The Abstract Syntax of VDM in sml

Roger Bishop Jones

ICL Defence Systems

1. INTRODUCTION

This is a transcription of the abstract syntax of VDM into sml. The material is transcribed from document no.8 pp 1-1->1-6.

```
type       Name        = string;
datatype   'a Op       = A of 'a | NIL;
datatype   'a list1    = L1 of ('a * 'a list);
type       'a set = 'a list;
datatype   Aadt_Ddeclns  =  AADT                    |
                            DDECLNS;
datatype   Classifier    =  OOP                     |
                            FFN                     |
                            CCN                     |
                            TTY;
datatype   Iimport_spec  =  AALL                    |
                            IMPORTS of (Classifier -> Name list)
and  TTY_type       =  PRIVATE                 |
                       Type_decl
and  Read_Write     =  READ                    |
                       WRITE
```

```
and Defn = TYPE of Type |
           SELECTOR_DECL of Selector_decl list
and Object_decl = TYPE_DECL of Type_decl |
                  CONSTANT_DECL of Constant_decl |
                  FUNCTION_DECL of Function_decl |
                  OPERATION_DECL of Operation_decl
and Type = SIMPLE_TYPE of Simple_type |
           TYPE_PARAMETER of Type_parameter |
           EXPRESSION of Expression
and Simple_type = TYPE_NAME of Type_name |
                  STATE_NAME of State_name |
                  BASIC_TYPE of Basic_type |
                  TYPE_UNION of Type_union |
                  UNARY_TYPE of Unary_type |
                  BINARY_TYPE of Binary_type
```

```
and Basic_type = BOOL |
                 CHAR |
                 TEXT |
                 NAT1 |
                 NAT0 |
                 INT |
                 REAL
and Unary_type_operator = SET |
                          LIST |
                          LIST1 |
                          OPTIONAL
and Binary_type_operator = MAP |
                           BIJECTIVE_MAP
and Type_Function_type = TYPE_TFT of Type |
                         FUNCTION_TYPE of Function_type
and Expression = PLACE of Place |
                 OLD_VAR of Old_var |
                 NEW_VAR of New_var |
                 CONSTANT_EXP of Constant |
                 LOCAL_DECL of Local_decl |
                 QUANTIFIED_EXPR of Quantified_expr |
                 RECORD_MODIFIER of Record_modifier |
                 APPLICATION of Application |
                 CONSTRUCTOR of Constructor |
                 LITERALS of Literals |
                 CONDITIONAL of Conditional
```

```
and Local_decl = LET_DECL of Let_decl |
                 WHERE_DECL of Where_decl
and Application = UNARY_APPLY of Unary_apply |
                  BINARY_APPLY of Binary_apply |
                  APPLY of Apply
and Unary_operator = NOT | MINUS | LEN | CARD | D_UNION |
                     D_INTERSECT | DOM | RNG | INDS | ELEMS |
                     CONC | HD | TL
and All_Ex_Un = AALL_AEU | EXISTS | UNIQUE
and Exp_or_Rator_exp
                  = EXPRESSION_ER of Expression |
                    RATOR_EXPRESSION_ER of Rator_expression
```

```
and Exp_or_Rator_exp_or_Type
                  = EXPRESSION_ERT of Expression |
                    RATOR_EXPRESSION_ERT of Rator_expression |
                    TYPE_ERT of Type
and Binary_operator = AND | OR | IMPLIES | EQUIV | ADD | SUB |
                      MULT | DIV | MOD | EXP | IDIV | EQ | NOT_EQ |
                      LT | GT | LE | GE | SUBSET | SUBSET_EQ |
                      S_UNION | S_SUB | INTERSECT | SUBRANGE |
                      CONCAT | M_RESTRICT | OVERWR | M_UNION |
                      M_DELETE | COMPOSE | ITERATION | MEM |
                      NOT_MEM
and Rator_expression = FUNCTION_NAME of Function_name |
                       RECORD_NAME of Record_name |
                       SELECTOR_NAME of Selector_name |
                       EXPRESSION_RE of Expression
and Pre_Post_Inv_Init = PRE | POST | INV | INIT
```

```
and Constructor = SET_CONSTRUCTOR of Set_constructor |
                  MAP_CONSTRUCTOR of Map_constructor
and Literals = LIST_DISPLAY of List_display |
               SET_DISPLAY of Set_display |
               MAP_DISPLAY of Map_display |
               TEXT_DISPLAY of Text_display |
               SIMPLE_LITERAL of Simple_literal
and Simple_literal = REAL_VALUE of Real_value |
                     INTEGER_VALUE of Integer_value |
                     ELEMENTARY_OBJECT of Elementary_object |
                     BOOLEAN of Boolean |
                     OPTIONAL_SL of Optional |
                     CHAR_VALUE of Char
```

```
and Optional = NIL_VALUE
and Conditional = IF of If |
                  CASES of Cases
and Case_of_Case_in = CASE_OF of Case_of_clause list |
                      CASE_IN of Case_in_clause list
and Rec_pat_List_pat = RECORD_PATTERN_RL of Record_pattern |
                       LIST_PATTERN_RL of List_pattern
and Pattern = DONT_CARE |
              PLACE_PAT of Place |
              SIMPLE_LITERAL_PAT of Simple_literal |
              RECORD_PATTERN of Record_pattern |
              LIST_PATTERN of List_pattern
and Pat_lis_or_Text_disp = PATTERN_LIST_PT of Pattern list |
                           TEXT_DISPLAY_PT of Text_display
and Dont_care_or_Place = DONT_CARE_DP |
                         PLACE_DP of Place
```

```
using Interface = { STATE_NAME : Name,
                    USAGE : Aadt_Ddeclns,
                    IIMPORT : Name -> Iimport_spec,
                    EEXPORT : Export_items }
and Type_decl = { DEFN : Defn Op,
                  INVARIANT : Function_defn Op }
and External_decl = { TYPE : Type,
                      MODE : Read_Write }
and Operation_decl = { INPUTS : Variable_decl list,
                       RESULTS : Variable_decl list,
                       EXTERNALS : Name -> External_decl,
                       LOCAL_DEFN : Binding list,
                       PPRE : Expression,
                       PPOST : Expression,
