Details of Occam-2 Grammar Definition Development

Larry Dickson
Space Sciences Corporation
9-20, 28 June 2018
6 July 2018

Summary

This long note is intended to be included with a copy of the relevant files cited here, in order to provide a step-by-step and verifiable description of the changes made in the original syntax, symbols, and keywords presented in the occam 2 Reference Manual [I] to generate the lex file OCCAMLIB.L (31 May 2018 at 15:45 PT) and the grammar file OCCAMLIB.Y (28 June 2018 at 10:29 PT), as well as a new BNF file derived from the grammar.

Tools

The system is a Mac OS X, Version 10.9.5. Its program Preview is used on [I] to extract pages 80-90, where the syntax description is, and its copy-paste capability is used with its program TextEdit to capture text in the file occlang/crippled-syntax-text.txt, which has Mac line ends (CR only) and other oddities. The program nano version 2.0.6, compiled 18:58:13, Aug 24 2013, is used to open, convert, and save this with DOS line ends but still many oddities in occlang/crippled-syntax-text2.txt, and the ordered syntax found there is hand-corrected (using the standard format book as a guide) to give occlang/CRI.txt, the starting point of this work. The file occlang/CRI.txt (3 May 2018 at 11:14 AM) is unchanged from the printed ordered syntax, using {o , x } for the subscript o comma repeater for x, and {1 g x } for the subscript 1 separator g repeater for x, where g is comma or semicolon.

All programs, except the original viewing with Preview and copy-paste to TextEdit, are command line Unix-like programs found in Mac BSD. These include grep 2.5.1-FreeBSD, sed (whose man file, dated May 10, 2005, refers to FreeBSD), flex 2.5.23 Apple(flex-31), and bison (GNU Bison) 2.3. These command-line programs are expected to be substantially identical to versions of grep, sed, flex (lex), and bison (yacc) found on other Unix-like systems such as Linux. The commands are:

```
yacc -v -d OCCAMLIB.Y
lex OCCAMLIB.L
yacc -d OCCAMLIB.Y
gcc lex.yy.c y.tab.c -o OCCAMLIB-COMPILER
./OCCAMLIB-COMPILER < filename.occ >filename.out 2>filename.err
```

And, in addition, it produces `y.tab.h` header file and `y.output` expansion of the full state description.

Progressive Discussion

(Reference: Appendices 2 and 3)

The following discussion will refer to changes made in time order. See Appendix 2 for the precise references. The changes are called "Substantive" if there is a change in which programs are legal, and "Organizational" if there is a change in how the grammar relates to the semantics, without an ultimate semantic change.

(A) 2018-05-14 occlang3/CRI.nl from Manual BNF (reference [I])

SUBSTANTIVE CHANGE

```
( value.process
)
```

is REMOVED from the definitions of `operand` and `expression.list` .

Summary: In-line value process is no longer allowed, thus allowing sub-line and multi-line syntactic objects to be strictly distinguished by descent. (This will make no essential syntactic change because it will be equivalently replaced by `ANY FUNCTION`, below.)

ORGANIZATIONAL CHANGES

New syntactic objects
`comms.type`
`counting.type`
`data.type`
are ADDED to subdivide `primitive.type` and restrict the BNF definitions of `definition`, `expression`, `literal`, and `simple.protocol` .

Summary: Types are subdivided to conform more closely to the description of the semantics in [I]. The grammar is still more permissive than the semantics, but not by as much. The descent is cleaned up, e.g. `literal` is now SIMPLE rather than equivalent to `expression`.

REFERENCE: Appendix 3 part A.

(B) 2018-05-25 occlang4/CRI.Y.tyo from occlang3/CRI.txt (essentially same as CRI.nl)

SUBSTANTIVE CHANGES

Error fix in [I] BNF:
`andor.expression`
is ADDED to subdivide `dyadic.operator` form of `expression`, and `expression` definition is expanded to accommodate.

Summary: The semantic description in [I] 7.2.6 Boolean expressions (page 48) is inconsistent with the BNF stated in [I] because it allows parentheses to be omitted in expressions involving AND and OR. Experimentation with the compiler shows this works, with evaluation starting at the left, and even if AND and OR are mixed. The change involving `andor.expression` models this correctly.

Replacement for inline value process:

```
ANY FUNCTION name ( )
  function_body
:
```

is ADDED to definition of `definition`, in order to replace inline value process. If placed directly above the line containing the former inline value process, with a unique name called at that point, it behaves provably the same.

Summary: See discussion of value process and function in [I].

Toplevel constructs:
`block.definition`
`block.definition.list`
`program`
are ADDED so that library-like program files can be compiled. A `block.definition` is broken out of `definition` and contains multi-line PROC and FUNCTION, a `block.definition.list` is a number of these on the same (top) level, and a `program` may be a `block.definition.list` or a `process`.

Summary: This creates a descent category TOP above PROCESS. It is needed if libraries are to be compilable, using yacc's approach of starting with the top syntactic object. The inclusion of both `process` and `block.definition.list` in `program` will need to be modified later.

Reference: Appendix 3 part B.

(C) 20180529 occlang4/CRI.txt from occlang4/CRI.Y.tyo (to format for yacc grammar)

ORGANIZATIONAL CHANGES

Change interior . to _ (C style names) - CRI.Y.tyq
Put single quotes around symbols - CRI.Z.tyr
Change BNF definition = to : - CRI.Z
Remove single quotes from { } (not symbols) - CRIol2.Y
Hand-reorder so that `program` is at top - CRI-20180525.Y

Name two-character symbols - CRI-curly.Y
Replace curly-bracket repeaters with new syntactic objects - CRI-nocurly.Y
Hand-add definitions of new _vlist, _olist, _clist, _slist objects - CRI-20180526.Y
Hand-insert NEWLINE, RIND, ROUTD, and definitions using lexer tokens - CRI.Y
Remove fold comments using origami or sed - CRI.txt

Summary: These changes though extensive, and involving new syntactic objects, are only organizational. The end result is in a format that can be handled by yacc. Spacing, newlines, and indentation are no longer significant in the grammar, being handled by NEWLINE, RIND, and ROUTD (though left in for convenience of viewing).

Reference: Appendix 2 between occlang4/CRI.Y.tyo and occlang4/CRI.txt.

(D) 20180606 occlang4/OCCAMLIB.Y from occlang4/CRI.txt

The changes to follow are designed to get improved results from yacc at each stage. They are organizational, not substantive, though they sometimes make considerable differences in the grammar; these differences are eliminated by semantic considerations, mainly declared type values.

Add ; line after each grammar definition; add tokens - OCCAMOLD.Y
OCCAMOLD.Y fails yacc due to port, timer, element conflicts. 6 S/R, 25 R/R
`process`-only `program`; merge `port`, `timer` with `channel` - OCCAMOL2.Y
Successful yacc (here and below); 6 Shift/Reduce, 12 Reduce/Reduce
Add `block_definition_list` back to `program` - OCCA.Y
6 Shift/Reduce, 21 Reduce/Reduce (9 extra due to `program` ambiguity)
Remove process from program definition - OCCAMLIB-20180529.Y
6 Shift/Reduce, 12 Reduce/Reduce (now LIB type only, no `program` ambiguity)
Remove redundancies from definitions, remove `variable_list` - OCCAML1.Y
4 Shift/Reduce, 8 Reduce/Reduce
Rename `tag` to `scalar` and include in `element`; new `indef_type` - OCCAMLIB-20180530.Y
6 Shift/Reduce, 5 Reduce/Reduce
Remove redundant channel '!' scalar from output - OCCAMLIB-201805312037.Y
6 Shift/Reduce, 4 Reduce/Reduce - NOW TEST SAMPLE .occ FILES . . .
Add %glr-parser - OCCAMLIB-20180531.Y
6 Shift/Reduce, 4 Reduce/Reduce
Enter %dprec to two ambiguities - OCCAMLIB-20180604.Y
6 Shift/Reduce, 4 Reduce/Reduce
Enter %dprec to one more ambiguity - OCCAMLIB.Y
6 Shift/Reduce, 4 Reduce/Reduce

Summary: Up until OCCAMLIB-201805312037 it is mostly removing redundant parts of definitions (i.e. completely contained in other parts of the same definition), plus `scalar` being used both for `tag` and for a `name` as an `element`, which are always distinguishable by semantics of declarations. After OCCAMLIB-201805312037, %glr-parser capability of yacc (actually bison) permits multiple paths during ambiguities and, with preferences imposed by %dprec, handles them all correctly.

References: Appendix 2 after occlang4/CRI.txt and Appendix 3 part C.

(E) 20180531 OCCAMLIB.L development

Summary: The tokens are handled slightly different than in [I], but equivalently. Actually they are more permissive, since uncounted lines (full-line comments, continuations, and Directives) do not take account of indentation. Because of the removal of inline value processes, the remaining lines always begin at even indentations, hence RIND and ROUTD as well as NEWLINE. The handling of multi-line string literals is equivalent to that defined in [I] 3.2 Literals (page 26). Directives are as defined in the INMOS occam Toolset ([IOT1] section 25.10), and are not active here, but treated as comments (e.g. `#INCLUDE`).

Reference: the part of Appendix 2 toward the end, from occlang3/undertok.nl to occlang4/OCCAMLIB.L.

(F) Fixup of OCCAMLIB.Y

Summary: One error was found: in `tagged_protocol`, the first variant (`scalar` alone) needed to be followed by a `NEWLINE`. Oddly, the uncorrected version was self-consistent (a bunch of lone tags on the same line), but unrealizable for the case when the protocol ended with a lone tag.

(G) Reconstruction of a BNF from final OCCAMLIB.Y.

Summary: This reconstruction was comparatively simple. See the end of Appendix 2. The reconstruction left `actual`, `expression_list`, and `output` with `%dprec` rankings.

Tests

(Reference: Appendix 4)

The parser OCCAMLIB-COMPILER should work on any occam program that is composed of a sequence of PROCs and FUNCTIONs at the top level, and that avoids in-line value processes, using only FUNCTION-enclosed value processes. Both the lexing and the parsing should be equivalent to the real occam-2 compiler. The run as shown will output a token list (with a few inserted comments having to do with the resolving of ambiguities) to the standard output, and an exhaustive parse dump, including %glr-parser parallel parsing with resolution based on branch failure or %dprec, to the standard error. Success is indicated by the last line to standard output being ENDFILE.

