The Essence of Parallel Algol

Stephen Brookes

Department of Computer Science, Carnegie Mellon University, Schenley Park, Pittsburgh, Pennsylvania 15213

Received March 1997; revised June 1998

We consider a parallel Algol-like language, combining procedures with shared-variable parallelism. Procedures permit encapsulation of common parallel programming idioms. Local variables provide a way to restrict interference between parallel commands. The combination of local variables, procedures, and parallelism supports a form of concurrent object-oriented programming. We provide a denotational semantics for this language, simultaneously adapting possible worlds to the parallel setting and generalizing transition traces to the procedural setting. This semantics supports reasoning about safety and liveness properties of parallel programs, and validates a number of natural laws of program equivalence based on noninterference properties of local variables. The semantics also validates familiar laws of functional programming. We also provide a relationally parallel semantics. This semantics supports standard methods of reasoning about representational independence, adapted to shared-variable programs. The clean design of the programming language and its semantics shows that procedures and shared-variable parallelism can be combined smoothly.

1. INTRODUCTION

The programming language Algol 60 has had a major influence on the theory and practice of language design and implementation [10]. Algol shows how to combine imperative programming with an essentially functional procedure mechanism, without destroying the validity of laws of program equivalence familiar from functional programming. Moreover, procedures and local variables in Algol can be used to support an "object-oriented" style of programming. Although Algol itself is no longer widely used, an idealized form of the language has stimulated a great deal of innovative research [10]. Idealized Algol, as characterized by Reynolds [14], augments a simple sequential imperative language with a procedure mechanism based on the simply typed call-by-name \( \lambda \)-calculus; procedure definitions, recursion, and the conditional construct are uniformly applicable to all phrase types. Reynolds identified these features as embodying the essence of Algol.

Although Algol 60 and Idealized Algol are sequential programming languages the utility of procedures and local variables is certainly not limited to the sequential setting. Nowadays there is much interest in parallel programming, because of the potential for implementing efficient parallel algorithms by concurrent processes designed to cooperate in solving a common task. In this paper we focus on one of the most widely known paradigms of parallel programming, the shared-variable model, in which parallel commands (or "threads") interact by reading and writing to shared memory. The use of procedures in such a language permits encapsulation of common parallel programming idioms. Local variable declarations provide a way to delimit the scope of interference: a local variable of one process is not shared by any other process and is therefore unaffected by the actions of other processes running concurrently.

To illustrate the use of procedures as a means of encapsulation, a procedure for implementing mutual exclusion [2] with a binary semaphore can be written (in sugared form) as:

```plaintext
procedure mutex(n_1, c_1, n_2, c_2);
    boolean s;
    begin
        s := true;
        while true do (n_1; await s then s := false; c_1; s := true)
        || (while true do (n_2; await s then s := false; c_2; s := true))
    end
```

Here \( c_1 \) and \( c_2 \) are parameters representing "critical" regions of code, and \( n_1 \) and \( n_2 \) represent non-critical code. The local boolean variable \( s \) represents the semaphore. The correctness of this procedure,