Math. Struct. in Comp. Science (2006), *vol.* 16, *pp.* 713–733. © 2006 Cambridge University Press
doi:10.1017/S0960129506005421 Printed in the United Kingdom

On light logics, uniform encodings and polynomial time

UGO DAL LAGO[†] and PATRICK BAILLOT[‡]

[†]*Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy.*
Email: dallago@cs.unibo.it
[‡]*Laboratoire d'Informatique de Paris-Nord (UMR 7030 CNRS), Université Paris 13, Intsitut Galilée, 99 avenue Jean-Baptiste Clément, 93430 Villetaneuse, France.*
Email: patrick.baillot@lipn.univ-paris13.fr

Received 27 July 2005; revised 31 January 2006

Light affine logic is a variant of linear logic with a polynomial cut-elimination procedure. We study the extensional expressive power of light affine logic with respect to a general notion of encoding of functions in the setting of the Curry–Howard correspondence. We consider light affine logic with both fixpoints of formulae and second-order quantifiers, and analyse the properties of polytime soundness and polytime completeness for various fragments of this system. In particular, we show that the implicative propositional fragment is not polytime complete if we place some reasonable conditions on the encodings. Following previous work, we show that second order leads to polytime unsoundness. We then introduce simple constraints on second-order quantification and fixpoints, and prove that the fragments obtained are polytime sound and complete.

1. Introduction

Characterising the class of functions that a logic can represent helps in understanding the computational expressive power of the logic. If the system under consideration enjoys a Curry–Howard correspondence, the analysis can be even more valuable – the class of representable functions becomes the class of functions that the underlying programming language can compute. These investigations become a crucial issue in the context of light logics, which have been defined precisely to capture relevant function classes, namely complexity classes.

Light linear logic, **LLL** (Girard 1998), was proposed by Girard as a variant of linear logic, **LL** (Girard 1987), characterising the class **FP** of deterministic polynomial time functions. It was later simplified by Asperti into light affine logic, **LAL** (Asperti 1998). The limitation to the computational power of **LLL** (or **LAL**) is obtained by considering a

[†] Work partially supported by project FOLLIA (MIUR).
[‡] Work partially supported by projects GEOCAL (ACI), NO-CoST (ANR), *Interaction et Complexité* (cooperation project CNR-CNRS, no 16251).

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

weaker modality ! for resource reuse than that used in plain linear logic. **LAL** has been the subject of many investigations from syntactical, semantical and programming language perspectives (Murawski and Ong 2004; 2000; Roversi 2000; Terui 2002). Another line of research in this direction is Lafont's soft linear logic, **SLL** (Lafont 2004), which is another variant of **LL** for polynomial time.

Still, one can observe that these characterisations of **FP** via the Curry–Howard correspondence (in **LLL**, **LAL** or **SLL**) only hold if data are encoded by bounded-depth proofs (the notion of depth is linked to the modalities and to the notion of a box). Recently, Mairson and Neergaard (Neergaard and Mairson 2002) proved that dropping the bounded box-depth assumption makes **LAL** complete for doubly exponential time. In their setting, data are represented by proofs having unbounded box-depth and different conclusions. Alternative notions of encodings have also been considered in Mairson and Terui (2003) for various subsystems of **LL**.

An important point is that the encodings given in Girard (1998) and Asperti and Roversi (2002) make extensive use of second-order quantification, which allows programming with polymorphism in the style of System **F**. This is an elegant and general approach, but second-order quantification brings difficulties of its own, which are not related to **LAL** itself. For instance, it makes the issues of provability decision problems, type-inference and semantics far more delicate. One may wonder how much second-order power is really needed in **LAL** to get polynomial time expressivity.

This question is all the more pertinent as **LAL** and **SLL** are compatible with another feature: fixpoints. Indeed, the fixpoints of formulae were one of the original intuitions underlying the definition of **LLL** (see the introduction of Girard (1998)). They are also *definable* in light set theory, **LST** (Girard 1998; Terui 2004), in which they can be used to write function definitions; one can then prove the termination of such functions in **LST**. Alternatively, when considering **LAL** and **SLL** as type systems, fixpoints correspond to recursive types. In particular, the expressivity of **SLL** with fixpoints has been examined in Baillot and Mogbil (2004).

So, as there are several notions of encoding, and a large range of connectives and computational features are available in **LAL**, we think it is important first to establish a *reasonable* notion of an encoding, and then determine the expressivity of small fragments of this logic. This will help us identify well-behaved fragments that might then be used for various purposes, such as type inference, proof of program termination or proof-search.

In previous work (Dal Lago 2003), we started focusing our attention on constrained representation schemes, called *uniform coding schemes*. We proved, in particular, that light affine logic is not polytime sound if the power of second-order quantification is fully exploited.

The encodings presented in Girard (1998), Asperti and Roversi (2002) and Baillot and Mogbil (2004) fit into the definition of uniform encodings, while some of those in Neergaard and Mairson (2002) and Mairson and Terui (2003) do not. In the latter, for instance, different function calls are encoded by (cut-free) proofs having different conclusions (in the style of boolean circuits, with one circuit for each size of input). This comes as no surprise, since the authors were interested in studying the complexity of

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

cut-elimination as a computational problem. However, these encodings are not acceptable if we want to study the expressive power of a logic as a programming language, as we do.

Our notion of uniformity is rather general. In particular, we do not impose any constraint on the shape of formulae for inputs and outputs. This is in contrast to similar results from the literature (Fortune *et al.* 1983; Leivant and Marion 1993).

In this paper, we systematically investigate the expressive power of (various fragments of) light affine logic, but always considering uniform coding schemes. First, we prove that if we impose some (fairly reasonable) conditions on the notion of an encoding, the propositional implicative fragment of **LAL** is *not* complete for **FP**. Then we introduce simple constraints on second-order quantification and fixpoints, and prove that the resulting fragments are polytime sound and complete.

A preliminary version of this work was presented at the workshop on *Logics for Resources, Processes, and Programs*, 2004 (Dal Lago and Baillot 2004).

2. Uniform encodings

In this short section we recall the notion of uniform encoding that was introduced in Dal Lago (2003).

A uniform encoding $\mathscr{E}(f)$ of $f : (\{0,1\}^*)^n \to \{0,1\}^*$ into a logic consists of:

- A proof π with conclusion $A_1, \ldots, A_n \vdash B$, (where $A_1, \ldots, A_n, B$ can be different).
- For every $i \in \{1, \ldots, n\}$, a suitable correspondence Φ_i between elements of $\{0,1\}^*$ and cut-free proofs having conclusions $\vdash A_i$. These correspondences must be computable in logarithmic space.
- A correspondence Ψ between cut-free proofs having conclusion $\vdash B$ and elements of $\{0,1\}^*$. This correspondence must be logspace computable.

Clearly, the following diagram should commute:

$$
\begin{array}{ccccccc}
\{0,1\}^* & \times \ldots \times & \{0,1\}^* & \xrightarrow{\;f\;} & \{0,1\}^* \\
\downarrow \Phi_1 & & \downarrow \Phi_n & & \uparrow \Psi \\
A_1 & , \; \ldots \; , & A_n & \xrightarrow{\;\pi\;} & B
\end{array}
$$

This definition is strongly inspired by the Curry–Howard correspondence.

For example, consider second-order implicative intuitionistic logic, that is to say, System **F** by the Curry–Howard correspondence. Let W stand for a type for binary words, for instance,

$$W = \forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha).$$

Then a proof of the sequent $W, \ldots, W \vdash W$ gives a uniform encoding of a function $f : (\{0,1\}^*)^n \to \{0,1\}^*$. But these are not the only uniform encodings in System **F**, since we can define others by considering other representations of binary lists. In particular, the types for the various arguments and for the result need not be the same.