The runs with sample .occ files are, of course, not an exhaustive test. However, the behavior at every ambiguity is checked by the small parser-occam files, and a typical real occam file works (from a real test program that runs on a real legacy Transputer array, with a lot of continuation lines added) in the first file. It should be possible to duplicate these tests using any reasonable version of the tools on any Unix-like system. Note that the .occ, .L and .Y files were all created using the fold editor origami.exe, which runs in DOS or a DOS emulator like DOSBOX (hence the DOS line ends).

The references below are to y.output which is generated as described in **Tools** above, which lists 262 rules (numbered 1 through 262) and 574 states (numbered 0 through 573). Of these states, eight exhibit ambiguities, all of which are solved using `%glr-parser`, either via failure of one of the branches or via a preference denoted by `%dprec`. Two of the eight states exhibit two ambiguities each, thus the total number of ambiguities is ten, but in each of the double-ambiguity cases, the two are related.
In the discussion below, the term "fixed token" will refer either to a symbol or a keyword, to distinguish it from a "variable token" such as a number or a name.

State 84 conflicts: 1 shift/reduce
State 141 conflicts: 1 shift/reduce

These are basically the same, found in a formals sequence that can continue with either a name or with a new type. A type is always a fixed token, unlike a name, so one branch of %glr-parser will fail, with no need for %dprec.

Test files: small.occ, small2.occ, small3b.occ, small3c.occ.

State 226 conflicts: 1 shift/reduce

This is a type followed by a name, and branches according as it is a type in a declaration, or a specifier in a definition or an abbreviation (a RETYPES or IS). The latter case always hits keywords RETYPES or IS before the colon, so the branches are distinguished without %dprec.

Test file: small6.occ (both branches).

State 310 conflicts: 1 shift/reduce

This needs a %dprec to prefer the scalar branch over the element branch, preventing wasted effort if there is a mere name before the semicolon in an output.

Test files: small4.occ, small4b.occ.

State 330 conflicts: 2 reduce/reduce

In both cases (detecting a ')' or a ','), this is found ultimately only in an instance (state 271), and is solved by a %dprec of actual element over operand.

Test files: small5.occ, small5b.occ.

State 350 conflicts: 1 shift/reduce

This comes in from a lot of sources, but ultimately reduces to an alternative, a choice, or an option. See small7.occ. It can be either a conversion in a live line of code, not ending in a colon, or a type in a declaration, ending in a colon. This distinction means a %dprec is not needed.

Test file: small7.occ (both cases).

State 406 conflicts: 1 shift/reduce

Although apparently dissimilar, this arises because of the same aliasing in an alternative, a choice, or an option as State 350 ambiguity. In one case (the live line) it is a member of a table, in the other it is a two-dimensional type. Distinction is via the colon as in State 350.

Test file: small8.occ (both cases).

State 509 conflicts: 2 reduce/reduce

In this case, in an explicit FUNCTION IS, or a RESULT or an ASSIGN, after the function instance with the parenthesized expression_olist is closed by an upcoming colon or NEWLINE, a %dprec prefers interpreting it as an expression_list rather than an operand. This is all that is needed when it is "naked", not followed by an operator.

Test files: small3.occ, small3b.occ, small3c.occ.

Appendix 1. DESCENT PROCEDURE

The Descent Procedure starts with a file CRI.nl that consists of multi-line BNF entries of the form

```
syntactic-object-name = syntax { | syntax }
```

where each syntax is a combination of syntactic-object-names, symbols, and keywords which may spread over one or more lines. `{ xxx }` means "repeat `xxx` zero or more times". An example of such a multi-line BNF entry is

```
conditional = IF
```

```
                { choice }
| IF replicator
    choice
```

The multi-line entries are separated by single empty lines.

CRI.nl

```
          sed -e'/^$/d' (Remove all empty lines)
```

CRI2.nl

```
          for i in `cat CRI2.nl`; do echo $i; done|sort -u
```

tokens.nl

```
          grep "^[^[:space:]]* =" CRI2.nl | sed -e"s/ =.*//"
```

CRI3.nl

```
          grep -n "^[^[:space:]]* =" CRI2.nl | sed -e"s/:.*//"
```

CRI4.nl

copy CRI4.nl, remove first line, add line at end that is `wc -l CRI2.nl`+1

CRI5.nl

```
          paste CRI3.nl CRI4.nl CRI5.nl
```

CRI6.nl

```
          mkdir CRI
          cd CRI
          cat ../CRI6.nl | while read line ; do j=0 ; for x in `echo $line` ; do j=$(($j+1)) ; if [ $j -eq 1 ] ; then s1=$x ; elif [ $j -eq 2 ] ; then s2=$x ; elif [ $j -eq 3 ] ; then s3=$x ; fi ; done  ; echo $s1 W $s2 W $s3 ; head -n $(($s3-1)) ../CRI2.nl | tail -n $(($s3-$s2)) > $s1 ; done
          cd ..
```

CRI/*

```
          mkdir CRItgt
          cd CRItgt
          cp -pi ../CRI/* .
          for i in `ls` ; do sed -i .tyo -e"s/^[^[:space:]]* = //" $i ; done
          rm -f *.tyo
          cd ..
```

CRItgt/*

```
          cd CRItgt
          for i in `ls` ; do ls -l $i ; cat $i ; done
          cd ..
```

CRI7

```
          mkdir CRInovptgt
          cp -pi CRItgt/* CRInovptgt
          cd CRInovptgt
```

remove (value process) from operand, expression.list if necessary

```
          cd ..
```

CRInovptgt/*

```
          cd CRInovptgt
          for k in `ls` ; do for i in `cat $k` ; do echo $i ;
done > ../tyog ; echo $k INCLUDES ; for i in `ls` ; do if grep -
q "^$i$" ../tyog ; then echo $i ; fi ; done ; done > ../CRI9.nl
          cd ..
```

CRI9.nl

```
          mkdir CRInovp9
          cd CRInovp9
          cat ../CRI9.nl | while read line ; do if echo $line |
grep -q " INCLUDES" ; then i=`echo $line | sed -e"s/
INCLUDES//"` ; touch $i ; else echo $line >> $i ; fi ; done
          cd ..
```

CRInovp9/*

```
          mkdir des9
          cd CRInovp9
          for i in `ls` ; do ../descend $i ; done
          cd ..
```

des9/*

```
          cd des9
          wc `ls -S *.des` > ../des.lst
          cd ..
```

des.lst

The bash script `descend` is:

```
echo $1 > ../des9/$1.des
ind=0
cur=$1
while [ $ind -lt `wc -l < ../des9/$1.des` ] ; do
  ind=$(($ind+1))
  cur=`head -n $ind ../des9/$1.des | tail -n 1`
  echo DESCENDANT $cur AT INDEX $ind
  for i in `cat $cur` ; do
    if grep -q "^$i\$" ../des9/$1.des ; then
      echo $i REPEAT
    else
      echo $i >> ../des9/$1.des
      echo $i NEW
    fi
  done
done
```

The description of each stage of the descent procedure is as follows:

The creation of CRI2.nl removes all empty lines.

The creation of tokens.nl creates a sorted list of tokens of CRI2.nl, where tokens are defined as non-whitespace-including character sequences separated by whitespace. Each token is included only once. Tokens that are part of BNF syntax (like | = { }) are included without distinction. This is a check step against the list in [I].

The creation of CRI3.nl generates the syntactic-object-name for each multi-line BNF entry in CRI2.nl.

The creation of CRI4.nl generates the start line number for each multi-line BNF entry in CRI2.nl.

The creation of CRI5.nl generates the (end line number)+1 for each multi-line BNF entry in CRI2.nl.

The creation of CRI6 generates a file with three tokens on each line: syntactic-object-name, start line number, (end line number)+1.

The creation of CRI/* generates a file for each syntactic object, named by its name, and containing its multi-line BNF entry.

The creation of CRItgt/* generates a file for each syntactic object, named by its name, and the same as in CRI except the heading "syntactic-object-name = " is removed.

The creation of CRI7.nl brings each CRItgt file into a single list headed by its ls -l file listing.

The creation of CRInovptgt/* copies CRItgt/* but if necessary (i.e. only in `occlang`) removes the BNF definition

```
( value.process
)
```

from both `expression.list` and `operand`. This is required to create a descent distinction between within-line (e.g. expression or operand) syntactic objects and multi-line (e.g. process) syntactic objects. It also simplifies the indentation rules. The capability removed here is equivalently restored (without requirement of semantic analysis) by the later addition of the

```
ANY FUNCTION name ( )
  function.body
:
```

function definition variant.

The creation of CRI9.nl makes a file in which each syntactic-object-name is followed by the word REQUIRES on the same line, followed by a single line for each syntactic-object-name that is part of its BNF definition in CRInovptgt. Other tokens, like symbols or fixed keywords, are not mentioned.

The creation of CRInovp9/* creates for each syntactic-object-name a file by that name which includes its list of syntactic-object-names needed for its BNF definition as found in CRI9.nl. This is the first step of descent list.

The creation of des9/* recurses through the files of CRInovp9 to create a file for each syntactic-object-name that list all the syntactic-object-names needed by its BNF definition at any level of descent. Anything included in an object's descent list also has its own entire descent included in the object's descent list.

The creation of des.lst generates a size-ordered length description from all the descents of des9/*. Due to the last sentence of the preceding paragraph, all members of a circular descent-dependency set of syntactic objects will have the same specifications in des.lst, while a strict inclusion will always have strictly greater specification in des.lst. A check that either of these dependencies actually holds is easily confirmed by finding one in the list of the other.

Note: the most important circular dependency sets are the one that is equivalent to `process` and the one that is equivalent to `expression` or `operand`. The elimination of the embedded value processes prevents these from being the same. Between them is a set of (non-equivalent) syntactic objects that each consist of at least one full syntactic line. The set of multi-line syntactic objects and the set of sub-line syntactic objects are mutually exclusive, and together comprise all syntactic objects. No multi-line syntactic object includes partial lines. (A syntactic line may include continuations.)

Appendix 2. Exact file creation description

All of the files listed on the left hand side should be part of the tarball that comes with this article. Command line program calls (mainly sed) work by having the previous file as a parameter and redirecting into the next file. Hand work and transitions are described, with techniques for checking.

