Proofs as Efficient Programs

Ugo Dal Lago and Simone Martini

Alma Mater Studiorum – Università di Bologna
Dipartimento di Scienze dell'Informazione

There may, indeed, be other uses of the system than its use as a logic.
A. Church [8]

Logic and theory of computation have been intertwined since their first days. The formalized notion(s) of effective computation are at first technical tools for the investigation of first order systems, and only ten years later – in the hands of John von Neumann – become the blueprints of engineered physical devices. Generally, however, one tends to forget that in those same years, in the newly-born proof-theory of Gerhard Gentzen [20] there is an implicit, powerful notion of computation – an effective, combinatorial procedure for the simplification of a proof. However, the complexity of the rules for the elimination of cuts (especially the commutative ones, in the modern jargon) hid the simplicity and generality of the basic computational notion those rules were based upon. We had to wait thirty more years before realizing in full glory that Gentzen's simplification mechanism and one of the formal systems for computability (Church's λ-calculus) were indeed one and the same notion.

As far as we know, Haskell Curry is the first to explicitly realize [11] that the *types* of some of his basic combinators correspond to axioms of intuitionistic implicational calculus, and that, more generally, the types assignable to expressions made up of combinators are exactly the provable formulae of intuitionistic implicational logic. It is William Howard in 1969 to extend this *formulas as types* correspondence to the more general *proofs as programs* isomorphism ([27], published in 1980 but widely circulated before). Under this interpretation, the two dynamics – proof normalization on one hand, and β-reduction on the other – are identified, so that techniques and results from one area are immediately available to the other.

In this paper, we will discuss the use of the Curry-Howard correspondence in *computational complexity theory*, the area of theoretical computer science concerned with the definition and study of complexity classes and their relations. The standard approach to this discipline is to fix first a machine model (e.g., Turing machines) equipped with an explicit cost (e.g., number of transitions) and define then the (inherent) complexity of a function as the minimal cost needed to compute it with those machines. More formally, assuming Turing machines with unitary cost per transition, one says first that a TM M works in time h iff for any input x, $M(x)$ terminates in at most $h(|x|)$ steps, where $|\ |$ is a suitable notion of size for the input. A function f has complexity h iff there exists a TM computing f working in time h. At this point one may define the *complexity class of* h as the set of all functions with complexity h. Main examples of such classes are the polynomial functions (FPTIME), the exponential functions (FEXPTIME), or the Kalmar-elementary ones (FELEMTIME). In building up this theory, one of the first tasks is to show that the definition of a complexity class is somehow independent from the machine model adopted at first. Here comes the notion of *reasonable* machine models [38]:

Reasonable machines can simulate each other within a polynomially-bounded overhead in time and a constant-factor overhead in space.

Complexity classes like FPTIME and the others we mentioned before are clearly invariant with respect reasonable machine models, and are thus amenable to general theoretical treatment.

If these classes are so robust, however, there should be characterizations of them independent from explicit machine models. This is the subject of *implicit computational complexity* (ICC) whose main aim is the description of complexity classes based on language restrictions, and not on external measure conditions or on an explicit machine model. It borrows techniques and results from mathematical logic (model theory, recursion theory, and proof-theory) and in doing so it has allowed the incorporation of aspects of computational complexity into areas such as formal methods in software development and programming language design. The most developed area of implicit computational complexity is probably the model-theoretic one – finite model theory being a very successful way to describe complexity problems. In the design of programming language tools (e.g., type-systems), however, syntactical techniques prove more useful, and in the last twenty years much work has been devoted to restricted ways of formulating recursion theory, and to proof-theoretical techniques to enforce resource bounds on programs.

1 Proofs as Programs

The Curry-Howard correspondence between natural deduction proofs and lambda-terms is summarized in Fig. 1. Formally, we start by defining *λ-terms* as:

$$M ::= x \mid \lambda x.M \mid MM\ ,$$

where x ranges over a denumerable set of *variables*. As usual, λ binds variables in its scope. *Formulas* (or *types*) are defined starting from a base type (o) as

$$A ::= \mathrm{o} \mid A \to A\ .$$

The system in Fig. 1 defines judgments of the form $\Gamma \vdash M : A$, where M is a term, A is a formula, and Γ is a set of pairs $x : A$, where x is a variable and all the variables in Γ are distinct.

Fig. 1. Natural deduction proofs and typed lambda-terms

$$\Gamma, x : A \vdash x : A$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ {\scriptstyle (\to,E)} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ {\scriptstyle (\to,I)}$$

If we forget terms, we are left with the usual natural deduction system for propositional implicational intuitionistic logic. Terms may be seen just as a convenient, linearized way to write the proofs, instead of the usual two-dimensional tree-like notation. On the other hand, a computer scientist will recognise the rules assigning types to functional programs. This correspondence between proofs and programs is completed by observing that the notion of proof normalization (that is, the elimination of the detours composed of an introduction immediately followed by an elimination) is the same as the reduction of the corresponding term (see Fig. 2, where we have adopted for more clarity the standard two-dimensional notation for proofs); $M[x \leftarrow N]$ denotes the substition of N for the free occurrences of x in M.

