Asynchronous Communication Model Based on Linear Logic

Naoki Kobayashi and Akinori Yonezawa
Department of Information Science, University of Tokyo, Tokyo, Japan

Keywords: Linear logic programming; Asynchronous concurrent computation

Abstract. We propose a new framework called ACL for concurrent computation based on linear logic. ACL is a kind of linear logic programming framework, where its operational semantics is described in terms of proof construction in linear logic. We also give a model-theoretic semantics based on phase semantics, a model of linear logic. Our framework well captures concurrent computation based on asynchronous communication. It will, therefore, provide us with a new insight into other models of asynchronous concurrent computation from a logical point of view. We also expect ACL to become a formal framework for analysis, synthesis and transformation of concurrent programs by the use of techniques for traditional logic programming. ACL’s attractive features for concurrent programming paradigms are also discussed.

1. Introduction

For future massively parallel processing environments, concurrent programming languages based on asynchronous communication would become more and more important. Due to the difficulty of writing and debugging programs in such environments, computers would need to aid programmers for transforming and verifying of concurrent programs, hence the role of formal frameworks for concurrent computation would be significant. Recently, several applications of Girard's linear logic [Gir87] to logic programming were proposed and shown that they correspond to reactive paradigms [AnP91a] [AnP91b] [Mil92].

We propose a new framework called ACL (Asynchronous Communication based on Linear logic) for concurrent computation along this line. Computation...
in ACL is described in terms of \textit{proof construction} in linear logic. We restrict the inference rules and formulas in linear sequent calculus so that the restricted rules have a proof power equivalent to the original rules for the restricted formulas. The resulting computational framework contains rich mechanisms for concurrent computation, such as message-passing style asynchronous communication, identifier creation, and hiding operator. They are all described in a \textit{purely logical} form. We also give a model-theoretic semantics as a natural extension of \textit{phase semantics}, a model of linear logic, by using a popular fixpoint construction. ACL inference rules can be proven to be sound and complete w.r.t. this model-theoretic semantics. Our framework well captures concurrent computation based on asynchronous communication. It will, therefore, provide us a new insight into other models of concurrent computation from a \textit{logical} point of view. In fact, the actor model [Agh86] and asynchronous CCS [Mil83] [Mil89] can be directly translated into our ACL framework. We also expect ACL to become a formal framework for analysis, synthesis and transformation of concurrent programs with techniques used in traditional logic programming. ACL also exhibits attractive features as a concurrent programming paradigm, subsuming the actor computation [Agh86] and providing mechanisms for multiple messages receptions, distributed information sharing, inheritance, etc.

The rest of this paper is organised as follows. Section 2 describes the syntax and operational semantics of the basic fragment of ACL. Section 3 gives a model-theoretic semantics and proves the soundness and completeness theorems. In Section 4, we extend ACL to include mechanisms for value passing, identifier creation, and hiding. Section 4 also gives an extension where processes can be consumed as resources, and also discusses the use of incomplete models as active types of concurrent processes. Section 5 shows the translation from the actor model and asynchronous CCS into ACL. Section 6 summarises features of ACL as concurrent programming paradigms. Section 7 compares ACL to the previous work. Section 8 concludes this paper.

2. ACL Framework

In this section, we introduce a syntax and an operational semantics for the basic (propositional) fragment of ACL. In ACL, computation is described in terms of a bottom-up proof search for some restricted formula in linear logic. Transition rules are, therefore, given in a form of inference rules.

2.1. Program Syntax

First, we define the ACL program clause.

\begin{definition}
\textbf{(program, clause, goal formula)} A program is a set of clauses, which are defined as follows:

\begin{align*}
\text{Clause} &::= \text{Head} \rightarrow \text{Goal} \\
\text{Head} &::= A_P \\
\text{Goal} &::= \text{Terminator} | \text{Suicide} | A_P | A_m \mid \text{ParComp} \mid \text{Fork} \mid \text{Choice} \\
\text{Terminator} &::= \top \\
\text{Suicide} &::= \bot \\
\text{ParComp} &::= \text{Goal} \& \text{Goal} \\
\text{Fork} &::= \text{Goal} & \text{Goal}
\end{align*}
\end{definition}