We say that a logic $\mathscr{L}$ is *polytime sound* if the class of functions $f : (\{0,1\}^*)^n \to \{0,1\}^*$ uniformly encodable in $\mathscr{L}$ is included in **FP** and that it is *polytime complete* if this class contains **FP**.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

Identity and cut

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

Structural rules

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

Implicative logical rules

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

Exponential logical rules

$$\frac{A \vdash B}{!A \vdash !B}\ P_!^1 \quad \frac{\vdash A}{\vdash !A}\ P_!^2 \quad \frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}\ P_\S$$

Fig. 1. Implicative intuitionistic light affine logic, $\mathbf{ILAL}_{\multimap}$

3. Syntax

Following the existing literature, we will use the intuitionistic variant of light affine logic **LAL**, called **ILAL**, as our reference system. *Formulae* are generated by the grammar

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

where α ranges over a set $\mathscr{L}$ of *atoms*. *Sequents* have the form $A_1, \ldots, A_n \vdash B$, where $A_1, \ldots, A_n, B$ are all formulae.

An **ILAL** *proof* is simply a tree whose nodes are labelled with sequents according to **ILAL** rules. A proof π having conclusion $\Gamma \vdash A$ is sometimes denoted by $\pi : \Gamma \vdash A$. We will define the *size* of the proof, denoted by $|\pi|$, as the number of rules in the proof.

We will study various fragments of **ILAL**. The core will be $\mathbf{ILAL}_{\multimap}$, which is defined in Figure 1.

Recall that in **ILAL** the contraction rule is restricted, as in linear logic, to !-marked formulae (rule C from Figure 1). The main difference compared with linear logic lies in the way !-marked formulae are introduced, which is more constrained: with rules $P_!^1$ and $P_!^2$ the ! modalities are introduced at the same time on the left- and right-hand sides of formulae, and the sequent can have at most one formula on the left-hand side. Alternatively, one can introduce ! modalities on the left-hand side using the $P_\S$ rule, but the remaining formulae must be marked with the new modality §.

The modality § can be thought of as a kind of degenerate !, in the sense that it does not allow for contraction. The rule $P_\S$ is a weak analogue of the dereliction rule of linear logic. Recall that the following principles (called *dereliction* and *digging*, respectively) are *not* provable in **LAL**, but are provable in linear logic:

$$!A \vdash A, \qquad !A \vdash !!A.$$

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ L_\otimes \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ R_\otimes$$

Fig. 2. Tensor logical rules

$$\frac{\Gamma, C[A/\alpha] \vdash B}{\Gamma, \forall\alpha.C \vdash B}\ L_\forall \qquad \frac{\Gamma \vdash C \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \forall\alpha.C}\ R_\forall$$

Fig. 3. Second-order rules

$$\frac{\Gamma, A[\mu\alpha.A/\alpha] \vdash B}{\Gamma, \mu\alpha.A \vdash B}\ L_\mu \qquad \frac{\Gamma \vdash A[\mu\alpha.A/\alpha]}{\Gamma \vdash \mu\alpha.A}\ R_\mu$$

Fig. 4. Fixpoint rules

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

Fig. 5. Derivable unfolding rules

$$\frac{\dfrac{\overline{A[\mu\alpha.A/\alpha] \vdash A[\mu\alpha.A/\alpha]}\ I}{A[\mu\alpha.A/\alpha] \vdash \mu\alpha.A}\ R_\mu \qquad \Gamma, \mu\alpha.A \vdash B}{\Gamma, A[\mu\alpha.A/\alpha] \vdash B}\ U$$

Fig. 6. A derivation for rule L'_μ

As we will see in Theorem 1, these restricted rules for the modalities are the key to the complexity bound on the cut-elimination procedure.

We can add other connectives to this core $\mathbf{ILAL}_{\multimap}$ to obtain more powerful logics. For example, we can add tensor ($\otimes$, see Figure 2) and second-order quantification ($\forall$, see Figure 3). Another interesting connective that can be added to the logic is the fixpoint operator (μ, see Figure 4). Note that the rules L'_μ and R'_μ of Figure 5 can be derived from L_μ and R_μ; a derivation of L'_μ is given in Figure 6.

In this way, we can build several fragments of intuitionistic light affine logic, such as $\mathbf{ILAL}_{\multimap\otimes\forall}$ or $\mathbf{ILAL}_{\multimap\otimes\forall\mu}$. It is important to stress that **ILAL** admits cut-elimination. This stems from two facts:

- All connectives admit cut-elimination steps. In particular, the number of rules decreases with one step of cut-elimination on a $\mu\alpha.A$ formula.
- A particular strategy allows us to eliminate all cuts: see Girard (1998) and Asperti and Roversi (2002).

Note that adding fixpoints to intuitionistic logic or linear logic breaks the cut-elimination property, but this is not the case with **LAL**. This is because in this system, as in elementary, light or soft linear logic (Lafont 2004; Baillot and Mogbil 2004), the cut-elimination argument does not depend on the size of the cut formulae but on the size and

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

depth of proofs (the latter will be defined soon). Actually, this was one motivation for the definition of light linear logic that was originally stressed by Girard (Girard 1998).

ILAL can also be thought of as a type assignment system for the following term calculus:

$$M, N ::= x \mid \lambda x.M \mid MM \mid (M, M) \mid \ \textbf{let} \ M \ \textbf{be} \ (x, x) \ \textbf{in} \ M.$$

In this setting, rules for !, §, ∀ and μ do not influence the underlying term. When this does not cause any ambiguity, we will denote an **ILAL** sequent calculus proof by the term it types. If $A_1, \ldots, A_n \vdash B$ types term M, then the free variables appearing in M will be named $x_1, \ldots, x_n$ of type $A_1, \ldots, A_n$, respectively. Most results for **ILAL** are traditionally given using proof-nets, which are handy for studying the dynamics of proofs (see Asperti and Roversi (2002)). However, to keep a concise presentation, we have chosen to present **ILAL** as a sequent calculus. Many sequent calculus proofs differing only in the order of application of rules could correspond to the same proof-net. Anyway, here we are just using sequent calculus as a convenient notation, and there would be no problem in converting the proofs into proof-nets if one wanted to examine the normalisation issues.

Definition 1. Given an **ILAL** proof π, the *box-depth* $\partial(\pi)$ of π is the maximum integer n such that there is a path in π from a leaf to the root that crosses n instances of rules $P_!^1$, $P_!^2$ or $P_\S$.

It is easy to check that this definition of box-depth is equivalent to the one traditionally given on **ILAL** proof-nets (Asperti and Roversi 2002), namely the maximal nesting level of boxes in the proof-net.

An **ILAL** fragment is said to be *reflective* if there is a function f (from sequents to natural numbers) such that $\partial(\pi) \leqslant f(\Gamma \vdash A)$ whenever $\pi : \Gamma \vdash A$ is a cut-free proof. This means that given a formula, one can bound the box-depth of cut-free proofs with this conclusion.

Now, we recall the main result of **ILAL**.

Theorem 1 (ILAL normalisation complexity (Asperti and Roversi 2002)). The normalisation of an **ILAL** proof π can be done in time $O(|\pi|^k)$, where the exponent k only depends on $\partial(\pi)$.

As a direct consequence, we have the following proposition.

Proposition 1. Any reflective fragment of **ILAL** is polytime sound.

4. The full case

We start with the fragment $\textbf{ILAL}_{\multimap\otimes\forall}$. We know from Asperti and Roversi (2002) that this fragment is polytime complete. In spite of this, it is not reflective since the rule $L_\forall$ can be used to build proofs with fixed conclusion but arbitrary box-depth. Indeed, $\textbf{ILAL}_{\multimap\otimes\forall}$ is polytime unsound if the full power of second-order quantification is exploited, as we are going to show.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

Binary lists can be represented in $\mathbf{ILAL}_{\multimap\otimes\forall}$ by cut-free proofs with conclusion

$$SOBinaryLists = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha).$$

The cut-free proof with conclusion $\vdash SOBinaryLists$ corresponding to string $s \in \{0,1\}^*$ will be denoted by $\ulcorner s \urcorner$.