Fig. 2. Normalization is beta-reduction

$$\frac{\dfrac{\begin{matrix}[x : A] \\ \pi \\ M : B\end{matrix}}{\lambda x.M : A \to B} \quad \begin{matrix}\delta \\ N : A\end{matrix}}{(MN) : B} \quad \Longrightarrow \quad \begin{matrix}\delta \\ N : A \\ \pi \\ M[x \leftarrow N] : B\end{matrix}$$

We may now apply logical methods to the study of computation as term rewriting, and vice versa. From the point of view of computational complexity, however, λ-calculus is a rather awkward computational model, since it misses a basic notion of elementary step with bounded cost. Indeed, it is evident that a step in a Turing machine can be "actually performed" in constant time, thanks to the finiteness of the states and the symbols. Even for machine models with non-constant time step – like Unbounded Register Machines (URM, see, e.g., [12]), where in a single step one may store or increment an arbitrary large natural number – one can usually give a *natural* and plausible cost model (for URMs one can assume, for instance, that one step accounts for the logarithm of the length of the manipulated number). For λ-calculus all these considerations seem to vanish, since beta-reduction is inherently too complex to be considered as an atomic operation, at least if we stick to explicit representations of lambda terms. Indeed, in a beta step

$$(\lambda x.M)N \rightarrow M[x \leftarrow N] ,$$

there can be as many as $|M|$ occurrences of x inside M. As a consequence, $M[x \leftarrow N]$ can be as big as $|M| \times |N|$. And this applies to any step during a reduction, where the terms N and M have no longer a direct evident connection with the original term.

As a result, in the literature the actual cost of normalizing a lambda-term has been studied without any reference to the number of beta steps to normal form. One of the main results of this field is Statman's theorem [35], showing that there is no Kalmar-elementary function of the length of a typed λ-term bounding the work needed to compute its normal form. This result, in a sense, matches in λ-calculus the series of results in proof theory giving bounds on the size of a *normal* proof, as a function of the size of the proof before normalization (see, e.g., [22]). These results could be summarized by saying that, in any reasonable logic, if the (natural deduction) proof π normalizes to ρ, then the size of ρ can be (hyper-)exponential in the size of π. But then we are faced with another problem in the direct application of the Curry-Howard correspondence to computational complexity. How could we use a tool that, in any interesting case, has a hyper-exponential complexity? How is it possible to use it for studying interesting complexity classes, and especially the most important PTIME?

The solution comes from a different perspective, the natural one from the programming view-point. We should forget the natural symmetry, present in λ-calculus, between programs (terms) and data (just other terms), and recall that, in computing, the roles of programs and data *are not* symmetric. We are instead interested in the behavior of a single, fixed program as a function of the size of its data. That is, under the proofs-as-programs isomorphism, the behavior of a fixed proof (say $\Gamma \vdash M : A \rightarrow B$) when composed with all sensible proofs (that is when "cut" against – applied to – all proofs of the shape $\Gamma \vdash N : A$). In explicit terms, we should study the time to normalize MN, with M in normal form, as a function of $|N|$ only.

We may borrow some terminology from software engineering and distinguish two main ways of applying logical notions and techniques to computational complexity.

1. Implicit computational complexity *in-the-large*, using logic to study complexity classes. More specifically, given a complexity class $\mathcal{C}$, find a logical system G such that:

Fig. 3. Intuitionistic Multiplicative Exponential Linear Logic

$$A \vdash A \ {\scriptstyle (Ax)} \qquad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B} \ {\scriptstyle (Cut)}$$

$$\frac{\Gamma \vdash C}{\Gamma, !A \vdash C} \ {\scriptstyle (Weak.)} \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \ {\scriptstyle (Contr.)}$$

$$\frac{\Gamma \vdash A \quad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C} \ {\scriptstyle (\multimap, l)} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \ {\scriptstyle (\multimap, r)}$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \ {\scriptstyle (\otimes_i, l)} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \ {\scriptstyle (\otimes, r)}$$

$$\frac{A_1, \ldots, A_n \vdash B}{!A_1, \ldots, !A_n \vdash !B} \ {\scriptstyle (!)}$$

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \ {\scriptstyle (\epsilon)} \qquad \frac{\Gamma, !!A \vdash B}{\Gamma, !A \vdash B} \ {\scriptstyle (\delta)}$$

$$\frac{\Gamma, T[S/t] \vdash C}{\Gamma, \forall t.T \vdash C} \ {\scriptstyle (\forall, l)} \qquad \frac{\Gamma \vdash C}{\Gamma \vdash \forall t.C} \ {\scriptstyle t \not\in FV(\Gamma)} \ {\scriptstyle (\forall, r)}$$

   Soundness: for any interesting type A and $M : A \to A$ in the system G, there is a function $f_A \in \mathcal{C}$ such that for any $N : A$, the cost of normalizing MN is bounded by $f_A(|N|)$.
   Completeness: for any function F computable in complexity $\mathcal{C}$, there is a proof M_F (in system G) "representing' F.