                       EXCEPTIONS : Name -> Error_predicates }
```

```
and  Operation_type  = {  ARG_TY_L       : Type list,
                          RES_TY_L       : Type list          }
and  Variable_decl   = {  NAME           : Name,
                          TYPE           : Type               }
and  Selector_decl   = {  NAME           : Name,
                          TYPE           : Type               }
and  Constant_decl   = {  TYPE           : Type,
                          EXPR           : Expression Op      }
and  Function_defn   = {  PARAMETERS     : Pattern list list1,
                          EXPR           : Expression         }
and  Function_decl   = {  SIGNATURE      : Function_type,
                          PPRE           : Function_defn Op,
                          PPOST          : Function_defn Op,
                          BODY           : Function_defn Op   }
and  Specification   = {  STATE_NAME     : Name,
                          USAGE          : Aadt_Ddeclns,
                          VARIABLES      : Variable_decl list,
                          STATE_INV      : Function_defn Op,
                          STATE_INIT     : Function_defn Op,
                          CONSTRUCTS     : Name -> Object_decl }
```

```
and  Export_items    = {  FFN          : (Name -> Function_type),
                          OOP          : (Name -> Operation_type),
                          CCN          : (Name -> Type),
                          TTY          : TTY_type           }
and  Module          = {  IINTERFACE          : Interface Op,
                          SPECIFICATION : Specification Op }
and  Error_predicates = {  CONDITION    : Expression,
                          ACTION       : Expression         }
and  Type_name       =    Name
and  State_name      =    Name
and  Type_union      = {  TYPES          : Type set           }
and  Unary_type      = {  OPERATOR       : Unary_type_operator,
                          OPERAND        : Type               }
and  Binary_type     = {  LEFT           : Type,
                          OPERATOR       : Binary_type_operator,
                          RIGHT          : Type                    }
and  Type_parameter  = {  VAR_NAME       : Name                    }
and  Function_type   = {  SOURCES  : Tuple_type list1,
                          RESULT     : Type               }
```

```
and  Tuple_type        =     Type_Function_type list
and  Place             = {   NAME       : Name       }
and  Old_var           = {   NAME       : Name       }
and  New_var           = {   NAME       : Name       }
and  Constant          = {   NAME       : Name       }
and  Let_decl          = {   BINDING    : Binding,
                             EXPRESSION : Expression }
and  Where_decl        = {   EXPRESSION : Expression,
                             BINDING    : Binding    }
and  Binding           = {   CONSTANT   : Pattern -> Typed_expression,
                             CHOICE     : Pattern -> Typed_expression,
                             FUNCTIONS  : Name -> Function_decl }
and  Typed_expression  = {   TYPE       : Type Op,
                             EXPR       : Expression }
and  Quantified_expr   = {   KIND       : All_Ex_Un,
                             BINDING    : Name -> Type,
                             PREDICATE  : Expression }
and  Record_modifier   = {   RECORD     : Expression,
                             MODIFIERS  : Name -> Expression }
and  Unary_apply       = {   OPERATOR   : Unary_operator,
                             OPERAND    : Expression }
```

```
and  Binary_apply      = {   LEFT       : Exp_or_Rator_exp,
                             OPERATOR   : Binary_operator,
                             RIGHT      : Exp_or_Rator_exp_or_Type }
and  Apply             = {   RATOR      : Rator_expression,
                             RAND       : Argument list1 }
and  Argument          =     Expression list
and  Function_name     = {   PREFIX     : Pre_Post_Inv_Init,
                             NAME       : Name,
                             INSTANCE   : Name -> Type }
and  Record_name       = {   NAME       : Name       }
and  Selector_name     = {   NAME       : Name       }
and  Set_constructor   = {   EXPR       : Expression,
                             SET_MEMBER : Name -> Type,
                             PREDICATE  : Expression }
and  Map_constructor   = {   MAPLET     : Maplet,
                             SET_MEMBER : Name -> Type,
                             PREDICATE  : Expression }
and  List_display      = {   VALUE      : Expression list }
and  Set_display       = {   VALUE      : Expression list }
and  Map_display       = {   VALUE      : Maplet list }
and  Maplet            = {   DOM_ELEMENT: Expression }
```

```
and  Text_display      = {  VALUE          : Char list          }
and  Real_value        = {  VALUE          : real          }
and  Integer_value     = {  VALUE          : int          }
and  Elementary_object      = {  VALUE          : Name          }
and  Boolean           = bool
and  If                = {  PREDICATE      : Expression,
                            THEN           : Expression,
                            ELSE           : Expression    }
and  Cases             = {  EXPR           : Expression,
                            CASES          : Case_of_Case_in,
                            ELSE           : Expression Op      }
and  Case_of_clause    = {  PATTERN        : Rec_pat_List_pat,
                            EXPR           : Expression    }
and  Case_in_clause    = {  TYPE           : Type,
                            EXPR           : Expression    }
and  Record_pattern    = {  TAG            : Name,
                            FIELDS         : Pattern list  }
and  List_pattern      = {  PREFIX   : Pat_lis_or_Text_disp,
                            MIDDLE   : Dont_care_or_Place,
                            POSTFIX  : Pat_lis_or_Text_disp  }
and  Char  = int;
```