IMPORTANT NOTE: In all `sed` and other quotes below, `^M` is the single character "carriage return" (hex 0D) which is typed in as the sequence Ctrl v Ctrl m on the Mac keyboard, and in other ways on other systems, and displays this way in Mac BSD.

oc20refman.pdf

(Source [I]) Copy paste into Mac TextEdit and save

occlang/crippled-syntax-text.txt

Open with nano -w, tweak to force store, save as DOS line ends

occlang/crippled-syntax-text2.txt

Hand convert to equivalent to [I] G.2 Ordered syntax, page 86-90

occlang/CRI.txt

`sed -e's/^M$//'` (converts DOS line ends to Unix line ends)

occlang/CRI.nl

DESCENT PROCEDURE

occlang/des.lst and others

Hand rearrangement of CRI.TXT into five folds using origami.exe

occlang/CRICAT.TXT	Master of following five sets of BNF definitions
PROCESS.EQU	Multi-line, circular dependency equivalents of `process`
LINE.EQU	Full line, below `process`, above `expression`
MIDDLE.EQU	Part line, below PROCESS and LINE, above `expression`
EXPRESON.EQU	Part line, circular dependency equivalents of `expression`
SIMPLE.EQU	Part line, below `expression`

Hand revision of CRI.txt (see Appendix 3 part A after next step)

occlang3/CRI.txt

`sed -e's/^M$//'` (converts DOS line ends to Unix line ends)

occlang3/CRI.nl

DESCENT PROCEDURE

occlang3/des.lst and others

Hand rearrangement of CRI.TXT into five folds using origami.exe

occlang3/CRICAT.TXT	Master - same as the one in occlang
PROCESS.EQU	Multi-line, circular dependency equivalents of `process`
LINE.EQU	Full line, below `process`, above `expression`
MIDDLE.EQU	Part line, below PROCESS and LINE, above `expression`
EXPRESON.EQU	Part line, circular dependency equivalents of `expression`
SIMPLE.EQU	Part line, below `expression`

Remove all the syntactic-object-names from tokens.nl, insert "> "

occlang3/undeftok.nl

Copy occlang3/CRI.nl without change to occlang4/CRIold.nl

occlang4/CRIold.nl

Hand revision of occlang3/CRI.txt (see Appendix 3 part B)

occlang4/CRI.Y.tyo

`sed -e"s/[.]/_/g"` (BEGIN CONVERTING TO YACC GRAMMAR)

occlang4/CRI.Y.tyq

```
            sed -e"s/^/ /" -e"s/^M$/ ^M/"
```

occlang4/CRI.Z.tyq

```
            sed -e"s/^\(.*\) \([^ |=0-9]\) \(.*\)$/\1 '\2' \3/" -
e"s/^\(.*\) \([^ |=0-9]\) \(.*\)$/\1 '\2' \3/" -e"s/^\(.*\) \([^
|=0-9]\) \(.*\)$/\1 '\2' \3/" -e"s/^\(.*\) \([^ |=0-9]\) \(.*\)
$/\1 '\2' \3/" -e"s/^\(.*\) \([^ |=0-9]\) \(.*\)$/\1 '\2' \3/" -
e"s/^\(.*\) \([^ |=0-9]\) \(.*\)$/\1 '\2' \3/" -e"s/^\(.*\) \([^
|=0-9]\) \(.*\)$/\1 '\2' \3/"
```

occlang4/CRI.Z.tyr

```
            sed -e"s/ = / : /"
```

occlang4/CRI.Z

```
            sed -e"s/^ //" -e"s/ ^M$/^M/"
```

occlang4/CRI.Y.tyr

```
            sed -e"s/'{'/{/g" -e"s/'}'/}/g"
```

occlang4/CRIol2.Y

Hand-reordered in origami to the folds TOP, PROCESS, LINE, MIDDLE, EXPRESSON, and SIMPLE, which are equivalent to the folds of similar names in CRICAT.TXT in occlang and occlang3, except for adding `program` and `block_definition_list` in TOP strictly above process, and hand revisions described in Appendix 3 part B (from occlang3/CRI.txt to occlang4/CRI.Y.tyo); see chkreord.tgz, in which pre.Y is identical to CRIol2.Y and post.Y is identical to CRIol2.Y, and post2.Y is identical to post.Y except for removal of the fold comments, and is checked to have the same content as pre.Y.

occlang4/CRI-20180525.Y

```
sed -e"s/:=/ASSIGN/" -e"s/::/DOUBLE_COLON/" -e"s/\[\]/
INDEF_SPEC/" -e"s/=/'='/"
```

occlang4/CRI-curly.Y

```
sed -e"s/^\(.*\){1 ';' \([^ ]*\) }\(.*\)$/\1\2_slist
\3/" -e"s/^\(.*\){1 ',' \([^ ]*\) }\(.*\)$/\1\2_clist\3/" -e"s/^
\(.*\){1 ',' \([^ ]*\) }\(.*\)$/\1\2_clist\3/" -e"s/^\(.*\){o
',' \([^ ]*\) }\(.*\)$/\1\2_olist\3/" -e"s/^\(.*\){ \([^ ]*\) }\
(.*\)$/\1\2_vlist\3/"
```

occlang4/CRI-nocurly.Y

Hand-add the definitions of every new vlist, slist, clist, and olist syntactic object, taking the place of the curly bracket constructions in BNF. Each is placed in the fold with its singleton. Each olist requires a corresponding clist to be defined. At this point, the vlist can contain 0 members, as defined in [I].

occlang4/CRI-20180526.Y

Hand-insert NEWLINE, RIND (= relative indent of two spaces), and ROUTD (= relative outdent of two spaces) where needed to eliminate need for indentation awareness within the grammar definition. Also redefine every vlist so that it must contain at least one member, and add extra options to those definitions that use a vlist, to separately account for the zero member case. This is needed because there is no way to detect a RIND/ROUTD with nothing between. Also in fold SIMPLE, define `andor_operator`, `byte`, `dyadic_operator`, `integer`, `monadic_operator`, `name`, `real`, `string`, and `string_head` in terms of lexer tokens.

occlang4/CRI.Y

Remove fold comments using origami or sed.

occlang4/CRI.txt

`sed -e's/^M$//'` (converts DOS line ends to Unix line ends)

occlang4/CRI.nl

DESCENT PROCEDURE

occlang4/des.lst and others

Starting from CRI.txt:

Apply the changes found in Appendix 3 part C.

Then change every blank line by inserting ; at the beginning. This yields the final version OCCAMLIB.Y.

(In fact it was done in the other order. By doing diffs between successive entries in the list, it can be seen how the progress was actually made. For instance, the change from CRI.txt to OCCAM.Y.tyq is done by adding all the tokens, and the change from OCCAM.Y.tyq to OCCAMOLD.Y is done by inserting ; at the beginning of every

blank line. The change from OCCAMOLD.Y to OCCAMOL2.Y is done by taking out reference to block_definition_list, leaving only process as program, and conflating port and timer with channel (eliminating a yacc error). But OCCA.Y retains the reference to block_definition_list in program. The output OCCAM.output is successful yacc compile from OCCA.Y, with 6 shift/reduce and 21 reduce/reduce ambiguities, whereas eliminating block_definition_list in OCCAMOL2.Y leads to 6 shift/reduce and12 reduce/reduce. From then on, all further progress is checked by looking at yacc output files.)
occlang4/OCCAM.Y.tyq
occlang4/OCCAMOLD.Y
occlang4/OCCAMOL2.Y
occlang4/OCCA.Y
occlang4/OCCAM.output
occlang4/OCCAMLIB-20180529.Y
occlang4/OCCAML1.Y
occlang4/OCCAMLIB-20180530.Y
occlang4/OCCAMLIB-201805311523.Y
occlang4/OCCAMLIB-201805312037.Y
occlang4/OCCAMLIB-20180531.Y
occlang4/OCCAMLIB-20180604.Y
occlang4/OCCAMLIB-201806060942.Y
occlang4/OCCAMLIB.Y (JUNE 6 VERSION)
occlang5/OCCAMLIB.Y (CURRENT VERSION - ONE CORRECTION)

occlang3/undeftok.nl (repeated from above - lexer files continued)

Devise input file for lex based on occlang3/undeftok.nl: separate sub-parser state A from INITIAL, define D=[0-9], L=[a-zA-Z], H=[A-F0-9], E=[Ee][+-]?{D}+, S=[CcNnTtSs"'*], list all Directives if <INITIAL>, full-line comments if <INITIAL>, STRING_LITERAL_MIDDLE and STRING_LITERAL_END if <INITIAL>, otherwise count indentation if <INITIAL>; and detect undeftok.nl tokens plus IDENTIFIER, HEX_CONSTANT, INT_CONSTANT, BYTE_CONSTANT, REAL_CONSTANT, STRING_LITERAL, and STRING_LITERAL_START if <A>, and also detect continuations.

occlang4/OCCAM.L

Add sub-parser state N, detect NEWLINE, RIND, and ROUTD from non-Directive, non-continuation, non-comment indentations.

occlang4/OCCAMLIB-20180530.L

Strip some C commentary and add single quotes to some; no substantive changes.

occlang4/OCCAMLIB-201805311132.L

Start returning values in the C code, as required to help yacc.

occlang4/O2.L

Return values for the keywords in the C code.

occlang4/O3.L

Split CHAN OF and PORT OF into separate keywords.

occlang4/OCCAMLIB-201805311349.L

```
          Return STRING_LITERAL_MIDDLE, STRING_LITERAL_END,
NEWLINE, RIND, ROUTD, IDENTIFIER, HEX_CONSTANT, INT_CONSTANT,
BYTE_CONSTANT, REAL_CONSTANT, STRING_LITERAL,
STRING_LITERAL_START, and values for two-character symbols, plus minor end code
changes.
occlang4/OCCAMLIB-201805311416.L
          Fix commentary C, no substantive changes.
occlang4/OCCAMLIB.L (JUNE 6 VERSION)
          Fix tagged_protocol: see Appendix 3(D)
occlang5/OCCAMLIB.Y (FINAL VERSION)
```

Reconstruction of BNF from Final Version of OCCAMLIB.Y:

```
occlang5/OCCAMLIB.Y
          cp -pi OCCAMLIB.Y CRI.TXT
          sed -i .tyo -e"/[^'];[}]/d" CRI.TXT
occlang5/CRI.TXT.tyo
          cp -pi CRI.TXT CRI.TXT.typ
occlang5/CRI.TXT.typ
          <Remove stuff before %% and after second %% from CRI.TXT.>
          sed -i .tyq -e"s/ : / = /" CRI.TXT
occlang5/CRI.TXT.tyq
          sed -i .tyr -e"s/^;//" CRI.TXT
occlang5/CRI.TXT.tyr
occlang5/CRI.TXT
```

Appendix 3. Hand file changes (supplement to Appendix 2)

The following show substantive changes done by hand to the BNF on its transformation to a grammar file.

