Atelier B

Type Checker

Error Message Manual

version 3.6

CLEARSY
SYSTEM ENGINEERING

ATELIER B
Type Checker Error Message Manual
version 3.6

Document made by CLEARSY.

CLEARSY
ATELIER B maintenance
Parc de la Duranne
320 avenue Archimède
Les Pléiades III - Bât.A
13857 Aix-en-Provence Cedex 3
France

Tél 33 (0)4 42 37 12 99
Fax 33 (0)4 42 37 12 71
email : maintenance.atelierb@clearsy.com

Contents

Chapter 1

Introduction

In this manual, different error and warning messages originating from the Type Checker are presented. The goal is to define the origin of errors for each message so as to help the user : once the source of error has been correctly located, it is much easier to correct one's specification. For more complex and detailled information, please refer to the B Language Reference Manual.

The messages from the Type Checker are all flanked by :

> *Type Checking <machine/refinement/implementation><comp_name>*
> *...*
> *End of Type checking*

For each effective control, an information message is posted . For example :

> *Checking operation Read*

informs the user that the Type Checker is verifying the Read operation. The error or warning messages which will follow, will refer to the Read operation. However, they will specify the extract from the source code where the error has been localised. In fact, the expressions *A <: B* et *A <<: B* are normalised in *A : POW(B)*. If there are any associated messages they will quote the normalised expression.

This manual is made up of four chapters. The first one defines the terms used in the message explanations. The following three chapters present, in order, the warning, error, and internal error messages.

The messages are classified according to alphabetical order. Symbols, apart from figures, are not included in the classification. So, the message :

> *<exp> and <ident> have incompatible type in a CASE substitution*

is classified under the A letter. It therefore comes before the message :

> *Bound <ident> of <exp> should be an integer*

Each message is presented in a table, as follows:

Example	**Comment**
Component name <ident> should be an identifier	*wording of the message*
Name of a component, which must be a simple name, i.e., a correct B identifier.	*description of the error made*

```
/*The name of the machine below is incorrect because
it contains a dot.*/
MACHINE
  M1.N2
END
```
	example of a specification generating the message

Chapter 2

Definitions

This chapter defines certain terms used henceforth in the manual.

constant denotes indifferently an abstract or concrete constant .

component denotes indifferently a machine, a refinement or an implementation.

B identifier chain of characters verifying the following rules:

- at least two characters
- beginning with a letter
- composed solely of letters (A-Z, a-z), digits (0-9) and underscore (_)

keyword identifier with a particular meaning. The list of keywords in the B Language is presented in the B Language Reference Manual. It is necessary to complete it with the following list which is the list of identifiers reserved for the tools : ARI, CATL, DED, DEF, END, FLAT, FORWARD, FORWARDTHEORIES, GEN, HYP, IS, LMAP, MAP, MODR, NEWV, NORMAL, NORMALTHEORIES, PROOFLEVEL, PROOFMETHOD, RES, REV, RULE, SET, SHELL, SPESPE, SUB, THEORY, THEORIES, WRITE, bUpident, band, bappend, bcall, bcall1, bcall2, bcatl, bclean, bclose, bcompile, bconnect, bcrel, bcrelr, bcrer, bctrule, bdef1, bdef2, bdump, berv, bfalse, bfwd, bget, bgethyp, bgetresult, bgoal, bguard, bhalt, bident, binhyp, blemma, blen, blenf, blent, blident, bload, blvar, bmark, bmatch, bmodr, bnewv, bnlmap, bnmap, bnot, bnum, bpattern, bpop, bprintf, bproved, breade, breadf, brecompact, bresetcomp, bresult, brev, brule, bsearch, bsetmode, bshell, bslmap, bsmap, bsparemem, bsrv, bstatistics, bstring, bsubfrm, btest, bunproved, bvrb, bwritef, bwritem, trace.

typing predicate predicate of the form *"Expression op Identifier"* where *op* is either belonging ($\in$), or inclusion ($\subset$ or $\subseteq$), or equality ($=$). These predicates are described in details in the B Language Reference Manual.

variable denotes an abstract or concrete variable.

Chapter 3

Warning messages

Warning messages from the type checker are preceeded by :

Warning :

They permit the user to anticipate a future error message from the B0Checker. They can also indicate potential problems concerning code readability.

Concrete constant <ident_cst> has not been valued

All of the concrete constants defined during refinement must be valued in the implementation's VALUES clause. This warning anticipates an error message from B0Checker.

```
IMPLEMENTATION M1_1
REFINES M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : INTEGER --> BOOL
END  /* cc is not valued */
```

Concrete constant <ident_cst> is not an implementable array

The concrete constant <ident_cst> is not implementable in B0: its domain must be an interval or a listed set.

```
MACHINE
  M1
CONSTANTS
  Sequence, Relation
SETS
  EE
PROPERTIES
  Sequence : seq(EE) &
  Relation : INT <-> INT
  /* Sequence and Relation are not implementable */
END
```

Concrete constant <ident_cst> may not be implementable

The type checker is not yet able to determine whether the constant <ident_cst> is implementable. This warning may appear after an error in the type calculation. In this case other messages will detail the problem.

```
MACHINE
  M1
CONSTANTS
  c1
PROPERTIES
  c1 = FctUnknown(1)
END
```

Constant <ident_cst> is not an implementable record : it uses a non implementable array

Concrete constant <ident_cst> is not implementable in B0: one of its fields is a non implementable array (its domain should be an interval or an enumerated set).

```
MACHINE
  M1
CONSTANTS
  Record1, Record2
SETS
  EE
PROPERTIES
  Record1  : struct(seq1 : seq(EE), bb ; BOOL) &
  Record2 : struct(rel1 : INT <-> INT, xx : INT)
  /* Record1 and Record2 are not implementable. */
END
```

Deferred set <ident_set> has not been valued

All of the abstract sets defined during refinement must be valued in the implementation's VALUES clause. This warning anticipates an error message from the B0Checker.

```
IMPLEMENTATION M1_1
REFINES M1
SETS SS
END  /* SS is not valued */
```

Identifier <ident> is already used

The identifier <ident> is used more than once in the analyzed component. These two definitions do not risk a conflict and the specification is correct. This warning simply highlights a potential problem in the understanding of sources when read.

```
REFINEMENT M1_1
REFINES M1
OPERATIONS
  op =
  VAR vv IN
    vv : (vv : NAT & !vv.(vv : BOOL => 0 = 0))
    /* the second vv does not conflict with the first one but
       may interfere with the understanding of the operation */
  END
END
```

Local variable <ident> may be read before being initialised

This message is generated for a machine or a refinement. Local variable <ident> is a variable defined in a VAR substitution or in the list of output parameters for an operation. It was used while not completely initialised by a branch substitution.

```
MACHINE M1
OPERATIONS
  ss, tt <-- op(ii) = PRE ii : NAT THEN
    IF ii > 1 THEN
      ss := 2
    END;
    tt := ss
    /* ss was not initialised in all branches of the IF condition */
  END
END
```

Local variable <ident> may not be initialised

Local variable <ident>, defined in a VAR substitution, is not properly initialised or is initialised in only some paths of a branch substitution.

```
REFINEMENT
  M1_1
OPERATIONS
  op = VAR vv IN skip END
END
```

Local variables <list_ident> may not be initialised

Local variables <list_ident>, defined in a VAR substitution, are not properly initialised or are initialised in only some paths of a branch substitution.

```
REFINEMENT
  M1_1
OPERATIONS
  op(ii) = PRE ii : NAT THEN
    VAR vv, ww IN
      IF ii = 1 THEN
        vv := 2
      END
END
```

Output parameter <ident> may not be initialised

This message is generated for a machine or a refinement. The output parameter <ident> from the operation being type checked was not initialised in all of the branches of the branch substitutions used in the body of this operation.

```
MACHINE M1
OPERATIONS
  ss <-- op(ii) = PRE ii : NAT THEN
    IF ii > 1 THEN
      ss := 2
    END
  END
END
/* ss was not initialised in all of the branches of the IF condition */
```

Output parameters <list_ident> may not be initialised

This message is generated for a machine or a refinement. The <list_ident> output parameters from the operation being type checked were not initialised in all branches of the branch substitutions in the body of this operation.

```
MACHINE M1
OPERATIONS
  ss, tt <-- op(ii) = PRE ii : NAT THEN
    IF ii > 1 THEN
      ss := 2
    ELSE
      tt := 3
    END
  END
END
/* ss and tt were not typed in all branches of the IF condition */
```

Chapter 4

Error messages

Error messages from the type checker are preceeded by :

Error :

As far as possible, the type checker does not stop after an error. If, however, it finds it impossible to continue, the following final message indicates that the verification has been interrupted :

TypeCheck aborted

$0 is not allowed: <ident>$0

Expression $0 is only allowed in the "becomes such as" and "WHILE" substitutions. This message is thrown in all other cases.

```
MACHINE
  M1
CONCRETE_VARIABLES
  vv
INVARIANT
  vv : NAT
INITIALISATION
  vv := 1
OPERATIONS
  op = vv := vv$0
END
```

Abstract and concrete headers of local operation <ident_op> differ

Headers of implementation of local operations must be strictly identical to the headers of their abstraction: the number of input and output parameters must be retained, the parameter names must be the same.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
CONCRETE_VARIABLES
  v1
INVARIANT
  v1:NAT
INITIALISATION
  v1:=0
LOCAL_OPERATIONS
  oper1 =
    skip;
  oper2(xx) = PRE
    xx:NAT
  THEN
    v1:=xx
  END;
  res <-- oper3 =
    res := v1;
  res <-- oper4(xx) = PRE
    xx:NAT
  THEN
    res:=xx+v1
  END
OPERATIONS
  oper1(xx) = skip
       /*xx is too many*/;
  out <-- oper2 = BEGIN
       /*out is too many,
         xx is missing */
    out:=0
  END;
  out <-- oper3 = BEGIN
       /*out instead of res*/
     out := v1
  oper4 = skip
       /* res and xx are missing*/
END
```

Abstract and concrete headers of operation <ident_op> differ

In a refinement or an implementation, the headers of refined operations must be strictly identical to the abstract machine headers: the number of input and output parameters must be retained, the parameter names must be the same. In the same way, when refining an operation with a promoted operation, the headers must be identical.

```
MACHINE
  M1
VARIABLES
   v1
INVARIANT
  v1:NAT
INITIALISATION
  v1:=0
OPERATIONS
  oper1 =
    skip;
  oper2(xx) = PRE
    xx:NAT
  THEN
    v1:=xx
  END;
  res <-- oper3 =
    res := v1;
  res <-- oper4(xx) = PRE
    xx:NAT
  THEN
    res:=xx+v1
  END;
  out <-- opincluse(in) = PRE
    in:NAT
  THEN
    out:=in+1
  END;
END
```

```
IMPLEMENTATION
  M1_1
REFINES
  M1
IMPORTS
  M2
PROMOTES
  opincluse
OPERATIONS
  oper1(xx) = skip
       /*xx is too much*/;
  out <-- oper2 = BEGIN
       /*out is too much,
         lacks xx*/
    out:=0
  END;
  out <-- oper3 = skip;
       /*out in place of res*/
  oper4 = skip
       /*lacks res and xx*/
END
```

```
MACHINE
  M2
OPERATIONS
  res <-- opinclue(xx) =
  /* res and xx in place of out and in */
  PRE xx:NAT THEN res:=xx+1 END
END
```

Abstract constant <ident_cst> cannot be used in <ident_mach> instanciation

The abstract constants of a machine or a refinement M cannot be used in the instanciation of the machines referenced in the INCLUDES and EXTENDS clauses in M.

```
MACHINE M1
ABSTRACT_CONSTANTS
  cc
PROPERTIES
  cc : POW(NAT) * POW(NAT)
INCLUDES
  M2(cc)
END
```

Abstract constant <ident> has not been typed

All abstract constants must be typed in the PROPERTIES clause using a typing predicate (refer to the definition in Chapter 1).