The encodings of functions considered in Asperti and Roversi (2002) used sequents of the form $SOBinaryLists \vdash \S^k SOBinaryLists$, with k an integer. These are particular uniform encodings, but in the present paper we will also consider other encodings.

Lemma 1. For every $n \in \mathbb{N}$, there is a cut-free $\mathbf{ILAL}_{\multimap\otimes\forall}$ proof

$$\rho_n : SOBinaryLists \vdash \S^{n+1} SOBinaryLists$$

such that ρ_n is a uniform encoding of the function $p^n : \{0,1\}^* \to \{0,1\}^*$, where $p^n(s) = 1^{|s|^n}$ for every $s \in \{0,1\}^*$.

Proof. In this proof, we will use BL as an abbreviation for $SOBinaryLists$. Since the case $n = 0$ is trivial, we can assume $n \geqslant 1$. For every $m \geqslant 1$, we can inductively define Γ_m as follows. First, $\Gamma_1 = BL$; moreover, $\Gamma_m = \Gamma_{m-1}, \S^{m-2}!BL$ for every $m > 1$. Similarly, A_1 denotes BL, while for every $m > 1$ $A_m = A_{m-1} \otimes \S^{m-2}!BL$. We now prove, by induction on m, that there is a proof $\sigma_m : \Gamma_m \vdash \S^m BL$ encoding function $f_m : (\{0,1\}^*)^m \to \{0,1\}^*$ where $f_m(s_1, \ldots, s_m) = 1^{|s_1|\cdots|s_m|}$ for every $s_1, \ldots, s_m \in \{0,1\}^*$. If $m = 1$, then σ_m is

$$\cfrac{\varphi : \vdash !(BL \multimap BL) \quad \varphi : \vdash !(BL \multimap BL) \quad \cfrac{\cfrac{\psi : \vdash BL \quad \overline{BL \vdash BL}}{BL \multimap BL \vdash BL}}{\S(BL \multimap BL) \vdash \S BL}}{BL \vdash \S BL}$$

where:

— φ is the proof

$$\cfrac{\cfrac{\xi : BL \vdash BL}{\vdash BL \multimap BL}}{\vdash !(BL \multimap BL)}$$

— ξ is a cut-free proof encoding function $g : \{0,1\}^* \to \{0,1\}^*$ with $g(s) = 1s$.

— ψ encodes the string ε.

If $m > 1$, then σ_m is

$$\cfrac{\theta \quad \theta \quad \cfrac{\cfrac{\psi : \vdash BL \quad \sigma_{m-1} : \Gamma_{m-1} \vdash \S^{m-1} BL}{BL \multimap BL, !BL, \ldots, \S^{m-3}!BL \vdash \S^{m-1} BL}}{\S(BL \multimap BL), \S!BL, \ldots, \S^{m-2}!BL \vdash \S^m BL}}{\Gamma_m \vdash \S^m BL}$$

where:

— θ is the proof

$$\cfrac{\cfrac{\omega : BL, BL \vdash BL}{BL \vdash BL \multimap BL}}{!BL \vdash !(BL \multimap BL)}$$

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

— ω encodes the function $h : (\{0,1\}^*)^2 \rightarrow \{0,1\}^*$ such that $h(s,t) = st$ for every $s,t \in \{0,1\}^*$.

We are now able to build ρ_n:

$$\dfrac{\dfrac{\dfrac{\eta : A_n \vdash A_n}{\vdash A_n \multimap A_n}}{\vdash !(A_n \multimap A_n)} \qquad \dfrac{\dfrac{\eta : A_n \vdash A_n}{\vdash A_n \multimap A_n}}{\vdash !(A_n \multimap A_n)} \qquad \dfrac{\dfrac{\tau : \vdash A_n \qquad \dfrac{\sigma_n : \Gamma_n \vdash \S^n BL}{A_n \vdash \S^n BL}}{A_n \multimap A_n \vdash \S^n BL}}{\S(A_n \multimap A_n) \vdash \S^{n+1} BL}}{BL \vdash \S^{n+1} BL}$$

Here, η and τ are generalisations of φ and ψ, respectively. Notice that ρ_n, as we have defined it, is cut-free and can be built in logarithmic space (on n). □

We have just proved that, for every $n \in \mathbb{N}$, ρ_n uniformly encodes the function p^n. Now, if all the different ρ_n had the same type, it would be easy to build a proof δ such that $\delta(\rho_n)$ reduces to $\rho_n(\ulcorner 11 \urcorner)$, and then normalise it to a proof similar to $\lceil m \rceil$ where $m = 1^{2^n}$. Actually, every ρ_n has a conclusion that is different from the conclusion of any other ρ_m. This problem, however, can be circumvented by building another sequence of proofs $\{\chi_n\}_{n \in \mathbb{N}}$. Every such χ_n behaves similarly to ρ_n, but all the proofs in the sequence have the same conclusion. In this way, we can find a uniform encoding inside $\mathbf{ILAL}_{\multimap\otimes\forall}$ of an intrinsically exponential function over the algebra $\{0,1\}^*$.

Proposition 2. There is a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ that can be uniformly represented in $\mathbf{ILAL}_{\multimap\otimes\forall}$ and is not computable in polynomial time.

Proof. $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is the function defined by letting

$$f(s) = 1^{2^{|s|}}$$

whenever $s \in \{0,1\}^*$. Clearly, f cannot belong to **FP**, because the length of the output is exponential in the length of the input. For every $n \in \mathbb{N}$, the proof χ_n is defined as follows:

$$\dfrac{\dfrac{\dfrac{\rho_n : BL \vdash \S^{n+1} BL \qquad \dfrac{}{\alpha \vdash \alpha}}{BL, \S^{n+1} BL \multimap \alpha \vdash \alpha}}{BL, \forall\beta.(\beta \multimap \alpha) \vdash \alpha}}{\vdash BL \multimap (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}$$

where ρ_n is as in Lemma 1. For every $m \in \mathbb{N}$, the proof τ_m is defined as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\lceil 1^m \rceil : \vdash BL}{\vdash \S^{\lceil lg m \rceil + 1} BL} \qquad \dfrac{}{\alpha \vdash \alpha}}{\S^{\lceil lg m \rceil + 1} BL \multimap \alpha \vdash \alpha}}{\forall\beta.(\beta \multimap \alpha) \vdash \alpha}}{\vdash (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}$$

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

Now let π be the proof

$$\dfrac{\lceil 11 \rceil : \vdash BL \quad \overline{(\forall\beta.(\beta \multimap \alpha)) \multimap \alpha \vdash (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}}{BL \multimap (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha \vdash (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}$$

Consider the following diagram:

$$\begin{array}{ccc} \{0,1\}^* & \xrightarrow{\quad f \quad} & \{0,1\}^* \\ \Big\downarrow \Phi & & \Big\uparrow \Psi \\ BL \multimap (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha & \xrightarrow{\pi} & (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha \end{array}$$

Φ is the function defined by letting $\Phi(s) = \chi_{|s|}$ for every $s \in \{0,1\}^*$, and Ψ is defined by letting $\Psi(\tau_m) = 1^m$ and $\Psi(\rho) = \varepsilon$ whenever ρ is not in the form τ_m. Both Φ and Ψ are obviously logspace computable. It is easy to check that the above diagram commutes. $\square$

The question we consider now is: can we restrict $\mathbf{ILAL}_{\multimap\otimes\forall}$ to reach a polytime sound and complete system? This is the main subject of this paper. The solution pursued in Dal Lago (2003) consisted of restricting the class of permitted *encodings*, by forbidding the use of $L_\forall$ in proofs representing inputs and outputs. Here, we use a different approach: we try to restrict the *logic*, without modifying the coding schemes.

5. $\mathbf{ILAL}_{\multimap}$ and polynomial time

How much can we restrict $\mathbf{ILAL}_{\multimap\otimes\forall}$ while keeping polytime completeness? Let us start by considering the smallest fragment, $\mathbf{ILAL}_{\multimap}$. In this section, we will prove that, under reasonable assumptions on the encodings, $\mathbf{ILAL}_{\multimap}$ is not polytime complete.