2. Implicit computational complexity *in-the-small*, using logic to study single machine-free models of computation. More specifically, in the context we just described, giving *natural* cost models for λ-calculus reduction, showing that it is indeed a reasonable model (that is, polynomially related to Turing machines).

2 ICC in the Large

Finding a system characterizing interesting complexity classes is far from obvious, all standard logics having a too high complexity of reduction. The breakthrough came with Girard's *light linear logic* [25]. Linear logic [21] put under scrutiny the structural rules (especially contraction, responsible for arbitrary duplication of subproofs during cut-elimination) and their role during normalization. Intuitionistic logic, however, may be embedded in linear logic, and therefore the latter cannot be used as a limited complexity formal system. One has to weaken linear logic, limiting the use of contraction.

Second order multiplicative linear logic is summarized in Fig. 3, as a sequent calculus. Rule (*Contr.*) is the culprit for exponential blow-up, but since it may only be applied to !-marked formulas, the rules for (!) are our main source of concern. If we look at the logic from an axiomatic point of view, one of the fundamental properties of the logic is the law $!A \equiv !A \otimes !A$ ($A \equiv B$ is short for $A \multimap B$ and $B \multimap A$), which is obtained from rules (*Contr.*), (*Weak.*) and (!). Other important laws are of course derivable (! acts much the same as $\Box$ in modal logic S4), namely $!A \multimap A$ (from (ϵ), "dereliction") and $!A \multimap !!A$ (from (δ), "digging"). The interplay between these rules is the main source for complexity of normalization (and for the expressivity of the logic, of course). Girard's insight was to weaken the operator !, still maintaining enough expressivity to code interesting functions.

If we drop (ϵ) and (δ), we get *Elementary linear logic* (ELL), which is sound and complete (in the meaning explained above) for Kalmar-elementary functions, a complexity class still too big for computer science. To get just the polynomial functions we must drastically restrict (!) to have at most one formula to the left of $\vdash$. But now the system would be too weak (far from complete from polytime, that is) and to fix this Girard adds another modality (§), to compensate for the loss of a full (!)-rule. The system is best formulated in an affine form (i.e., full weakening is allowed), following [2]. The relevant rules for *Light affine logic* (LAL) are summarized in Fig. 4 (they replace rules (*Weak.*), (*Contr.*), (!), (ϵ), and (δ) of Fig. 3; in (§), $n, m \geq 0$; in (!u), A can be absent).

Fig. 4. Light affine logic, LAL

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ (AWeak.) \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ (Contr.)$$

$$\frac{A_1, \ldots, A_n, C_1, \ldots, C_m \vdash B}{!A_1, \ldots, !A_n, \S C_1, \ldots, \S C_m \vdash \S B}\ (\S) \qquad \frac{A \vdash B}{!A \vdash !B}\ (!u)$$

With this key idea, one obtains indeed a sound and complete system for polytime. The following theorem expresses the soundness result, the most delicate and interesting one. We observe, first, that there is natural way to code in the logic, as a type, interesting data types; the theorem may be stated with respect to $\ulcorner BS \urcorner$, the type coding binary strings.

Theorem 1. *Let* $\vdash_{\text{LAL}} M : \ulcorner BS \urcorner \to \ulcorner BS \urcorner$. *For any* $\vdash_{\text{LAL}} W : \ulcorner BS \urcorner$, *we can compute the normal form of* MW *in time proportional to* $(d+1)|MW|^{2d}$, *where* d *depends only on* M.

Observe that, when M is fixed and in normal form, the theorem says that M is a program with complexity $O(|W|^{2d})$, that is, polynomial in its input. Care should be taken, however, on the correct reading of the normalization result. In order the theorem to hold as stated, we cannot simply reduce MW

as a lambda-term (because even terms typable in LAL could have esponentially long normal forms, though "computable" in polynomial time [6]). *Proof-nets* (the intended notation for proofs in linear logic) should be used instead.

Once discovered, the technology has been applied, *mutatis mutandis*, to several other formal systems. In addition to LLL and LAL, we have EAL (for elementary time), SLL (for polynomial time [29]), STA$_B$ (for logarithmic space [19]), etc. Moreover, we now have also systems where, contrary to LAL, the soundness for polynomial time holds for lambda-calculus reduction, like DLAL [6] and other similar systems. As a result, the general framework of light logics is now full of different systems, and of variants of those systems. It is urgent to get *general*, *unifying* results to compare the systems and improve on them.

As already mentioned, soundness and completeness are the key results to be obtained when trying to characterize complexity classes by way of logical systems. Completeness is always of an *extensional* nature: one proves that any function in the complexity class under consideration can be represented by a proof in the logical system, without taking care of the *intensional* aspect of the proof itself. Since the same function can be represented by many, distinct proofs (in particular those corresponding to unnatural algorithms, in computer science terminology), proving completeness does not say much about the usefulness of the considered system as a programming language. On the other hand, soundness is a key property: any representable function is in a complexity class. This is often proved by showing that *any* proof in the logical system can be normalized with a bounded amount of resources. This way, soundness proofs give insights on the *reasons* why normalization is a relatively easy computational process. As a consequence, soundness proofs are interesting on their own:

- The intensional expressive power of two or more apparently unrelated systems can be compared through their soundness proofs. This requires the soundness proofs to be phrased in the same framework.
- The possibility of extending sound systems with new rules and features can be made easier if their soundness proofs are designed to be adaptable to extensions in the underlying syntax.