A. Changes between occlang/CRI.nl and occlang3/CRI.nl

Filename: occlang3/CRI.from.occlang.txt (20180608 4:13 PM)

```
======
--- ../occlang/CRI.nl    2018-05-03 11:16:20.000000000 -0700
+++ CRI.nl     2018-05-14 15:21:31.000000000 -0700
@@ -50,6 +50,10 @@
 | specification
   choice

+comms.type = CHAN OF protocol
+| TIMER
+| PORT OF type
+
 conditional = IF
                { choice }
 | IF replicator
@@ -64,6 +68,17 @@

 count = expression

+counting.type = BYTE
+| INT
+| INT16
+| INT32
+| INT64
+
+data.type = counting.type
+| BOOL
+| REAL32
+| REAL64
+
 declaration = type {1 , name } :

 definition = PROTOCOL name IS simple.protocol :
@@ -75,10 +90,10 @@
 | PROC name ( {o , formal } )
     procedure.body
   :
-| {1 , primitive.type } FUNCTION name ( {o , formal } )
+| {1 , data.type } FUNCTION name ( {o , formal } )
     function.body
   :
-| {1 , primitive.type } FUNCTION name ( {o , formal } ) IS
expression.list :
+| {1 , data.type } FUNCTION name ( {o , formal } ) IS
expression.list :
 | specifier name RETYPES element :
 | VAL specifier name RETYPES expression :

@@ -96,12 +111,10 @@
 | operand dyadic.operator operand
 | conversion
 | operand
-| MOSTPOS type
-| MOSTNEG type
+| MOSTPOS data.type
+| MOSTNEG data.type

-expression.list = ( value.process
-                  )
-| name ( {o , expression } )
+expression.list = name ( {o , expression } )
 | {1 , expression }

 formal = specifier {1 , name } | VAL specifier {1 , name }
@@ -136,9 +149,9 @@

 literal = integer
 | byte
-| integer ( type )
-| byte ( type )
-| real ( type )
+| integer ( data.type )
+| byte ( data.type )
+| real ( data.type )
 | string
 | TRUE | FALSE

@@ -146,8 +159,6 @@
          process

 operand = element | literal | table | ( expression )
-| ( value.process
-  )
 | name ( {o , expression } )

 option = {1 , case.expression }
@@ -185,17 +196,8 @@

 port = element

-primitive.type = CHAN OF protocol
-| TIMER
-| BOOL
-| BYTE
-| INT
-| INT16
-| INT32
-| INT64
-| REAL32
-| REAL64
-| PORT OF type
+primitive.type = comms.type
+| data.type
```

```
 procedure.body = process

@@ -223,7 +225,7 @@

 sequential.protocol = {1 ; simple.protocol }

-simple.protocol = type | primitive.type :: [] type
+simple.protocol = type | counting.type :: [] type

 specification = declaration | abbreviation | definition

======
```

B. Changes between occlang3/CRI.txt and occlang4/CRI.Y.tyo

Filename: occlang4/diff-u-occlang3-CRI-txt-vs-CRI-Y-tyo (20180604 9:06 AM)

```
======
--- ../occlang3/CRI.txt  2018-05-14 15:21:05.000000000 -0700
+++ CRI.Y.tyo  2018-05-25 15:34:38.000000000 -0700
@@ -27,6 +27,9 @@
 | boolean & channel ? CASE
     { variant }

+andor.expression = andor.expression andor.operator operand
+| operand andor.operator operand
+
 array.type = [ expression ] type

 assignment = variable := expression
@@ -34,6 +37,17 @@

 base = expression

+block.definition = PROC name ( {o , formal } )
+                     procedure.body
+                   :
+| {1 , data.type } FUNCTION name ( {o , formal } )
+    function.body
+  :
+
+block.definition.list = block.definition
+| block.definition.list
+  block.definition
```

```
+
 boolean =  expression

 byte = ' character '
@@ -87,15 +101,13 @@
     CASE
       { tagged.protocol }
   :
-| PROC name ( {o , formal } )
-    procedure.body
-  :
-| {1 , data.type } FUNCTION name ( {o , formal } )
+| ANY FUNCTION name ( )
     function.body
   :
 | {1 , data.type } FUNCTION name ( {o , formal } ) IS
expression.list :
 | specifier name RETYPES element :
 | VAL specifier name RETYPES expression :
+| block.definition

 delayed.input = timer ? AFTER expression

@@ -109,6 +121,7 @@

 expression = monadic.operator operand
 | operand dyadic.operator operand
+| andor.expression
 | conversion
 | operand
 | MOSTPOS data.type
@@ -207,6 +220,9 @@
 | allocation
   process

+program = process
+| block.definition.list
+
 protocol = name | simple.protocol | ANY

 real = digits.digits | digits.digitsEexponent

======
```

C. Changes between occlang4/CRI.txt and sed -e's/^;//' occlang4/OCCAMLIB.Y

Filename: occlang4/CRIdiffu.txt (20180607 11:42 AM)

```
======
--- CRI.txt    2018-05-29 10:12:12.000000000 -0700
+++ /dev/fd/63 2018-06-07 11:42:56.000000000 -0700
@@ -1,6 +1,29 @@
-
-program : process
-| block_definition_list
+%{
+#include <stdio.h>
+#include <string.h>
+#define YYDEBUG 1
+
+int yyerror(const char *str);
+%}
+
+%token '!' '&' '(' ')' '*' '+' ',' '-' '/'
+%token ':' ';' '<' '=' '>' '?' '[' '\\' ']' '~'
+%token AFTER ALT AND AND_OP ANY ASSIGN AT BITAND
+%token BITNOT BITOR BOOL BYTE BYTE_CONSTANT CASE
+%token CHAN DOUBLE_COLON ELSE FALSE FOR FROM FUNCTION
+%token GE_OP HEX_CONSTANT IDENTIFIER IF INDEF_SPEC INT
+%token INT16 INT32 INT64 INT_CONSTANT IS LEFT_OP LE_OP
+%token MINUS MOSTNEG MOSTPOS NEWLINE NE_OP NOT OF OR
+%token OR_OP PAR PLACE PLACED PLUS PORT PRI PROC
+%token PROCESSOR PROTOCOL REAL32 REAL64 REAL_CONSTANT
+%token REM RESULT RETYPES RIGHT_OP RIND ROUND ROUTD
+%token SEQ SIZE SKIP STOP STRING_LITERAL STRING_LITERAL_END
+%token STRING_LITERAL_MIDDLE STRING_LITERAL_START TIMER
+%token TIMES TRUE TRUNC VAL VALOF WHILE XOR_OP
+
+%glr-parser
+%%
+program : block_definition_list

 block_definition_list : block_definition
 | block_definition_list
@@ -61,8 +84,7 @@
 construction : sequence | conditional | selection | loop
 | parallel | alternation

-definition : PROTOCOL name IS simple_protocol ':' NEWLINE
-| PROTOCOL name IS sequential_protocol ':' NEWLINE
```

```
+definition : PROTOCOL name IS sequential_protocol ':' NEWLINE
 | PROTOCOL name NEWLINE
     RIND CASE NEWLINE
       RIND tagged_protocol_vlist ROUTD ROUTD
@@ -186,20 +208,22 @@

 instance : name '(' actual_olist ')' NEWLINE

-tagged_protocol : tag | tag ';' sequential_protocol NEWLINE
+tagged_protocol : scalar | scalar ';' sequential_protocol
NEWLINE

 tagged_protocol_vlist : tagged_protocol
 | tagged_protocol_vlist
   tagged_protocol

-actual : element | expression
+actual : element %dprec 2
+ {printf("<[< actual from element, not expression >]>\n");}
+| expression     %dprec 1
+ {printf("<[< actual from expression, not element >]>\n");}

 actual_clist : actual | actual_clist ',' actual

 actual_olist :  | actual_clist

-assignment : variable ASSIGN expression
-| variable_list ASSIGN expression_list
+assignment : variable_clist ASSIGN expression_list

 base : expression

@@ -211,10 +235,12 @@

 channel : element

-delayed_input : timer '?' AFTER expression
+delayed_input : channel '?' AFTER expression

-expression_list : name '(' expression_olist ')'
-| expression_clist
+expression_list : name '(' expression_olist ')' %dprec 2
+ {printf("<[< expressipn_list from FUNCTION name, not
expression_clist >]>\n");}
+| expression_clist                              %dprec 1
```

```
+ {printf("<[< expression_list from expression_clist, not
FUNCTION name >]>\n");}

 formal : specifier name_clist | VAL specifier name_clist

@@ -222,31 +248,27 @@

 formal_olist :  | formal_clist

-input : channel '?' variable
-| channel '?' input_item
-| channel '?' input_item_slist
+indef_type : INDEF_SPEC type
+| INDEF_SPEC indef_type
+| '[' expression ']' indef_type
+
+input : channel '?' input_item_slist
 | channel '?' CASE tagged_list
-| timer_input
 | delayed_input
-| port '?' variable

 input_item : variable | variable DOUBLE_COLON variable

 input_item_slist : input_item | input_item_slist ';' input_item

-output : channel '!' expression
-| channel '!' output_item
-| channel '!' output_item_slist
-| channel '!' tag
-| channel '!' tag ';' output_item_slist
-| port '!' expression
+output : channel '!' output_item_slist      %dprec 1
+ {printf("<[< output from slist, not scalar>]>\n");}
+| channel '!' scalar ';' output_item_slist  %dprec 2
+ {printf("<[< output from scalar, not slist >]>\n");}

 output_item : expression | expression DOUBLE_COLON expression

 output_item_slist : output_item | output_item_slist ';'
output_item

-port : element
-
 replicator :  name '=' base FOR count
 selector : expression
@@ -255,22 +277,14 @@

 simple_protocol_slist : simple_protocol | simple_protocol_slist
';' simple_protocol

-specifier : primitive_type
-| INDEF_SPEC specifier
-| '[' expression ']' specifier
-
-tagged_list : tag | tag ';' input_item_slist
+specifier : type | indef_type

-timer : element
-
-timer_input : timer '?' variable
+tagged_list : scalar | scalar ';' input_item_slist

 variable : element

 variable_clist : variable | variable_clist ',' variable

-variable_list : variable_clist
-
 andor_expression : andor_expression andor_operator operand
 | operand andor_operator operand

@@ -288,7 +302,7 @@

 element : element '[' subscript ']'
 | '[' element FROM subscript FOR subscript ']'
-| name
+| scalar

 expression : monadic_operator operand
 | operand dyadic_operator operand
@@ -319,6 +333,7 @@
 | '[' table FROM subscript FOR count ']'

 type : primitive_type | array_type
+
 andor_operator : AND | OR

 byte : BYTE_CONSTANT
@@ -358,9 +373,26 @@
```

```
 real : REAL_CONSTANT

+scalar : name
+
 string : STRING_LITERAL | string_head STRING_LITERAL_END

 string_head : STRING_LITERAL_START | string_head
STRING_LITERAL_MIDDLE

-tag : name

+%%
+
+int yyerror(const char *str) {
+  fprintf(stderr, "error: %s\n", str);
+  return 1;
+}
+
+int yywrap() {
+  return 1;
+}
+
+int main() {
+  extern int yydebug;
+  yydebug = 1;
+  yyparse();
+}

======
```

