Information and Computation 241 (2015) 62–95

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Quantitative classical realizability

Aloïs Brunel

LIPN – UMR CNRS 7030 – Université Paris 13, Villetaneuse, France

ARTICLE INFO

Article history:
Received 20 January 2012
Available online 17 October 2014

ABSTRACT

Introduced by Dal Lago and Hofmann, quantitative realizability is a technique used to define models for logics based on Multiplicative Linear Logic. A particularity is that functions are interpreted as bounded time computable functions. It has been used to give new and uniform proofs of soundness of several type systems with respect to certain time complexity classes. We propose a reformulation of their ideas in the setting of Krivine's classical realizability. The framework obtained generalizes Dal Lago and Hofmann's realizability, and reveals deep connections between quantitative realizability and a linear variant of Cohen's forcing.

1. Introduction

Ever since its introduction by J.L. Krivine [12], the theory of classical realizability has raised a growing interest. Initially designed to study the computational content of classical proofs through the Curry–Howard correspondence, it has led to promising results in various fields. One could mention the recent advances [15] made by Krivine in the elaboration of new models of the **ZF** axiomatic set theory. Another success has been its use to define and justify a classical extraction procedure for the proof assistant Coq [21].

Forcing — Forcing is a technique designed by Cohen [4] to prove the independence of the Continuum Hypothesis (CH) from **ZFC**. The idea is to define a formula transformation which turns every formula A into a new one noted $p \Vdash A$, where p is a *forcing condition*. By choosing a suitable set of forcing conditions, one can prove the statement $p \Vdash \neg CH$. It has been recently shown by Krivine [14] that combining classical realizability and forcing is possible. This construction can be seen as a generalization of forcing iteration and makes possible a study of forcing through the Curry–Howard isomorphism: Krivine has shown that the forcing technique not only provides a logical translation but also a program transformation. Following that work, Miquel [22] has introduced an abstract machine (the Krivine Forcing Abstract Machine, or **KFAM**) that internalizes the computational behavior of programs obtained via this transformation. One remarkable feature of this machine is that it provides sophisticated programming features like memory cells or program execution tracing.

Resource sensitive realizability — Realizability techniques have also been fruitfully applied to *implicit complexity*. This research field aims at providing machine-independent characterizations of complexity classes (such as polynomial time or logspace functions). One of the possible approaches is to use linear logic based type systems to constrain programs enough so that they enjoy bounded-time normalization properties. Proving these properties can be achieved using semantic techniques. Following different works [9,10], Dal Lago and Hofmann have introduced in [18] a *quantitative* (another word for

E-mail address: alois.brunel@ens-lyon.org.

http://dx.doi.org/10.1016/j.ic.2014.10.007
0890-5401/

resource sensitive) framework based on Kleene realizability [11]. One of the crucial ideas behind Dal Lago and Hofmann's work is to consider bounded-time λ-terms as realizers. Bounds are described using elements of a *resource monoid*. No matter what resource monoid is chosen, their framework always yields a model of second-order Multiplicative Affine Logic (**MAL**). Various systems extending **MAL** are then dealt with by choosing a suitable resource monoid, while the basic realizability constructions are unchanged. This work has offered new and uniform proofs of the soundness theorems for **LAL**, **EAL**, **SAL** and **BLL** with respect to the associated complexity classes [16–18]. In [3], Terui and the author gave a new characterization of the complexity class **FP** (the functions computable in polynomial time) and used a variant of Dal Lago and Hofmann's realizability to show the soundness part of this result.

The present work aims at applying methodology and tools coming from classical realizability to generalize the framework proposed by Dal Lago and Hofmann, and to reveal deep connections between quantitative realizability and forcing techniques.

Quantitative classical realizability — We propose a new quantitative framework, based on Munch's classical realizability for focalizing system **L** (or $\mathbf{L}_{foc}$) [23], a term calculus for classical logic **LC** [8]. We extend this realizability using the notion of *quantitative monoid*, which derives from the *resource monoid* structure introduced by Dal Lago and Hofmann. We show that, whatever the quantitative monoid, this framework always gives rise to a model of the Multiplicative Affine fragment of Higher-order Classical Arithmetic (abbreviated **MAL**ω). By choosing different quantitative monoids, we obtain models of logics extending **MAL**ω. Because all resource monoids in the sense of [18] are also quantitative monoids, we can in principle obtain models for all the systems treated in [17,18], although we only exhibit a model of Soft Affine Logic (**SAL**) [2].

Quantitative reducibility candidates — By carefully setting parameters of classical realizability, one can retrieve the notion of *reducibility candidates* (presented using orthogonality, as in [7,19,24,25]), which is used to prove normalization properties. Similarly, in our setting, we are able to define a quantitative extension of this technique, which we call *quantitative reducibility candidates*. It allows us to semantically prove complexity properties of programs that are typable in the logic we interpret. Moreover, because we work with a term calculus which generalizes both call-by-name and call-by-value classical λ-calculi, these complexity properties are transferred for free to these calculi. Hence, we are able to retrieve and generalize the bounded-time termination results proved in [17,18].

A forcing decomposition — Quantitative classical realizability is deeply connected with a certain notion of forcing, which we propose to study. We formalize inside **MAL**ω a forcing transformation on Multiplicative Linear Logic (**MAL**) formulas, called *linear forcing*. Then, following Miquel's methodology [22], we propose an abstract machine designed to execute programs obtained by a specific linear forcing instance. A **connection lemma** is proved, which shows that composing this instance of linear forcing with a non-quantitative realizability built upon this machine always yields a quantitative realizability model. Finally, using this result, we show how quantitative reducibility candidates (restricted to **MAL**) arise from the composition of usual reducibility candidates with forcing.

Outline — Sections 2 and 3 introduce **MAL**ω and its quantitative realizability interpretation. The model of quantitative reducibility candidates is then defined and used to prove a bounded time termination property of **MAL**ω. In Section 4, we show by taking **SAL** as an example that this interpretation and the corresponding complexity result can be extended to larger type systems. Finally, we introduce in Section 5 the linear forcing interpretation of **MAL** and prove the accompanying decomposition results.

2. The calculus

In this section, we describe the system **MAL**ω. It is based on the **MAL** type system for Munch's focalizing system **L** [23], extended with higher-order quantifications and arithmetical operations. Logically, it is a fragment of classical higher-order Peano arithmetic (abbreviated by **PA**ω). The syntax of **MAL**ω is divided in three distinct layers: the *terms*, the *type constructors* and the *kinds*. The language of terms, which we shall use to express both proof-terms and realizers, is based on the multiplicative fragment of $\mathbf{L}_{foc}$, extended with extra instructions. The type constructors layer is an adaptation of the higher-order terms syntax of **PA**ω [22] to linear logic: it can be seen as a combination of the languages of **PA**ω and classical **F**ω [19]. Finally, kinds are used as a simple type system for type constructors.