Unfortunately, soundness has been traditionally proved with *ad hoc* techniques which cannot be easily generalized. Different systems has been proved sound with very different methodologies. As an example, Lafont's SLL has been proved to be polytime sound by a simple mathematical argument, while Asperti's LAL requires a complicated argument, in particular if strong soundness is needed [36].

In the last years, much effort has been put in the task of finding powerful, simple and unifying semantic framework for implicit computational complexity. In the rest of the section we will give an account of some of these frameworks, with particular emphasis on authors' contribution. What we would like to convey are not the technicalities, of course, but the flexibility of the approaches – simple modifications to a general framework allow for the semantical description (and sometimes also for syntactical results, like soundness) of a wide spectrum of formal systems.

2.1 Context Semantics

Context semantics [26] is a powerful framework for the analysis of proof and program dynamics. It can be considered as a model of Girard's geometry of interaction [24, 23] where the underlying algebra consists of *contexts*. As such, it is a dynamic, interactive semantical framework, with many similarities to game semantics [3]. In these semantic frameworks, one manages to identify those proofs interacting in the same way with the environment (i.e., that cannot be distiguished, through normalization, no matter the context in which they are used in a larger proof; technically: observationally equivalent proofs) by proving that the semantic objects which interpret them are the same. In some cases, one can prove that the converse implication holds – this property is called *full-abstraction* in the literature. Context semantics and geometry of interaction have been used to prove the correctness of optimal reduction algorithms [26] and in the design of sequential and parallel interpreters for the λ-calculus [31, 33].

Notice that whenever a proof π is obtained from another proof ρ by cut-elimination (or normalization), π and ρ have the same interactive behaviour and, as a consequence, cannot be distinguished in the semantics. This implies that the "complexity" of normalizing a proof π *cannot* be read out of the interpretation of π itself, in any standard semantical approach. What is appealing in context semantics is that its formulation, differently from the majority of other approaches, allows for easy modifications of the semantics of π (without altering the basic mathematical concepts needed), in such a way that the interpretation of π somehow "reflects" the computational difficulty of normalizing it. A first example of this flexibility of the context semantics, is the well known fact that strongly normalizing proofs are exactly the ones having finitely many so-called regular paths in the geometry of interaction [18]. As a second example, a class of proofs which are not just strongly normalizing but normalizable in elementary time can still be captured in the geometry of interaction framework, as suggested by Baillot and Pedicini [5]. Until recently it was not known, however, whether this correspondence scales down to smaller complexity classes, such as the one of polynomial time computable functions. The usual measure based on the length of regular paths cannot be used, since there are proofs which can be normalized in polynomial time but whose regular paths have exponential length [18].

The solution to the above problem is relatively simple: not *every* (regular) path should be taken into account, but only those having something to do with duplication. This way, one can define the *weight* W_π of any proof as the total number of paths in this restricted ¡class. On one hand, we may prove that W_π is an upper bound to the computational difficulty of normalizing π:

Theorem 2. *There is a polynomial $p : \mathbb{N}^2 \to \mathbb{N}$ such that for every proof π, π normalizes in at most $p(W_\pi, |\pi|)$ steps and the size of any reduct ρ of π is at most $p(W_\pi, |\pi|)$.*

On the other hand, W_π can be proved to be a lower bound to the complexity of normalizing π:

Theorem 3. *For every proof π there is another proof ρ such that π normalizes in n steps to ρ and $W_\pi \geq n + |\rho|$.*

The two results above highlight a strong correspondence between W_π and some crucial quantitative attributes of π, i.e. the (maximal) number of steps to normal form for π and the (maximal) size of reducts of π. Perhaps the most interesting application of this correspondence are new, simple proofs of soundness for elementary, light and soft linear logic. After all, W_π can be defined for any (multiplicative and exponential) proof π, while proving bounds on W_π when some of the rules are forbidden (as in the cited logical systems) turns out to be very simple. Details about the results sketched in this paragraph can be found in [14].

The same technique can be applied to similar, but different, systems. A first example consists in linear lambda calculi with higher-order recursion [13]. There, a parameter influencing the complexity of any term M is the (maximal) size of ground subterms of reducts of M, called the *algebraic potential size* of M. Context semantics is powerful enough to induce bounds on the algebraic potential size of terms, in the same vein as in Theorem 2. Another example is optimal reduction [1], where context semantics allows to transfer known complexity results from global reduction, where proofs are normalized in the usual way, to local reduction, where a slightly different syntax for proofs allows a completely local notion of reduction [4], itself necessary to get optimality for lambda calculus.