```
/*In the example below, constants valmin and valmed have not been
typed.*/
MACHINE MACH_CONST
ABSTRACT_CONSTANTS
  valmax,
  valmin,
  valmed
PROPERTIES
  valmax = 100 &
  valmin < valmax    /* This does not type valmin */
END                  /* valmed not typed */
```

Abstract constant <ident_hcst> has not the same type in <ident_comp1> and in <ident_comp2>

<ident_comp1> designates the component refined by the analyzed component.
<ident_comp2> designates a machine which is directly requested by the analyzed component.
The abstract constant <ident_hcst> of <ident_comp1> can not be implemented by an abstract or concrete homonym constants which have a different type in <ident_comp2>.

```
MACHINE
  other
ABSTRACT_CONSTANTS
  cc
PROPERTIES
  cc : BOOL
END
```

```
MACHINE
  M1
ABSTRACT_CONSTANTS
  cc
PROPERTIES
  cc : NAT
  /* Replace cc : NAT
     with cc : BOOL */
END
```

```
IMPLEMENTATION
  M1_i
REFINES
  M1
IMPORTS
  other
END
```

Abstraction and refinement have the same name

The names of components in a vertical development must all be distinct. In general, the position n refinement of machine M1 is named M1_n.

```
REFINEMENT
  MACH  /* illegal */
REFINES
  MACH
END
```

```
REFINEMENT
  MACH_1   /* write recommended */
REFINES
  MACH
END
```

Abstract set name <ident> should be an identifier, or invalid list separator

A set name must be a B language identifier (refer to the definition in Chapter 1). Each set definition must be separated by a semi colon.

```
MACHINE
  M1
SETS
  2; "string"; combined name
END
```

<exp> and <ident> have incompatible type in a CASE substitution

Discriminant <exp> of a CASE substitution and branch selector <ident> should have the same type.

```
MACHINE
  M1
SETS
  EE = {c1, c2}
VARIABLES
  vv
INVARIANT
  vv : NAT
INITIALISATION
  vv :: NAT
OPERATIONS
op =
  CASE vv OF
  EITHER c1 THEN skip
  OR TRUE THEN skip
  ELSE skip
  END
  END
  /* c1 and TRUE do not have the same type as vv */
END
```

<ident_op> and another operation of <ident_mach> are called simultaneously

Two included operations cannot be called in parallel.

```
MACHINE M1
VARIABLES
  v1,v2
INVARIANT
  v1:NAT & v2:NAT & v1<=v2
INITIALISATION
  v1:=0 || v2:=0
OPERATIONS
  increment = PRE
    v1<v2
  THEN
    v1:=v1+1
  END
;
  decrement = PRE
    v1<v2
  THEN
    v2:=v2-1
  END
END
```

```
MACHINE M0
INCLUDES
  M1
OPERATIONS
  op_errone = PRE
    v1<v2
  THEN
    increment || decrement
    /* the invariant is lost */
  END
END
```

A record element whithout label can not be used in <Expression>

Two record elements whithout label can not be compared. This is because a record element without label has a generic type.

```
MACHINE
  M1
ABSTRACT_VARIABLES
  xx,yy
INVARIANT
  xx : NAT &
  yy : BOOL &
  rec(xx,yy) = rec(2,TRUE)
  /* The expression rec(xx,yy) = rec(2,TRUE) is not correct */
  /* xx = 2 &  yy = TRUE is correct              */
INITIALISATION
  xx := 2 ||
  yy := TRUE
END
```

Bound <ident> of <exp> should be an integer

The two boundaries of an interval should be integers.

```
MACHINE
  M1
CONSTANTS
  cc, dd
PROPERTIES
  cc = TRUE..7 &       /* TRUE is not an integer */
  dd = 2..Binconnue    /* Binconnue is not an integer */
END
```

<ident> can not be typed by {}

This message is sent when the identifier <ident> is typed by the empty set.

```
MACHINE
        test
ABSTRACT_VARIABLES
        vv
INVARIANT
        vv = {} /* vv has not been typed. For example, you must write
                   vv <: NAT &
                   vv = {}
                 */
INITIALISATION
        vv := {}
END
```

Component name <ident> is a keyword

The <ident> identifier is a reserved language component (refer to Chapter 1). It is illegal to use it to rename a component.

```
MACHINE
  MAXINT
END
```

Component name <ident> should be an identifier

A component name must be a simple name, i.e. a correct B language identifier (refer to the definition in Chapter 1).

```
/*The machine name below is incorrect as it contains a dot.*/
MACHINE
  M1.N2
END
```

Concrete variable <ident> is implicitly implemented with a variable of <ident> which has not the same type

In an implementation, a concrete variable may be implicitly implemented with a variable of the same name taken from an imported machine.
In the case of this message, the variable to implement and the one which is imported do not have the same type, which is illegal.

```
MACHINE
  M1
CONCRETE_VARIABLES
  vv
INVARIANT
  vv : NAT
INITIALISATION
  vv := 1
END
```

```
MACHINE
  M0
CONCRETE_VARIABLES
  vv
INVARIANT
  vv : BOOL
INITIALISATION
  vv := TRUE
END
```

```
IMPLEMENTATION
  M1_1
REFINES
  M1
IMPORTS
  M0
END
```

Constant <ident> has not been typed

All constants must be typed in the PROPERTIES clause using a typing predicate (refer to the definition in Chapter 1).

```
/*In the example below, constants valmin and valmed have not been
typed.*/
MACHINE MACH_CONST
CONSTANTS
  valmax,
  valmin,
  valmed
PROPERTIES
  valmax = 100 &
  valmin < valmax    /* This does not type valmin */
END                  /* valmed not typed */
```

Constant <ident> is not an implementable array

This message is generated for an implementation. An array is not implementable in B0 if its array is not an interval or an enumerated set.

```
IMPLEMENTATION M1_1
REFINES M1
VISIBLE_CONSTANTS
  cc
PROPERTIES
  cc : INTEGER --> BOOL
VALUES
  cc = INTEGER * {TRUE}
  /* INTEGER is not bounded */
END
```

Constants should be defined in the PROPERTIES clause

The component analyzed is not the PROPERTIES clause although it contains constants.

```
MACHINE MACH_CONST
CONSTANTS
  valmin, valmax
END      /* the PROPERTIES clause is missing */
```

<ident> declaration is not visible

The analyzed component refers to an object called <ident> that does not belong to the set of visible objects. This situation occurs after a data entry error or when the visibility constraints are violated.

```
MACHINE M1
OPERATIONS
vv <-- op = vv := UnknownId
    /* UnknownId is not a visible identifier */
END
```

Distinct definitions of enumerated set <ident_set>

In implementation, a given listed set may be defined in one of the refined components (or in the implementation) and in a machine that is seen or imported. However, the two definitions must be identical: same number of elements, same name for each element, same order of the elements.

```
MACHINE
  M1
SETS
  Enum1 = {bb};
  Enum2 = {E2a, E2b}
END
```

```
MACHINE
  M2
SETS
  Enum1 = {aa};
  Enum2 = {E2b, E2a}
END
```

```
IMPLEMENTATION
  M1_1
REFINES
  M1
SEES
  M2
  /* Enum1 and Enum2 do not have the same definition in M1 and M2 */
END
```

<ident> does not exist or is not a visible operation

The operation called <ident> does not belong to the set of visible operations. This situation occurs after an entry error or when visibility constraints are not met.

```
/*The unknown operation in the following
machine does not belong to the included
machine, therefore it is not possible to
promote it:*/
MACHINE
  M1
INCLUDES
  M2
PROMOTES
  opinconnue
END
```

```
MACHINE
  M2
END
```

Element <ident_elt> of set <ident_set> is already defined

This is an identifier conflict.

```
MACHINE MACH
SETS
  COLOURS = { red, green, blue }
;  GREEN = { green }      /* green is in conflict */
END
```

Enumerated set name in definition <enum_def> should be an identifier

A set name must be a B language identifier (refer to the definition in Chapter 1).

```
MACHINE
  M1
SETS
  2 = {aa};
  "string" = {bb};
  combined name = {cc}
END
```

<ident_cst> has not the same type in <ident_mach1> (or in an abstraction <ident_mach1>) and in <ident_mach2>

The <ident_cst> constant is implicitly valued by a constant with the same name belonging to a seen or imported machine. <ident_cst> type is defined in the PROPERTIES clause of the abstraction of the analyzed component and the one which is defined in the seen or imported machine must therefore be identical.

```
MACHINE M1
CONSTANTS
  cst
PROPERTIES
  cst : NAT
END
```

```
MACHINE M2
CONSTANTS
  cst
PROPERTIES
  cst : BOOL
END
```

```
IMPLEMENTATION M1_1
REFINES
  M1
SEES
  M2  /* implicit valuation of cst */
END
```

Identifier <ident> is a keyword

Identifier <ident> is a language keyword (refer to Chapter 1). It cannot be used to name another entity.

```
MACHINE MACH(skip)
END
```

Identifier <ident> is already defined

This message reminds the user of the presence of an identifier conflict when analyzing a specific clause.

```
MACHINE
  MACH
SEES
  SEE01
CONSTANTS
  cst1
  /*conflict with SEE01*/
PROPERTIES
  cst1 : NAT
END
```

```
MACHINE
SEE01
CONSTANTS
  cst1
PROPERTIES
  cst1 : NAT
END
```

Identifier <ident_cst> is already valued

A constant or a set of the analyzed component is valued twice, which is illegal.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
VALUES
  val1 = 2 ;
  val1 = 2     /*val1 is valued twice*/
END
```

Identifier <ident> is defined in <ident_mach1> and in <ident_mach2>

This message indicates an identifier conflict between two machines covered by a visibility clause. The use of a renaming prefix may resolve this conflict.

```
MACHINE INCO1
VARIABLES
  v_conflict   /*conflict*/
INVARIANT
  v_conflict : NAT
INITIALISATION
  v_conflict := 0
END
```

```
MACHINE INCO2
VARIABLES
  v_conflict   /*conflict*/
INVARIANT
  v_conflict : BOOL
INITIALISATION
  v_conflict := FALSE
END
```

```
MACHINE  GLOBAL
INCLUDES
  INCO1, INCO2
  /* a correct write includes:
INCLUDES
  i1.INCO1, i2.INCO2 */
END
```

Identifier <ident> is defined in <ident_mch1> and in an included renamed machine of <ident_mch2>

This message indicates an identifier conflict between two machines covered by a visibility clause. The use of a renaming prefix may resolve this conflict.

Identifier <ident> is defined in <ident_mch1> and in <ident_mch2> (or in an abstraction of <ident_mch2>)

This message indicates an identifier conflict between two machines covered by a visibility clause. The use of a renaming prefix may resolve this conflict.

```
MACHINE MACH
INCLUDES
  INC01
END
```

```
MACHINE INC01
VARIABLES
  v_conflict   /*conflict*/
INVARIANT
  v_conflict : NAT
INITIALISATION
  v_conflict := 0
END
```

```
REFINEMENT MACH_1
REFINES MACH
END
```

```
REFINEMENT MACH_2
REFINES MACH_1
CONCRETE_CONSTANTS
  v_conflict /* conflict */
INVARIANT
  v_conflict : BOOL
INITIALISATION
  v_conflict := FALSE
  /* v_conflict in INC01 is still
    visible, hence the conflict*/
END
```

Identifier <ident> is defined in an included (possibly renamed) machine of <ident_mch1> and in an included (possibly renamed) machine of <ident_mch2>

This message indicates an identifier conflict between two machines covered by a visibility clause. The use of a renaming prefix may resolve this conflict.

Identifier <ident> is defined in an included renamed machine of <ident_mch1> and in <ident_mch2>

This message indicates an identifier conflict between two machines covered by a visibility clause. The use of a renaming prefix may resolve this conflict.