$\mathbf{ILAL}_{\multimap}$ can be seen as a type assignment system for pure lambda-calculus. If a pure lambda-term M can be typed by an $\mathbf{ILAL}_{\multimap}$ proof, then it is simply-typable. Moreover, if M can be typed by a cut-free $\mathbf{ILAL}_{\multimap}$ proof, then it is necessarily a β-normal form, but it may contain η-redexes.

An encoding of f into $\mathbf{ILAL}_{\multimap}$ is said to be *extensional* if all the correspondences $\Phi_1, \ldots, \Phi_n, \Psi$ map distinct elements of $\{0,1\}^*$ to $\mathbf{ILAL}_{\multimap}$ proofs whose underlying lambda-terms are distinct and η-normal.

Now, we can recall a theorem by Statman.

Theorem 2 (Finite completeness theorem (Statman 1982)). Let M be a closed term having simple type A. There exists a finite model $\mathscr{M}(M)$ such that $\mathscr{M}(M) \models M = N$ if and only if $M =_{\beta\eta} N$.

The function *equality* : $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is defined by

$$equality(s,t) = \begin{cases} 1 & if \quad s = t \\ 0 & otherwise. \end{cases}$$

Now we get the following proposition.

Proposition 3. *equality* is not extensionally encodable in $\mathbf{ILAL}_{\multimap}$.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

Proof. Basically, we use the fact that an extensional encoding of a function f in $\mathbf{ILAL}_{\multimap}$ induces a corresponding encoding of f into the simply typed lambda-calculus. Suppose, to show a contradiction, that an extensional encoding $\mathscr{E}$ of *equality* into $\mathbf{ILAL}_{\multimap}$ exists. Then, there are simply typable closed terms

$$\begin{aligned} M &: A \to B \to C \\ T &: C \\ F &: C \\ T &\neq_{\beta\eta} F \end{aligned}$$

where T and F encode the values 0 and 1, respectively; and for every $s \in \{0,1\}^*$, there are simply typable closed terms encoding s into types A and B, respectively:

$$\begin{aligned} P(s) &: A \\ Q(s) &: B \end{aligned}$$

such that, for every $s, t \in \{0,1\}^*$ with $s \neq t$,

$$\begin{aligned} MP(s)Q(s) &\to_\beta^* T \\ MP(t)Q(s) &\to_\beta^* F. \end{aligned}$$

From the extensionality hypothesis, we know that both T and F are η-normal. We use $\mathscr{M}$ to denote the model $\mathscr{M}(T)$ obtained by Theorem 2 applied to the term T. It is a finite model, so there must be $s, t \in \{0,1\}^*$ with $s \neq t$ such that $\mathscr{M}$ interprets both $P(s)$ and $P(t)$ with the same semantical value. Obviously, $MP(s)Q(s) =_{\beta\eta} T$ and $MP(t)Q(s) =_{\beta\eta} F$; so, by soundness, we have

$$\begin{aligned} \mathscr{M} &\models MP(s)Q(s) = T \\ \mathscr{M} &\models MP(t)Q(s) = F. \end{aligned}$$

But $\mathscr{M}$ must interpret $MP(s)Q(s)$ and $MP(t)Q(s)$ in the same way, so it follows that

$$\mathscr{M} \models T = F.$$

By Theorem 2, this implies $T =_{\beta\eta} F$, which is a contradiction. □

6. Polynomials and $\mathbf{ILAL}_{\multimap\otimes}$

Because of the previous negative result (Section 5), from now on we will consider fragments containing $\mathbf{ILAL}_{\multimap\otimes}$. A necessary condition for polytime completeness is the ability to represent polynomials. In this section we will show that $\mathbf{ILAL}_{\multimap\otimes}$ is sufficient for the representation of polynomials using a Church-style encoding for numerals.

In fact, throughout this paper, we use 'polynomial' to mean a polynomial with positive integer coefficients.

For every $\mathbf{ILAL}_{\multimap\otimes}$ formula A, $PInt(A)$ will be the type $!(A \multimap A) \multimap \S(A \multimap A)$. The class of *integer formulae* is the smallest class satisfying the following conditions:

— For every formula A, $PInt(A)$ is an integer formula.
— If B is an integer formula, $!B$ and $\S B$ are integer formulae.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

In other words, integer formulae are given by the following grammar:

$$B ::= PInt(A) \mid !B \mid \S B$$

where A ranges over $\mathbf{ILAL}_{\multimap\otimes}$ formulae.

Lemma 2. For every $\mathbf{ILAL}_{\multimap\otimes}$ formula A, there are proofs

$$\begin{aligned} \pi_{+1} &: PInt(A) \vdash PInt(A) \\ \pi_{+} &: PInt(A), PInt(A) \vdash PInt(A) \\ \pi_{\times} &: PInt(PInt(A)), !PInt(A) \vdash \S PInt(A) \end{aligned}$$

representing successor, addition and multiplication, respectively.

Proof. We just give the underlying terms for π_{+1}, π_{+} and $\pi_{\times}$, which are

$$\begin{aligned} M_{+1} &= \lambda x.\lambda y.x((x_1 x)y) \\ M_{+} &= \lambda x.\lambda y.(x_1 x)((x_2 x)y) \\ M_{\times} &= x_1(\lambda x_1.M_{+})(\lambda x.\lambda y.y), \end{aligned}$$

respectively (remember the convention fixed in Section 3 for the naming of free variables in typed lambda terms). □

The class of basic arithmetical functions is the smallest class satisfying the following constraints:

— The identity $id : \mathbb{N}^1 \to \mathbb{N}$ on natural numbers is a basic arithmetical function.
— For every $n \in \mathbb{N}$, the constant $n : \mathbb{N}^0 \to \mathbb{N}$ is a basic arithmetical function.
— If $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^m \to \mathbb{N}$ are basic arithmetical functions, then $f + g : \mathbb{N}^{n+m} \to \mathbb{N}$, defined by

$$(f + g)(x_1, \dots, x_n, y_1, \dots, y_m) = f(x_1, \dots, x_n) + g(y_1, \dots, y_m)$$

is a basic arithmetical function.
— If $f : \mathbb{N}^n \to \mathbb{N}$ is a basic arithmetical function, then $\widetilde{f} : \mathbb{N}^{n+1} \to \mathbb{N}$ defined by

$$\widetilde{f}(x_1, \dots, x_n, x) = x \cdot f(x_1, \dots, x_n)$$

is a basic arithmetical function.

Lemma 3. For every formula A and for every basic arithmetical function $f : \mathbb{N}^n \to \mathbb{N}$, there is an $\mathbf{ILAL}_{\multimap\otimes}$ proof $\pi_f : A_1, \dots, A_n \vdash \S^k PInt(A)$ representing f, where $A_1, \dots, A_n$ all are integer formulae.