2.2 Realizability Semantics

Kleene's realizability and its variations have been used for various purposes in modern logic (see [37] for an introduction). In particular, realizability has been used in connection with complexity-constrained computation in several places. The most prominent one is Cook and Urquhart's work [9], where terms of a language called PV^ω are used to realize formulas of bounded arithmetic. There, realizability is used to show polytime soundness of a logic (for a slightly different notion of soundness than the one considered here). Realizers in [9] are typed and very closely related to the logic that is being realized. This implies that any modification to the underlying logical system would require changing as well the language of realizers.

On the other hand, one can use untyped realizers and interpret formulas as partial equivalence relations on those, this way going towards a unifying framework. This has been done by Crossley et al. [10], in the same spirit as

Kreisel's untyped realizability model HEO [28]. The proof that the untyped realizers (in this case of formulas of bounded arithmetic) can indeed be normalized in bounded time, has to be performed "externally" (that is, it is not obtained directly from the model). In other words, proving soundess of a logic has been reduced to proving soundness of a certain class of realizers.

Recently, a realizability framework has been introduced by one of the authors, as a joint work with Martin Hofmann [15]. The framework is designed as a refinement of the existing ones. In standard realizability, formulas are usually interpreted as sets of pairs (t, f), where t (a realizer, e.g., a lambda term) realizes f (a denotational object, e.g., a function); in our approach formulas are interpreted as sets of triples (α, t, f), where t realizes f as usual but, moreover, t is *majorized* by α (itself an element of an algebraic structure called a *resource monoid*). The notions of majorization and of resource monoid are relatively simple and designed to guarantee that whenever $(\alpha, t, f) \in A \to B$ and $(\beta, u, g) \in A$, t computes a realizer for $f(g)$ in time bounded by $\mathcal{F}(\alpha+\beta)$ when applied to u ($\mathcal{F}$ is a function from majorizers to natural numbers which is part of the definition of the underlying resource monoid). Proving that $\mathcal{F}$ is bounded (e.g., by something like a polynomial) is usually easy, once a resource monoid is fixed. Therefore, by choosing a proper resource monoid, "intensional" soundness can be obtained as a corollary of "extensional" soundness while giving meaning to a logic. Notice, however, that this way we do not prove anything about the time needed to normalize *proofs*, but we rather prove something about *realizers* corresponding to the interepreted proofs.

Once a resource monoid $\mathcal{M}$ has been fixed, one can present the above construction as a category $\mathscr{L}(\mathcal{M})$: objects are *length spaces* (i.e., the ternary relations described above) while morphisms are realized and majorized functions between them. Noticeably, $\mathscr{L}(\mathcal{M})$ is symmetric monoidal closed for every $\mathcal{M}$. This means that any category $\mathscr{L}(\mathcal{M})$ is a model of (second order) multiplicative linear logic, and that one must only guarantee that the *exponential* structure can be justified when interpreting systems like LAL or SLL in such a category. This has been done indeed [15], obtaining as a byproduct new proofs of soundness for LAL, SAL (the affine version of SLL) and EAL. Noticeably, even systems going beyond the realm of linear logic (e.g., Hofmann's LFPL) can be proved sound.

2.3 Other Frameworks

In the last two sections, we focused on semantic frameworks where relevant, quantitative properties of the interpreted proof can be "read out" from the underlying semantic object. This way, semantics can be effectively used as a tool when proving soundness.

Semantics, however, has also a more traditional role (in ICC, as in any other domain), somehow orthogonal to the goals we had so far. A mathematical model of a formal system is, by its very essence, a different description of the syntax, expressed with new concepts, and embedding the syntactical objects in the structures the model is built upon. The more simple, non-syntactical, and mathematically elegant a model is, the more it is interesting and able to give us a fresh look on our formal system. Also in this sense, therefore, semantics is much welcomed in ICC.

Once a mathematical object $\mathscr{M}$ is proved to be a fully complete model of a logical system, itself sound and complete with respect to a complexity class $\mathscr{C}$, the model $\mathscr{M}$ can be understood as a new, presumably simpler, presentation of $\mathscr{C}$. Here, we are not interested in inferring intensional properties of proofs from $\mathscr{M}$ – indeed, not much about the intensional expressive power of a logic can be inferred from any fully complete model for it. We would like, instead, to get global insight on the structure (and properties) of $\mathscr{C}$. As pointed out by Girard [25] denotational models of light logics could thus help in shedding some light on the nature of polytime from an extensional point of view, maybe allowing to attack one of the many open problems in computational complexity. In the rest of this section, we briefly recall three proposals designed around these lines.

First, one should mention Baillot's *stratified coherent spaces* [7], a refinement on Girard's coherent spaces in which semantic stratification mimicks the absence of digging and dereliction in ELL. Indeed the two principles mentioned are not valid in stratified coherent spaces. By a further enrichment, one can obtain a model for LLL through *stratified measured coherence spaces.* In all these cases, however, full completeness does not hold, due to the inherent incompleteness of coherent spaces (already at the level of linear logic).