Identifier <ident> is defined in <ident_mch1> (or in <ident_mch1>'s abstractions) and in <ident_mch2>

This message indicates an identifier conflict between two machines covered by a visibility clause. The use of a renaming prefix may resolve this conflict.

<ident> in <expr> can not be typed by a record element without label

This message is produced when one tries to type a data with a record value where some labels where omitted.

```
MACHINE Mach
CONSTANTS cc
PROPERTIES
  cc = rec(1, TRUE)
  /* correct version : cc = rec(l1 : 1, l2 : TRUE) */
END
```

Incompatible types in <exp>

The syntax of <exp> implies certain conditions for the types. This message indicates a violation of these conditions.
For example, in expression ff(xx), xx must belong to the starting domain of ff. In the same way, in substitution vv := {aa, bb, cc}, the three elements aa, bb and cc must have the same type.

```
MACHINE
  M1
SETS
  SS; TT
CONSTANTS
  relation, ff
PROPERTIES
  relation : SS <-> TT &
  ff : INT --> SS
OPERATIONS
  vv <-- op1 = vv := relation[{1}];
            /* 1 does not belong to SS */
  vv <-- op2 = vv := [1, 2, TRUE, 6];
            /* TRUE is not the same type as 6 */
  vv <-- op3 = vv := {1, TRUE, 2};
            /* TRUE is not the same type as 2 */
  vv <-- op4 = vv := ff(TRUE)
            /* TRUE is not an integer */
END
```

<exp1> in <exp2> has not been typed

Expression <exp1> contains one or more identifiers that were not typed prior to use in <exp2>.

```
MACHINE
  M1
CONSTANTS
  ff, xx
PROPERTIES
  ff : NAT --> NAT &
  ff(xx) = 5
END
```

<exp1> in <exp> should be a couple of sets

The operator used in <exp> expects as an argument a couple of sets.

```
MACHINE
  M1
SETS
  EE
CONSTANTS
  cc, dd
PROPERTIES
  cc = prj1(EE) &     /* EE is not a couple of sets */
  dd = prj2(Unknown)  /* Unknown is not a couple of sets */
END
```

<exp1> in <exp> should be a function

In an expression in the form f(x), f must have been defined as a function.

```
MACHINE
  M1
CONSTANTS
  c1, c2
PROPERTIES
  c1 = TRUE(1) &    /* TRUE is not a function */
  c2 = Unknown(1)   /* Unknown is not a function */
END
```

<exp1> in <exp> should be a list of distinct identifiers

<exp1> must be a list of B language identifiers, distinct from each other and separated by commas. The definition of a B language identifier is provided in Chapter 1.

```
MACHINE
  M1
CONSTANTS
  cc
PROPERTIES
  !(xx, xx). (cc = xx) &
      /*xx appears twice */
  cc = PI(xx; yy).(xx : NAT & yy : NAT | 1) &
      /*use ';' in place of ',' */
  cc = SIGMA(xx, _1).(xx : NAT | 1) &
      /* _1 is not an identifier */
  cc = UNION(xx, 1).(xx : NAT | {xx})
      /* 1 is not an identifier */
END
```

<exp1> in <exp> should be an expression

This message is generated for a lambda expression: in the notation %L.(P | E), E must be an expression.

```
MACHINE
  M1
CONSTANTS
  cc, dd, ee
PROPERTIES
  cc = %(xx).(xx : NAT | skip) &
             /* skip is not an expression */
  dd = %(xx).(xx : NAT | UnknownExp) &
             /* UnknownExp is not an expression */
  ee = %(xx).(xx : NAT | xx = 2)
             /* xx = 2 is not an expression,
                xx := 2 is correct */
END
```

<exp1> in <exp> should be an integer

The operators used in <exp> require that <exp1> should be an integer.

```
MACHINE
  M1
CONSTANTS
  Relation
PROPERTIES
  Relation : INT <-> INT
OPERATIONS
  vv <-- op1 = vv := SIGMA(xx).(xx: 1..100 | bool(xx <= 20));
             /* bool(xx <= 20) is a Boolean value*/
  vv <-- op2 = vv := iterate(Relation, UnknownInteger)
             /* UnknownInteger does not have a type*/
END
```

<exp1> in <exp> should be an integer set or an enumerated set

The operator used in <exp> requires that <exp1> represents an integer set or an enumerated set.

```
MACHINE
  M1
SETS
  AA
OPERATIONS
  vv <-- opMinAbst = vv := min(AA);         /*AA is an abstract set */
  vv <-- opMinScal = vv := min(3);          /*3 is not a set */
  vv <-- opMaxInc = vv := max(UnknownEns)   /*UnknownEns is not a set*/
END
```

<exp1> in <exp> should be a relation

The operator used in <exp> requires that <exp1> represents a relation.

```
MACHINE
  M1
SETS
  SS
CONSTANTS
  cc, tt
PROPERTIES
  cc = ran(6) &                 /* 6 is not a relation */
  tt : NAT
OPERATIONS
  vv <-- op1 = vv := rel(tt);    /* tt is not a relation */
  vv <-- op2 = vv := Unknown~;   /* Unknown is not a relation */
  vv <-- op3 = vv := fnc(SS)     /* SS is not a relation */
END
```

<exp1> in <exp> should be a relation between a set and itself

The operator used in <exp> expects as an argument a relation between a set and itself.

```
MACHINE
  M1
SETS
  EE, FF
CONSTANTS
  Rel, Rel6, Clos
PROPERTIES
  Rel : EE <-> FF &
  Rel6 = iterate(Rel, 6)  /* error as EE /= FF */
END
```

<exp1> in <exp> should be a sequence of sequences

The operator used in <exp> expects a sequence of sequences as its argument.

```
MACHINE
  M1
CONSTANTS
  Sequence
PROPERTIES
  Sequence : seq(INT)
OPERATIONS
  vv <-- opConc = vv := conc(Sequence);
    /* Sequence is not a sequence of sequences */
  vv <-- opConc2 = vv := conc(UnknownSeq)
    /* UnknownSeq is not a sequence of sequences */
END
```

<exp1> in <exp> should be a set
The operators used in <exp> require that <exp1> represents a set.

```
MACHINE
  M1
CONSTANTS
  cc, dd, ee
SETS
  EE
PROPERTIES
  cc : UnknownEns & /* UnknownEns should be a set */
  ee : NAT &
  dd /: ee          /* ee is not a set */
OPERATIONS
  vv <-- opInter = vv := INTER(xx).(xx : NAT | ee);
                    /* ee is not a set */
  vv <-- opCard = vv := card(UnknownEns);
                    /* UnknownEns is not a set */
  vv <-- opSeq = vv := seq(1)
                    /* 1 is not a set */
END
```

<exp1> in <exp> should be a set of sets of same type
The operators used in <exp> require that <exp1> represent a set of sets of the same type.

```
MACHINE M1
CONSTANTS
  aa, bb
PROPERTIES
  aa = union(UnknownEns) &     /* UnknownEns does not have a type */
  bb = inter({1, 2})           /* {1, 2} is a set of integers */
END
```

Internal name clash between identifier <ident> and a renamed identifier of the abstraction of <ident_mach>

When a component renames a machine with the "pp" prefix, and when the latter has an identifier called "ident", the proof obligation generator and the prover handle the "ppident" identifier and not "pp.ident". If a "ppident" identifier is also defined in a non renamed machine or in the component itself, a conflict occurs.
This conflict is detected so that there are never any incorrect proof obligations, this is only due to the internal operation of Atelier B.

```
MACHINE M1
INCLUDES
  pp.M2
END
```

```
MACHINE M2
VARIABLES
  var
INVARIANT
  var : NAT
INITIALISATION
  var := 0
END
```

```
REFINEMENT M1_1
REFINES M1
VARIABLES
  ppvar
INVARIANT
  ppvar : BOOL  /*conflict*/
INITIALISATION
  ppvar := TRUE
END
```

Invalid assignement for a record element in <Expression>

This message is sent when a record element assignement is not correct.

```
MACHINE
        test
CONCRETE_VARIABLES
        xx
INVARIANT
        xx : INT --> struct(l1 : BOOL, l2 : 1..10)
INITIALISATION
        xx :: INT --> struct(l1 : BOOL, l2 : 1..10)
OPERATIONS
        op1 = BEGIN xx(1)'l1 := TRUE END
                /* The syntaxe xx(1)'l1 is not allowed.
                   xx(1) := rec(TRUE,1) is correct. */
END
```

Invalid call of <ident_op>: wrong number of input parameters

When an operation is called up, the number of effective parameters must equal the number of formal parameters.

```
MACHINE  M1
INCLUDES
  M2
OPERATIONS
  oper02 = BEGIN
    oper01(10,10)
  END
END
```

```
MACHINE M2
OPERATIONS
  oper01(xx) = PRE
    xx:NAT
  THEN
    skip
  END
END
```

Invalid call of <ident_op>: wrong number of output parameters

When calling up an operation, the number of effective parameters must equal the number of formal parameters.

```
MACHINE  M1
INCLUDES
  M2
OPERATIONS
  vv, ww <-- opM1 =
    vv, ww <-- opM2
END
```

```
MACHINE M2
OPERATIONS
  vv <-- opM2 = vv := 1
END
```

Invalid constant <expression> in a branch of CASE

A constant listed set or a constant character set was used in a branch of a CASE substitution. Only numerical constants or identifiers are allowed.

```
MACHINE M1
VARIABLES
  ww
INVARIANT
  ww : NAT
INITIALISATION
  ww:=0
OPERATIONS
  uu <-- OP = BEGIN
    CASE ww OF
    EITHER {0,1,2} THEN uu:=0
        /* 0,1,2 without brackets is correct*/
    OR "3,4,5" THEN uu:=1
        /* 3,4,5 without brackets is correct */
    OR _1 THEN uu:=2
        /* _1 is not an identifier*/
    ELSE uu:=3
    END
  END
END
END
```

Invalid extended machine <ident_mach>, it uses other machines

A machine that performs a USES cannot be referenced in an IMPORTS clause. It cannot therefore appear in the EXTENDS clause of an implementation, as this would result in importing it.
Note that this message only appears in an implementation. In an abstract machine or in a refinement, the extension implies an inclusion, therefore it remains authorized.

```
MACHINE
  M2
USES
  M3
ENDD
```

```
IMPLEMENTATION
  M1_1
EXTENDS
  M2
END
```

Invalid formula in VALUES clause

A syntax error was detected in the VALUES clause. The different valuations must be separated by semi colons, each valuation is indicated by a '=' character.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
VALUES
  c3 = 3 &
  c4 = " " &
  c5 = " "
/* c3 = 3 ; c4 = " " ; c5 = " " is correct */
END
```

Invalid identifier or invalid list separator <ident>

A syntax error was detected in a list of identifiers. It may be either an incorrect B language identifier, or the use of a character other than a comma to separate the elements in the list. The definition of a B language identifier is given in Chapter 1.

```
MACHINE
  M1
CONSTANTS
  c1;c2
PROPERTIES
  c1 : NAT &
  c2 : NAT
END
```

Invalid imported machine <ident_mach>, it uses other machines

A machine that performs USES cannot be referenced in an IMPORTS clause.

```
MACHINE M2
USES M3
END
```

```
IMPLEMENTATION M1_1
REFINES  M1
IMPORTS  M2
END
```

Invalid input format

The specification text contains an incorrectly placed character. This may be a character string that is not closed.

```
MACHINE M1
CONSTANTS
  message
PROPERTIES  /* this string is not closed */
  message = "message title.
END
```

Invalid inputs in <op_header>

The input parameters of an operation must be B language identifiers, separated by commas and distinct from each other. The definition of a B identifier is given in Chapter 1.

```
MACHINE M1
OPERATIONS
  op1(_1) = ...         /* _1 is not an identifier */
;  vv <-- op2(b) = ...  /* b is not an identifier */
;  op3(vv, vv) = ...    /* vv appears twice */
;  op4(vv; ww) = ...    /* the separator must be a comma */
END
```

Invalid label <ident_label> in <ident_elem_rec>'<ident_label>

This message is sent when <ident_label> is not an item of the record element <ident_elem_rec>.

