On Blocks: locality and asynchronous communication
(Extended abstract)

F.S. de Boer¹, J.N. Kok², C. Palamidessi³, J.J.M.M. Rutten⁴

¹Department of Computer Science, Technical University Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

²Department of Computer Science, Utrecht University,
P.O. Box 80089, 3508 TB Utrecht, The Netherlands

³Dipartimento di Informatica, Università di Pisa,
Corso Italia, 40, 56125 Pisa, Italy

⁴CWI,
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

Abstract. A general construct for locality in languages based on asynchronous communication is introduced which allows a uniform semantic description of such apparently diverse notions as the introduction of local variables in concurrent imperative languages with shared variables and the hiding of logical variables in concurrent constraint languages.

Keywords: Parallelism, locality, block structure, asynchronous communication.

Contents

0 Introduction
1 Operational semantics
   1.1 The language $\mathcal{L}$
   1.2 The operational model
2 Two instantiations of $\mathcal{L}$
   2.1 An imperative language
   2.2 Concurrent constraint programming
3 Compositional semantics
  3.1 The model
  3.2 Semantic operators

0 Introduction

In [dBKPR91], an abstract language ($L$) is introduced as the basis for a semantic theory for asynchronous communication in concurrent languages. It is built from a set $A$ of atomic actions and three binary operators for sequential composition, nondeterministic choice and parallel composition. An operational semantics for $L$ is given by first interpreting atomic actions as state transformations (assuming an abstract set of states), and next defining the meaning of composite statements in terms of a labelled transition system in Plotkin's SOS style ([Plo81]). This semantics formally captures the following computational intuition. Statements represent (compositions of) computing agents or processes. These act on a global state (or store)—which can be seen as a data structure common to all processes—by performing atomic actions. Such actions can either query the state, and possibly suspend in case the required information is not present, or update the state by adding or changing information. (A combination of the two aspects in one and the same atomic action is also possible.) The kind of communication that can be established in this way is of an asynchronous nature because such query and update actions are independent in the sense that they can happen at different moments.

Then it is shown that $L$ actually is a family of languages, the members of which are obtained by making a specific choice for the set $A$. Each of these members has its own semantics that is obtained as an instance of the semantics for $L$, by fixing a specific interpretation of the atomic actions.

An example is an imperative language obtained by taking for $A$ the collection of guarded assignments, $A = \{b.z := e \mid z \in Var, b, e \in Exp\}$ (where $Var$ and $Exp$ are given sets of variables and expressions), and for states the usual function from variables to values. The interpretation of $b.z := e$ in a given state $\sigma$ amounts to first evaluating the Boolean expression $b$ in $\sigma$; if this results to true then the outcome of the evaluation of the expression $e$ in $\sigma$ is assigned to $z$. If $b$ evaluates to false, however, the whole action suspends.

As a second example, concurrent constraint languages ([Sar89]) can also be obtained as an instance of $L$. The set of atomic action is defined as the collection of ask and tell primitives, and states are constraints. The interpretation of these actions follows the standard semantics for constraint languages. (See [dBKPR91] for more examples of instantiations of $L$.)

As was observed in the concluding section of the above paper, one important notion has not been addressed, namely that of locality. Many parallel programming languages, whether their basic communication mechanism is of a synchronous or asynchronous nature, contain some kind of block structure for modelling local computations. Examples are the block structure in imperative programming (like $let \ z = v \ in \ s$), and the projection (or hiding) operator (like $\exists X.s$) in constraint programming.

Therefore we shall in this paper extend the abstract language $L$ with a block construct that generalizes these two examples. We assume given an abstract set of variables $\mathcal{Var}$,