If one forgets about coherence in coherent spaces, what is left is relational semantics, maybe the simplest model of linear logic. Again, one can define a restricted notion of this model, this time through *obsessional cliques* [30]. Of course, the obtained model is not fully complete with respect to any of the known light logics. However, one can prove relative completeness for ELL: the clique interpreting a MELL proof π is obsessional (in a certain technical sense) *if and only if* π is an ELL proof. A similar result holds for SLL.

The only example of a semantic object which is fully complete with respect to a light logic is the model based on Murawski and Ong's *discreet games* [32]. They can be constructed as appropriate restrictions on usual game structures, already known to be fully complete with respect to MELL.

3 ICC in the Small

Giving a *reasonable* cost model for λ-calculus grounded on natural notions is a non trivial challenge. From the logical point of view, the most interesting parameter is the number of β-reductions, but, as we briefly discussed in Sect. 1, it is difficult to give general arguments proving that this simple approach gives rise to a reasonable model. As usual, the problem stems from duplication; in a single step $(\lambda x.M)N \rightarrow M[x \leftarrow N]$, the term N may be duplicated as many times as the number of occurrences of x in M. While in general this precludes to use the number of beta steps as the main ingredient in a cost model, there are restrictions of the reduction for which something can be done. In particular, we will concentrate on pure untyped λ-calculus endowed with *weak* (that is, we never reduce under an abstraction) *call-by-value reduction.* The following definitions are standard.

Definition 1.

- Terms *are defined as usual:* $M ::= x \mid \lambda x.M \mid MM$, *where* x *ranges over a denumerable set.* Λ *denotes the set of all lambda terms.*
- *Values are defined as* $V ::= x \mid \lambda x.M$.
- *Weak call-by-value reduction is denoted by* $\rightarrow_v$ *and is obtained by closing call-by-value reduction under any applicative context:*

$$\frac{}{(\lambda x.M)V \rightarrow_v M[x \leftarrow V]} \quad \frac{M \rightarrow_v N}{ML \rightarrow_v NL} \quad \frac{M \rightarrow_v N}{LM \rightarrow_v LN} \ .$$

  Here M *ranges over terms, while* V *ranges over values. We denote with* $\twoheadrightarrow$ *the reflexive and transitive closure of* $\rightarrow_v$.
- *The length* $|M|$ *of* M *is defined as follows, by induction on* M*:* $|x| = 1$, $|\lambda x.M| = |M| + 1$ *and* $|MN| = |M| + |N| + 1$.

Weak call-by-value reduction enjoys many nice properties. In particular, the one-step diamond property holds and, as a consequence, the number of beta-steps to normal form (if any) is invariant on the reduction order [16]. These properties enable the definition of interesting cost models for this calculus. We will discuss two different points of view.

First, we consider the case in which λ-terms are represented explicitly. Then, we will discuss the situation where we are allowed to choose more compact (and therefore efficient) representations.

3.1 Explicit Representation

With explicit representation, a term M is represented by...itself (that is, with a data structure of size $|M|$). In particular, from the representation of a term in normal form, the term is immediately available (a situation which will change for implicit representations). In this case, we define the *difference cost* of a single step of β-reduction $M \rightarrow_v N$ as $\max\{1, |N| - |M|\}$. The following definition and results are from [16].

Definition 2 (Difference cost model). *If* $M \twoheadrightarrow N$*, where* N *is a normal form, then* $Time_d(M)$ *is defined as* $|M|$ *plus the sum of the difference costs of the individual steps of the sequence* $M \twoheadrightarrow N$*. If* M *diverges, then* $Time_d(M)$ *is infinite.*

In view of the invariance of the number of beta-steps to normal form, this is a good definition. As an example, consider the term $\underline{n}\ \underline{2}$, where $\underline{n} \equiv \lambda x.\lambda y.x^n y$ is the Church numeral for n. It reduces to normal form in one step, because we do not reduce under the abstraction. To force reduction, consider $E \equiv \underline{n}\ \underline{2}\ x$, where x is a (free) variable; E reduces to

$$F \equiv \lambda y_n.(\lambda y_{n-1} \ldots (\lambda y_2.(\lambda y_1.x^2 y_1)^2 y_2)^2 \ldots)^2 y_n$$

in $\Theta(n)$ beta steps. However, $\mathit{Time}_d(E) = \Theta(2^n)$, since at any step the size of the term is duplicated. Indeed, the size of F is exponential in n.

We are left to state that this cost model makes weak call by value reduction a reasonable machine. That is, (i) we can simulate Turing machine computations with β-reduction, and (ii) we can implement β-reduction over Turing machines, in both cases with a polynomial overhead on the cost with Turing machines.

Theorem 4. *If $f : \Sigma^* \to \Sigma^*$ is computed by a Turing machine M in time g, then there are a λ-term N_M and a suitable encoding $\ulcorner\cdot\urcorner : \Sigma^* \to \Lambda$ such that $N_M\ulcorner v\urcorner$ normalizes to $\ulcorner f(v)\urcorner$ and $\mathit{Time}_d(N_M\ulcorner v\urcorner) = O(g(|v|))$.*

Theorem 5. *We can construct a (nine tapes) Turing machine $\mathcal{R}$ computing the normal form of any term M in $O((\mathit{Time}_d(M))^4)$ steps.*

3.2 Compact Representation

If we are allowed to use more efficient representations, we can get more parsimonious cost models. In particular, under certain conditions, we can in fact choose the number of β-steps to normal form as a reasonable cost model.