```
MACHINE
  M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : struct(aa : BOOL, bb : BOOL, ee : NAT) &
  cc'dd = 3
      /* cc does not contain the label dd.
         cc'ee = 3 is correct             */
END
```

Invalid label <ident_label> in a record expression

This message is sent when the same label <ident_label> appears more than once in a record expression and when this label is not a B language identifier.

```
MACHINE
  M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : struct(aa : BOOL, bb : BOOL, 2cc : NAT, 2cc : NAT)
    /* 2cc is not a B language identifier.
       2cc appears more than once in the same record expression */
END
```

Invalid list of identifiers in enumerated set definition <enum_def>

An element in an enumerated set must be a B language identifier (refer to the definition in Chapter 1). These elements must all be distinct and separated by commas.

```
MACHINE
  M1
SETS
  ACTIONS = {open-door, close-door};
  E1 = {"string"};
  E2 = {1};
  E4 = {aa, aa};
  E5 = {aa; bb}
END
```

Invalid number of arguments for <subst>

The 'becomes equal' substitution is used with an incorrect number of parameters: the number of variables is different from the number of values to assign.

```
MACHINE M1
VARIABLES
  var1, var2
INVARIANT
  var1 : NAT &
  var2 : NAT
INITIALISATION
  var1, var2 := 0
  /* correct initialisation: var1, var2 := 0,0 */
END
```

Invalid operation call for <ident> assignment

The operation call cannot be used to assign this type of variables.

Invalid operation call for <ident> assignment in <exp>

The operation call cannot be used for the assignment of this type of variables.

Invalid output parameter <exp>

The effective parameter returned by a called up operation cannot be in the form f(x). It is necessary to use an intermediate variable.

```
MACHINE
  M1
INCLUDES
  M2
VARIABLES
  ff
INVARIANT
  ff : 1..5 --> INT
INITIALISATION
  ff :: 1..5 --> INT
OPERATIONS
  op = ff(1) <-- opincluse
END
```

Invalid output parameters in <op_header>

The output parameters of an operation must be B language identifiers, separated by commas and distinct from each other. The definition of a B language identifier is given in Chapter 1.

```
MACHINE M1
OPERATIONS
  a <-- op1 = ...;            /* a is not an identifier */
  _1 <-- op2(ii) = ...;       /* _1 is not an identifier " */
  (tt, tt) <-- op3 = ...;     /* tt appears twice */
  (tt; uu) <-- op4 = ...;     /* the separator must be a comma */
END
```

Invalid predicate <pred>

The predicate <pred> is syntactically incorrect.
This message may be generated when a substitution or an expression is used when a predicate is expected. For example, do not confuse the assignment sign ':=' used in the substitutions only, and the equals sign '=' reserved for predicates.

```
MACHINE MACH
VARIABLES
    var1, var2
INVARIANT
   var1 : NAT &
   var2 : NAT &
   var1 := var2  /* var1 = var2 is correct */
INITIALISATION
    var1:=1 || var2:=1
END
```

Invalid seen machine <ident_mach>, it uses other machines

A machine performing USES cannot be referenced in a SEES clause.

```
MACHINE MAC02
USES
  UMAC01
END
```

```
MACHINE MACH
SEES
  MAC02
END
```

Invalid sequence in <exp>

The operator used in <exp> expects a sequence as an argument.

```
MACHINE
  M1
CONSTANTS
  c1, c2
PROPERTIES
  c1 = size(TRUE) &        /* TRUE is not a sequence */
  c2 = first(UnknownSeq)  /* UnknownSeq is not a sequence */
```

Invalid substitution <subst>

The substitution <subst> is syntactically incorrect. In the case of an operation call-up, the message may be generated if the operation does not exist or is not visible (especially the modification operations from a machine that is seen cannot be used in the ”indicator” component).

```
MACHINE MACH
OPERATIONS
  op1 = BEGIN
    opinc(0)   /* opinc: unknown operation */
  END
;  op2 = BEGIN
    MAXINT     /* MAXINT is not a substitution */
  END
;  op3 = BEGIN
    v1 = v2    /* v1 := v2 is correct */
  END
END
```

Invalid syntax for substitution CASE <subst>

The CASE substitution of the B component analyzed is syntactically incorrect. This message is generated when a mandatory part of the CASE substitution is missing.

```
/*In the following CASE substitution, the second THEN is missing.*/
MACHINE
  M1
OPERATIONS
  op(xx) = PRE xx : NAT THEN
    CASE xx OF
      EITHER 0,1,2 THEN skip
      OR 3,4,5
      END
    END
  END
END
```

Invalid syntax for substitution IF <subst>

The IF substitution in the analyzed component is syntactically incorrect. This message is generated when a mandatory part of the IF is missing.

```
MACHINE
  M1
OPERATIONS
  op1(xx) = PRE xx : NAT THEN
    IF xx = 3 END    /*THEN is missing*/
  END;
  op2(xx) = PRE xx : NAT THEN
    IF xx < 2 THEN skip
    ELSIF xx = 10    /*THEN in ELSIF is missing*/
    END
  END
END
```

Invalid syntax for substitution SELECT <subst>

The SELECT <subst> substitution is syntactically incorrect. This message may be generated when a required part of SELECT is missing.

```
MACHINE
  M1
OPERATIONS
  op(vv) = PRE vv : NAT THEN
    SELECT vv>10   /* THEN is missing */
    WHEN vv=0 THEN
      skip
    ELSE
      skip
    END
  END
END
```

Invalid syntax in operation definition <op>

The operation definition <op> could not be analyzed. This may be due to a syntax problem, or due to a priority level problem. Remember that two operations must be separated by a semi colon.

```
/* In the following OPERATIONS
clause, an analysis error is due
to the precedence of ‘||’ in
relation to ‘=’ */
MACHINE M1
OPERATIONS
  op1 = skip || skip
END
```

```
/* Using BEGIN ... END in this case
will resolve the problem */
MACHINE M1
OPERATIONS
  op1 = BEGIN skip || skip END
END
```

Invalid type for <ident> ; <Expression> contains a record element without label

<ident> designates a not typed data.
<ident> can not be typed by a record element whitout label.

```
MACHINE
  M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc=rec(2,3)   /* rec(2,3) can not be used for typing cc. */
                /* The expression cc = rec(item1:2,item2:3) is correct */
END
```

Invalid use of a record element without label

Two record elements without label can not be compared. This is because a record element without label has a generic type.

```
MACHINE
  M1
ABSTRACT_VARIABLES
  xx,yy
INVARIANT
  xx : NAT &
  yy : BOOL &
  rec(xx,yy) = rec(2,TRUE)
  /* The expression rec(xx,yy) = rec(2,TRUE) is not correct */
  /* xx = 2 &  yy = TRUE is correct              */
INITIALISATION
  xx := 2 ||
  yy := TRUE
END
```

Invalid valuation of <ident_const>

The rules that allow valuing sets and constants were violated. The types of the formal constants defined in the abstraction PROPERTIES clause and the types of the values assigned in the implementation must be identical.
Note that in addition, a set cannot be valued by another set from the same component.

```
MACHINE MACH
SETS
  S1
;  S2
CONSTANTS
  c1
PROPERTIES
  c1 = 1
END
```

```
IMPLEMENTATION MACH_imp
REFINES MACH
VALUES
  S1 = NAT    /* ok */
;  S2 = S1    /* no */
;  c1 = TRUE  /* no */
END
```

<ident_mach> is not a machine

A USES, SEES, INCLUDES, EXTENDS or IMPORTS clause in the analyzed component refers to <ident_mach> which is a refinement or an implementation. Only abstract machines may be covered by a visibility clause.

```
IMPLEMENTATION IMP_1
REFINES IMP
END
```

```
MACHINE MACH
SEES
  IMP_1
  /*an implementation cannot be
    seen*/
END
```

<ident> is not an identifier

Identifier <ident> breaks the syntax rules that define B language identifiers (refer to Chapter 1).

```
MACHINE
  M1
CONSTANTS
  5, _1    /* 5 and _1 are not identifiers */
PROPERTIES
  5 : NAT &
  _1 : INT
END
```

Left hand side and right hand side of <exp> have incompatible type

When using an equals, not equals, an assignment, etc..., the types of the left hand and right hand parts must be identical.
When using operator such as , ><, /:, etc..., some of the rules on types must be verified.
For example, when composing two relations:
relation1; relation2
so that relation1 : A <-> B and relation2 : C <-> D, B and C must be identical.
If this is not the case, the error message is generated.

```
MACHINE MACH
VARIABLES
  v1, v2, v3
CONSTANTS
  relation1
SETS
  EE; FF; GG
PROPERTIES
  relation1 : EE <-> FF
INVARIANT
  v1:NAT &
  v2:BOOL &
  v3:STRING &
  v2/: NAT &                          /* incompatibility */
  v1/=v2                              /* incompatibility */
INITILISATION
  v1:=0 || v2:=TRUE || v3:=""
OPERATIONS
  op1 = v1:= v3                       /* incompatibility */
  vv <-- op2 = vv := 1..2 /\ BOOL;    /* incompatibility */
  vv <-- op5 = vv := relation1 |>> GG;  /* incompatibility */
  vv <-- op7 = vv := EE - FF          /* incompatibility */
END
```

Left hand side in valuation <val> should be an identifier

The left hand side of a valuation must be a B language identifier (refer to the definition in Chapter 1).

```
IMPLEMENTATION
  M1_1
REFINES
  M1
VALUES
  1 = TRUE;
  _1 = 2
END
```

Left hand side of comparison <exp> has not been typed

The left hand side of <exp> has not be typed. This message may be generated when the typing predicates are placed after property <exp>. The definition of a typing predicate is detailed in Chapter 1.

```
MACHINE
  M1(pp)
CONSTRAINTS
  pp <= 1 &   /* pp has not yet been typed*/
  pp : NAT
CONSTANTS
  cc
PROPERTIES
  cc < 2 &    /* cc has not yet been typed*/
  cc : NAT
VARIABLES
  vv
INVARIANT
  vv >= 3 &   /* vv has not yet been typed*/
  vv : NAT
INITIALISATION
  vv := 0
OPERATIONS
  op(ii) = PRE ii >4 & ii : NAT THEN skip END
              /* ii has not yet been typed */
END
/* To correct this specification, simply reverse the predicates */
```

Left hand side of comparison <exp> should be an integer

A comparison can only be made between integers.

```
MACHINE
  M1
CONSTANTS
  cc
PROPERTIES
  cc : BOOL &
  cc >= 1
END
```

Left hand side of <exp> has not been typed

The left hand side of <exp> has not been typed. This message may be generated when the typing predicates are placed after the <exp> property. The definition of a typing predicate is described in Chapter 1.

```
REFINEMENT
  M1
CONSTANTS
  pp
PROPERTIES
  pp /= 1 &   /* pp has not yet been typed*/
  pp : NAT
OPERATIONS
  uu, vv <-- op = BEGIN
    uu := vv; /* vv has not yet been typed*/
    vv := 1
  END
END
```

Left hand side of <exp> should be an integer

The operator used in <exp> expects an integer on its left hand side.

```
MACHINE
  M1
OPERATIONS
  vv <-- op1 = vv := UnknownVar * 2;
  vv <-- op2 = vv := TRUE - 2;
  vv <-- op3 = vv := TRUE mod FALSE
END
```

Left hand side of <exp> should be a relation

The operator used in <exp> expects a relation on its left hand side.

```
MACHINE
  M1
SETS
  EE; FF
VARIABLES
  relation, var
INVARIANT
  relation : EE <-> FF & var : EE
INITIALISATION
  relation :: EE <-> FF || var :: EE
OPERATIONS
  v1 <-- op1 = v1 := (var || relation);
          /* var is not a relation */
  v2 <-- op2 = v2 := (Rinconnue >< relation)
          /* Rinconnue is not a relation */
END
```

Left hand side of <exp> should be a sequence

The operator used in <exp> expects a sequence on its left hand side.

