \otimes \exists\gamma.\gamma}}{\vdash \alpha \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)}}{\vdash \mathbb{B}}$$

  They simply are $\mathtt{state}_1$ and $\mathtt{state}_2$.

  The use of garbage in the type $\mathbb{B}$ allows us to garbage collect into $\mathbb{B}$ through:

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

  Conversely, garbage can be extracted from an element of $\mathbb{B}$ to obtain a garbage-free boolean in $\mathbb{B}_1$:

$$\mathtt{b2bl} = \dfrac{\mathtt{0}_1 \quad \mathtt{1}_1}{\mathbb{B} \vdash \mathbb{B}_1 \otimes \exists\gamma.\gamma}$$

- We represent Church numerals with the type $\mathbb{C} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$.

  Let P be a polynomial, it can be represented as a proof of $!\mathbb{C} \vdash !\mathbb{C}$ (see [Gir98]), meaning that if `n` represents the Church numeral n the following proof reduces to the proof associated with $P(n)$ followed by a promotion rule:

$$\dfrac{\dfrac{}{\vdash \mathbb{C}}\,\mathtt{n} \qquad P}{\dfrac{\vdash !\mathbb{C} \qquad !\mathbb{C} \vdash !\mathbb{C}}{\vdash !\mathbb{C}}}$$

- We represent (Church style) binary words with the type $\mathbb{W} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$.

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

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

- 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 type $\mathbb{S} = \mu\beta.\forall\alpha.(\beta \multimap \alpha) \multimap (\beta \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)$ (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 operations of pushing elements on top of a word $\mathtt{cons}_0$ and $\mathtt{cons}_1$ are obtained from the two possible choices of the splitting of the occurrences of $\mathbb{S} \multimap \alpha$ in:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{S} \vdash \mathbb{S}} \quad \dfrac{}{\alpha \vdash \alpha}}{\mathbb{S}, \mathbb{S} \multimap \alpha \vdash \alpha} \qquad \mathtt{gc}\ \dfrac{}{\mathbb{S} \multimap \alpha, \alpha \vdash \exists\gamma.\gamma}}{\mathbb{S}, \mathbb{S} \multimap \alpha, \mathbb{S} \multimap \alpha, \alpha \vdash \alpha \otimes \exists\gamma.\gamma}}{\mathbb{S} \vdash (\mathbb{S} \multimap \alpha) \multimap (\mathbb{S} \multimap \alpha) \multimap \alpha \multimap (\alpha \otimes \exists\gamma.\gamma)}}{\mathbb{S} \vdash \mathbb{S}}$$

  and the empty word is:

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

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{0}}{\vdash \mathbb{B}} \quad \dfrac{}{\mathbb{S} \vdash \mathbb{S}}}{\mathbb{S} \vdash \mathbb{B} \otimes \mathbb{S}}}{\vdash \mathbb{S} \multimap \mathbb{B} \otimes \mathbb{S}} \qquad \dfrac{\dfrac{\dfrac{\mathtt{1}}{\vdash \mathbb{B}} \quad \dfrac{}{\mathbb{S} \vdash \mathbb{S}}}{\mathbb{S} \vdash \mathbb{B} \otimes \mathbb{S}}}{\vdash \mathbb{S} \multimap \mathbb{B} \otimes \mathbb{S}} \qquad \dfrac{\mathtt{0} \quad \mathtt{nil}}{\vdash \mathbb{B} \otimes \mathbb{S}} \qquad \dfrac{\dfrac{\dfrac{\mathtt{gc}_{\mathbb{B}}}{\mathbb{B}, \exists\gamma.\gamma \vdash \mathbb{B}} \quad \dfrac{}{\mathbb{S} \vdash \mathbb{S}}}{\mathbb{B}, \mathbb{S}, \exists\gamma.\gamma \vdash \mathbb{B} \otimes \mathbb{S}}}{\mathbb{B} \otimes \mathbb{S} \otimes \exists\gamma.\gamma \vdash \mathbb{B} \otimes \mathbb{S}}}{(\mathbb{S} \multimap \mathbb{B} \otimes \mathbb{S}) \multimap (\mathbb{S} \multimap \mathbb{B} \otimes \mathbb{S}) \multimap (\mathbb{B} \otimes \mathbb{S}) \multimap (\mathbb{B} \otimes \mathbb{S} \otimes \exists\gamma.\gamma) \vdash \mathbb{B} \otimes \mathbb{S}}}{\mathbb{S} \vdash \mathbb{B} \otimes \mathbb{S}}$$