D. Changes between occlang4/OCCAMLIB.Y and occlang5/OCCAMLIB.Y

Filename: occlang5/diffu-occlang4-occlang5-OCCAMLIBY.txt (20180706 10:03 AM)

```
======
--- ../occlang4/OCCAMLIB.Y    2018-06-06 16:45:31.000000000
-0700
+++ OCCAMLIB.Y 2018-06-28 10:29:41.000000000 -0700
@@ -208,7 +208,8 @@
 ;
 instance : name '(' actual_olist ')' NEWLINE
 ;
-tagged_protocol : scalar | scalar ';' sequential_protocol
NEWLINE
+tagged_protocol : scalar NEWLINE
```

```
+| scalar ';' sequential_protocol NEWLINE
 ;
 tagged_protocol_vlist : tagged_protocol
 | tagged_protocol_vlist
======
```

Appendix 4. Test files

The first file is genuine, compilable occam. The others are parser-occam and may not be compilable because of stuff left out for simplicity of output. All are tested and work with `./OCCAMLIB-COMPILER` . File text is enclosed by lines ====== .

```
-rw-r--r--  1 tjoccam  staff  828 May 22 10:25 childcs.occ
======
#INCLUDE "hostio.inc" --  -- contains SP protocol
--{{{  PROC waitandtrigger(CHAN OF SP fmas, tmas)
PROC waitandtrigger(CHAN OF SP fmas,
  tmas)
  INT thewait,
    thetime:
  [2]INT thewaits :
  VAL INT thesize IS SIZE "Beware the jabberwock my son,* -- Start
                          * the jaws that bite,* -- Example of middle
                          * the claws that catch." : -- Example of end
  TIMER clock:
  INT16 len:
  SEQ
    SEQ j = 0 FOR 2
      SEQ
        []BYTE thewaitbytes RETYPES thewait:
        fmas ? len::thewaitbytes
        thewaits[j] :=
          thewait
    SEQ j = 0 FOR 2
      SEQ
        thewait := thewaits[j]
        clock ? thetime
        clock ? AFTER (thetime PLUS thewait)
        []BYTE thewaitbytes RETYPES thewait:
        tmas ! len::thewaitbytes
:
--}}}
======
```

```
-rw-r--r--  1 tjoccam  staff  92 May 31 14:52 small.occ
======
INT FUNCTION add(VAL INT a, VAL INT b)
```

```
  INT c:
  VALOF
    c := a + b
    RESULT c
:
======

-rw-r--r--  1 tjoccam  staff  84 May 31 15:33 small2.occ
======
INT FUNCTION add(VAL INT a, b)
  INT c:
  VALOF
    c := a + b
    RESULT c
:
======

-rw-r--r--  1 tjoccam  staff  142 Jun  4 15:00 small3.occ
======
INT FUNCTION awk(VAL INT a, VAL INT b)
  INT c, d, e:
  VALOF
    SEQ
      c, d := a, sin(b)
      c, e := quot(a, b)
    RESULT c
:
======

-rw-r--r--  1 tjoccam  staff  82 Jun  6 09:02 small3b.occ
======
REAL32 FUNCTION awk(REAL32 a, REAL32 b)
  VALOF
    SKIP
    RESULT sin(b)
:
======

-rw-r--r--  1 tjoccam  staff  77 Jun  6 09:08 small3c.occ
======
REAL32 FUNCTION awk(REAL32 a, b)
  VALOF
    SKIP
    RESULT a+sin(b)
:
======
-rw-r--r--  1 tjoccam  staff  135 Jun  6 09:25 small3d.occ
======
REAL32 FUNCTION awk(REAL32 a, REAL32 b)
  REAL32 FUNCTION whistle(REAL32 x) IS sin(x) :
  VALOF
    SKIP
    RESULT whistle(b)
:
======

-rw-r--r--  1 tjoccam  staff  58 Jun  4 17:49 small4.occ
======
PROC awl(CHAN OF ANY a, b)
  INT c, d :
  a ! c ; d
:
======

-rw-r--r--  1 tjoccam  staff  60 Jun  6 17:09 small4b.occ
======
PROC awl(CHAN OF ANY a, b)
  INT c, d :
  a ! c+d ; d
:
======

-rw-r--r--  1 tjoccam  staff  78 Jun  6 09:34 small5.occ
======
INT FUNCTION awk(VAL INT a, VAL INT b)
  VALOF
    yaw(a)
    RESULT b
:
======

-rw-r--r--  1 tjoccam  staff  80 Jun  6 16:58 small5b.occ
======
INT FUNCTION awk(VAL INT a, VAL INT b)
  VALOF
    yaw(a+b)
    RESULT b
:
======

-rw-r--r--  1 tjoccam  staff  91 Jun  7 16:56 small6.occ
```

```
======
PROC awl(CHAN OF ANY a)
  INT c :
  []BYTE d RETYPES c:
  BYTE e IS d[0] :
  a ! e
:
======

-rw-r--r--  1 tjoccam  staff  140 Jun  7 17:53 small7.occ
======
PROC awl(CHAN OF ANY a, b)
  INT c, d :
  SEQ
    ALT
      BOOL c & a ? d
        SKIP
      BOOL e :
      b ? e
        SKIP
:
======

-rw-r--r--  1 tjoccam  staff  155 Jun  7 18:42 small8.occ
======
PROC awl(CHAN OF ANY a, b)
  INT d :
  BOOL e :
  SEQ
    ALT
      [e][0] & a ? d
        SKIP
      [d][1]BOOL c :
      b ? c
        SKIP
:
======
```

Appendix 5. Parser source files

Following are complete copies of the final source files OCCAMLIB.L and OCCAMLIB.Y. File text is enclosed by lines ====== .