```
MACHINE
  M1
CONSTANTS
  sequence
PROPERTIES
  sequence : seq(INT)
OPERATIONS
  vv <-- op1 = vv :=  2 ^ sequence;    /*2 is not a sequence*/
  vv <-- op2 = vv := UnknownSeq <- 2  /*UnknownSeq is not a sequence*/
END
```

Left hand side of <exp> should be a set

The operator used in <exp> expects a set on its left hand side.

```
MACHINE
  M1
SETS
  SS; TT
VARIABLES
  relation,
  relation2
INVARIANT
  relation : SS <-> TT &
  relation2 : 2 <-> SS         /*2 is not a set*/
INITIALISATION
  relation :: SS <-> TT ||
  relation2 :: UnknownEns <-> SS
                               /* UnknownEns is not a set*/
OPERATIONS
  vv <-- op1 = vv := 3 \/ 1..2;
                               /*3 is not a set*/
  vv <-- op2 = vv := (5 <| relation);
                               /*5 is not a set*/
  vv <-- op4 = vv := TRUE * SS
                               /*TRUE is not a set*/
END
```

Local operation <ident_op> has not been implemented

In an implementation, every local operation defined in the LOCAL_OPERATIONS clause must be implemented in the OPERATIONS clause.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
LOCAL_OPERATIONS
  op = skip
END
/*op should be implemented*/
```

Local variable <ident> is read before being initialised

This message is only generated in implementation. The <ident> local variable is a variable defined in a VAR substitution or in the list of output parameters for an operation. It is used when it has not been initialised by a substitution.

```
IMPLEMENTATION M1_1
REFINES M1
OPERATIONS
  ss, tt <-- op(ii) =
    IF ii > 1 THEN
      ss := 2
    END;
    tt := ss
    /* ss was not initialised in all of the branches of IF */
END
```

Machine <ident_mach> can not be refined, it uses other machines

A machine that performs a USES action cannot be refined, it is an abstract module.

```
MACHINE M1
USES
  M2
END
```

```
REFINEMENT M1_1
REFINES
  M1
  /*refinement impossible*/
END
```

Machine <ident_mach1> should be included in <ident_mach2> : it has been included in the abstraction of <ident_mach2>

Machine <ident_mach1> is included in a component that refines <ident_mach2>. However, <ident_mach2> does not include <ident_mach1>, while some of its abstractions do. This is illegal.
If a component Mi includes a machine N, then:
- none of its refinements includes or imports an N, or
- one of its refinements Mj includes or imports an N, and in this case ALL of the components of the refinement string between Mi and Mj must include N.
This constraint is used to avoid certain identifier conflicts.

```
MACHINE M1
INCLUDES M2
END
```

```
REFINEMENT M1_1
REFINES M1
END
```

```
REFINEMENT M1_2
REFINES M1_1
INCLUDES M2  /* illegal if M1_1 does not include M2*/
END
```

Machine <ident_mach1> should be seen by <ident_mach2>

The analyzed component M1 includes two machines M2 and M4 so that M2 used M4. The gluing invariant that links the variables of M2 and M4 is defined in the INVARIANT clause of M2 but must be proven at M1 level.
In the context that generates this message, M2 sees a machine M3 and the variables of M3 are involved in the gluing invariant linking M2 and M4. However component M1 does not see M3, and therefore it does not know anything about its variables. The proof is bound to fail. It is therefore necessary to add M3 to the SEES clause in M1.

```
MACHINE M3
VARIABLES
  v3
INVARIANT
  v3 : NAT
INITIALISATION
  v3 := 0
END
```

```
MACHINE M4
VARIABLES
  v4
INVARIANT
  v4 : NAT
INITIALISATION
  v4 := 10
END
```

```
MACHINE M2
SEES
  ss.M3
USES
  uu.M4
VARIABLES
  v2
INVARIANT
  v2 : NAT &
  /*gluing invariant M2/M4 : */
  v2 < ss.v3 + uu.v4
INITIALISATION
  v2 :: NAT
END
```

```
MACHINE M1
INCLUDES
  M2,
  uu.M4
/* is missing:
SEES
  ss.M3
*/
END
```

Machine <ident_mach1> should be seen by <ident_mach2> (it is seen by <ident_mach3>)

If a machine is seen by a component, it must remain so by all of the components that come after it in the refinement string. This message is therefore generated if a component M is refined by a component N so that machine S appears in the SEES clause of M but not in that of N.

```
MACHINE M1
SEES M2
END
```

```
REFINEMENT M1_1
REFINES M1
/* missing:
SEES M2 */
END
```

Machine <ident_mach> should have parameters

The analyzed component contains a CONSTRAINTS clause while it does not have parameters, but this clause only allows defining the properties of component parameters.

```
MACHINE
  M1
CONSTANTS
  c1
CONSTRAINTS
  c1 : NAT   /* predicate to place in a PROPERTIES clause */
END
```

Machine <ident_mach1> uses <ident_mach2> which is neither included nor extended

When a machine that performs a USES action is included, all of the machines used must also be included. For example, if M1 uses M2 that uses M3, then if M2 is included, M1 and M3 must also be included.

```
MACHINE MAC02
USES
  UMAC01
END
```

```
MACHINE MACH
INCLUDES
  MAC02
  /* UMAC01 must also be included */
END
```

Missing symbol => in predicate <pred>

This message concerns expressions in the form !X.A. It is generated when A is not in the form (P => Q). Predicate P must contain the typing predicates for the variables of X. The definition of a typing predicate is described in Chapter 1.

```
MACHINE
  M1
CONSTANTS
  vv
PROPERTIES
  vv : 1..10 &
  !xx.(xx : NAT & xx >5 & xx > vv)
  /* correct notation:
  !xx.(xx : NAT & xx > 5 => xx > vv)
  */
END
```

Multiple assignment of <ident_var> in parallel substitutions

The same variable cannot be assigned in more than one branch in a simultaneous substitution.

```
/*The following machine attempts
to give variable v1, the value 0
and the value 1 in parallel.
It is incorrect.*/
MACHINE
  MACH
VARIABLES
  v1
INVARIANT
  v1:NAT
INITIALISATION
  v1:=0 ||
  v1:=1
END
```

```
/*The following machine proposes
multiple solutions to correctly
express the intuitive idea that
was implemented opposite,
i.e. for v1 to equal 0, or 1.*/
MACHINE
  MACH
VARIABLES
  v1
INVARIANT
  v1:NAT
INITIALISATION
  v1 :: {0,1}
/*v1:(v1=0 or v1=1)
  CHOICE v1:=0 OR v1:=1 END
  are also possible*/
END
```

Multiple assignment of <ident> when calling local operation <ident_op>

Local operation <ident_op> modifies variable <ident>. When called, one of its effective output parameter is also variable <ident>. Thus, the operation call is incorrect.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
CONCRETE_VARIABLES
  vv
INVARIANT
  vv : INT
INITIALISATION
  vv := 1
LOCAL_OPERATIONS
  ss <-- loc_op = BEGIN ss :: INT || vv :: NAT END
OPERATIONS
  ss <-- loc_op = BEGIN ss := 1; vv := 2 END;
  op = BEGIN vv <-- loc_op END
  /* a correct version would be :
       VAR tmp IN tmp <-- loc_op; vv := tmp END */
END
```

Multiple definition of identifier <ident> (because of the INCLUDES clause transitivity used for <ident_mch1>)

Identifier <ident> is defined both in the analyzed component and in a visible component. This conflict may be due to the transitivity of the INCLUDES clause.

Multiple definition of identifier <ident> in <ident_mach>

The analyzed component contains an internal identifier conflict.

```
MACHINE MACH
CONSTANTS
  cst1,cst2,cst2    /* cst2 appears twice */
PROPERTIES
  cst1 : NAT & cst2 : NAT
END
```

Multiple promotion of operation <ident_op>

Each promoted operation must only be mentioned once.

```
MACHINE M1
INCLUDES M2
PROMOTES
    op1, op1
END
```

Multiple reference of machine <ident_mach>

The same machine must only appear once in the INCLUDES, IMPORTS, EXTENDS, SEES, USES clauses of the same component.

```
MACHINE M1
INCLUDES M2
SEES M2
/* problem as M2 appears twice */
END
```

Multiple use of constant <ident_cst> in branches of CASE

The same constant <ident_cst> appears more than once in the branches of a CASE substitution, whereas the different cases in a substitution must be mutually exclusive.

```
MACHINE
  M1
OPERATIONS
  out <-- op(in) =
  PRE in : NAT THEN
    CASE in OF
      EITHER 0,1,2 THEN out:=0
      OR 2,3,4 THEN out:=1  /* 2 appears again */
      END
    END
  END
END
```

Multiple use of identifier <ident> in branches of CASE

The same constant appears more than once in the branches of a substitution CASE, whereas the different cases in a substitution must be mutually exclusive.

```
MACHINE
  M1
CONSTANTS
  yy
PROPERTIES
  yy : NAT
OPERATIONS
  out <-- op(in) = PRE in : NAT THEN
    CASE in OF
      EITHER yy THEN out := 1
      OR yy THEN out := 2    /*yy appears again */
      ELSE out := 3
      END
    END
  END
END
```

Multiple use of label <ident_label> in a record expression

The labels contained in a record set or in a record element must be distinct from each other.

```
MACHINE
  M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : struct(aa:NAT,bb:BOOL,aa:0..9)
    /* Replace the expression aa:0..9 by ee:0..9 */
END
```

Object <ident> cannot be valued

The <ident> object is valued, when it is not valuable or unknown. This may be a typing error or a visibility problem.

```
MACHINE M1
SETS
  S1; S2
CONSTANTS
  c1
PROPERTIES
  c1 = 1
END
```

```
IMPLEMENTATION M1_1
REFINES M1
VALUES
  S1 = NAT
; S2 = NAT1
; c1 = 1
; c2 = 1  /* c2 unknown */
END
```

<ident_op> of machine <ident_mch> is called simultaneously with a modification of variable <ident_var>

A local operation can modify directly an imported variable. This message is produced when one modifies an imported variable in parallel with a call to an operation of the same imported machine.

```
IMPLEMENTATION M1_1
REFINES M1
IMPORTS M0
LOCAL_OPERATIONS
  loc_op = BEGIN
    increment ||
    v2 := v2-1
  END
  /* M0 invariant is broken up */
END
```

```
MACHINE M0
VARIABLES
  v1,v2
INVARIANT
  v1:NAT & v2:NAT & v1<=v2
INITIALISATION
  v1:=0 || v2:=0
OPERATIONS
  increment = PRE
    v1<v2
  THEN
    v1:=v1+1
  END
END
```

Only one ABSTRACT_CONSTANTS clause is allowed

This message is produced when an ABSTRACT_CONSTANTS clause should not take place in the analyzed component. In particular, there cannot be two ABSTRACT_CONSTANTS clauses in the same component, or an ABSTRACT_CONSTANTS clause and a HIDDEN_CONSTANTS clause. Both keywords have the same meaning indeed.

```
MACHINE M1
ABSTRACT_CONSTANTS
  cst1
HIDDEN_CONSTANTS
  cst2
PROPERTIES
  cst1 : NAT & cst2 : NAT
END
```

Only one ABSTRACT_VARIABLES clause is allowed

This message is produced when an ABSTRACT_VARIABLES clause should not take place in the analyzed component. In particular, it is illegal to have two ABSTRACT_VARIABLES clauses in the same machine, or an ABSTRACT_VARIABLES clause anda VARIABLES or HIDDEN_VARIABLES clause, as these three keywords have the same meaning.

```
MACHINE MACH
ABSTRACT_VARIABLES
    v1
VARIABLES
    v2
INVARIANT
    v1:NAT &
    v2:NAT
INITIALISATION
    v1:=0 ||
    v2:=0
END
```

Only one ASSERTIONS clause is allowed

This message is sent when an ASSERTIONS clause should not take place in the analyzed component. In particular having two ASSERTIONS clauses is forbidden.

