THE SCRIPTIC PROGRAMMING LANGUAGE

André van Delft
Delftware Technology B.V., Gentesstraat 165,
2587 HP Den Haag, The Netherlands.
E-mail: delft@fwu.uva.nl

INTRODUCTION

Over the last years a vast number of parallel languages have been developed. A large part of these are not available on 'normal' computers, or they do not cooperate easily with other languages. This paper describes Scriptic, a parallel language that extends widely used sequential languages (C, C++). Scriptic offers great expressiveness by incorporating many concepts from process theory dealing with concurrency and communication. Its availability on normal computers allows many software developers to experiment with parallel programming.

Basicly, Scriptic extends the C language with powerful and concise alternatives for the sequence operator: instead of the semicolon you can write for instance a comma for a parallel composition and a bar for choice. Scriptic has already been applied successfully in areas such as simulations, language parsers and graphical user interfaces.

This article is intended as a brief introduction to Scriptic programming. It will discuss some theory, explain most Scriptic operators and language features, and then illustrate their use through some real-world examples. Although Scriptic has also been designed as an extension to Pascal and Modula-2, these versions will not be discussed in this paper. Basic knowledge of C is assumed.

THE THEORY

The main flow structure in most programming languages is the sequence. It is often represented by a semicolon such as in Pascal and C. In the last decade a family of process theories has emerged: CSP, CCS, Process Algebra [2, 3, 4, 6, 7]. These suggest that other kinds of flow, such as choice and parallelism, can be covered in a comparable way. So if we already have a semicolon for sequence, then why not add a bar for choice, a comma for parallelism, and others?

Scriptic was founded on Process Algebra (PA). This theory is about process expressions and other representations of processes, which are well comparable to programs. In PA atomic actions are the building bricks (rather than assignments etc.) and constructs such as sequence and choice act as a kind of glue. PA basically defines axioms for sequence and (exclusive) choice, e.g.,

\[ \times | y = y | \times \]
This axiom states that choice is symmetric: the order of operands is irrelevant. Other basic axioms state laws of associativity and distribution for sequence and choice. Sequence and choice are somewhat comparable with mathematical multiplication and addition, and with and or in logic. Additional axioms in PA define constructs as parallelism in terms of sequence and choice. The atomic actions in parallel processes do not happen synchronously, but in an interleaved mode. They cannot be preempted by other atomic actions.

PA also defines axioms for two special processes: deadlock and the empty process (or immediate success). These correspond with the mathematical zero and one, or with the truth values false and true in Boolean Algebra (BA), a mathematical treatment of logic to which PA is closely related.

Currently research is done on combining BA and PA into a single theory dealing both with actions and truth values, or booleans. It seems that deadlock and the empty process are equal to the boolean values false and true. We see mixtures of actions and truth values already in conventional programming languages, where boolean expressions that are guards in if-statements determine which actions will be executed. The combination of BA and PA appears more directly in Scriptic. It suggests amongst others two kinds of parallelism: and-parallelism and or-parallelism. These are pure process generalisations of and and or for truth values. Although born as theoretical piece of work, or-parallelism has shown of significant practical use in Scriptic programs.

PRIMARY LANGUAGE CONSTRUCTS

Scriptic gets Process Algebra to work by letting fragments of C code placed between braces playing the role of atomic actions. (Unlike the situation in C, code fragments need not to end with a semicolon before the right-hand brace.). With symbols as the semicolon for sequence and the bar for exclusive choice you can make script expressions which have much in common with statement sequences in C. There is a function-like refinement construct named script which can have parameters and local variables. Unlike in C function calls, empty parameter lists in script calls may be omitted. One can specify groups of scripts in a scripts section, as in

```plaintext
scripts
Hello  = {printf ("Hello world!")}
Goodbye = {printf ("Goodbye!")}
main    = Hello; Goodbye
```

In later examples we will leave out the keyword scripts. Various operators are available for Script expressions, as shown by table 1 in their priority order. Several operators denote parallelism:

- the comma for normal parallelism or each-parallelism: each operand should succeed in order to let this parallel composition succeed.
- the plus for or-parallelism: as soon as one operand terminates successfully the others are discarded.
- the ampersand for and-parallelism: as soon as one operand enters a deadlock state the others are discarded.