Abstract. When we cut a multiplicative proof-net of linear logic in two parts we get two modules with a certain border. We call pretype of a module the set of partitions over its border induced by Danos-Regnier switchings. The type of a module is then defined as the double orthogonal of its pretype. This is an optimal notion describing the behaviour of a module: two modules behave in the same way precisely if they have the same type.

In this paper we define a procedure which allows to characterize (and calculate) the type of a module only exploiting its intrinsic geometrical properties and without any explicit mention to the notion of orthogonality. This procedure is simply based on elementary graph rewriting steps, corresponding to the associativity, commutativity and weak-distributivity of the multiplicative connectives of linear logic.

1. Introduction

“Switchings should be seen as dense subset of para-proofs.”

The notion of modularity plays a central role in the current programming languages and techniques: the major part of them make sensitive use of concepts like modules, objects, components, etc. When we build a large program it is natural to cut it into reusable sub-units, or modules, which should result to be correct; this correctness must be checked locally, without involving the whole program. Moreover, when we branch together some modules in order to get a unit, the inner part of these modules should play no role; actually, the only information required should be the specifications of their interfaces, also called types. The type of a module tells
us about the behavior of this module. In particular, it says that if the branching of modules is done according to their types then the correctness of the whole resulting structure will only depend on the local correctness of the involved modules. At the end of the 80ths J.-Y. Girard has shown in [Gir87a] that proof-nets of linear logic [Gir87] (at least the multiplicative fragment) are suitable “in order to prove that the respect of specification implies correctness”. Let us recall the question of modularity of proof-nets, as formulated in [Gir87a]:

“Asume I am given a program \( P \) and I cut it in two parts, arbitrarily. I create two (very bad) modules, linked together by their border. How can I express that my two modules are complementary, in other terms, that I can branch them by identification of their common border? One would like to define the type of the modules as their plugging instructions: these plugging instructions should be such that they authorize the restoring of the original \( P \).”

Formally a module of multiplicative linear logic is a structure with a specified border, consisting of all of the hypotheses and some of the conclusions. The conclusions not belonging to the border are called the proper conclusions (see Section 2). Our main question concerns how to code the behavior of a module as a function on the border only.

By a slight modification of the original terminology adopted in [DR89] we call pretype of a module the set of partitions over the border induced by the systems of Danos-Regnier switchings. Then the type of a module is defined as the double orthogonal of its pretype, according to [Gir87a] (see Section 3). The type is an optimal notion describing the behavior of a module: two modules behave in the same way precisely when they have same type.

We now give a characterization of the type of a module which does not make explicit mention to the notion of orthogonality. This characterization only relies on a constructive procedure which allows us to calculate the type of a module, simply by starting from its Danos-Regnier type, here called pretype. The procedure iterates inside the module some elementary steps of graph rewriting, illustrated in Section 4 and only based on the associativity and commutativity of \( \otimes \) and \( \wp \) and the weak-distributivity laws. The weak-distributive laws correspond to the following (well known) theorems of linear logic: \( A \otimes (B \wp C) \vdash (A \otimes B) \wp C \) and \( A \otimes (B \wp C) \vdash (A \otimes C) \wp B \) (see [CS97], [AJ94] and Section 8 for a discussion of these works).

In order to show that our rewrite method is complete w.r.t. the type of a module we cut our reasoning in two parts: in Section 6 we study the case when a module is a formula-tree, then in Section 7 we discuss the general case (a module with axioms).

We claim, in Section 8, that our characterization of the type of module can have nice applications to the design of (distributed) theorem provers based on proof-nets.

---

1 In [Gir87a] is given an other definition of the type of a module in terms of permutations instead of partitions of the border formulas. This choice is justified by the different style of the correctness criterion of proof-nets formulated in terms of trip over the switched proof-structures.