µOCCAM

Chapter 1

The Grammar for μOCCAM

μOCCAM is essentially a subset of the OCCAM programming language, as described in the occam 2.1 reference manual, SGS-Thomson 1995. This is available electronically from the course web pages, and copies are distributed at the introductory lecture. For further copies, ask at the Informatics Teaching Office, JCMB room 1502. There are also books on occam in the JCM library and in the machine halls.

We assume that the reader has already glanced at a book on OCCAM, particularly at the examples, and is familiar with the general approach followed by this language. The simplifications in μOCCAM are primarily in the area of types. The only types supported are integers, channels, and one-dimensional arrays of these. Furthermore, channels can only carry integers. We treat booleans in the same way as C, i.e. as integers with 0 denoting **false**, and any other value denoting **true**. In some cases we have altered the syntax of the language where this aids syntactic or semantic analysis.

The language μOCCAM is described by the following grammar, where we have used the same indentation and repetition conventions as in Appendix H of the reference manual.

```
process      =   STOP | SKIP | action | construction | instance
             |   specification
                 process

action       =   assignment | input | output

assignment   =   variable := expression

input        =   channel ? variable
output       =   channel ! expression

construction =   sequence | conditional | loop | parallel | alternation

sequence     =   SEQ
                   { process }
             |   SEQ replicator
                   process

conditional  =   IF
                   { choice }
             |   IF replicator
                   choice
choice       =   boolean
                   process
boolean      =   expression
```

```
loop            =   WHILE boolean
                        process

parallel        =   PAR
                      { process }
                  |  PAR replicator
                        process

alternation     =   ALT
                      { alternative }
                  |  ALT replicator
                        alternative

alternative     =   guard
                        process
guard           =   input  |  boolean & input

replicator      =   name = base FOR count
base            =   expression
count           =   expression

type            =   primitive.type
                  |  [ expression ] primitive.type

primitive.type  =   INT  |  CHAN

literal         =   integer

element         =   name  |  name [ subscript ]

subscript       =   expression
variable        =   element
channel         =   element

operand         =   variable  |  literal  |  ( expression )

expression      =   monadic.operator operand
                  |  operand dyadic.operator operand
                  |  operand

specification   =   declaration  |  definition

declaration     =   type name :
                  |  INT name = expression :
                  |  VAL name IS expression :

definition      =   PROC name ( {0, formal } )
                        procedure.body
                        :

formal          =   primitive.type name
procedure.body  =   process

instance        =   name ( {0, actual } )
```

actual	=	element
monadic.operator	=	`-` | `NOT`
dyadic.operator	=	`+` | `-` | `*` | `/` | `\` | `=` | `<` | `>` | `<=` | `>=` | `<>` | `AND` | `OR`

Notes:

The following items clarify certain points about the language, and highlight differences from the language as given in the occam 2.1 reference manual.

- Integer literals consist of one or more digits, and a name consists of a sequence of alphanumeric characters and dots starting with an alphabetic character (reference manual page 105).
- Indentation carries significant meaning in μ`OCCAM`. See pages 3 and 4 of the reference manual.
- TAB characters. Any line may begin with some TAB characters, each counting for four levels of indentation, followed by some spaces, two spaces per indentation level. Once you are on to spaces, no more TAB characters are allowed. TAB characters are not legal in the middle of a line.
- Comments. There may be comments anywhere within a program except within a continuation line.
- Blank lines. There may be blank lines anywhere within a program except within a continuation line.
- End of file. The last statement of a program need not have a terminating newline character. That is, the end of a file may appear either at the end of the last program line, or on a line of its own. This is necessary because not all editors allow the user easy control of this.
- Procedure parameters are passed by *value*, just as in C. This is different from the reference manual.
- Invalid processes should be treated like `STOP` (reference manual page 101).
- The count in a replicated `ALT`, the expression in a `VAL` declaration, and the array bounds in an array declaration must all be constant expressions, *i.e.* expressions whose value can be calculated at compile time. The function `constant_expression` in Section 5.8.2 can help to check this.
- The base and count of a replicated `PAR` do not have to be constant expressions as we are not running on a real distributed system.
- Two channels, `stdin` and `stdout`, are predefined and connected to the standard input and output at the start of execution.

When you find an illegal program (e.g. with lexical, syntactic or semantic errors) you should call the error function (see Section 6.3). This will stop the program and set the return code appropriately. You do not need to spend time adding error recovery mechanisms to the parser. Obviously this is something you would like in a real compiler, but there is insufficient time in this practical for such niceties. When completed, the compiler should check for type errors such as integers used as channels, procedures called with the wrong number of arguments *etc.* However, you do not need to check that variables are only updated by one process, or that only one process inputs or outputs on a channel. As the language has arrays, such constraints can only be partially checked at compile time anyway. However, we will assume that all programs passed to the compiler satisfy these constraints.

Hopefully the grammar, in addition to the `OCCAM` manual, will answer most of your questions about μ`OCCAM`. When in doubt, ask the project organiser rather than just guessing.