One the most appealing representations for λ-terms is the graph-based one, known at least since Wadsworth's thesis [39]. A term is represented first with its *syntax tree*, linking then the nodes representing bound variables to the λ-node binding them. The crucial idea, at this point, is to perform reduction without textual substitution, and updating – as much as possible – the arcs of the graph (i.e., performing update of pointers in the implementation). When we have a redex $(\lambda x.M)N \to M[x \leftarrow N]$, just erase the λ and the application nodes of the redex and link the arcs representing x (from the graph of M) to the root of the graph representing N. If there is more than one occurrence of x, N is not duplicated, but simply "shared" by all the arcs represented the different x's. The problem is that, sooner or later, some portion of a redex has to be *actually* duplicated (see [1], pages 14-15). The cost of normalization, thus, has to take into account the size of the duplicated part, which has no relation, in general, with the size of the original term (or with other sensible parameters). In other words, we would be back at the cost model of Sect. 3.1, taking into account the size of the intermediate terms. However, in case of weak (call-by-value and call-by-name) reduction we can drastically improve on Sect. 3.1 and show that the number of β-reduction steps *is indeed* a polynomially invariant cost model. The crucial observation is that, for these reductions, there exist ways to represent λ-terms in which subterms can be efficiently shared. Moreover, for this representation, any subterm needing to be duplicated during the reduction is a subterm *of the original term.* This has been obtained first by Sands, Gustavsson, and Moran [34], by a fine analysis of a λ-calculus implementation based on a stack machine.

We may give a more formal treatment [17], by exploiting techniques as defunctionalization and graph-rewriting. By using defunctionalization, we first encode a λ-term M into a term $[M]_{\Phi}$ in a (first-order) constructor term rewrite system, where any λ-abstraction is represented by an atomic constructor.

Theorem 6. *Weak (call-by-value and call-by-name) β-reduction can be simulated step by step by first-order rewriting.*

At this point we are left to show that first-order rewriting can be implemented so efficiently that the number of reduction steps can be taken as the actual cost of the reduction (up to a fixed polynomial). Without going into technical details, we may define a further encoding $[\cdot]_{\Theta}$ that, applied to a first order term, returns a graph. Our original term M would then be translated into $[[M]_{\Phi}]_{\Theta}$, composing the two encodings. We prove, further, that graph-rewriting simulates step by step constructor rewriting. We can eventually exploit the crucial observation about the first-order encoding of a λ-term: during the reduction of $[M]_{\Phi}$ we always manipulate subterms of the original term M.

Definition 3 (Unitary cost model). *For any λ-term M, $Time_u(M)$ is defined as* the *number of β-steps to normal form, under weak reduction. If M diverges, then $Time_u(M)$ is infinite.*

Some simple combinatorics allows to prove the following theorem.

Theorem 7. *There is a polynomial $p : \mathbb{N}^2 \to \mathbb{N}$ such that for every lambda term M, the normal form of $[[M]_{\Phi}]_{\Theta}$ (under weak reduction) can be computed in time at most $p(|M|, Time_u(M))$.*

It is worth observing that $[[N]_{\Phi}]_{\Theta}$ is in general very different from N, the former being a *compact* graph representation of the latter. Additional "read-back" work must in general be performed to obtain from $[[N]_{\Phi}]_{\Theta}$ its explicit, sequentialized form N. To factor out this time from our cost model should be considered a feature of the model. Indeed, we cannot do miracles. Consider again the term $E \equiv \underline{n}\ \underline{2}\ x$, introduced earlier. Under weak call by value reduction this term reduces to normal form in $\Theta(n)$ beta steps, but there is an exponential gap between this quantity and the time just needed to write the normal form, which is of length $\Theta(2^n)$.

4 Conclusions

Proof-theoretical implicit computational complexity is a young research area which has given in the last ten years a fresh look to a number of different subjects. In this short paper we tried to give a (*very*) personal overview of some of its main achievements and of its techniques. Still, we have to admit it is a very fragmented area. Too many systems, too few general results. We have suggested some semantical paths, but it is not sufficient.

An important problem we do not know how to attack is *intensional expressiveness*. We mentioned several complete systems for certain complexity classes. Completeness is an *extensional* notion – for any function in a complexity class, there is a term (proof, program) in our complete system. For the systems we mentioned, the term guaranteed to exists is seldom the most intuitive one, from a programming point of view. Moreover, there are a lot of natural algorithms (terms, proofs, programs) which *do have* the right complexity, but which *are not* expressible in our systems (they are not typable, or get the wrong type). Filling the gap between extensional completeness and the miserable intensional expressiveness of these systems is probably the most interesting challenge of the field.