- A translation form the Church style representation of binary words $\mathbb{W}$ into their Scott style representation can be obtained by:

$$\texttt{w2s} = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{cons}_0}{\mathbb{S} \vdash \mathbb{S}}}{\vdash \mathbb{S} \multimap \mathbb{S}}}{\vdash !(\mathbb{S} \multimap \mathbb{S})} \quad \dfrac{\dfrac{\dfrac{\texttt{cons}_1}{\mathbb{S} \vdash \mathbb{S}}}{\vdash \mathbb{S} \multimap \mathbb{S}}}{\vdash !(\mathbb{S} \multimap \mathbb{S})} \quad \dfrac{\dfrac{\dfrac{\texttt{nil}}{\vdash \mathbb{S}} \quad \dfrac{}{\mathbb{S} \vdash \mathbb{S}}}{\mathbb{S} \multimap \mathbb{S} \vdash \mathbb{S}}}{!(\mathbb{S} \multimap \mathbb{S}) \vdash !\mathbb{S}}}{!(\mathbb{S} \multimap \mathbb{S}) \multimap !(\mathbb{S} \multimap \mathbb{S}) \multimap !(\mathbb{S} \multimap \mathbb{S}) \vdash !\mathbb{S}}}{\mathbb{W} \vdash !\mathbb{S}}$$

- The representation of the execution of Turing machines with two symbols and s states is based on the following type of configurations $\textsc{Config} = \mathbb{S} \otimes \mathbb{B} \otimes \mathbb{S} \otimes \mathbb{E}_s$. 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.

  Given the representation in $\mathbb{S}$ of a binary word, one can build the initial configuration by:

$$\texttt{init} = \dfrac{\dfrac{\texttt{nil}}{\vdash \mathbb{S}} \quad \dfrac{\texttt{pop}}{\mathbb{S} \vdash \mathbb{B} \otimes \mathbb{S}} \quad \dfrac{\texttt{state}_1}{\vdash \mathbb{E}_s}}{\mathbb{S} \vdash \textsc{Config}}$$

  assuming that 1 is the initial state.

  A transition of the machine is computed through the proof `step` of $\textsc{Config} \vdash \textsc{Config}$:

$$\dfrac{\dfrac{\dfrac{\cdots \quad \dfrac{\dfrac{\text{transit}_0^s}{\mathbb{S}, \mathbb{S} \vdash \textsc{Config}} \quad \dfrac{\text{transit}_1^s}{\mathbb{S}, \mathbb{S} \vdash \textsc{Config}}}{\mathbb{S}, \mathbb{B}, \mathbb{S} \vdash \textsc{Config} \otimes \exists\gamma.\gamma} \quad \cdots}{\mathbb{S}, \mathbb{B}, \mathbb{S}, \mathbb{E}_s \vdash \textsc{Config} \otimes \exists\gamma.\gamma \otimes \exists\gamma.\gamma} \quad \dfrac{\dfrac{\dfrac{\dfrac{\texttt{gc}_{\mathbb{B}}}{\mathbb{B}, \exists\gamma.\gamma, \exists\gamma.\gamma \vdash \mathbb{B}} \quad \dfrac{}{\mathbb{S} \otimes \mathbb{S} \otimes \mathbb{E}_s \vdash \mathbb{S} \otimes \mathbb{S} \otimes \mathbb{E}_s}}{\mathbb{B}, \mathbb{S} \otimes \mathbb{S} \otimes \mathbb{E}_s, \exists\gamma.\gamma, \exists\gamma.\gamma \vdash \textsc{Config}}}{\textsc{Config}, \exists\gamma.\gamma, \exists\gamma.\gamma \vdash \textsc{Config}}}{\textsc{Config} \otimes \exists\gamma.\gamma \otimes \exists\gamma.\gamma \vdash \textsc{Config}}}{\dfrac{\mathbb{S}, \mathbb{B}, \mathbb{S}, \mathbb{E}_s \vdash \textsc{Config}}{\textsc{Config} \vdash \textsc{Config}}}$$

  The proof transit_i^s encodes the transition table of the Turing machine. For that we only need to consider the following two parametrized proofs which, given a state k and a symbol b, move the head of the machine to one direction (left or right) and build the new configuration from the left part and the right part of the previous tape:

$$\texttt{transit}_\texttt{l}(k, b) = \dfrac{\dfrac{\texttt{pop}}{\mathbb{S} \vdash \mathbb{S} \otimes \mathbb{B}} \quad \dfrac{\texttt{cons}_b}{\mathbb{S} \vdash \mathbb{S}} \quad \dfrac{\texttt{state}_k}{\vdash \mathbb{E}_s}}{\mathbb{S}, \mathbb{S} \vdash \textsc{Config}}$$

$$\texttt{transit}_\texttt{r}(k, b) = \dfrac{\dfrac{\texttt{cons}_b}{\mathbb{S} \vdash \mathbb{S}} \quad \dfrac{\texttt{pop}}{\mathbb{S} \vdash \mathbb{B} \otimes \mathbb{S}} \quad \dfrac{\texttt{state}_k}{\vdash \mathbb{E}_s}}{\mathbb{S}, \mathbb{S} \vdash \textsc{Config}}$$

