Information and Computation 261 (2018) 23–54

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Paths-based criteria and application to linear logic subsystems characterizing polynomial time

Matthieu Perrinel

LIP, ENS Lyon, 46 allée d'Italie, 69007 Lyon, France

ARTICLE INFO

Article history:
Received 13 July 2015

ABSTRACT

Several variants of linear logic have been proposed to characterize complexity classes in the proofs-as-programs correspondence. Light linear logic (LLL) ensures a polynomial bound on reduction time, and characterizes in this way polynomial time (*Ptime*). In this paper we study the complexity of linear logic proof-nets and propose three semantic criteria based on context semantics: stratification, dependence control and nesting. Stratification alone entails an elementary time bound, the three criteria entail together a polynomial time bound.

These criteria can be used to prove the complexity soundness of several existing variants of linear logic. We define a decidable syntactic subsystem of linear logic: *SDNLL*. We prove that the proof-nets of SDNLL satisfy the three criteria, which implies that *SDNLL* is sound for *Ptime*. Several previous subsystems of linear logic characterizing polynomial time (*LLL*, mL^4, maximal system of *MS*) are embedded in *SDNLL*, proving its *Ptime* completeness.

1. Introduction

Motivations for a type-system capturing polynomial time Programming is a notoriously error-prone process. The behaviours of the programs written by programmers on their first attempt often differ from their expected behaviours. Type systems can detect some of those mistakes so that programmers can correct them more easily. In this work, the property we are interested in is time complexity: the execution time of a program as a function of the size of its input. A type system *S* enforcing a polynomial bound on the time complexity of a program would be useful in several ways:

- In some real-time applications (e.g. car control systems) programs can never miss a deadline, otherwise the whole system is a failure. It is not enough to verify that the system reacted fast enough during tests, we need an absolute certainty.
- For some software, it seems enough to get an empirical estimate of the complexity by running tests. In this case, *S* could be useful to find the origin of the slowness observed during tests (this requires the type inferred to give useful information when it fails to type a term) in the same way existing static type systems make it easier for humans to fix the kind of errors they check [18].
- In complexity theory, the main method to prove that a problem is *NP*-complete, is to define a polynomial time reduction from another *NP*-complete problem. If *S* is well-trusted, it could be used as a specialized proof assistant: the fact

E-mail address: matthieu.perrinel@ens-lyon.fr.

https://doi.org/10.1016/j.ic.2018.05.004
0890-5401/

that the reduction is typable in S would increase the trust in the proof. More generally, S could be used in any proof relying on a complexity bound for a program [23,29].

In this work, we define a subsystem $SDNLL$ of linear logic such that every proof-net normalizes in polynomial time. This property is called *Ptime soundness*, and, for every function f computable in polynomial time there exists a $SDNLL$ proof-net G_f which computes f. This property is called *Ptime extensional completeness*.

Determining if a proof-net normalizes in polynomial time is undecidable. So for every such system S, either determining if a proof-net G belongs to S is undecidable, or S is not *intensional complete*: i.e. there exist programs which normalize in polynomial time and are not typable by S. The subsystem $SDNLL$ is in the second case. We take inspiration from previous decidable type systems characterizing *Ptime* and relax conditions without losing neither soundness nor decidability. The more intensionally expressive S is (i.e. the more terms are typable by S), the more useful S is. Indeed, the three motivations for systems characterizing polynomial time we described earlier require S to type programs written by non-specialists: people who may not have a thorough understanding of S.

Linear logic and proof-nets Linear logic (LL) [12] can be considered as a refinement of System F where we focus especially on how the duplication of formulae is managed. In linear logic, the structural rules (contraction and weakening) are only allowed for formulae of the shape $!A$:

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

With the three following additional rules (promotion, dereliction and digging), linear logic is as expressive as System F, so the elimination of the *cut* rule (corresponding to the β-reduction of λ-calculus) can not be bounded by a primitive recursive function (let alone a polynomial) in the size of the proof-net.

$$\frac{A_1, \cdots, A_n \vdash B}{!A_1, \cdots, !A_k \vdash !B}\ !P \qquad \frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\ ?D \qquad \frac{\Gamma, !!A \vdash B}{\Gamma, !A \vdash B}\ ?N$$

However, because the structural rules are handled by 5 distinct rules, one can enforce a subtle control on the use of resources by modifying one of them. If we restrict some of those rules, it restricts the duplication of formulae. For instance, in the absence of $?D$ and $?N$ rules, the cut-elimination normalizes in elementary time [9]. The set of such proofs is defined as Elementary Linear Logic (ELL).[1]

Proof-nets [13] are an alternative syntax for linear logic, where proofs are considered up-to meaningless commutations of rules. Proof-nets are graph-like structures where nodes correspond to logical rules. One of the reasons we use proof-nets instead of proof derivations is that context semantics, the main tool we use in this article, is much simpler to define and use in proof-nets.

Context semantics Context semantics is a presentation of geometry of interaction [16,10] defined by tokens travelling across proof-nets according to some rules. The paths defined by those tokens are stable by reduction so they represent the reduction of the proof-net. Context semantics has first been used to study optimal reduction [17].

Recently, it has been used to prove complexity bounds on subsystems of System T [6] and linear logic [4,7]. In [7], Dal Lago defines for every proof-net G a weight $W_G \in \mathbb{N} \cup \{\infty\}$ based on the paths of context semantics such that, whenever G reduces to H, $W_G \geq W_H + 1$. Thus W_G is a bound on the length of the longest path of reduction starting from G. Then we can prove theorems of the shape "whenever G satisfies some property (for instance if G belongs to a subsystem such as LLL), W_G satisfies some bound (for instance $W_G \leq P(|G|)$ with P a polynomial and $|G|$ the size of G)."

From this point of view, context semantics has two major advantages compared to the syntactic study of reduction. First, its genericity: some common results can be proved for different variants of linear logic, which allows to factor out proofs of complexity results for these various systems. Moreover, the bounds obtained stand for any strategy of reduction. On the contrary, most bounds proved by syntactic means are only proved for a particular strategy. There are several advantages to strong bounds:

- Let us suppose we know a strong complexity bound for a system S'. We can prove the same strong complexity bound on a system S if we find an embedding ϕ of S programs in S' programs such that, whenever t reduces to u in S, $\phi(t)$ reduces to $\phi(u)$ in S' (with at least one step). We use such an embedding in Section 5.4 to prove a strong bound for λ-terms typed by $SDNLL$. If we only had a weak complexity bound for system S', we would have to prove that the reduction from $\phi(t)$ to $\phi(u)$ matches the reduction strategy entailing the bound, which is not always possible.
- The languages we study here are confluent. However, if we consider an extension of linear logic or λ-calculus with side-effects (such as $\lambda^{!R}$ considered by Madet and Amadio in [22]), the reduction strategy influences the result of a

[1] In the original presentation of Danos and Joinet [9], the $!P$ rule is separated into a rule *der* introducing the ? connective and a rule !-box introducing the ! connective. They define ELL as the proofs such that: a $?A$ formula introduced by a *der* rule has at least one descendant in a context of a !-box rule (it corresponds to forbidding $?N$ rules) and at most one descendant in such a context (it corresponds to forbidding $?D$ rules).