A Lotos *Vade Mecum*

K. J. Turner, University of Stirling

16th April 1996

Syntax classes are given in *italics*, keywords in **bold**. '...' denotes a repeated element (usually optional). The precedence of behaviour operators is given as 1 (loosest binding) to 8 (tightest binding).

<table>
<tr><td>action</td><td>i</td><td>(internal)</td></tr>
<tr><td></td><td>gate_id ! value or ? var_decl ...</td><td>(observable)</td></tr>
<tr><td></td><td>gate_id ! value or ? var_decl ... [guard]</td><td>(sel. pred.)</td></tr>
<tr><td>behaviour</td><td>let var_decl = value, ... in behaviour</td><td>1</td></tr>
<tr><td></td><td>choice var_decl or gate_decl, ... [] behaviour</td><td>1</td></tr>
<tr><td></td><td>par gate_decl, ... ||| or |[gate_id, ...]| or || behaviour</td><td>1</td></tr>
<tr><td></td><td>hide gate_id, ... in behaviour</td><td>1</td></tr>
<tr><td></td><td>behaviour ≫ behaviour</td><td>2 (enables)</td></tr>
<tr><td></td><td>behaviour ≫ accept var_decl, ... in behaviour</td><td></td></tr>
<tr><td></td><td>behaviour ▷ behaviour</td><td>3 (disabled by)</td></tr>
<tr><td></td><td>behaviour ||| or |[gate_id, ...]| or || behaviour</td><td>4 (parallel)</td></tr>
<tr><td></td><td>behaviour [] behaviour</td><td>5 (choice)</td></tr>
<tr><td></td><td>[guard] -> behaviour</td><td>6</td></tr>
<tr><td></td><td>action; behaviour</td><td>7</td></tr>
<tr><td></td><td>stop</td><td>8</td></tr>
<tr><td></td><td>exit</td><td>8</td></tr>
<tr><td></td><td>exit (value or any sort_id, ...)</td><td></td></tr>
<tr><td></td><td>proc_id [gate_id, ...] (value, ...)</td><td>8</td></tr>
<tr><td></td><td>(behaviour)</td><td>8 (grouping)</td></tr>
<tr><td>definition</td><td>behaviour where type or process ...</td><td></td></tr>
<tr><td>digit</td><td>0..9</td><td></td></tr>
<tr><td>equation</td><td>value = value</td><td>(unconditional)</td></tr>
<tr><td></td><td>[guard] => value = value</td><td>(conditional)</td></tr>
<tr><td>equations</td><td>forall var_decl, ...
  ofsort sort_id
    forall var_decl, ...
      equation; ...
  ...</td><td></td></tr>
<tr><td>formal_pars</td><td>[gate_id, ...] (var_decl, ...) : noexit</td><td></td></tr>
<tr><td></td><td>[gate_id, ...] (var_decl, ...) : exit</td><td></td></tr>
<tr><td></td><td>[gate_id, ...] (var_decl, ...) : exit (sort_id, ...)</td><td></td></tr>
<tr><td>gate_decl</td><td>gate_id, ... in [gate_id, ...]</td><td></td></tr>
<tr><td>guard</td><td>value = value</td><td></td></tr>
<tr><td></td><td>bool_value</td><td>(Boolean)</td></tr>
<tr><td>id</td><td>letter, letter or digit or underscore ...</td><td>(normal)</td></tr>
<tr><td></td><td>digit, letter or digit or underscore ...</td><td>(operation)</td></tr>
<tr><td></td><td>special_char ...</td><td>(operation)</td></tr>
<tr><td>letter</td><td>a..z A..Z</td><td></td></tr>
</table>

lib_type_id	***type_id***	***sort_id***	***opn_ids***
	Bit	*Bit*	*0..1 eq ge gt le lt NatNum ne*
	BitString	*BitString*	*+ ++ BitString eq Length NatNum ne Reverse*
	Boolean	*Bool*	*and eq false iff implies ne not or true xor*
	DecDigit	*DecDigit*	*0..9 eq ge gt le lt NatNum ne*
	DecString	*DecString*	*+ ++ DecString eq Length NatNum ne Reverse*
	Element	*Element*	*eq ne*
	FBoolean	*FBool*	*not true*
	HexDigit	*HexDigit*	*0..F eq ge gt le lt NatNum ne*
	HexString	*HexString*	*+ ++ eq HexString Length NatNum ne Reverse*
	NaturalNumber	*Nat*	*0 + * ** eq ge gt le lt ne Succ*
	NonEmptyString	*NonEmptyString*	*+ ++ eq Length ne Reverse String*
	OctDigit	*OctDigit*	*0..7 eq ge gt le lt NatNum ne*
	OctString	*OctString*	*+ ++ eq Length NatNum ne OctString Reverse*
	Octet	*Octet*	*Bit1..Bit8 eq ne Octet*
	OctetString	*OctetString*	*<> + ++ eq Length ne OctetString Reverse*
	Set	*Set*	*{} Card eq Includes Insert Ints IsIn IsSubsetOf Minus ne NotIn Remove Union*
	String	*String*	*<> + ++ eq Length ne Reverse String*

operation *pref_opn_id* or *_ inf_opn_id _ , ... : sort_id, ... → sort_id*

process
process *proc_id formal_pars* :=
  definition
endproc

special_char #%&*+−./<=>@\^~{}

specification
specification *spec_id formal_pars*
  type ...
  behaviour or **behavior** *definition*
endspec

type
library
  lib_type_id, ...
endlib

type *type_id* **is** *type_id, ...*
  formalsorts *sort_id, ...*
  formalopns *operation ...*
  formaleqns *equations*
  sorts *sort_id, ...*
  opns *operation ...*
  eqns *equations*
endtype

type *type_id* **is** *type_id*
  renamedby *type_repl*
endtype

type *type_id* **is** *type_id*
  actualizedby *type_id, ...* **using** *type_repl*
endtype

type_repl
sortnames *sort_id* **for** *sort_id ...*
opnnames *opn_id* **for** *opn_id ...*

value
var_id
pref_opn_id (*value, ...*) (prefix)
value inf_opn_id value (infix)
value **of** *sort_id* (coercion)
(*value*) (grouping)

var_decl *var_id, ... : sort_id*