  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:

$$\texttt{run} = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{step}}{\textsc{Config} \vdash \textsc{Config}}}{\vdash \textsc{Config} \multimap \textsc{Config}}}{\vdash !(\textsc{Config} \multimap \textsc{Config})} \quad \dfrac{\dfrac{\dfrac{\texttt{init}}{\mathbb{S} \vdash \textsc{Config}} \quad \dfrac{}{\textsc{Config} \vdash \textsc{Config}}}{\textsc{Config} \multimap \textsc{Config}, \mathbb{S} \vdash \textsc{Config}}}{!(\textsc{Config} \multimap \textsc{Config}), !\mathbb{S} \vdash !\textsc{Config}}}{!(\textsc{Config} \multimap \textsc{Config}) \multimap !(\textsc{Config} \multimap \textsc{Config}), !\mathbb{S} \vdash !\textsc{Config}}}{\mathbb{C}, !\mathbb{S} \vdash !\textsc{Config}}$$

The acceptance of a configuration is tested through the proof:

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

assuming the state s is the accepting one.

- The result of the evaluation of the PTIME Turing machine on an input represented in $!\mathbb{W}$ is obtained by:

$$\dfrac{\dfrac{\dfrac{\overset{\texttt{length}}{\mathbb{W} \vdash \mathbb{C}}}{!\mathbb{W} \vdash !\mathbb{C}} \qquad \overset{P}{!\mathbb{C} \vdash !\mathbb{C}}}{!\mathbb{W} \vdash !\mathbb{C}} \qquad \dfrac{\overset{\texttt{w2s}}{\mathbb{W} \vdash !\mathbb{S}} \qquad \overset{\texttt{run}}{\mathbb{C}, !\mathbb{S} \vdash !\textsc{Config}}}{\mathbb{C}, \mathbb{W} \vdash !\textsc{Config}}}{!\mathbb{C}, !\mathbb{W} \vdash !!\textsc{Config}} \qquad \dfrac{\overset{\texttt{accept}}{\textsc{Config} \vdash \mathbb{B}}}{!\textsc{Config} \vdash !\mathbb{B}} \qquad \dfrac{\dfrac{\overset{\texttt{b2bl}}{\mathbb{B} \vdash \mathbb{B}_{\mathbb{1}} \otimes \exists\gamma.\gamma}}{!\mathbb{B} \vdash !(\mathbb{B}_{\mathbb{1}} \otimes \exists\gamma.\gamma)}}{!!\mathbb{B} \vdash !!(\mathbb{B}_{\mathbb{1}} \otimes \exists\gamma.\gamma)}}{!\mathbb{W}, !\mathbb{W} \vdash !!(\mathbb{B}_{\mathbb{1}} \otimes \exists\gamma.\gamma)}}{!\mathbb{W} \vdash !!(\mathbb{B}_{\mathbb{1}} \otimes \exists\gamma.\gamma)}$$

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.

Theorem 2 (PTIME Completeness)
If $\mathcal{P}$ is a PTIME *problem, there exists an e-net which represents $\mathcal{P}$.*

Proof: Sequent calculus proofs used above can be turned into e-nets:

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

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

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

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

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

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

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

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

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

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

This transforms the proof of $!\mathbb{W}$ representing a binary word w into its e-net representation, and a proof $!!(\mathbb{B}_{\mathbb{1}} \otimes \exists\gamma.\gamma)$ is turned into an e-net representing a boolean.

This means that the sequent calculus proof built above from the description of a PTIME Turing machine is translated into an e-net which represents the same problem as the Turing machine. □

References

[Bai11] Patrick Baillot. Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In Hongseok Yang, editor, *Proceedings of the 9th Asian Symposium on Programming Languages and Systems (APLAS)*, volume 7078 of *Lecture Notes in Computer Science*, pages 337–352. Springer, December 2011.

[DLB06] Ugo Dal Lago and Patrick Baillot. On light logics, uniform encodings and polynomial time. *Mathematical Structures in Computer Science*, 16(4):713–733, 2006.

[DR89] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[Gir98] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, June 1998.

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 2003.