```
MACHINE MACH
ASSERTIONS
    TRUE
ASSERTIONS
    TRUE
END
```

Only one component can be refined: <ident_mach> is chosen for the Type Check continuation

The REFINES clause in the analyzed refinement or the implementation refers to a number of machines. This is illegal, as two components cannot be refined at the same time.
In this case the check continues with as refined component the last in the list. This is the component name that appears in the error message.

```
REFINEMENT M1_1
REFINES
    M1a, M1b
END
```

Only one CONCRETE_CONSTANTS clause is allowed

This message is generated when a CONCRETE_CONSTANTS clause should not take place in the analyzed component. In particular, there cannot be two CONCRETE_CONSTANTS clauses in the same component, or a CONCRETE_CONSTANTS clause and a VISIBLE_CONSTANTS or CONSTANTS clause, as these three keywords have the same meaning.

```
MACHINE M1
CONCRETE_CONSTANTS
  cst1
CONSTANTS
  cst2
PROPERTIES
  cst1 : NAT & cst2 : NAT
END
```

Only one CONCRETE_VARIABLES clause is allowed

This message is generated when a CONCRETE_VARIABLES clause does not have its place in the analyzed component. In particular, it is illegal to have two CONCRETE_VARIABLES clauses, or a CONCRETE_VARIABLES clause and a VISIBLE_VARIABLES clause, as these two keywords have these same meaning.

```
MACHINE MACH
CONCRETE_VARIABLES
    v1
VISIBLE_VARIABLES
    v2
INVARIANT
    v1:NAT &
    v2:NAT
INITIALISATION
    v1:=0 ||
    v2:=0
END
```

Only one CONSTANTS clause is allowed

This message is generated when a CONSTANTS clause does not have its place in the analyzed component. In particular, there cannot be two CONSTANTS clauses in the same component, or a CONSTANTS clause and a VISIBLE_CONSTANTS or CONCRETE_CONSTANTS clause. This is because these three keywords have the same meaning.

```
MACHINE M1
CONSTANTS
  cst1
CONCRETE_CONSTANTS
  cst2
PROPERTIES
  cst1 : NAT & cst2 : NAT
END
```

Only one CONSTRAINTS clause is allowed

This message is generated when a CONSTRAINTS clause does not have its place in the analyzed component. In particular, having two CONSTRAINTS clauses in the same component is not allowed.

```
MACHINE
  M1(xx, yy)
CONSTRAINTS
  xx : NAT
CONSTRAINTS
  yy : NAT
END
```

Only one EXTENDS clause is allowed

This message is generated when an EXTENDS does not have its place in the analyzed component. In particular, having two EXTENDS clauses in the same component is impossible.

```
MACHINE MACH
EXTENDS
    MAC1(NAT)
EXTENDS
    MAC2(1..100,BOOL)
END
```

Only one HIDDEN_CONSTANTS clause is allowed

This message is generated when a HIDDEN_CONSTANTS clause does not have its place in the analyzed component. In particular, there cannot be two HIDDEN_CONSTANTS clauses in the same component, or a HIDDEN_CONSTANTS clause and an ABSTRACT_CONSTANTS clause. This is because these two keywords have the same meaning.

```
MACHINE M1
HIDDEN_CONSTANTS
  cst1
ABSTRACT_CONSTANTS
  cst2
PROPERTIES
  cst1 : NAT & cst2 : NAT
END
```

Only one HIDDEN_VARIABLES clause is allowed

This message is generated when a HIDDEN_VARIABLES clause does not have its place in the analyzed component. In particular, it is illegal to have two HIDDEN_VARIABLES clauses in the same machine, or a HIDDEN_VARIABLES clause and a VARIABLES or ABSTRACT_VARIABLES clause. These three keywords have the same meaning.

```
MACHINE MACH
HIDDEN_VARIABLES
    v1
ABSTRACT_VARIABLES
    v2
INVARIANT
    v1:NAT &
    v2:NAT
INITIALISATION
    v1:=0 ||
    v2:=0
END
```

Only one IMPORTS clause is allowed

This message is generated when an IMPORTS clause does not have its place in the analyzed implementation. In particular, it is illegal to have two IMPORTS clauses in the same implementation.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
IMPORTS
  M2
IMPORTS
  M3
END
```

Only one INCLUDES clause is allowed

This message is generated when an INCLUDES clause does not have its place in the analyzed component. In particular, it is illegal to have two INCLUDES clauses in the same component.

```
MACHINE
  M1
INCLUDES
  M2
INCLUDES
  M3
END
```

Only one INITIALISATION clause is allowed

This message is generated when an INITIALISATION clause does not have its place in the analyzed component. In particular, it is illegal to have two INITIALISATION clauses in the same component.

```
MACHINE MACH
VARIABLES
    v1,v2
INVARIANT
    v1:NAT &
    v2:NAT
INITIALISATION
    v1:=0    /* v1:=0 || v2:=0 is correct */
INITIALISATION
    v2:=0
END
```

Only one INVARIANT clause is allowed

This message is generated when an INVARIANT does not have its place in the analyzed component. In particular, it is illegal to have two INVARIANT clauses in the same component.

```
MACHINE MACH
VARIABLES
    v1,v2
INVARIANT
    v1:NAT    /* v1:NAT & v2:NAT are correct */
INVARIANT
    v2:NAT
INITIALISATION
    v1:=0 ||
    v2:=0
END
```

Only one LOCAL_OPERATIONS clause is allowed

This message is produced when a LOCAL_OPERATIONS clause should not take place in the analysed component. In particular, it is forbidden to have two LOCAL_OPERATIONS clause in the same component.

```
IMPLEMENTATION MM_1
REFINES MM
LOCAL_OPERATIONS
    op1 = BEGIN
      skip
    END
LOCAL_OPERATIONS
    op2 = BEGIN
      skip
    END
END
```

Only one OPERATIONS clause is allowed

This message is sent when an OPERATIONS clause no longer has its place in the analyzed component. In particular, it is illegal to have two OPERATIONS clauses in the same component.

```
MACHINE MACH
OPERATIONS
    op1 = BEGIN
      skip
    END
OPERATIONS
    op2 = BEGIN
      skip
    END
END
```

Only one PROMOTES clause is allowed

This message is generated when a PROMOTES clause does not have its place in the analyzed component. In particular, it is illegal to have two PROMOTES clauses in the same component. All of the promoted operations must appear in the same PROMOTES clause, even if they come from different machines.

```
MACHINE MACH
INCLUDES MAC01(10), MAC02(1..1000, BOOL)
PROMOTES
    op_01
PROMOTES
    op_02  /* not correct */
END
```

Only one PROPERTIES clause is allowed

This message is sent when a PROPERTIES clause does not have its place in the analyzed component. In particular, it is illegal to have two PROPERTIES clauses in the same component.

```
MACHINE MACH
CONSTANTS
    c1, c2
PROPERTIES
    c1 :NAT
PROPERTIES
    c2 :NAT
END
```

Only one REFINES clause is allowed

This message is generated when a REFINES clause does not have its place in the analyzed component. In particular, it is illegal to have two REFINES clauses in the same component. This is illegal as the two components cannot be refined at the same time.

```
REFINEMENT
  M1_1
REFINES
  M1
REFINES
  M2
END
```

Only one SEES clause is allowed

This message is generated when a SEES clause does not have its place in the analyzed component. In particular, it is illegal to have two SEES clauses in the same component.

```
MACHINE MACH
SEES
  SEE01
SEES
  SEE02
END
```

Only one SETS clause is allowed

This message is generated when a SETS clause does not have its place in the analyzed component. In particular, it is illegal to have two SETS clauses in the same component.

```
MACHINE MACH
SETS
    S1
SETS
    S2
END
```

Only one USES clause is allowed

This message is generated when a USES clause does not have its place in the analyzed component. In particular, it is illegal to have two USES clauses in the same.

```
MACHINE MACH
USES
  MAC1    /* MAC1, MAC2 is correct */
USES
  MAC2
END
```

Only one VALUES clause is allowed

This message is generated when a VALUES clause does not have its place in the analyzed component. In particular, it is illegal to have two VALUES clauses in the same implementation.

```
IMPLEMENTATION IMP
REFINES
  REF
VALUES
  S1 = NAT
VALUES
  S2 = INT
END
```

Only one VARIABLES clause is allowed

This is generated when a VARIABLES clause does not have its place in the analyzed component. In particular, it is illegal to have two VARIABLES clauses in the same component, or a VARIABLES clause and a HIDDEN_VARIABLES or ABSTRACT_VARIABLES clause, as these three keywords have the same meaning.

```
MACHINE MACH
VARIABLES
    v1
ABSTRACT_VARIABLES
    v2
INVARIANT
    v1:NAT &
    v2:NAT
INITIALISATION
    v1:=0 ||
    v2:=0
END
```

Only one VISIBLE_CONSTANTS clause is allowed

This message is generated when a VISIBLE_CONSTANTS clause does not have its place in the analyzed component. In particular, there cannot be two VISIBLE_CONSTANTS clauses in the same component, or a VISIBLE_CONSTANTS clause and a CONSTANTS or CONCRETE_CONSTANTS clause. This is because these three keywords have the same meaning.

```
MACHINE M1
VISIBLE_CONSTANTS
  cst1
CONCRETE_CONSTANTS
  cst2
PROPERTIES
  cst1 : NAT & cst2 : NAT
END
```

Only one VISIBLE_VARIABLES clause is allowed

This message is generated when a VISIBLE_VARIABLES clause does not have its place in the analyzed component. In particular, it is illegal to have two VISIBLE_VARIABLES clauses, or a VISIBLE_VARIABLES clause and a CONCRETE_VARIABLES clause, as the two keywords have the same meaning.

```
MACHINE MACH
VISIBLE_VARIABLES
    v1
CONCRETE_VARIABLES
    v2
INVARIANT
    v1:NAT &
    v2:NAT
INITIALISATION
    v1:=0 ||
    v2:=0
END
```

Operation <ident_op> does not exist in <mach>

The operation <ident_op> appears in the PROMOTES clause of the analyzed component, but is not defined in its abstraction.
When an operation is promoted, it is considered as having been written in the component itself. However, in a refinement, local operations can only be refinements of abstract machine operations, with exactly the same signature.

```
MACHINE M1
OPERATIONS
  res <-- op2 (xx,yy
  PRE
    xx:1..100 & yy:1..100
  THEN
    res :: BOOL
  END
END
```

```
MACHINE M2(ENS)
OPERATIONS
  op1 = skip
; res <-- op2 (xx,yy) = PRE
    xx:ENS & yy:ENS
  THEN
    res:=bool(xx<=yy)
  END
END
```

```
REFINEMENT M1_1
REFINES M1
EXTENDS
  M2(NAT) /* op1 produces an error message as it does not correspond to
       any operation in machine M1 */
END
```

Operation <ident_op> does not exist in abstraction

The local operations of a refinement or an implementation must be specified in the abstract machine. You cannot define a new operation in a refinement.

```
MACHINE M1
OPERATIONS
  res <-- op1 (xx,yy) =
  PRE
    xx:1..100 & yy:1..100
  THEN
    res :: BOOL
  END
END
```

```
REFINEMENT M1_1
REFINES M1
OPERATIONS
  op2 = skip
  /*op2 does not exist in M1*/
END
```

Operation <ident_op> has not been implemented

In an implementation, all of the operations defined in the abstract machine must be implemented.

```
MACHINE
  M1
OPERATIONS
  op = skip
END
```

```
IMPLEMENTATION
  M1_1
REFINES
  M1
END
/*op must be implemented*/
```

Operation name <ident_op> in <op_header> is a keyword

<ident_op> is a reserved word in B language (refer to Chapter 1): it cannot be used to name an operation.

```
MACHINE M1
OPERATIONS
  MAXINT(xx) = ...     /* MAXINT: reserved word */
;  res <-- skip = ...  /* skip: reserved word */
END
```