Proof. We proceed by induction on the definition of basic arithmetical functions f. The base cases are straightforward, so we can concentrate on the two inductive cases. If $f = g + h$, where $g : \mathbb{N}^n \to \mathbb{N}$ and $h : \mathbb{N}^m \to \mathbb{N}$, then, by the induction hypothesis, there must be proofs

$$\begin{aligned} \pi_g &: A_1, \dots, A_n \vdash \S^k PInt(A) \\ \pi_h &: B_1, \dots, B_m \vdash \S^l PInt(A) \end{aligned}$$

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

representing g and h, respectively, where all the A_i and B_j are integer formulae. π_f will be

$$\dfrac{\dfrac{\pi_g : A_1, \ldots, A_n \vdash \S^k PInt(A)}{\S^l A_1, \ldots, \S^l A_n \vdash \S^{l+k} PInt(A)} \quad \rho}{\S^l A_1, \ldots, \S^l A_n, \S^k B_1, \ldots, \S^k B_m \vdash \S^{l+k} PInt(A)}$$

where ρ is

$$\dfrac{\dfrac{\pi_h : B_1, \ldots, B_m \vdash \S^l PInt(A)}{\S^k B_1, \ldots, \S^k B_m \vdash \S^{k+l} PInt(A)} \quad \dfrac{\pi_+ : PInt(A), PInt(A) \vdash PInt(A)}{\S^{l+k} PInt(A), \S^{l+k} PInt(A) \vdash \S^{l+k} PInt(A)}}{\S^k B_1, \ldots, \S^k B_m, \S^{l+k} PInt(A) \vdash \S^{l+k} PInt(A)}$$

If $f = \widetilde{g}$, where $g : \mathbb{N}^n \to \mathbb{N}$, then, by the induction hypothesis, there must be a proof

$$\pi_g : A_1, \ldots, A_n \vdash \S^k PInt(PInt(A))$$

representing g where all the A_i and B_j are integer formulae. The proof π_f will be

$$\dfrac{\pi_g : A_1, \ldots, A_n \vdash \S^k PInt(PInt(A)) \quad \dfrac{\pi_\times : PInt(PInt(A)), !PInt(A) \vdash \S PInt(A)}{\S^k PInt(PInt(A)), \S^k !PInt(A) \vdash \S^{k+1} PInt(A)}}{A_1, \ldots, A_n, \S^k !PInt(A) \vdash \S^{k+1} PInt(A)}$$

This concludes the proof. □

Proposition 4. For every formula A and for every polynomial $f : \mathbb{N} \to \mathbb{N}$, there is an $\mathbf{ILAL}_{\multimap\otimes}$ proof $\pi_f : PInt(B) \vdash \S^k PInt(A)$ representing f.

Proof. Any polynomial $f : \mathbb{N} \to \mathbb{N}$ with integer coefficients can be written as

$$f(x) = \sum_{i=1}^{n} \prod_{j=1}^{m(i)} a_i^j$$

where a_i^j is either an integer constant or the indeterminate x. We can arrange all the constants in a sequence $a_{cp(1)}^{ca(1)}, \ldots, a_{cp(p)}^{ca(p)}$ and all the x occurrences in another sequence $a_{ip(1)}^{ia(1)}, \ldots, a_{ip(q)}^{ia(q)}$. Let m be $\sum_{i=1}^{n} m(i)$. The function $g : \mathbb{N}^m \to \mathbb{N}$ defined by

$$g(x_1^1, \ldots, x_1^{m(1)}, \ldots, x_n^1, \ldots, x_n^{m(n)}) = \sum_{i=1}^{n} \prod_{j=1}^{m(i)} x_i^j$$

is a basic arithmetical function. So, by Lemma 3, there is an $\mathbf{ILAL}_{\multimap\otimes}$ proof

$$\pi_g : A_1^1, \ldots, A_1^{m(1)}, \ldots, A_n^1, \ldots, A_n^{m(n)} \vdash \S^k PInt(A)$$

encoding g. Now, the function f is obtained from g by:

(i) Substituting the constant $a_{cp(i)}^{ca(i)}$ for each $x_{cp(i)}^{ca(i)}$ $(1 \leqslant i \leqslant p)$.

(ii) Identifying all $x_{ip(i)}^{ia(i)}$ $(1 \leqslant i \leqslant q)$ with the same variable x.

An idea to define from π_g a proof π_f representing f is thus:

— For (i), perform p cuts of π_g with proofs representing the integers $a_{cp(1)}^{ca(1)}, \ldots, a_{cp(p)}^{ca(p)}$.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

— For (ii), cut the proof with another proof ρ that, intuitively, transforms an integer k into q copies of k.

The proof ρ can in fact be defined without using contraction, by simply applying the iteration scheme associated with an integer formula: the term underlying ρ : $PInt(A^{ia(1)}_{ip(1)} \otimes \ldots \otimes A^{ia(q)}_{ip(q)}) \vdash \S(A^{ia(1)}_{ip(1)} \otimes \ldots \otimes A^{ia(q)}_{ip(q)})$ is

$$x_1(M_{+1}, \ldots, M_{+1})(M_0, \ldots, M_0)$$

where

$$\begin{aligned} M_{+1} &= \lambda x.\lambda y.\lambda z.y((xy)z) \\ M_0 &= \lambda x.\lambda y.y. \end{aligned}$$

The proof π_f can then be defined as

$$\dfrac{\rho \qquad \dfrac{\dfrac{\omega(a^{ca(p)}_{cp(p)}) :\vdash A^{ca(p)}_{cp(p)} \qquad \dfrac{\omega(a^{ca(1)}_{cp(1)}) :\vdash A^{ca(1)}_{cp(1)} \qquad \pi_g : A^1_1, \ldots, A^{m(n)}_n \vdash \S^k PInt(A)}{A^{ca(p)}_{cp(p)}, A^{ia(1)}_{ip(1)}, \ldots, A^{ia(q)}_{ip(q)} \vdash \S^k PInt(A)}}{A^{ia(1)}_{ip(1)}, \ldots, A^{ia(q)}_{ip(q)} \vdash \S^k PInt(A)}}{\S(A^{ia(1)}_{ip(1)} \otimes \ldots \otimes A^{ia(q)}_{ip(q)}) \vdash \S^{k+1}(PInt(A))}}{PInt(A^{ia(1)}_{ip(1)} \otimes \ldots \otimes A^{ia(q)}_{ip(q)}) \vdash \S^{k+1} PInt(A)}$$

For every i, the term underlying $\omega(a^{ca(i)}_{cp(i)}) :\vdash A^{ca(i)}_{cp(i)}$ is the $a^{ca(i)}_{cp(i)}$-th Church numeral. This concludes the proof. □

7. Linear quantifiers and fixpoints

We saw that $\mathbf{ILAL}_{\multimap}$ is not polytime complete while $\mathbf{ILAL}_{\multimap\otimes\forall\mu}$ is not polytime sound. We thus would like to consider an intermediate system enjoying both properties. A possible approach for this is to try to limit the power of quantifiers and fixpoints.

Note that quantification with instantiation on formulae with modality § was a crucial component of the counter-example of Proposition 2 in Section 4. Indeed, the rules $L_\forall$, L_μ and R_μ (see Figures 3, 4) used with formulae A containing modalities are responsible for **ILAL** not being reflective because, reading the proof from the bottom up, they introduce new occurrences of modalities that can allow new $P_!$ or $P_\S$ rules. Thus, a natural remedy is to restrict the use of $\forall$ and μ.

We say an $\mathbf{ILAL}_{\multimap\otimes\forall\mu}$ formula A is *linear* if it does not contain any instance of ! or §. We use $\mathscr{L}$ to denote the class of **ILAL** linear formulae.

We want to replace rules $L_\forall$, L_μ and R_μ by rules that can be applied only when A is a linear formula. To achieve this, we introduce two new connectives $\overline{\forall}$ and $\overline{\mu}$, defined by the rules in Figure 7.

We use $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ to denote this new fragment. The following proposition can be verified.

Proposition 5. The fragment $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ is stable by cut-elimination.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

$$\frac{\Gamma, C[A/\alpha] \vdash B \quad A \in \mathscr{L}}{\Gamma, \overline{\forall}\alpha.C \vdash B} L_{\overline{\forall}} \qquad \frac{\Gamma \vdash C \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \overline{\forall}\alpha.C} R_{\overline{\forall}}$$

$$\frac{\Gamma, A[\overline{\mu}\alpha.A/\alpha] \vdash B \quad A \in \mathscr{L}}{\Gamma, \overline{\mu}\alpha.A \vdash B} L_{\overline{\mu}} \qquad \frac{\Gamma \vdash A[\overline{\mu}\alpha.A/\alpha] \quad A \in \mathscr{L}}{\Gamma \vdash \overline{\mu}\alpha.A} R_{\overline{\mu}}$$

Fig. 7. Linear quantifiers and fixpoints

Observe that when read bottom-up the rules $L_{\overline{\forall}}$, $L_{\overline{\mu}}$, $R_{\overline{\mu}}$ do not introduce new occurrences of ! or §. It follows that the number of rules $P_!^1$, $P_!^2$ and $P_\S$ in a cut-free $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ proof is bounded by the number of occurrences of ! and § in its conclusion; therefore we have the following fact.

