1. Introduction

The current paper defines a proof system for the language of Distributed Processes, henceforth called DP, given in Brinch Hansen [2]. An important aspect of our approach is that proofs of the individual processes of a program are completely isolated from each other. This allows us to consider an individual process without making assumptions about the behavior of the other processes. In fact, the most important idea underlying our work is that the individual processes of a concurrent program must be dealt with in isolation if we are to avoid getting lost in the complex interactions between the processes. Since the semantics of a construct is independent of the rest of the program, we have the flexibility to change any one construct without causing a change in the verification of the other processes in the program. Once we obtain the individual process proofs, we only need to combine the post-conditions of the processes in order to obtain the behavior of the entire program.

This aspect can be contrasted with the system in Gerth et. al.[3] where individual process proofs are closely related to each other. In their system, the properties of the individual processes are proved using assumptions about the behavior of the remaining processes in the program. Their parallel composition rule contains an "interference freedom test"[3] since this test is needed to substantiate these assumptions. The proof of interference freedom is often the most difficult part of the proof. Moreover, the interference freedom test requires the use of the complete proof outlines of each of the constructs within the program. Therefore, a modification to any construct would require the modification of potentially all the other processes defined within the program.

For these reasons, we feel that considering constructs entirely in isolation from the other processes leads to simpler proofs. This concept is also the motivation behind the development of the axiomatic semantics of CSP defined in [7,8,9].

2. The Language Distributed Processes

DP was proposed for real-time applications controlled by microcomputer networks
with distributed storage. A DP program consists of a fixed number of sequential processes that are executed concurrently and exist forever. Each process can only access its own variables — there are no shared variables. A process may call the common procedures defined within the other processes. Processes are synchronized by means of nondeterministic statements called guarded regions.

2.1 Programs, Processes, & Procedures

If \( P_1, P_2, \ldots, P_N \) denote \( N \) processes, a program is denoted as:

\[
[P_1] [P_2] \ldots [P_N]
\]

A process is defined as follows:

- Process name
- Variables local to the process
- Common Procedures
- Process Initialization
- Variables local to Process Initialization

Each process contains a number of variables that are accessible to all the procedures of the process. A procedure in one process may call a procedure in another process, but not a procedure within the same process. Each process also includes a process initialization section and the execution of a process begins with this section.

A procedure is defined as follows:

- Proc name(input parameters\#output parameters)
- Variables local to procedure
- Procedure body

There are two types of parameters: in and out. A procedure can be called by the process initialization of the process within which the procedure is defined (i.e. internal calls). It can also be called by any other procedures or process initializations that are defined within any of the other processes (i.e. external calls).

A process \( P \) can call the procedure \( R \), defined within process \( Q \), by the statement:

\[
\text{CALL } Q.R(\text{expressions}\#\text{variables})
\]

When a procedure call is accepted, a new incarnation of the called procedure is created for the calling process. Previous to the execution of the procedure \( R \), the expression values of the call are assigned to the in parameters. When the execution of the procedure terminates, the out parameters are assigned to the variables listed