Operational semantics and a distributed implementation of CSP

Gerlinde Schreiber (*)

Abstract

In this paper an operational semantics for Hoare's CSP is presented. This semantics definition is used to develop an interpreter for CSP in a distributed environment. The correctness of this interpreter is proved.

Introduction

To increase the efficiency of computers, developments in computer architecture suggest the use of multiprocessor systems instead of monoprocessor machines. There are two fundamentally different ways to work with such a multiprocessor system:
- hide the architecture of the machine from the user (e.g. by a multiprogrammed operating system)
- leave the use of the multiprocessor to the programer by providing special parallel programming languages.

(*) The author is employed at Siemens AG, ESTE 366, 8520 Erlangen, West Germany.
In his paper "Communicating Sequential Processes" [H087] Hoare has made a widely regarded proposal for the main features of such a parallel language. CSP is designed for a system of similar processors, each of them working on its own private memory. The processors are connected by named channels. CSP offers a parallel command for the creation of processes: \[\langle \text{process} \rangle \cap \ldots \cap \langle \text{process} \rangle\] denotes the generation of \(n\) processes that are to be executed in parallel on different processors. As the processes can’t communicate by shared variables, Hoare introduces special input and output commands. In these commands a channel connecting the two communicating processes is referenced: \[\langle \text{channelid} \rangle \rightarrow \langle \text{varid} \rangle\] means the assignment of the value presented on the channel \[\langle \text{channelid} \rangle\] to the variable \[\langle \text{varid} \rangle\]. Besides the exchange of data between two processes, the communication commands support their synchronisation: an input command in one process and an output command in another process concerning the same channel are executed at the same time. The execution of one communication command is delayed until a corresponding communication command is ready. The attempt to communicate with a terminated process ends in a deadlock.

Another main feature of CSP are nondeterministic commands:
\[\langle \text{guard} \rangle_1; \langle \text{commandlist} \rangle_1; \ldots; \langle \text{guard} \rangle_n; \langle \text{commandlist} \rangle_n\] denotes the nondeterministic choice between \(n\) alternatives that is controlled by guards. These guards are constructed of a boolean expression and a communication command. Only such an alternative can be chosen whose boolean guard evaluates to true and whose communication guard can be executed. If all guards fail (i.e. the boolean guards evaluate to false or the communication partners are terminated) the nondeterministic command fails.

Up to now a lot of work has been done in specifying the semantics of CSP [H080], [H081], [HBR81], [CH81], [PL83], [OH84]. A few CSP-like parallel languages have been developed and implemented [FS81], [MS84], [RE83], the most famous of them being OCCAM®. Of these implementations, just one [RE83] is based on a formal semantics description, namely on the model of the interleaving semantics by Hoare and Olderog [OH84]. This model uses as one component of the semantics definition the observation of program-execution. In a parallel program many observable actions are executed at the same time, but at one time just one observation can be made. In the model, the parallel actions are represented by a sequence of observations made in arbitrary order. Thus parallelism is represented in a sequential way. The implementation of the model makes use of this property and realizes CSP on a monoprocessor machine.

In this paper the semantics of CSP is defined by labelled transitions. As in [OH84], one component of the semantics definition is observation - but instead of the observation of program-execution the observation of one single process is considered. This approach leads to a semantics definition that can be transformed into an interpreter for CSP in a multiprocessor environment. Some examples illustrate the problems of such a distributed implementation. Up to now these problems have led to constraints in the implemented CSP-dialect [MS84] or they could only be managed by additional requirements on the