Fact 1. The fragment $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ is reflective.

So, by Theorem 1, we have the following proposition.

Proposition 6. The system $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ is polytime sound.

We will now show that this fragment is also polytime complete.

A Turing Machine $\mathscr{M}$ is described by:

— A finite alphabet $\Sigma = \{a_1, \ldots, a_n\}$, where a_1 is considered to be the blank symbol.
— A set $Q = \{q_1, \ldots, q_m\}$ of states, where q_1 is considered to be the starting state.
— A transition function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \rightarrow, \downarrow\}$.

A configuration for $\mathscr{M}$ is a quadruple in $\Sigma^* \times \Sigma \times \Sigma^* \times Q$. For example, if $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$, then $\mathscr{M}$ evolves from (sa_p, a_j, t, q_i) to $(s, a_p, a_k t, q_l)$ (and from $(\varepsilon, a_j, t, q_i)$ to $(\varepsilon, a_1, a_k t, q_l)$).

Linear quantifiers and fixpoints are enough to code a transition function. First, we define the following formulae (parameterised on k):

$$\begin{aligned}
PString_k(\alpha) &= \overline{\mu}\beta.\underbrace{(\beta \multimap \alpha) \multimap \ldots \multimap (\beta \multimap \alpha)}_{k\ times} \multimap \alpha \multimap \alpha \\
PChar_k(\alpha) &= \underbrace{\alpha \multimap \ldots \multimap \alpha}_{k\ times} \multimap \alpha \\
PState_k(\alpha) &= \underbrace{\alpha \multimap \ldots \multimap \alpha}_{k\ times} \multimap \alpha \\
SOString_k &= \overline{\forall}\alpha.PString_k(\alpha) \\
SOChar_k &= \overline{\forall}\alpha.PChar_k(\alpha) \\
SOState_k &= \overline{\forall}\alpha.PState_k(\alpha).
\end{aligned}$$

For every $i \in \{1, \ldots, n\}$, the symbol a_i will be represented by

$$projection_i^n = \lambda x_1 \ldots \lambda x_n . x_i,$$

which, viewed as a proof, has conclusion $SOChar_n$. Analogously, state q_i will be represented by $projection_i^m$. The counterparts of strings in Σ^* are defined by induction:

— The empty string ε is represented by $projection_{n+1}^{n+1}$.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

— If $t \in \Sigma^*$ is represented by M, then, for every $i \in \{1, \dots, n\}$, the string $a_i t$ is represented by

$$\lambda x_1 \dots \lambda x_n \lambda x_{n+1}.x_i M.$$

This encoding can be seen as a variant of lists in the Scott numeral representation (Wadsworth 1980).
The term $append_i^n : SOString_n \multimap SOString_n$ encodes the juxtaposition of a_i to the input string:

$$append_i^n = \lambda x.\lambda x_1 \dots \lambda x_{n+1}.x_i x.$$

Configurations become cut-free proofs for

$$SOConfig_n^m = SOString_n \otimes SOChar_n \otimes SOString_n \otimes SOState_m.$$

Lemma 4. For any transition function of a Turing machine there exists an $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ proof of $\vdash SOConfig_n^m \multimap SOConfig_n^m$ representing it.

Proof. We construct such a proof $step_{\mathscr{M}}$. The λ-term corresponding to $step_{\mathscr{M}}$ is

$$\lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (s, a, t, q)\ \ \mathbf{in}\ \ (q\ M_1\ \dots\ M_m)(s, a, t) \tag{1}$$

where, for every i, term M_i has type $SOString_n \otimes SOChar_n \otimes SOString_n \multimap SOConfig_n^m$. Note that in this proof the $\overline{\forall}$ quantifier of the $SOState_m$ type of q is instantiated with the type of M_i, $SOString_n \otimes SOChar_n \otimes SOString_n \multimap SOConfig_n^m$.

Now, M_i is itself given by

$$\lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (s, a, t)\ \ \mathbf{in}\ \ (a\ N_i^1\ \dots\ N_i^n)(s, t). \tag{2}$$

Each $N_i^j : SOString_n \otimes SOString_n \multimap SOConfig_n^m$ encodes the value of $\delta(q_i, a_j)$. Note that in the proof for M_i the $\overline{\forall}$ quantifier of the type $SOChar_n$ of a is instantiated with the type of the N_i^j.

Finally, we describe how the N_i^j are defined. If, as in the example above, $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$, N_i^j will be the term

$$\lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (s, t)\ \ \mathbf{in}\ \ (s\ P^1\ \dots\ P^n\ R)t \tag{3}$$

where, for every r, term $P^r : SOString_n \multimap SOString_n \multimap SOConfig_n^m$ is

$$\lambda s.\lambda t.(s, projection_r^n, append_k^n\ t, projection_l^m)$$

and $R : SOString_n \multimap SOConfig_n^m$ is

$$\lambda t.(projection_{n+1}^{n+1}, projection_1^n, t, projection_l^m).$$

In the proof for N_i^j the quantifier $\overline{\forall}$ of the type $SOString_n$ of s is instantiated with the type of R. □

We can finally prove the following result.

Theorem 3. $f : \{0, 1\}^* \to \{0, 1\}^*$ is computable by a polynomial time Turing Machine iff f is uniformly encodable into $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$. Thus $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ is polytime sound and complete.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

$$\frac{\Gamma \vdash A}{\Gamma, 1 \vdash A} L_1 \qquad \frac{}{\vdash 1} R_1$$

Fig. 8. Rules for constant 1

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C} L_\oplus \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} R^1_\oplus \qquad \frac{\Gamma \vdash A}{\Gamma \vdash B \oplus A} R^2_\oplus$$

Fig. 9. Rules for $\oplus$

Proof. Let $\mathscr{M}$ be a Turing machine running in time $f : \mathbb{N} \to \mathbb{N}$. If f is a polynomial, Proposition 4 gives us a proof

$$\pi_f : PInt(B) \vdash \S^k PInt(SOConfig^m_n)$$

encoding f. Using Lemma 4, the term underlying $step_{\mathscr{M}}$ can be typed by

$$\S^k !(SOConfig^m_n \multimap SOConfig^m_n).$$

Putting these two ingredients together, we obtain a proof

$$\pi_{\mathscr{M}} : PInt(B) \otimes \S^{k+1} SOConfig^m_n \vdash \S^{k+1} SOConfig^m_n,$$

which is a uniform encoding for the function computed by $\mathscr{M}$. $\square$

8. Additive connective $\oplus$ and fixpoints

Note that we have used linear quantification in the previous section essentially to deal with case distinction. This can, in fact, also be done using another feature of linear logic: the additive connectives & and $\oplus$. In the intuitionistic setting that we are considering, it is even enough for our purposes to consider the connective $\oplus$ only. We give the rules for $\oplus$ in Figure 9. We will also use the constant for the connective $\otimes$, denoted 1: the corresponding rules are given in Figure 8.

The term language is extended accordingly with the following new productions:

$$M ::= 1 \mid \mathbf{inl}(M) \mid \mathbf{inr}(M) \mid \mathbf{case}\ \ M\ \ \mathbf{of}\ \ \mathbf{inl}(x) \Rightarrow M, \mathbf{inr}(x) \Rightarrow M.$$

The fragment we are dealing with now is thus **ILAL** with $\multimap$, $\otimes$, 1, $\oplus$, $\overline{\mu}$, but no quantification. We will now show that the step function of a Turing machine can be encoded in this fragment. Using the previous encoding of polynomials, we will then be able to deduce that polytime Turing machines can be simulated in this fragment. We use the connective $\oplus$ to define enumeration types and case distinction on those types (in particular, a conditional test with boolean type).

