DENOTATIONAL SEMANTICS FOR OCCAM

A.W. Roscoe
Oxford University Computing Laboratory
8-11 Keble Road, Oxford OX1 3QD, United Kingdom.

ABSTRACT. A denotational semantics is given for a large subset of occam, a programming language for concurrent systems. The semantic domain used is a "failure-sets" model modified to allow machine states to be properly dealt with. The applications of the semantics are discussed briefly, and we see how the natural congruence induced by the semantics allows us to prove simple algebraic laws relating occam programs.

0. Introduction The occam programming language [10] was designed with the philosophy of eliminating unnecessary complexities, thus keeping the language simple and elegant. Fundamental to its design was the idea that the language should have close ties with formal methods of designing and verifying programs. The main aim of this paper is to develop a theory which will allow us to apply existing techniques of program verification to occam, as well as giving us a rigorous mathematical framework for developing new ones.

The main difficulty in developing a useful theory of occam is the fact that it is a concurrent language. Indeed the idea of concurrency is central in occam, making an adequate treatment of concurrency essential in such a theory. There has been a considerable amount of effort expended in recent years in developing theories of concurrency. This subject is rather harder than the study of purely sequential computation, because of the emergence of such phenomena as nondeterminism, deadlock and livelock. Fortunately occam is close in spirit to CSP [6], a language which has been one of the main vehicles for research into concurrency.

Since the problems of dealing with concurrency in isolation are quite considerable, many authors have chosen to omit several "conventional" programming constructs from the example languages they have used when reasoning about concurrent systems. In particular they have often omitted those constructs, such as assignment and declaration, which deal with a machine's internal state (or store). Several of the most successful theories of concurrency have been based on these "purely parallel" languages. To handle occam we need a theory which, while retaining its full capacity for dealing with concurrency, is extended to handle machine states.

This paper presents one possible approach to this problem by constructing a mathematical model for communicating processes with internal states. As a basis for our treatment of concurrency we take the "failure-sets" (or "refusal-sets") model for communicating processes, originally developed as a model for a purely parallel version of CSP. It was introduced in [4], and developed and improved in [3,5,12]. It provides a reasonably simple mathematical structure within which most of the important features of concurrency are easy to study. The fact that it was developed as a model for CSP makes it well
suited to occam. It is necessary to add to the model a mechanism for dealing with the state transformations induced by occam's "conventional" constructs. The framework chosen for this is the well-known idea of regarding a program as a relation between initial and final states. This is sufficient because it turns out that knowledge of intermediate states is not required. One of the aims when putting these models together was that on purely parallel programs the results obtained would correspond closely to the old failure-sets model, and that on sequential programs the results would be relations on states.

The first part of the paper is concerned with the construction of the revised model. The second part shows how it can be used to give a denotational semantics in the style of [11,13,14] to occam. The third section discusses a few applications of these semantics, and derives some algebraic laws relating occam terms.

Throughout this paper $\mathcal{P}(X)$ will denote the full powerset of $X$ (the set of all subsets of $X$), while $\mathcal{P}(X)$ will denote the finite powerset of $X$ (the set of all finite subsets of $X$). $X^*$ will denote the set of all finite sequences of elements of $X$. $<>$ denotes the empty sequence, and $<a,b,\ldots,z>$ denotes the sequence containing $a,b,\ldots,z$ in that order. If $s,t \in X^*$, then $st$ denotes the concatenation of $s$ and $t$ (e.g. $<abc><de> = <abcde>$). If $s,t \in X^*$ then $s \preceq t$ (s is a prefix of t) if there is some $u \in X^*$ with $su = t$.

1. Constructing the model

Our semantic domain for occam is based on the failure-sets model for communicating processes. The following is a brief summary of its construction; much fuller descriptions and motivations can be found in [3,4,5,12]. The version described here is that of [5].

The failure-sets model has as its only primitives the set $\Sigma$ of atomic communications between processes. Communications are events which occur when the participating processes agree to execute them. In themselves they have no direction - there is no inputting process or outputting process. Input and output are modelled at a higher level by varying the set of a process' possible communications (an outputting process will typically have one possible communication, while an inputting process will have many). Each process communicates with its environment. This might be some other process or processes, or it might be some external observer. No distinction is made between these cases. We will think of a process proceeding by accepting (i.e. communicating) symbols in $\Sigma$ which are offered by the environment. Only one symbol can be communicated at a time, and only finitely many in any finite time.

A process is modelled as a pair. The first component is a relation between the possible traces of the process (the elements of $\Sigma^*$ which are the possible sequences of communications of the process up to some time) and the sets of symbols which the process can refuse to respond to after the traces (refusals). A failure of a process is a pair $(s,X)$ where the process can refuse to communicate when offered the set $X$ by its environment after the trace $s$. The first component of our representation of a process is the set of all its possible failures.