A. OCCAMLIB.L (lex source)

```
-rw-r--r--  1 tjoccam  staff  15806 May 31 15:45 OCCAMLIB.L
======
%{
/* A Lex program for OCCAM language.
 * Stripped down. LJD 20180531
*/
#include <stdio.h>
#include "y.tab.h"
#define YY_SKIP_YYWRAP
#define yywrap() 1
int lineno = 1;
int indent = 0;
int cntinu = 0; /* next is continuation line if at end */
int wasindent = -1;
int realindent = -1;
int initialindent = -1;
void count();
%}

/* sub-parser state A normal, N nesting change */
%s N
%s A

strippedline            [^\r\n]+
D                       [0-9]
L                       [a-zA-Z]
H                       [A-F0-9]
E                       [Ee][+-]?{D}+
S                       [CcNnTtSs'"*]
%%
<INITIAL>" "*#" "*INCLUDE[^\r\n]* { indent = strspn(yytext, " ");
                                    /*
                                    printf("INDENT %d, LINENO %d ", indent, lineno);
                                    printf(" directive: %s\n", yytext+indent);
                                    */
                                    BEGIN(A); }
<INITIAL>" "*#" "*USE[^\r\n]* { indent = strspn(yytext, " ");
                                    /*
                                    printf("INDENT %d, LINENO %d ", indent, lineno);
                                    printf(" directive: %s\n", yytext+indent);
                                    */
                                    BEGIN(A); }
<INITIAL>" "*#" "*IMPORT[^\r\n]* { indent = strspn(yytext, " ");
                                    /*
                                    printf("INDENT %d, LINENO %d ", indent, lineno);
                                    printf(" directive: %s\n", yytext+indent);
                                    */
                                    BEGIN(A); }
<INITIAL>" "*#" "*COMMENT[^\r\n]* { indent = strspn(yytext, " ");
                                    /*
                                    printf("INDENT %d, LINENO %d ", indent, lineno);
                                    printf(" directive: %s\n", yytext+indent);
                                    */
                                    BEGIN(A); }
<INITIAL>" "*#" "*OPTION[^\r\n]* { indent = strspn(yytext, " ");
                                    /*
                                    printf("INDENT %d, LINENO %d ", indent, lineno);
                                    printf(" directive: %s\n", yytext+indent);
                                    */
                                    BEGIN(A); }
<INITIAL>" "*#" "*PRAGMA[^\r\n]* { indent = strspn(yytext, " ");
                                    /*
                                    printf("INDENT %d, LINENO %d ", indent, lineno);
                                    printf(" directive: %s\n", yytext+indent);
                                    */
                                    BEGIN(A); }
<INITIAL>" "*--[^\r\n]* { indent = strspn(yytext, " ");
                          /*
                          printf("INDENT %d, LINENO %d ", indent, lineno);
                          printf(" full-line-comment: %s\n", yytext+indent);
                          */
                          BEGIN(A); }
<INITIAL>" "*[*]([*]{S}|[^*\r\n]|[*]#{H}{H})*[*] { cntinu = 1;
                                    indent = strspn(yytext, " ");
                                    /* printf("INDENT %d, LINENO %d ", indent,
lineno); */
                                    printf("STRING_LITERAL_MIDDLE\t\t%s\n", yytext
+indent);
                                    count(NULL);
                                    BEGIN(A); return STRING_LITERAL_MIDDLE; }
<INITIAL>" "*[*]([*]{S}|[^*\r\n]|[*]#{H}{H})*["] { cntinu = 0;
                                    indent = strspn(yytext, " ");
                                    /* printf("INDENT %d, LINENO %d ", indent,
lineno); */
                                    printf("STRING_LITERAL_END\t\t%s\n", yytext
+indent);
                                    count(NULL);
                                    BEGIN(A); return STRING_LITERAL_END; }
<INITIAL>" " { indent++; }
<INITIAL><<EOF>> {
                   if (realindent >= 0) {
                     int delindent = initialindent - realindent;
                     printf("NEWLINE\n");
                     if (delindent&1) {
                       printf("!!!ERROR - ODD DEL INDENTATION %d", delindent);
                       yyterminate();
                     } else {
                       BEGIN(N);
                       return NEWLINE;
                     }
                   } else {
                     yyterminate();
                   }
                 }
<INITIAL>. {
             int delindent = 0;
             int wasnewline = 0;
             if (!cntinu) {
               if (realindent >= 0) {
                 wasindent = realindent;
                 realindent = indent;
                 delindent = realindent - wasindent;
                 printf("NEWLINE\n");
                 wasnewline = 1;
                 if (delindent&1) {
                   printf("!!!ERROR - ODD DEL INDENTATION %d", delindent);
                   yyterminate();
                 } else if (delindent) {
                   BEGIN(N);
                 } else {
                   /*
                   printf("\n");
                   printf("INDENT %d, LINENO %d\n", indent, lineno);
                   */
                 }
               } else {
                 initialindent = indent;
                 realindent = indent;
                 /* printf("INDENT %d, LINENO %d\n", indent, lineno); */
               }
             } else {
               /* printf("INDENT %d, LINENO %d\n", indent, lineno); */
             }
             if (*yytext) unput(*yytext);
             if (!delindent) {
               BEGIN(A);
             }
             if (wasnewline) return NEWLINE;
           }

<N><<EOF>> { if (realindent >= 0) {
               int delindent = initialindent - realindent;
               if (delindent&1) {
                 printf("!!!ERROR - ODD DEL INDENTATION %d", delindent);
                 yyterminate();
               } else if (delindent>0) {
                 printf("RIND\n"); realindent += 2;
                 return RIND;
               } else if (delindent<0) {
                 printf("ROUTD\n"); realindent -= 2;
                 return ROUTD;
               } else {
                 printf("ENDFILE\n");
                 yyterminate();
               }
             } else {
               yyterminate();
             }
           }
<N>. { int delindent = 0;
       int wasrind = 0;
       int wasroutd = 0;
       if (!cntinu) {
         if (realindent >= 0) {
           delindent = realindent - wasindent;
           if (delindent&1) {
             printf("!!!ERROR - ODD DEL INDENTATION %d", delindent);
             yyterminate();
           } else if (delindent>0) {
             printf("RIND\n"); wasindent += 2;
             wasrind = 1;
           } else if (delindent<0) {
             printf("ROUTD\n"); wasindent -= 2;
             wasroutd = 1;
           } else {
             /*
             printf("\n");
             printf("INDENT %d, LINENO %d\n", indent, lineno);
             */
           }
         }
       }
       if (*yytext) unput(*yytext);
       if (!delindent) {
         BEGIN(A);
       }
       if (wasrind) return RIND;
       if (wasroutd) return ROUTD;
     }

<A>--[^\r\n]*                    { count(NULL); }
<A>"AFTER"                  { cntinu = 1; count("");  return AFTER; }
<A>"ALT"                  { cntinu = 0; count("");  return ALT; }
<A>"AND"                  { cntinu = 1; count("");  return AND; }
<A>"ANY"                  { cntinu = 0; count("");  return ANY; }
<A>"AT"                  { cntinu = 0; count("");  return AT; }
<A>"BITAND"                  { cntinu = 1; count("");  return BITAND; }
<A>"BITNOT"                  { cntinu = 1; count("");  return BITNOT; }
<A>"BITOR"                  { cntinu = 1; count("");  return BITOR; }
<A>"BOOL"                  { cntinu = 0; count("");  return BOOL; }
<A>"BYTE"                  { cntinu = 0; count("");  return BYTE; }
<A>"CASE"                  { cntinu = 0; count("");  return CASE; }
<A>"CHAN"                  { cntinu = 0; count("");  return CHAN; }
<A>"ELSE"                  { cntinu = 0; count("");  return ELSE; }
<A>"FALSE"                  { cntinu = 0; count("");  return FALSE; }
<A>"FOR"                  { cntinu = 1; count("");  return FOR; }
<A>"FROM"                  { cntinu = 1; count("");  return FROM; }
<A>"FUNCTION"                  { cntinu = 0; count("");  return FUNCTION; }
<A>"IF"                  { cntinu = 0; count("");  return IF; }
<A>"IS"                  { cntinu = 1; count("");  return IS; }
<A>"INT"                  { cntinu = 0; count("");  return INT; }
<A>"INT16"                  { cntinu = 0; count("");  return INT16; }
<A>"INT32"                  { cntinu = 0; count("");  return INT32; }
<A>"INT64"                  { cntinu = 0; count("");  return INT64; }
<A>"MINUS"                  { cntinu = 1; count("");  return MINUS; }
<A>"MOSTNEG"                  { cntinu = 0; count("");  return MOSTNEG; }
<A>"MOSTPOS"                  { cntinu = 0; count("");  return MOSTPOS; }
<A>"NOT"                  { cntinu = 1; count("");  return NOT; }
<A>"OF"                  { cntinu = 0; count("");  return OF; }
<A>"OR"                  { cntinu = 1; count("");  return OR; }
<A>"PAR"                  { cntinu = 0; count("");  return PAR; }
<A>"PLACE"                  { cntinu = 0; count("");  return PLACE; }
<A>"PLACED"                  { cntinu = 0; count("");  return PLACED; }
<A>"PLUS"                  { cntinu = 1; count("");  return PLUS; }
<A>"PORT"                  { cntinu = 0; count("");  return PORT; }
<A>"PRI"                  { cntinu = 0; count("");  return PRI; }
<A>"PROC"                  { cntinu = 0; count("");  return PROC; }
<A>"PROCESSOR"                  { cntinu = 0; count("");  return PROCESSOR; }
<A>"PROTOCOL"                  { cntinu = 0; count("");  return PROTOCOL; }
<A>"REAL32"                  { cntinu = 0; count("");  return REAL32; }
<A>"REAL64"                  { cntinu = 0; count("");  return REAL64; }
<A>"REM"                  { cntinu = 1; count("");  return REM; }
```

```
<A>"RESULT"                  { cntinu = 0; count("");  return RESULT; }
<A>"RETYPES"                  { cntinu = 0; count("");  return RETYPES; }
<A>"ROUND"                  { cntinu = 0; count("");  return ROUND; }
<A>"SEQ"                  { cntinu = 0; count("");  return SEQ; }
<A>"SIZE"                  { cntinu = 1; count("");  return SIZE; }
<A>"SKIP"                  { cntinu = 0; count("");  return SKIP; }
<A>"STOP"                  { cntinu = 0; count("");  return STOP; }
<A>"TIMER"                  { cntinu = 0; count("");  return TIMER; }
<A>"TIMES"                  { cntinu = 1; count("");  return TIMES; }
<A>"TRUE"                  { cntinu = 0; count("");  return TRUE; }
<A>"TRUNC"                  { cntinu = 0; count("");  return TRUNC; }
<A>"VAL"                  { cntinu = 0; count("");  return VAL; }
<A>"VALOF"                  { cntinu = 0; count("");  return VALOF; }
<A>"WHILE"                  { cntinu = 0; count("");  return WHILE; }
<A>{L}({L}|{D}|[.])*           { cntinu = 0; count("IDENTIFIER");
                                 return IDENTIFIER; }
<A>#{H}+          { cntinu = 0; count("HEX_CONSTANT"); return HEX_CONSTANT; }
<A>{D}+               { cntinu = 0; count("INT_CONSTANT");
                        return INT_CONSTANT; }
<A>'([*]{S}|[^*\r\n]|[*]#{H}{H})'       { cntinu = 0; count("BYTE_CONSTANT");
                                          return BYTE_CONSTANT; }
<A>{D}*"."{D}+({E})?  { cntinu = 0; count("REAL_CONSTANT");
                                          return REAL_CONSTANT; }
<A>{D}+"."{D}*({E})?  { cntinu = 0; count("REAL_CONSTANT");
                                          return REAL_CONSTANT; }
<A>["]([*]{S}|[^*\r\n]|[*]#{H}{H})*["] { cntinu = 0; count("STRING_LITERAL");
                                          return STRING_LITERAL; }
<A>["]([*]{S}|[^*\r\n]|[*]#{H}{H})*[*] { cntinu = 1;
                                         count("STRING_LITERAL_START");
                                         return STRING_LITERAL_START; }
<A>":="                    { cntinu = 1; count("ASSIGN");
                             return ASSIGN; }
<A>">>"                    { cntinu = 1; count("RIGHT_OP");
                             return RIGHT_OP; }
<A>"<<"                    { cntinu = 1; count("LEFT_OP");
                             return LEFT_OP; }
<A>"/\\"                    { cntinu = 1; count("AND_OP");
                             return AND_OP; }
<A>"\\/"                    { cntinu = 1; count("OR_OP");
                             return OR_OP; }
<A>"<="                    { cntinu = 1; count("LE_OP");
                             return LE_OP; }
<A>">="                    { cntinu = 1; count("GE_OP");
                             return GE_OP; }
<A>"><"                    { cntinu = 1; count("XOR_OP");
                             return XOR_OP; }
<A>"<>"                    { cntinu = 1; count("NE_OP");
                             return NE_OP; }
<A>"[]"              { cntinu = 0; count("INDEF_SPEC");
                             return INDEF_SPEC; }
<A>"::"                     { cntinu = 0; count("DOUBLE_COLON");
                             return DOUBLE_COLON; }
<A>[=]                    { cntinu = 1; count("\'=\'");  return '='; }
<A>";"                     { cntinu = 1; count("\';\'");  return ';'; }
<A>"["              { cntinu = 0; count("\'[\'");  return '['; }
<A>"]"              { cntinu = 0; count("\']\'");  return ']'; }
<A>","                     { cntinu = 1; count("\',\'");  return ','; }
<A>":"                     { cntinu = 0; count("\':\'");  return ':'; }
```

```
<A>"!"                     { cntinu = 0; count("\'!\'");  return '!'; }
<A>"?"                     { cntinu = 0; count("\'?\'");  return '?'; }
<A>"("                     { cntinu = 0; count("\'(\'");  return '('; }
<A>")"                     { cntinu = 0; count("\')\'");  return ')'; }
<A>"."                     { cntinu = 0; count("\'.\'");  return '.'; }
<A>"&"                     { cntinu = 0; count("\'&\'");  return '&'; }
<A>"~"                     { cntinu = 1; count("\'~\'");  return '~'; }
<A>"-"                     { cntinu = 1; count("\'-\'");  return '-'; }
<A>"+"                     { cntinu = 1; count("\'+\'");  return '+'; }
<A>"*"                     { cntinu = 1; count("\'*\'");  return '*'; }
<A>"/"                     { cntinu = 1; count("\'/\'");  return '/'; }
<A>"\\"                     { cntinu = 1; count("\'\\\'");  return '\\'; }
<A>"<"                     { cntinu = 1; count("\'<\'");  return '<'; }
<A>">"                     { cntinu = 1; count("\'>\'");  return '>'; }
<A>"#"                     { cntinu = 0; count("\'#\'");  return '#'; }
<A>[ \t\v\f]             { count(NULL); }
<A>[\r\n]+ {
             lineno++;
             indent = 0;
             /* if (cntinu) printf("CONTINUATION "); */
             BEGIN(INITIAL);
           }
<A>.                         ;
%%
int column = 0;

void count(const char *name)
{
        int i;

        for (i = 0; yytext[i] != '\0'; i++)
                if (yytext[i] == '\n')
                        column = 0;
                else if (yytext[i] == '\t')
                        column += 8 - (column % 8);
                else
                        column++;

        if (name) {
          if (*name) printf("%s\t\t", name);
          ECHO;
          printf("\n");
        }
}
======
```

