Abstract. We describe $\mathcal{LC}$, a formalism based on the proof theory of linear logic, whose aim is to specify concurrent computations and whose language restriction (as compared to other linear logic language) provides a simpler operational model that can lead to a more practical language core. The $\mathcal{LC}$ fragment is proved to be an abstract logic programming language, that is any sequent can be derived by uniform proofs. The resulting class of computations can be viewed in terms of multiset rewriting and is reminiscent of the computations arising in the Chemical Abstract Machine and in the Gamma model.

The fragment makes it possible to give a logic foundation to existing extensions of Horn clause logic, such as Generalized Clauses, whose declarative semantics was based on an ad hoc construction.

Programs and goals in $\mathcal{LC}$ can declaratively be characterized by a suitable instance of the phase semantics of linear logic. A canonical phase model is associated to every $\mathcal{LC}$ program. Such a model gives a full characterization of the program computations and can be obtained through a fixpoint construction.

Keywords: Linear Logic, Uniform Proofs, Concurrency, Phase Semantics, Chemical Abstract Machine.
1 Introduction

The availability of powerful environments for parallel processing has made particularly interesting the field of logic languages. Writing concurrent programs is quite difficult. Therefore it is desirable to have languages with a clear and simple semantics, so as to have a rigorous basis for the specification, the analysis, the transformation and the verification of programs. A programming framework based on logic seems to be well suited.

In this paper we investigate the expressive power of linear logic in a concurrent programming framework. This logic is gaining wide consensus in theoretical computer science and our attempt is not quite new in its kind. Linear logic has already given the basis to many proposals. Our approach is based on the paradigm of computation as proof search, typical of logic programming. We take as foundation the proof theoretical characterization of logic programming given by Miller [24, 23]. The definition of uniformity will lead us to single out a restricted fragment of linear logic capable of specifying an interesting class of parallel computations. We think the simple operational model can lead to a more practical language core.

The resulting framework is strongly related to the paradigm of multiset rewriting lying at the basis of the Gamma formalism [6] and of the Chemical Abstract Machine [7]. Actually it allows to specify a set of transformations that try to reduce an input multiset of goals to the empty multiset, returning as an output an answer substitution for the initial goal. More transformations can be applied concurrently to the multiset, thus making possible efficient implementations in parallel environments.

An important feature of the language is its ability to express in a simple way the synchronization and the communication between different computational flows, opening the way to distributed programming, as has already been showed in [11] and [8] for similar languages. In the case of $\mathcal{L}C$ however we have a declarative semantics for the symmetrical interactions. In fact in the last part of the paper we propose a semantics for the language obtained by instantiating the phase semantics of linear logic. The resulting abstract structure associated to programs allows to declaratively model the behaviour of our computations. By exploiting the similarities of our fragment with the language of Generalized Clauses [11, 8], a fixpoint characterization of the phase semantics is also presented.

The paper is organized as follows. In subsection 1.1 we introduce linear logic and its proof system. In section 1.2 we introduce the definition of uniformity for multiple conclusions sequent systems. Section 2 shows the fragment $\mathcal{L}C$ and its computational features. In section 3 we relate the $\mathcal{L}C$ framework to actual programming environments. Finally a semantics for $\mathcal{L}C$ programs in the style of the phase semantics will be shown in section 5 together with a fixpoint characterization.