Operation name <ident_op> in <op_header> should be an identifier

The name of the operations must be a simple name, i.e. a B language identifier (refer to the definition in Chapter 1).

```
MACHINE M1
OPERATIONS
  _1 <-- val = ...     /* _1 is not an identifier */
;  res <-- f(x) = ...  /* f is not an identifier */
END
```

Output parameter <ident> has not been initialised

This message is only generated in implementation. The <ident> output parameter for the operation currently being type checked was not initialised by the body of this operation.

```
IMPLEMENTATION M1_1
REFINES M1
OPERATIONS
  ss <-- op(ii) =
    IF ii > 1 THEN
      ss := 2
    END
END
/* ss was not initialised in all branches of IF */
```

Output parameters <list_ident> have not been initialised

This message is only generated in implementation. The <list_ident> output parameters for the operation currently being type checked were not initialised by the body of this operation.

```
IMPLEMENTATION M1_1
REFINES M1
OPERATIONS
  ss, tt <-- op(ii) =
    IF ii > 1 THEN
      ss := 2
    ELSE
      tt := 3
    END
END
/* ss and tt were not typed in all of the branches of IF */
```

Parameter <ident> has not been typed

All scalar parameters must be typed in the CONSTRAINTS clause using a typing predicate (refer to the definition in Chapter 1).

```
MACHINE MACH(par1,par2,par3)
CONSTRAINTS
  par1 : NAT &
  par2 < par1  /* par2 not typed */
END           /* par3 not typed */
```

Parameter <ident> of <ident_op> is already defined in <ident_mach>

A conflict between the input/output parameters of the promoted operation <ident_op> and a visible identifier of machine <ident_mach> was detected.

```
MACHINE M2
OPERATIONS
  op1(xx) = PRE xx:NAT THEN
    skip
  END
END
```

```
MACHINE M1
INCLUDES M2
PROMOTES
  op1
VARIABLES
  xx
 /* conflict with xx in op1 */
INVARIANT
  xx : NAT
INITIALISATION
  xx :: NAT
END
```

Parameters of abstraction <ident_mch1> and refinement <ident_mch2> differ

All of the refinements of a vertical development must have the same parameters as the abstract machine (the number and the name of the parameters must be identical).

```
MACHINE MACH(var1,var2,ENS)
CONSTRAINTS
  var1 : ENS &
  var2 : ENS
END
```

```
REFINEMENT MACH_1(var,ENS)
  /* var is surplus;
     var1 and var2 are missing */
REFINES
  MACH
END
```

Prefix <ident1> in <ident1>.<ident2> is a keyword

The <ident1> prefix is a reserved word in the language (refer to Chapter 1). It cannot be used to prefix a machine.

```
MACHINE M1
SEES skip.M0
END
```

Prefix in <ident> should be an identifier

A renaming prefix must be a correct B language identifier (refer to the definition in Chapter 1).

```
MACHINE MACH
INCLUDES
  1.MAC1 ,
  #10x.MAC1 ,
  <>.MAC1
END
```

Prefix <ident> is used twice

For a given component each renaming prefix can only be used once, even if it is renamed as a separate machine.

```
MACHINE MACH
INCLUDES
  pref.INC01
EXTENDS
  pref.INC02
END
```

<exp> ran(<exp>) should be a set of sets

The operator used in <exp> expects as its argument a function with a starting set that is a set of sets.

```
MACHINE
  M1
SETS
  SS; TT
CONSTANTS
  fonction,
  relation
PROPERTIES
  fonction : SS --> TT &
  relation = rel(fonction)
    /* TT should be a set of sets */
END
```

Read only or unknown left hand side <ident>

This error message is generated when the becomes "becomes equal" or "call-up operation" substitution attempts to modify an entity that cannot be modified. The visibility tables show which entities are accessible in write mode and which are not, depending on which clause is considered.

```
MACHINE M1
CONSTANTS
  c1
PROPERTIES
  c1 : NAT
OPERATIONS
  ini = (c1, UnknownId := 0, 0)
         /* c1 constant that cannot be modified,
            UnknownId unknown identifier */
END
```

Refined component <ident> cannot be renamed

The name of the component that appears in the REFINES clause is preceded by a renaming prefix. This is illegal.

```
/* Incorrect refinement: */
REFINEMENT M1_1
REFINES pp.M1
END
```

```
/* Correct refinement: */
REFINEMENT M1_1
REFINES M1
END
```

Right hand side of comparison <exp> has not been typed

The right hand side of <exp> has not been typed. This message may be generated when the typing predicates are placed after the <exp> property. The definition of a typing predicate is described in Chapter 1.

```
MACHINE
  M1(pp)
CONSTRAINTS
  1 < pp &   /* pp has not yet been typed*/
  pp : NAT
CONSTANTS
  cc
PROPERTIES
  2 <= cc &  /* cc has not yet been typed*/
  cc : NAT
VARIABLES
  vv
INVARIANT
  3 > vv &   /* vv has not yet been typed*/
  vv : NAT
INITIALISATION
  vv := 0
OPERATIONS
  op(ii) = PRE 4 >= ii & ii : NAT THEN skip END
             /* ii has not yet been typed*/
END
/* To correct this specification, simply reverse the predicates */
```

Right hand side of comparison <exp> should be an integer

A comparison can only be made between integers.

```
MACHINE
  M1
CONSTANTS
  cc
PROPERTIES
  cc : BOOL &
  2 <= cc
END
```

Right hand side of <exp> has not been typed

The right hand side of <exp> has not been typed. This message may be generated when the typing predicates are placed after the <exp> property. The definition of a typing predicate is described in Chapter 1.

```
REFINEMENT
  M1
VARIABLES
  pp
INVARIANT
  1 /= pp &   /* pp has not yet been typed*/
  pp : NAT
INITIALISATION
  pp := 4
OPERATIONS
  uu, vv <-- op = BEGIN
    uu := 1;
    IF vv = 1 THEN /* vv has not yet been typed*/
      vv := 2
    END
  END
END
```

Right hand side of <exp> should be an integer

The operator used in <exp> expects an integer on its right hand part.

```
MACHINE
  M1
OPERATIONS
  vv <-- op1 = vv := 2 * UnknownVar;
  vv <-- op2 = vv := 2 - TRUE;
  vv <-- op3 = vv := TRUE mod FALSE
END
```

Right hand side of <exp> should be a relation
The operator used in <exp> expects a relation on the right hand side.

```
MACHINE
  M1
SETS
  EE; FF
VARIABLES
  relation, var
INVARIANT
  relation : EE <-> FF & var : EE
INITIALISATION
  relation :: EE <-> FF || var :: EE
OPERATIONS
  v1 <-- op1 = v1 := (relation || var);
          /* var is not a relation */
  v2 <-- op2 = v2 := (relation >< Rinconnue)
          /* Rinconnue is not a relation */
END
```

Right hand side of <exp> should be a sequence
The operator used in <exp> expects a sequence on its right hand side.

```
MACHINE
  M1
PROPERTIES
  sequence : seq(INT)
OPERATIONS
  vv <-- op1 = vv := sequence ^ 2;
      /* 2 is not a sequence */
  vv <-- op2 = vv := a1 -> UnknownSeq
      /* UnknownSeq is not a sequence */
END
```

Right hand side of <exp> should be a set

The operator used in <exp> expects a set on the right hand side.

```
MACHINE
  M1
SETS
  SS; TT
VARIABLES
  relation1, relation2
INVARIANT
  relation1 : SS <-> TT
INITIALISATION
  relation1 :: SS <-> TT
OPERATIONS
  vv <-- op2 = vv := 1..2 /\ UnknownEns;
                                    /* UnknownEns is not a set */
  vv <-- op3 = (vv :: SS --> 5);    /*5 is not a set */
  vv <-- op4 = vv := SS - TRUE      /*TRUE is not a set*/
END
```

Seen machine <ident_mach> cannot be instanciated

Only the machines referenced in the INCLUDES, IMPORTS and EXTENDS clauses can be instanced.

```
MACHINE MACH
SEES
  MCH01(NAT)
END
```

Sequence in <exp> should not be empty

The operator used in <exp> expects a non empty sequence as an argument.

```
MACHINE
  M1
CONSTANTS
  c1
PROPERTIES
  c1 = first(<>)   /* first awaits as argument a non empty sequence */
END
```

Sequencing substitution is forbidden in a local operation specifications : <subst>

This message is produced when a sequencing substitution ";" is used in a local operation specification, as this substituion is not allowed in specification. The simultaneous substituion "||" is recommended instead.

```
IMPEMENTATION
  MM_1
REFINES
  MM
CONCRETE_VARIABLES
  v1, v2
INVARIANT
  v1 : NAT & v2 : NAT
INITIALISATION
  v1 := 0; v2 := 0
LOCAL_OPERATIONS
  op = BEGIN
    v1 := 0; v2 := 0
/* correct: v1:=0 || v2 := 0 or v1,v2 := 0,0 */
  END
OPERATIONS
  op = BEGIN
    v1 := 0; v2 := 0
  END
/* here, it's allowed */
END
```

Sequencing substitution is forbidden in a machine: <subst>

This message is generated when the sequencing substitution ";" is used in an abstract machine. However this substitution is only allowed in refinement and in implementation modes. However, the simultaneous substitution "||" is recommended in specification mode.

```
MACHINE MACH
VARIABLES
  v1, v2
INVARIANT
  v1 : NAT & v2 : NAT
INITIALISATION
  v1 := 0; v2 := 0
/*write correct: v1:=0 || v2 := 0 or v1,v2 := 0,0 */
END
```

Set <ident_set> is already defined

An identifier conflict involving the <ident_set> set was detected.

```
MACHINE MACH
SETS
    S1;S1
END
```

The ABSTRACT_CONSTANTS clause is not allowed in an implementation

The ABSTRACT_CONSTANTS clause cannot be used in an implementation. In this case it is preferable to use the VISIBLE_CONSTANTS clause.

```
IMPLEMENTATION M1_1
REFINES
  M1
ABSTRACT_CONSTANTS
  cst
PROPERTIES
  cst : NAT
END
```

The ABSTRACT_VARIABLES clause is not allowed in an implementation

The ABSTRACT_VARIABLES clause cannot be used in an implementation. In this case it is preferable to use the CONCRETE_VARIABLES clause.

```
IMPLEMENTATION
  M1
REFINES
  M1
ABSTRACT_VARIABLES
  v1
INVARIANT
  v1 : NAT
INITIALISATION
  v1 := 0
END
```

The component <ident_mach> cannot be referenced by itself

A B language component cannot be referenced by itself in one of its SEES, INCLUDES, EXTENDS or USES clauses.

```
MACHINE MACH(XX)
CONSTRAINTS
  card(XX)=5
INCLUDES
  MACH(1..5) /* illegal attempt at recursivity */
END
```

The CONSTRAINTS clause is only allowed in a machine

The analyzed component should not contain a CONSTRAINTS clause. This message is generated in a refinement or in an implementation when attempting to specify parameter constraints. These constraints must be specified exclusively in the abstract machine.

```
REFINEMENT
  M1(xx, yy)
REFINES
  M1
CONSTRAINTS
  xx : NAT & yy : NAT
END
```

The HIDDEN_CONSTANTS clause is not allowed in an implementation

The HIDDEN_CONSTANTS clause cannot be used in an implementation. In this case it is preferable to use the VISIBLE_CONSTANTS clause.

```
IMPLEMENTATION M1_1
REFINES
  M1
HIDDEN_CONSTANTS
  cst
PROPERTIES
  cst : NAT
END
```

The HIDDEN_VARIABLES clause is not allowed in an implementation

The HIDDEN_VARIABLES clause cannot be used in an implementation. In this case it is preferable to use the CONCRETE_VARIABLES clause.

```
IMPLEMENTATION
  M1
REFINES
  M1
HIDDEN_VARIABLES
  v1
INVARIANT
  v1 : NAT
INITIALISATION
  v1 := 0
END
```