We define $ABool_k = 1 \oplus \ldots \oplus 1$ (with k components) for $k \geqslant 1$. This formula represents the k-ary boolean type and we use $\underline{1}, \ldots, \underline{k}$ to denote its k normal proofs. We use

$$\frac{\Gamma \vdash M_1 : B \quad \ldots \quad \Gamma \vdash M_k : B}{\Gamma, x : ABool_k \vdash \mathbf{case}\ \ x\ \ \mathbf{of}\ \ \underline{1} \Rightarrow M_1, \ldots, \underline{k} \Rightarrow M_k : B}$$

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

as a shorthand term notation for the case distinction defined on $ABool_k$ using the previous rules.

To simulate a Turing machine $\mathcal{M}$, we set

$$\begin{aligned} AChar_k &= ABool_k \\ AState_k &= ABool_k, \\ AString_k &= \overline{\mu}\alpha.(1 \oplus (AChar_k \otimes \alpha)) \\ AConfig_n^m &= AString_n \otimes AChar_n \otimes AString_n \otimes AState_m. \end{aligned}$$

The empty string ε is represented by $\mathbf{inl}(1)$. The symbol a_i $(1 \leqslant i \leqslant n)$ is represented by $\underline{i}$ with conclusion $AChar_n$, and the state q_j $(j \leqslant m)$ by $\underline{j}$ with conclusion $AState_m$.

Then we can define proofs for

$$\begin{aligned} cons &: AChar_n \otimes AString_n \multimap AString_n \\ pop &: AString_n \multimap AChar_n \otimes AString_n \end{aligned}$$

by

$$\begin{aligned} cons &= \lambda x.\mathbf{let}\ x\ \mathbf{be}\ (a,s)\ \mathbf{in}\ \mathbf{inr}(a,s) \\ pop &= \lambda s.\mathbf{case}\ s\ \mathbf{of}\ \mathbf{inl}(x) \Rightarrow (\underline{1},\mathbf{inl}(1)), \mathbf{inr}(y) \Rightarrow y\,. \end{aligned}$$

The proof *pop* applied to a non-empty string returns its head and tail; by convention it returns $(\underline{1},\mathbf{inl}(1))$ when applied to the empty string.

Given the transition function δ of $\mathcal{M}$ we construct a proof $step_{\mathcal{M}} : AConfig_n^m \multimap AConfig_n^m$ implementing a step of execution of $\mathcal{M}$:

$$\begin{aligned} step_{\mathcal{M}} = \lambda x.&\mathbf{let}\ x\ \mathbf{be}\ (s,a,t,q)\ \mathbf{in} \\ &\mathbf{case}\ q\ \mathbf{of}\ (\ldots,\underline{i} \Rightarrow (\mathbf{case}\ a\ \mathbf{of}\ (\ldots,\underline{j} \Rightarrow M_{i,j},\ldots),\ldots) \end{aligned}$$

where $M_{i,j}$ is defined according to the value of $\delta(q_i,a_j)$. For instance, if $\delta(q_i,a_j) = (q_l,a_k,\leftarrow)$,

$$M_{i,j} = \mathbf{let}\ (pop\ s)\ \mathbf{be}\ (b,r)\ \mathbf{in}\ (r,b,(cons\ \underline{k}\ t),\underline{l}).$$

Then, arguing as in the previous section, we have the following proposition.

Proposition 7. The system $\mathbf{ILAL}_{\multimap\otimes\oplus\overline{\mu}}$ is polytime sound and complete.

9. Getting rid of second-order quantification

Looking at the encoding of Turing machines from Section 7, we can can see that (linear) second order is used in a very restricted way there. In the proof $step_{\mathcal{M}}$, there are just three instances of the $L_{\overline{\forall}}$ rule: acting on $SOString_n$, $SOChar_n$ and $SOState_m$, respectively. In particular:

— $SOState_m$ is instantiated with the formula

$$SOString_n \otimes SOChar_n \otimes SOString_n \multimap SOConfig_n^m$$

(see (1) in the proof of Lemma 4 in Section 7).

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

— $SOChar_n$ is instantiated with the formula

$$SOString_n \otimes SOString_n \multimap SOConfig_n^m$$

(see (2)).

— $SOString_n$ is instantiated with the formula

$$SOString_n \multimap SOConfig_n^m$$

(see (3)).

Thus, in order to type the terms from (1), (2) and (3), it would be sufficient to replace the use of $\overline{\forall}$ by fixpoint constructions allowing us to do three similar instantiations. This is what we are going to do here, obtaining in this way a suitable transition function in $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$, whose associated term is the same as the one in Lemma 4.

Two formulae A and B are said to be *congruent*, written $A \approx B$, if B can be obtained from A by applying (zero or more times) the rule $\overline{\mu}\alpha.A \approx A[\overline{\mu}\alpha.A/\alpha]$. In other words, $\approx$ is the reflexive and transitive closure of the above (symmetric) rule.

Note that if $A \approx B$, the identity term can be given type $A \multimap B$, and thus A and B are in particular isomorphic types.

We get the following proposition.

Proposition 8. For every $k, h \in \mathbb{N}$, there are $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ formulae $FPState_k^h$, $FPChar_k^h$ and $FPString_k^h$ such that

$$\begin{aligned}
FPState_k^h &\approx PState_h(FPString_k^h \otimes FPChar_k^h \otimes FPString_k^h \multimap FPConfig_k^h)\\
FPChar_k^h &\approx PChar_k(FPString_k^h \otimes FPString_k^h \multimap FPConfig_k^h)\\
FPString_k^h &\approx PString_k(FPString_k^h \multimap FPConfig_k^h)\\
\text{where } FPConfig_k^h &= FPString_k^h \otimes FPChar_k^h \otimes FPString_k^h \otimes FPState_k^h.
\end{aligned}$$

Proof. Consider the following definitions:

$$\begin{aligned}
FPString_k(\alpha,\beta) &= \overline{\mu}\gamma.PString_k(\gamma \multimap \gamma \otimes \beta \otimes \gamma \otimes \alpha)\\
FPChar_k(\alpha) &= \overline{\mu}\beta.PChar_k(FPString_k(\alpha,\beta) \otimes FPString_k(\alpha,\beta) \multimap\\
&\qquad FPString_k(\alpha,\beta) \otimes \beta \otimes FPString_k(\alpha,\beta) \otimes \alpha)\\
FPState_k^h &= \overline{\mu}\alpha.PState_h(FPString_k(\alpha, FPChar_k(\alpha)) \otimes\\
&\qquad FPChar_k(\alpha) \otimes FPString_k(\alpha, FPChar_k(\alpha)) \multimap\\
&\qquad FPString_k(\alpha, FPChar_k(\alpha)) \otimes FPChar_k(\alpha) \otimes\\
&\qquad FPString_k(\alpha, FPChar_k(\alpha)) \otimes \alpha\\
FPChar_k^h &= FPChar_k(FPState_k^h)\\
FPString_k^h &= FPString_k(FPState_k^h, FPChar_k(FPState_k^h)).
\end{aligned}$$

The thesis then follows easily. □

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

Lemma 5. For any transition function of a Turing machine there exists an $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ proof of $\vdash \mathit{FPConfig}_n^m \multimap \mathit{FPConfig}_n^m$ representing it.

Proof. We can type in $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ the terms of (1), (2), (3) in Section 7 by taking advantage of the congruences given in Proposition 8. □

We then get, as in Section 7, the following theorem.

Theorem 4. The system $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ is polytime sound and complete.

10. Discussion and perspectives

Several papers, such as Marion and Moyen (2000), Marion (2001) and Hofmann (2003), have advocated the study of *intensional* aspects of implicit computational complexity (ICC), that is to say, the study of *algorithms* representable in a given ICC language, as opposed to *functions*. Note that this is not the main focus of the present paper, as we have mainly studied functions representable in ICC systems and used encodings of Turing machines. Moreover, the notion of uniform encodings that we have considered is rather delicate from a programmer's point of view. It allows us to consider different data structures for different inputs and outputs, which, in a sense, gives us more flexibility. However, as it stands, finding the correct data structure would still be part of the programming task, which might be a bit too much to ask. In particular, this would be a problem for modular programming.