B. OCCAMLIB.Y (yacc source)

```
-rw-r--r--  1 tjoccam  staff  9567 Jun 28 10:29 OCCAMLIB.Y
======
%{
#include <stdio.h>
#include <string.h>
#define YYDEBUG 1

int yyerror(const char *str);
%}
```

```
%token '!' '&' '(' ')' '*' '+' ',' '-' '/'
%token ':' ';' '<' '=' '>' '?' '[' '\\' ']' '~'
%token AFTER ALT AND AND_OP ANY ASSIGN AT BITAND
%token BITNOT BITOR BOOL BYTE BYTE_CONSTANT CASE
%token CHAN DOUBLE_COLON ELSE FALSE FOR FROM FUNCTION
%token GE_OP HEX_CONSTANT IDENTIFIER IF INDEF_SPEC INT
%token INT16 INT32 INT64 INT_CONSTANT IS LEFT_OP LE_OP
%token MINUS MOSTNEG MOSTPOS NEWLINE NE_OP NOT OF OR
%token OR_OP PAR PLACE PLACED PLUS PORT PRI PROC
%token PROCESSOR PROTOCOL REAL32 REAL64 REAL_CONSTANT
%token REM RESULT RETYPES RIGHT_OP RIND ROUND ROUTD
%token SEQ SIZE SKIP STOP STRING_LITERAL STRING_LITERAL_END
%token STRING_LITERAL_MIDDLE STRING_LITERAL_START TIMER
%token TIMES TRUE TRUNC VAL VALOF WHILE XOR_OP

%glr-parser
%%
program : block_definition_list
;
block_definition_list : block_definition
| block_definition_list
  block_definition
;
alternation : ALT NEWLINE
                RIND alternative_vlist ROUTD
| ALT NEWLINE
| ALT replicator NEWLINE
    RIND alternative ROUTD
| PRI ALT NEWLINE
    RIND alternative_vlist ROUTD
| PRI ALT NEWLINE
| PRI ALT replicator NEWLINE
    RIND alternative ROUTD
;
alternative : guarded_alternative
| alternation
| specification
  alternative
| channel '?' CASE NEWLINE
    RIND variant_vlist ROUTD
| channel '?' CASE NEWLINE
| boolean '&' channel '?' CASE NEWLINE
    RIND variant_vlist ROUTD
| boolean '&' channel '?' CASE NEWLINE
;
alternative_vlist : alternative
| alternative_vlist
  alternative
;
block_definition : PROC name '(' formal_olist ')' NEWLINE
                     RIND procedure_body ROUTD
                   ':' NEWLINE
| data_type_clist FUNCTION name '(' formal_olist ')' NEWLINE
    RIND function_body ROUTD
  ':' NEWLINE
;
case_input : channel '?' CASE NEWLINE
               RIND variant_vlist ROUTD
```

```
| channel '?' CASE NEWLINE
;
choice : guarded_choice
| conditional
| specification
  choice
;
choice_vlist : choice
| choice_vlist
  choice
;
conditional : IF NEWLINE
                RIND choice_vlist ROUTD
| IF NEWLINE
| IF replicator NEWLINE
    RIND choice ROUTD
;
construction : sequence | conditional | selection | loop
| parallel | alternation
;
definition : PROTOCOL name IS sequential_protocol ':' NEWLINE
| PROTOCOL name NEWLINE
    RIND CASE NEWLINE
      RIND tagged_protocol_vlist ROUTD ROUTD
  ':' NEWLINE
| PROTOCOL name NEWLINE
    RIND CASE NEWLINE ROUTD
  ':' NEWLINE
| ANY FUNCTION name '(' ')' NEWLINE
    RIND function_body ROUTD
  ':' NEWLINE
| data_type_clist FUNCTION name '(' formal_olist ')' IS expression_list ':' NEWLINE
| specifier name RETYPES element ':' NEWLINE
| VAL specifier name RETYPES expression ':' NEWLINE
| block_definition
;
function_body : value_process
;
guarded_alternative : guard
                        RIND process ROUTD
;
guarded_choice : boolean NEWLINE
                   RIND process ROUTD
;
loop : WHILE boolean NEWLINE
         RIND process ROUTD
;
option : case_expression_clist NEWLINE
           RIND process ROUTD
| ELSE NEWLINE
    RIND process ROUTD
| specification
  option
;
option_vlist : option
| option_vlist
  option
;
```

```
parallel : PAR NEWLINE
             RIND process_vlist ROUTD
| PAR NEWLINE
| PAR replicator NEWLINE
    RIND process ROUTD
| PRI PAR NEWLINE
    RIND process_vlist ROUTD
| PRI PAR NEWLINE
| PRI PAR replicator NEWLINE
    RIND process ROUTD
| placedpar
;
placedpar : PLACED PAR NEWLINE
              RIND placedpar_vlist ROUTD
| PLACED PAR NEWLINE
| PLACED PAR replicator NEWLINE
    RIND placedpar ROUTD
| PROCESSOR expression NEWLINE
    RIND process ROUTD
;
placedpar_vlist : placedpar
| placedpar_vlist
  placedpar
;
procedure_body : process
;
process : SKIP NEWLINE
| STOP NEWLINE
| action | construction | instance | case_input
| specification
  process
| allocation
  process
;
process_vlist : process
| process_vlist
  process
;
selection : CASE selector NEWLINE
              RIND option_vlist ROUTD
| CASE selector NEWLINE
;
sequence : SEQ NEWLINE
             RIND process_vlist ROUTD
| SEQ NEWLINE
| SEQ replicator NEWLINE
    RIND process ROUTD
;
specification : declaration | abbreviation | definition
;
valof : VALOF NEWLINE
          RIND process
          RESULT expression_list NEWLINE ROUTD
| specification
  valof
;
value_process : valof
;
```

```
variant : tagged_list NEWLINE
            RIND process ROUTD
| specification
  variant
;
variant_vlist : variant
| variant_vlist
  variant
;
abbreviation : specifier name IS element ':' NEWLINE
| name IS element ':' NEWLINE
| VAL specifier name IS expression ':' NEWLINE
| VAL name IS expression ':' NEWLINE
;
action : assignment NEWLINE
| input NEWLINE
| output NEWLINE
;
allocation : PLACE name AT expression ':' NEWLINE
;
declaration : type name_clist ':' NEWLINE
;
guard : input NEWLINE
| boolean '&' input NEWLINE
| boolean '&' SKIP NEWLINE
;
instance : name '(' actual_olist ')' NEWLINE
;
tagged_protocol : scalar NEWLINE
| scalar ';' sequential_protocol NEWLINE
;
tagged_protocol_vlist : tagged_protocol
| tagged_protocol_vlist
  tagged_protocol
;
actual : element %dprec 2
 {printf("<[< actual from element, not expression >]>\n");}
| expression   %dprec 1
 {printf("<[< actual from expression, not element >]>\n");}
;
actual_clist : actual | actual_clist ',' actual
;
actual_olist :  | actual_clist
;
assignment : variable_clist ASSIGN expression_list
;
base : expression
;
boolean :  expression
;
case_expression : expression
;
case_expression_clist : case_expression | case_expression_clist ',' case_expression
;
channel : element
;
delayed_input : channel '?' AFTER expression
;
```

```
expression_list : name '(' expression_olist ')' %dprec 2
 {printf("<[< expressipn_list from FUNCTION name, not expression_clist >]>\n");}
| expression_clist                              %dprec 1
 {printf("<[< expression_list from expression_clist, not FUNCTION name >]>\n");}
;
formal : specifier name_clist | VAL specifier name_clist
;
formal_clist : formal | formal_clist ',' formal
;
formal_olist :  | formal_clist
;
indef_type : INDEF_SPEC type
| INDEF_SPEC indef_type
| '[' expression ']' indef_type
;
input : channel '?' input_item_slist
| channel '?' CASE tagged_list
| delayed_input
;
input_item : variable | variable DOUBLE_COLON variable
;
input_item_slist : input_item | input_item_slist ';' input_item
;
output : channel '!' output_item_slist      %dprec 1
 {printf("<[< output from slist, not scalar>]>\n");}
| channel '!' scalar ';' output_item_slist  %dprec 2
 {printf("<[< output from scalar, not slist >]>\n");}
;
output_item : expression | expression DOUBLE_COLON expression
;
output_item_slist : output_item | output_item_slist ';' output_item
;
replicator :  name '=' base FOR count
;
selector : expression
;
sequential_protocol : simple_protocol_slist
;
simple_protocol_slist : simple_protocol | simple_protocol_slist ';' simple_protocol
;
specifier : type | indef_type
;
tagged_list : scalar | scalar ';' input_item_slist
;
variable : element
;
variable_clist : variable | variable_clist ',' variable
;
andor_expression : andor_expression andor_operator operand
| operand andor_operator operand
;
array_type : '[' expression ']' type
;
comms_type : CHAN OF protocol
| TIMER
| PORT OF type
;
conversion : primitive_type operand
```

```
| primitive_type ROUND operand
| primitive_type TRUNC operand
;
count : expression
;
element : element '[' subscript ']'
| '[' element FROM subscript FOR subscript ']'
| scalar
;
expression : monadic_operator operand
| operand dyadic_operator operand
| andor_expression
| conversion
| operand
| MOSTPOS data_type
| MOSTNEG data_type
;
expression_clist : expression | expression_clist ',' expression
;
expression_olist :  | expression_clist
;
operand : element | literal | table | '(' expression ')'
| name '(' expression_olist ')'
;
primitive_type : comms_type
| data_type
;
protocol : name | simple_protocol | ANY
;
simple_protocol : type | counting_type DOUBLE_COLON INDEF_SPEC type
;
subscript :  expression
;
table : table '[' subscript ']'
| '[' expression_clist ']'
| '[' table FROM subscript FOR count ']'
;
type : primitive_type | array_type
;
andor_operator : AND | OR
;
byte : BYTE_CONSTANT
;
counting_type : BYTE
| INT
| INT16
| INT32
| INT64
;
data_type : counting_type
| BOOL
| REAL32
| REAL64
;
data_type_clist : data_type | data_type_clist ',' data_type
;
dyadic_operator : AFTER | BITAND | BITOR | MINUS | PLUS | REM
| TIMES | RIGHT_OP | LEFT_OP | AND_OP | OR_OP | LE_OP | GE_OP
| XOR_OP | NE_OP | '-' | '+' | '*' | '/' | '\\' | '<' | '>'
;
integer : INT_CONSTANT | HEX_CONSTANT
;
literal : integer
| byte
| integer '(' data_type ')'
| byte '(' data_type ')'
| real '(' data_type ')'
| string
| TRUE | FALSE
;
monadic_operator : BITNOT | MINUS | NOT | SIZE | '~' | '-'
;
name : IDENTIFIER
;
name_clist : name | name_clist ',' name
;
real : REAL_CONSTANT
;
scalar : name
;
string : STRING_LITERAL | string_head STRING_LITERAL_END
;
string_head : STRING_LITERAL_START | string_head STRING_LITERAL_MIDDLE
;

%%

int yyerror(const char *str) {
  fprintf(stderr, "error: %s\n", str);
  return 1;
}

int yywrap() {
  return 1;
}

int main() {
  extern int yydebug;
  yydebug = 1;
  yyparse();
}
======
```