The implementation <ident_mach> cannot be refined

The analyzed component refines an implementation. However, only abstract machines and refinements can be refined. The implementation is the final step in a vertical development (development by successive refinements).

```
IMPLEMENTATION IMP
REFINES MACH
END
```

```
REFINEMENT REF
REFINES IMP    /*error*/
END
```

The IMPORTS clause is only allowed in an implementation

This message is generated when an abstract machine or a refinement contains an IMPORTS clause. This is exclusively reserved for the implementation. However, the INCLUDES clause may be used.

```
MACHINE Mach
IMPORTS
  ImpMch0(10)
END
```

The INCLUDES clause is not allowed in an implementation

This message is generated when an implementation contains an INCLUDES clause. This is only allowed in abstract machines and in refinements. However, the IMPORTS clause, dedicated to the implementation, may be used.

```
IMPLEMENTATION M1_1
REFINES M1
INCLUDES
  IncMch04(10)
END
```

The LOCAL_OPERATIONS clause is only allowed in an implementation

This message is produced when an abstract machine or a refinement contains a LOCAL_OPERATIONS clause. The latter can only be used in implementations.

```
MACHINE Mach
LOCAL_OPERATIONS
  op = skip
END
```

The refined machine <ident_mach> cannot be required

The abstract machine refined by the analyzed component cannot appear in any of its visibility clauses.

```
REFINEMENT MAC02
REFINES MACH
INCLUDES
    MACH  /* MACH cannot be included */
END
```

The REFINES clause is not allowed in a machine

This message is sent when a REFINES clause appears in an abstract machine, when an abstract machine cannot refine a B language component. Only a refinement (identified by the first word of the REFINEMENT source) and an implementation (identified by the first word in the IMPLEMENTATION source) can (and must) contain a REFINES clause.

```
MACHINE M0
REFINES
  M1
END
```

The REFINES clause missing

The analyzed refinement or implementation does not have a REFINES clause. This clause is mandatory.

```
REFINEMENT REF_1
END
```

The USES clause is only allowed in a machine

This message is generated when a refinement or an implementation contains a USES clause. This clause is only allowed in an abstract machine.

```
REFINEMENT
  M1_1
REFINES
  M1
USES
  M2
END
```

The VALUES clause is only allowed in an implementation

This message is generated when an abstract machine or refinement contains a VALUES clause. The valuation of constants and sets is only possible in an implementation.
The PROPERTIES clause may possibly force a constant to take a given value, but it will still have to be valued, with the same value, in the implementation.

```
MACHINE
  M1
CONSTANTS
  c1
VALUES
  c1 = 0
END
```

The VARIABLES clause is not allowed in an implementation

This message is generated when an implementation contains a VARIABLES clause. This clause is equivalent to the HIDDEN_VARIABLES clause and it cannot therefore be used in an implementation. In this case it is preferable to use the CONCRETE_VARIABLES clause.

```
IMPLEMENTATION
  M1
REFINES
  M1
VARIABLES
  v1
INVARIANT
  v1 : NAT
INITIALISATION
  v1 := 0
END
```

Unknown renamed identifier: <ident1>.<ident2>

Form <ident1>.<ident2> is a renaming: it designates the identifier <ident2> defined in a requested machine renamed using the <ident1> prefix.
This message is generated when identifier <ident2> is visible in none of the machines renamed with the <ident1> prefix. This may be due to a typing error or violation of the visibility constraints.

```
MACHINE M1
SEES
  pp.M2
END
```

```
MACHINE M2
ABSTRACT_CONSTANTS
  cst2
PROPERTIES
  cst2 : NAT
END
```

```
REFINEMENT M1_1
REFINES M1
ABSTRACT_CONSTANTS
  pp.cst2
PROPERTIES
  pp.cst2 : NAT
END
```

Used machine <ident_mach> cannot be instanciated

Only the machines referenced in the INCLUDES, IMPORTS and EXTENDS clauses can be instanciated.

```
MACHINE MACH
USES
  MCH01(NAT)
END
```

Use of non implementable arrays in <exp>

This message is generated for an implementation. An array is not implementable in B0 if its array is not an interval or an enumerated set.

```
IMPLEMENTATION M1_1
REFINES M1
VISIBLE_CONSTANTS
  cc
PROPERTIES
  cc : INTEGER --> BOOL
CONCRETE_VARIABLES
  vv
INVARIANT
  vv : INTEGER --> BOOL
INITIALISATION
  vv := cc  /* cc is not a finite set of indices */
END
```

Variable <ident_var> has not been typed

All of the variables must be typed in the INVARIANT clause using a typing predicate (refer to the definition in Chapter 1).

```
MACHINE MACH
VARIABLES
  var1, var2, var3
INVARIANT
  var1 : NAT &
  var2 < var1   /* var2 must be typed */
                /* var3 must be typed */
INITIALISATION
  var1, var2, var3 := 5, 6, 7
END
```

Variable <ident> is not an implementable array

This message is generated for an implementation. An array is not implementable in B0 if its array is not an interval or an enumerated set.

```
IMPLEMENTATION M1_1
REFINES M1
CONCRETE_VARIABLES vv
INVARIANT
  vv : INTEGER --> BOOL
INITIALISATION
  vv := INTEGER * {TRUE}   /* INTEGER is not bounded */
END
```

Variable <ident> should be initialised

All of the variables defined in a component must be initialised in the INITIALISATION clause.

```
MACHINE MACH
VARIABLES
    xx,yy
INVARIANT
    xx:NAT & yy:NAT
INITIALISATION
    xx:=0    /* yy must be initialised */
END
```

Variant <exp> should designate a natural

In a WHILE loop, the variant must be an expression that designates a natural integer.

```
IMPLEMENTATION M1_1
REFINES M1
OPERATIONS
  opM1 = BEGIN
    WHILE 12 <0 DO skip INVARIANT 6 : NAT VARIANT "string" END;
    /* "string" is not a natural */
    WHILE 12 <0 DO skip INVARIANT 6 : NAT VARIANT ident_inconnu END
    /* ident_inconnu's type is unknown */
  END
END
```

VAR substitution is forbidden in a local operation specification : <subst>

The VAR substitution is a programming substitution reserved for refinement and implementation. In a local operation specification, a LET or ANY substitution must be used instead.

```
IMPLEMENTATION MM_1
REFINES MM
LOCAL_OPERATIONS
  op = VAR vv IN vv := 2 END
  /* incorrect specification:
     LET vv BE vv = 2 IN skip END is correct */
OPERATIONS
  op = VAR vv IN vv := 2 END
  /* correct implementation */
END
```

VAR substitution is forbidden in a machine: <subst>

The VAR substitution is a programming substitution reserved for refinements and implementations. In a machine, a LET or ANY substitution must be used instead.

```
/* Incorrect machine: */
MACHINE M1
OPERATIONS
  op = VAR vv IN vv := 2 END
END
```

```
/* Correct machine: */
MACHINE M1
OPERATIONS
  op = LET vv BE vv = 2 IN skip END
END
```

WHILE substitution is forbidden in a local operation specification : <subst>

This message is produced when a WHILE loop is used in a local operation specification. This instruction is not a specification substitution, indeed.

```
IMPEMENTATION
  MM_1
REFINES
  MM
CONCRETE_VARIABLES
  vv
INVARIANT
  vv : NAT
INITIALISATION
  vv := 0
LOCAL_OPERATIONS
  opWhile =
    WHILE vv > 10
    DO skip
    INVARIANT vv := NAT
    VARIANT vv
    /* forbidden */
    END
OPERATIONS
  opWhile =
    WHILE vv > 10
    DO skip
    INVARIANT vv := NAT
    VARIANT vv
    END
    /* allowed */
END
```

WHILE substitution is only allowed in an implementation: <subst>

This message is generated when a WHILE loop is used in an abstract machine or in a refinement. This substitution is only allowed in implementation mode, indeed.

```
MACHINE MACH
VARIABLES
  vv
INVARIANT
  vv : NAT
INITIALISATION
  vv := 0
OPERATIONS
  opWhile =
    WHILE vv > 10
    DO skip
    INVARIANT vv := NAT
    VARIANT vv
    END
END
```

Wrong number of parameters for instanciated machine <ident_mach>

For an inclusion with instancing, you must instance all of the parameters of the included machine.

```
MACHINE
  M1
INCLUDES
  M2(TRUE)
  /* value of p2 is missing */
END
```

```
MACHINE
  M2(p1, p2)
CONSTRAINTS
  p1 : BOOL & p2 : INT
END
```

Wrong type for actual input parameters of called operation <ident_op>

The formal input parameters for the called operation <ident_op> and the effective parameters are not of the same type. The types of formal operation parameters and the types of values set as arguments for a call-up, must be identical.

```
MACHINE MACH
INCLUDES
  MAC01
OPERATIONS
  op = oper01("error")
END
```

```
MACHINE MAC01
OPERATIONS
  oper01(x1) = PRE x1:NAT THEN
    skip
  END
END
```

Wrong type for actual output parameters of called operation <ident_op>	
The formal output parameters from the <ident_op> operation and the effective parameters are not of the same type. The types of the formal parameters of the operation and the types of variables that receive the returned value after call-up must be identical.	

`MACHINE MACH`
`INCLUDES`
`  MAC01`
`VARIABLES`
`  ww`
`INVARIANT`
`  ww : BOOL`
`INITIALISATION`
`  ww := TRUE`
`OPERATIONS`
`  op = ww <-- oper01`
`  /* ww is a Boolean value */`
`END`
	`MACHINE MAC01`
	`OPERATIONS`
	`  vv <--oper01 =`
	`    vv := 2`
	`  /* vv is an integer value */`
	`END`

Wrong type for actual parameter <ident_param> of machine <ident_mach>	
This actual parameters is used when the instancing of an included machine is not of the correct type. In practice, when performing an instantiated inclusion, the types of the included machine's formal parameters and the types of the effective parameters must be identical. This may also be caused by a syntax error (is <ident> a correct B language identifier?) or a visibility error (is the <ident> object visible?).	

`MACHINE`
`  M2(p1, p2, p3)`
`CONSTRAINTS`
`  p1 : NAT & p2 : BOOL & p3 : INT`
`END`
	`MACHINE M1`
	`INCLUDES`
	`  M2(UnknownParam, 67, _1)`
	`/*UnknownParam is unknown,`
	`  67 is not the correct type,`
	`  _1 is not a B ident*/`
	`END`

Wrong type for expression <exp> in a CASE substitution

The expression that should determine the performance of the CASE substitution has an illegal type. This expression must be an integer type, a Boolean type, or an element of an abstract set or of a listed set.

```
MACHINE
  M1
VARIABLES
  SS
INVARIANT
  SS <: NAT
INITIALISATION
  SS :: POW(NAT)
OPERATIONS
op1 =
  CASE "sting" OF
  EITHER 1 THEN skip
  ELSE skip
  END
  END;
op2 =
  CASE UnknownExp OF
  EITHER 1 THEN skip
  ELSE skip
  END
  END;
op3 =
  CASE SS OF    /*SS is part of NAT*/
  EITHER 1 THEN skip
  ELSE skip
  END
  END
END
```

Chapter 5

Internal error messages

The messages presented in this chapter do appear only in case of forbidden use of Atelier B - for example, manual use of files from the Data Base Project. It is therefore necessary to redo type checking for the component stated in the message.

Bad magic number for <ident_mach>.nf
The .nf file assigned to component <ident_mach> was not generated with the same version of the type checker. It cannot therefore be used by this version. Run the type checker again on <ident_mach>.

Cannot load information file of component <ident_mach>
The analyzed component references the <ident_mach> component whose .nf file does not exist or is empty.

Wrong Normal Form format for the refined structure.
The .nf file relating to the refined component was modified by an action external to Atelier B.