Nevertheless, our main point here was rather to stress that in the setting of light logics several interesting fragments or sublanguages are available. Before exploring intensional expressivity, one needs to study extensional expressivity, which is what we have done here for **LAL** and its variants. This gives us some criteria for comparing logical fragments or languages. Once relevant languages with suitable extensional properties have been isolated, they can be used to suggest new constants or programming primitives that are compatible with typing and preserving complexity properties. Note, for example, that the use of type fixpoints has enabled us to give simpler encodings of Turing machines than the original ones for second-order **LAL** (Asperti and Roversi 2002). We leave for future work a more complete study of the intensional aspects of the logical systems identified in this paper.

11. Conclusions

In this paper we have delineated the computational power of several fragments of light affine logic with respect to a general notion of the encoding of functions. The results are summarised in Figure 10, which shows which fragments of **ILAL** are known to be polytime sound and/or complete, or neither. In particular, we have shown, on the one hand, that the purely implicative propositional fragment of **ILAL** is not polytime complete (under a further natural assumption for the encoding), but, on the other hand, the extension with linear fixpoints is both polytime complete and sound.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

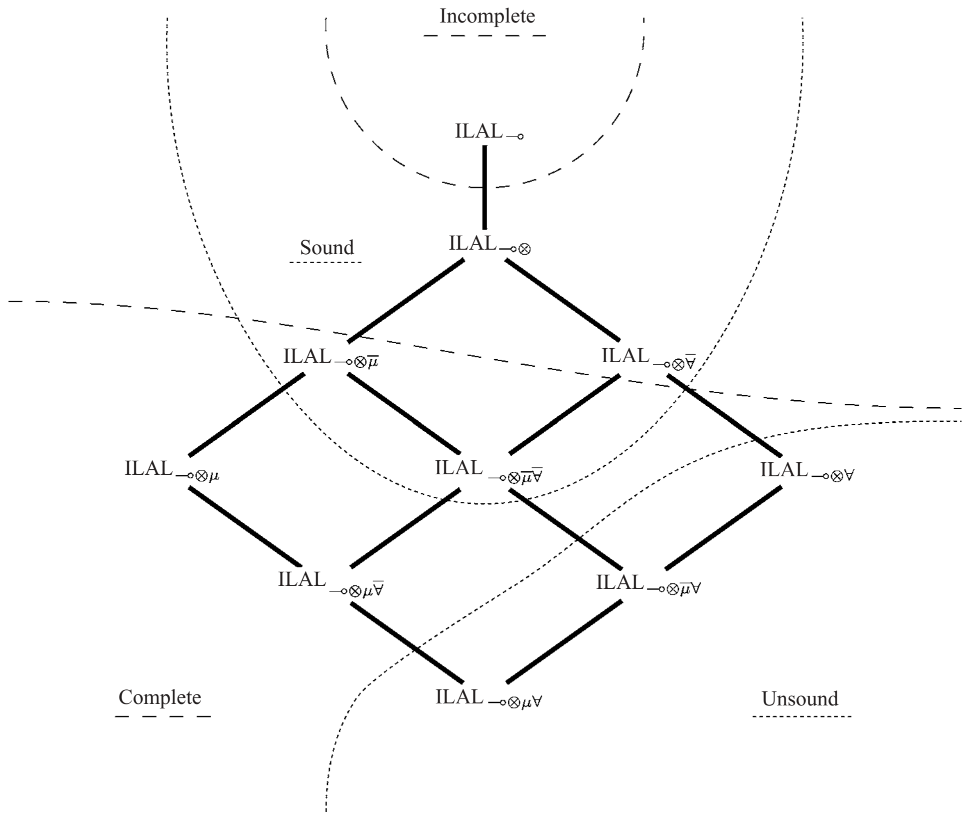

Fig. 10. Polytime soundness and completeness results for fragments of **ILAL**

References

Asperti, A. (1998) Light Affine Logic. In: *Proceedings of the 13th IEEE Syposium on Logic in Computer Science* 300–308.

Asperti, A. and Roversi, L. (2002) Intuitionistic Light Affine Logic. *ACM Transactions on Computational Logic* **3** (1) 137–175.

Baillot, P. and Mogbil, V. (2004) Soft lambda-calculus: a language for polynomial time computation. In: Proceedings of 7th International Conference on Foundations of Software Science and Computation Structures. *Springer-Verlag Lecture Notes in Computer Science* **2987** 29–41.

Dal Lago, U. (2003) On the Expressive Power of Light Affine Logic. In: Proceedings of 8th Italian Conference on Theoretical Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **2841** 216–227.

Dal Lago, U. and Baillot, P. (2004) On Light Logics, Uniform Encodings and Polynomial Time (preliminary version). Presented at the Workshop on Logics for Resources, Processes and Programs (affiliated with LICS and ICALP) 11 pages.

Fortune, S., Leivant, D. and O'Donnell, M. (1983) The expressiveness of simple and second-order type structures. *Journal of the ACM* **30** (1) 151–185.

Girard, J.-Y. (1987) Linear Logic. *Theoretical Computer Science* **50** 1–102.

Girard, J.-Y. (1998) Light Linear Logic. *Information and Computation* **143** (2) 175–204.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press

Hofmann, M. (2003) Linear types and non-size-increasing polynomial time computation. *Information and Computation* **183** (1) 57–85.

Lafont, Y. (2004) Soft Linear Logic and Polynomial Time. *Theoretical Computer Science* **318** (1-2) 163–181.

Leivant, D. and Marion, J.-Y. (1993) Lambda Calculus Characterizations of Poly-Time. *Fundamenta Informaticae* **19** (1/2) 167–184.

Mairson, H. G. and Terui, K. (2003) On the Computational Complexity of Cut-Elimination in Linear Logic. In: Proceedings of 8th Italian Conference on Theoretical Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **2841** 23–36.

Marion, J.-Y. (2001) Complexité implicite des calculs de la théorie à la pratique. Habilitation Thesis, Université Nancy 2.

Marion, J.-Y. and Moyen, J.-Y. (2000) Efficient First Order Functional Program Interpreter with Time Bound Certifications. In: Proceedings of 7th International Conference on Logic for Programming and Automated Reasoning. *Springer-Verlag Lecture Notes in Computer Science* **1955** 25–42.

Murawski, A. and Ong, L. (2000) Discreet Games, Light Affine Logic and PTIME Computation. In: Proceedings of 14th Annual Conference of the European Association of Computer Science Logic. *Springer-Verlag Lecture Notes in Computer Science* **1862** 427–441.

Murawski, A. and Ong, L. (2004) On an interpretation of safe recursion in light affine logic. *Theoretical Computer Science* **318** (1-2) 197–223.

Neergaard, P. M. and Mairson, H. G. (2002) LAL is Square: Representation and Expressiveness in Light Affine Logic. Presented at the Fourth International Workshop on Implicit Computational Complexity.

Roversi, L. (2000) Light Affine Logic as a Programming Language: a First Contribution. *International Journal of Foundations of Computer Science* **11** (1) 113–152.

Statman, R. (1982) Completeness, Invariance, and λ-definability. *Journal of Symbolic Logic* **47** (1) 7–26.

Terui, K. (2002) *Light logic and polynomial time computation*, Ph.D. thesis, Keio University.

Terui, K. (2004) Light Affine Set Theory: A Naive Set Theory of Polynomial Time. *Studia Logica* **77** (1) 9–40.

Wadsworth, C. (1980) Some unusual λ-calculus numeral systems. In: Seldin, J. and Hindley, J. (eds.) *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press.

https://doi.org/10.1017/S0960129506005421 Published online by Cambridge University Press