C. CRI.TXT (BNF)

```
-rw-r--r--  1 tjoccam  staff  8011 Jul  5 17:02 CRI.TXT
======
program = block_definition_list

block_definition_list = block_definition
| block_definition_list
  block_definition

alternation = ALT NEWLINE
                RIND alternative_vlist ROUTD
| ALT NEWLINE
| ALT replicator NEWLINE
    RIND alternative ROUTD
| PRI ALT NEWLINE
    RIND alternative_vlist ROUTD
| PRI ALT NEWLINE
| PRI ALT replicator NEWLINE
    RIND alternative ROUTD

alternative = guarded_alternative
| alternation
| specification
  alternative
| channel '?' CASE NEWLINE
    RIND variant_vlist ROUTD
| channel '?' CASE NEWLINE
| boolean '&' channel '?' CASE NEWLINE
    RIND variant_vlist ROUTD
| boolean '&' channel '?' CASE NEWLINE

alternative_vlist = alternative
| alternative_vlist
  alternative

block_definition = PROC name '(' formal_olist ')' NEWLINE
                    RIND procedure_body ROUTD
                   ':' NEWLINE
| data_type_clist FUNCTION name '(' formal_olist ')' NEWLINE
    RIND function_body ROUTD
  ':' NEWLINE

case_input = channel '?' CASE NEWLINE
               RIND variant_vlist ROUTD
| channel '?' CASE NEWLINE

choice = guarded_choice
| conditional
| specification
  choice

choice_vlist = choice
| choice_vlist
  choice

conditional = IF NEWLINE
                RIND choice_vlist ROUTD
| IF NEWLINE
| IF replicator NEWLINE
    RIND choice ROUTD

construction = sequence | conditional | selection | loop
| parallel | alternation

definition = PROTOCOL name IS sequential_protocol ':' NEWLINE
| PROTOCOL name NEWLINE
    RIND CASE NEWLINE
      RIND tagged_protocol_vlist ROUTD ROUTD
  ':' NEWLINE
| PROTOCOL name NEWLINE
```

```
    RIND CASE NEWLINE ROUTD
  ':' NEWLINE
| ANY FUNCTION name '(' ')' NEWLINE
    RIND function_body ROUTD
  ':' NEWLINE
| data_type_clist FUNCTION name '(' formal_olist ')' IS expression_list ':' NEWLINE
| specifier name RETYPES element ':' NEWLINE
| VAL specifier name RETYPES expression ':' NEWLINE
| block_definition

function_body = value_process

guarded_alternative = guard
                        RIND process ROUTD

guarded_choice = boolean NEWLINE
                   RIND process ROUTD

loop = WHILE boolean NEWLINE
         RIND process ROUTD

option = case_expression_clist NEWLINE
           RIND process ROUTD
| ELSE NEWLINE
    RIND process ROUTD
| specification
  option

option_vlist = option
| option_vlist
  option

parallel = PAR NEWLINE
             RIND process_vlist ROUTD
| PAR NEWLINE
| PAR replicator NEWLINE
    RIND process ROUTD
| PRI PAR NEWLINE
    RIND process_vlist ROUTD
| PRI PAR NEWLINE
| PRI PAR replicator NEWLINE
    RIND process ROUTD
| placedpar

placedpar = PLACED PAR NEWLINE
              RIND placedpar_vlist ROUTD
| PLACED PAR NEWLINE
| PLACED PAR replicator NEWLINE
    RIND placedpar ROUTD
| PROCESSOR expression NEWLINE
    RIND process ROUTD

placedpar_vlist = placedpar
| placedpar_vlist
  placedpar

procedure_body = process
```

```
process = SKIP NEWLINE
| STOP NEWLINE
| action | construction | instance | case_input
| specification
  process
| allocation
  process

process_vlist = process
| process_vlist
  process

selection = CASE selector NEWLINE
              RIND option_vlist ROUTD
| CASE selector NEWLINE

sequence = SEQ NEWLINE
             RIND process_vlist ROUTD
| SEQ NEWLINE
| SEQ replicator NEWLINE
    RIND process ROUTD

specification = declaration | abbreviation | definition

valof = VALOF NEWLINE
          RIND process
          RESULT expression_list NEWLINE ROUTD
| specification
  valof

value_process = valof

variant = tagged_list NEWLINE
            RIND process ROUTD
| specification
  variant

variant_vlist = variant
| variant_vlist
  variant

abbreviation = specifier name IS element ':' NEWLINE
| name IS element ':' NEWLINE
| VAL specifier name IS expression ':' NEWLINE
| VAL name IS expression ':' NEWLINE

action = assignment NEWLINE
| input NEWLINE
| output NEWLINE

allocation = PLACE name AT expression ':' NEWLINE

declaration = type name_clist ':' NEWLINE

guard = input NEWLINE
| boolean '&' input NEWLINE
| boolean '&' SKIP NEWLINE
```

```
instance = name '(' actual_olist ')' NEWLINE

tagged_protocol = scalar NEWLINE
| scalar ';' sequential_protocol NEWLINE

tagged_protocol_vlist = tagged_protocol
| tagged_protocol_vlist
  tagged_protocol

actual = element %dprec 2
| expression     %dprec 1

actual_clist = actual | actual_clist ',' actual

actual_olist =  | actual_clist

assignment = variable_clist ASSIGN expression_list

base = expression

boolean =  expression

case_expression = expression

case_expression_clist = case_expression | case_expression_clist ',' case_expression

channel = element

delayed_input = channel '?' AFTER expression

expression_list = name '(' expression_olist ')' %dprec 2
| expression_clist                              %dprec 1

formal = specifier name_clist | VAL specifier name_clist

formal_clist = formal | formal_clist ',' formal

formal_olist =  | formal_clist

indef_type = INDEF_SPEC type
| INDEF_SPEC indef_type
| '[' expression ']' indef_type

input = channel '?' input_item_slist
| channel '?' CASE tagged_list
| delayed_input

input_item = variable | variable DOUBLE_COLON variable

input_item_slist = input_item | input_item_slist ';' input_item

output = channel '!' output_item_slist      %dprec 1
| channel '!' scalar ';' output_item_slist  %dprec 2

output_item = expression | expression DOUBLE_COLON expression

output_item_slist = output_item | output_item_slist ';' output_item
```

```
replicator =  name '=' base FOR count

selector = expression

sequential_protocol = simple_protocol_slist

simple_protocol_slist = simple_protocol | simple_protocol_slist ';' simple_protocol

specifier = type | indef_type

tagged_list = scalar | scalar ';' input_item_slist

variable = element

variable_clist = variable | variable_clist ',' variable

andor_expression = andor_expression andor_operator operand
| operand andor_operator operand

array_type = '[' expression ']' type

comms_type = CHAN OF protocol
| TIMER
| PORT OF type

conversion = primitive_type operand
| primitive_type ROUND operand
| primitive_type TRUNC operand

count = expression

element = element '[' subscript ']'
| '[' element FROM subscript FOR subscript ']'
| scalar

expression = monadic_operator operand
| operand dyadic_operator operand
| andor_expression
| conversion
| operand
| MOSTPOS data_type
| MOSTNEG data_type

expression_clist = expression | expression_clist ',' expression

expression_olist =  | expression_clist

operand = element | literal | table | '(' expression ')'
| name '(' expression_olist ')'

primitive_type = comms_type
| data_type

protocol = name | simple_protocol | ANY

simple_protocol = type | counting_type DOUBLE_COLON INDEF_SPEC type

subscript =  expression
```

```
table = table '[' subscript ']'
| '[' expression_clist ']'
| '[' table FROM subscript FOR count ']'

type = primitive_type | array_type

andor_operator = AND | OR

byte = BYTE_CONSTANT

counting_type = BYTE
| INT
| INT16
| INT32
| INT64

data_type = counting_type
| BOOL
| REAL32
| REAL64

data_type_clist = data_type | data_type_clist ',' data_type

dyadic_operator = AFTER | BITAND | BITOR | MINUS | PLUS | REM
| TIMES | RIGHT_OP | LEFT_OP | AND_OP | OR_OP | LE_OP | GE_OP
| XOR_OP | NE_OP | '-' | '+' | '*' | '/' | '\\' | '<' | '>'

integer = INT_CONSTANT | HEX_CONSTANT

literal = integer
| byte
| integer '(' data_type ')'
| byte '(' data_type ')'
| real '(' data_type ')'
| string
| TRUE | FALSE

monadic_operator = BITNOT | MINUS | NOT | SIZE | '~' | '-'

name = IDENTIFIER

name_clist = name | name_clist ',' name

real = REAL_CONSTANT

scalar = name

string = STRING_LITERAL | string_head STRING_LITERAL_END

string_head = STRING_LITERAL_START | string_head STRING_LITERAL_MIDDLE

======
```

References

[I] Inmos Ltd: occam 2 Reference Manual. Prentice Hall, 1988, document number 72 occ 45 01. http://www.transputer.net/obooks/isbn-013629312-3/oc20refman.pdf

[IOT1] INMOS Ltd: occam 2 toolset user manual - part 1 (User guide and tools). INMOS, March 1991, document number 72 TDS 275 02. www.transputer.net/prog/72-tds-275-02/otdsug1.pdf

[IOT2] INMOS Ltd: occam 2 toolset user manual - part 2 (occam libraries and appendices). INMOS, March 1991, document number 72 TDS 276 02.www.transputer.net/prog/72-tds-276-02/otdsug2.pdf