

# The Syntax of Z

*Roger Bishop Jones*

ICL Defence Systems

## 1. INTRODUCTION

This document contains a specification of the syntax of Z in the variant of BNF accepted by the syntax translator in DBC/RBJ/060. The syntax description is translated into a form acceptable to the parser in DBC/RBJ/058 yielding a parser for Z.

I follow as closely as possible the presentation in ?.

```
structure M176 =  
struct  
open X056 X058 X060;
```

## 2. THE CONCRETE SYNTAX OF Z

### 2.1. Lexical Elements

Using pro-tem the sml scanner, we define here some syntactic categories which might be expected to be dealt with by the Z scanner.

```
WORD      =   Iden;
INT       =   Integer_lit;
DECOR     =   { "" | "?" | "!" };
BLANK_LINE = ";";
AX_START  =   "|a-";
SCH_START =   "|s-";
GEN_START =   "|g-";
BAR       =   "|-";
END       =   "-|";
Ident    =   WORD [ DECOR []];
```

```
specification    =   { paragraph ? ";"};
```

```
paragraph =
  Gen_Formals
  | Axiomatic_Box
  | Schema_Box
  | Generic_Box
  | WORD [ Gen_Formals ] "=" term
  | Def_lhs "==" Expression
  | term;
```

```
Gen_Formals    =   "[" { Ident ? "," } "]";
```

```
Axiomatic_Box =   AX_START Decl_Part BAR term END;
```

```
Schema_Box    =   SCH_START WORD [Gen_Formals []] Decl_Part BAR term END;
```

```
Generic_Box   =   GEN_START [Gen_formals []] Decl_Part BAR term END;
```

```
Decl_Part    =   { Basic_Decl ? ";" };
```

```
Def_Lhs     =
  Ident Ident [Ident]
  | Var_Name [Gen_Formals []];
```

```
Var_Name    =   Ident | "(" Op_Name ")";
```

```
Op_Name     =
```

```

    "_Ñ_Ò"
  |   "_" Ident ["_"]
  |   Ident "_";

```

```

aterm =
  Ident
  |   "(" term ")" [DECOR []]
  |   ["∀"|"∃"|"λ"] Schema_Text "•" term
  |   "μ" Schema_Text ["•" term []]
  |   "<" { term ? "," } ">"
  |   ">>" { term ? "," } "¼"
  |   "(" { term ? "," } ")"
  |   "{" { term ? "," } "}"
  |   "[" { Schema_Text ["•" term []] } "];

```

```

term = { aterm };

```

```

Schema_Text = Declaration ["|" term []];

```

```

Declaration = { Basic_Decl ? ";" };

```

```

Basic_Decl =
  { Decl_Name ? ";" } ":" term
  |   Schema_Ref
  |   "(" term ")";

```

```

Decl_Name = Ident | Op_Name;

```

```

Schema_ref = WORD DECOR [Gen_Actuals []];

```

```

Gen_Actuals = "[" { term ? "," } "];

```

### 3. TRANSLATION

```

local val Z_phrases =
  (diag:=false;
   translate_syntax res1 "/escher/usr2/rbj/sml/176.syn")
in   val Z_syntax = { initial = "term", phrases = Z_phrases }
end;

```

**4. TESTING**

```

fun open_stream (p as ref {stream, reserved, toklist, stat, tok}) =
  let   val {tokens, remains, status} = scan_to (!p)
  in    (p:= {stream=stream, reserved=reserved, toklist = tokens,
             stat = status, tok=tok});()
  end;

val nulltree = PLis [];
fun parse_zterm (p as ref {stream, reserved, toklist, stat, tok}) syntax =
  case toklist of
  [] => raise end_of_input |
  t   => (case parse_tree syntax t of
          result as Success {tokens=toks,tree} =>
            (p:= {stream=stream, reserved=reserved, toklist = toks,
                 stat=stat, tok=tok};
             if !diag
             then (print tree;
                  print "\nparse succeeded\n";
                  tree)
             else (print "\nparse succeeded\n";
                  tree)) |
          result => (print "\n\n*****PARSE FAILED*****\n\n";
                   print tokens>nulltree))
             handle ? =>
              (print "\n\n*****EXCEPTION*****\n\n";
               diag:=true;
               ((parse_tree syntax t;
                print toklist) handle ? => print toklist);
              nulltree);

fun parse_zed string = parse_tree;
fun next_dec stream = parse_zterm stream Z_syntax;
fun translate_prog stream = (open_stream stream; while not (next_dec stream = nulltree) do ());
fun translate_mod id = let val stream = mk_stream id (Res_al "") res1
                       in print ("\nparsing:" ^ id ^ "\n");translate_prog stream
                       end handle end_of_input => ();
fun test_run list = map translate_mod list;

```