References

1. A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge University Press, 1998.
2. A. Asperti and L. Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.
3. P. Baillot. *Approches dynamiques en sémantique de la logique lineaire: jeux et géometrie de l'interaction*. PhD thesis, Université Aix-Marseille 2, 1999.
4. P. Baillot, P. Coppola, and U. Dal Lago. Light logics and optimal reduction: Completeness and complexity. In *22nd LICS*, pages 421–430. IEEE Comp. Soc., 2007.
5. P. Baillot and M. Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2):1–31, 2001.
6. P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *LICS 2004*, pages 266–275. IEEE Comp. Soc., 2004.
7. Patrick Baillot. Stratified coherence spaces: a denotational semantics for light linear logic. *TCS*, 318(1-2):29–55, 2004.
8. A. Church. A set of postulates for the foundation of logic. *Ann. of Math. (2)*, 33:346–366, 1932.
9. S. Cook and A. Urquhart. Functional interpretations of feasible constructive arithmetic. *Annals of Pure and Applied Logic*, 63(2):103–200, 1993.
10. J. Crossley, G. Mathai, and R. Seely. A logical calculus for polynomial-time realizability. *Journal of Methods of Logic in Computer Science*, 3:279–298, 1994.
11. H. B. Curry and R. Feys. *Combinatory Logic, Volume I.* North-Holland, Amsterdam, 1958.
12. N. Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, Cambridge, 1980.
13. U. Dal Lago. The geometry of linear higher-order recursion. In *20th LICS*, pages 366–375. IEEE Comp. Soc., 2005.
14. U. Dal Lago. Context semantics, linear logic and computational complexity. In *21st LICS*, pages 169–178. IEEE Comp. Soc., 2006.
15. U. Dal Lago and M. Hofmann. Quantitative models and implicit complexity. In *Proc. Found. of Software Techn. and Theor. Comp. Sci.*, pages 189–200, 2005.
16. U. Dal Lago and S. Martini. An invariant cost model for the lambda calculus. In *Logical Approaches to Computational Barriers*, number 3988 in LNCS. Springer Verlag, 2006.

17. U. Dal Lago and S. Martini. Beta reduction is a cost model for the weak lambda calculus. Available from the authors, October 2007.
18. V. Danos and L. Regnier. Proof-nets and Hilbert space. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 307–328. Cambridge University Press, 1995.
19. M. Gaboardi, J.-Y. Marion, and S. Ronchi Della Rocca. A logical account of PSPACE. In *Proceedings of 35th ACM POPL*, 2008. To appear.
20. G. Gentzen. Untersuchungen über das logische Schließen. I and II. *Mathematische Zeitschrift*, 39(1):176–210; 405–431, 1935.
21. J.-Y. Girard. Linear logic. *TCS*, 50:1–102, 1987.
22. J.-Y. Girard. *Proof Theory and Logical Complexity, I.* Bibliopolis, Napoli, 1987.
23. J.-Y. Girard. Geometry of interaction 2: deadlock-free algorithms. In *COLOG-88: Proc. of the int. conf. on Computer logic*, volume 417 of *LNCS*, pages 76–93. Springer-Verlag, 1988.
24. J.-Y. Girard. Geometry of interaction 1: Interpretation of system F. In *Proc. Logic Colloquium '88*, pages 221–260. North-Holland, 1989.
25. J.-Y. Girard. Light linear logic. *Inform. and Comp.*, 143(2):175–204, 1998.
26. G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proc. 12th ACM POPL*, pages 15–26, 1992.
27. W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, Inc., New York, 1980.
28. G. Kreisel. Interpretation of analysis by means of constructive functions of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland, 1959.
29. Y. Lafont. Soft linear logic and polynomial time. *TCS*, 318:163–180, 2004.
30. O. Laurent and L. Tortora de Falco. Obsessional cliques: A semantic characterization of bounded time complexity. In *LICS*, pages 179–188, 2006.
31. I. Mackie. The geometry of interaction machine. In *Proc. 22nd ACM POPL*, pages 198–208, 1995.
32. A. S. Murawski and C.-H. L. Ong. Discreet games, light affine logic and ptime computation. In *CSL*, pages 427–441, 2000.
33. J. S. Pinto. Parallel implementation models for the lambda-calculus using the geometry of interaction. In *Proc. 5th International Conference on Typed Lambda Calculi and Applications*, pages 385–399, 2001.
34. D. Sands, J. Gustavsson, and A. Moran. Lambda calculi and linear speedups. In *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, number 2566 in LNCS. Springer Verlag, 2002.
35. R. Statman. The typed lambda-calculus is not elementary recursive. *TCS*, 9:73–81, 1979.
36. K. Terui. Light affine lambda calculus and polynomial time strong normalization. *Arch. Math. Log.*, 46(3-4):253–280, 2007.
37. A. S. Troelstra. Realizability. In Samuel Buss, editor, *Handbook of Proof Theory*, pages 407–473. Elsevier, 1998.
38. P. van Emde Boas. Machine models and simulation. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 1–66. MIT Press, 1990.
39. C. P. Wadsworth. *Semantics and pragmatics of the lambda-calculus.* Phd Thesis, Oxford, 1971. Chapter 4.