

**Atelier B**

# **Contrôleur de types**

**Manuel des messages d'erreur**

**version 3.6**



ATELIER B  
Contrôleur de types Manuel des messages d'erreur  
version 3.6

Document établi par CLEARSY.

Ce document est la propriété de CLEARSY et ne doit pas être copié, reproduit, dupliqué  
totalement ou partiellement sans autorisation écrite.

Tous les noms des produits cités sont des marques déposées par leurs auteurs respectifs.

CLEARSY  
Maintenance ATELIER B  
Parc de la Duranne - 320 avenue Archimède  
Les Pléiades III - Bat A  
13857 AIX EN PROVENCE Cedex 3  
FRANCE

Tél 33 (0)4 42 37 12 70

Fax 33 (0)4 42 37 12 71

mail : [contact@atelierb.eu](mailto:contact@atelierb.eu)

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Définitions</b>	<b>3</b>
<b>3</b>	<b>Messages d'avertissement</b>	<b>5</b>
<b>4</b>	<b>Messages d'erreur</b>	<b>9</b>
<b>5</b>	<b>Messages d'erreur interne</b>	<b>79</b>



# Chapitre 1

## Introduction

Ce manuel présente les différents messages d'erreur et d'avertissement du Contrôleur de types. Son but est de préciser l'origine des erreurs pour chaque message afin d'aider l'utilisateur : une fois la cause correctement ciblée, il est plus facile de corriger sa spécification. Pour des informations complètes et détaillées, il est conseillé de se reporter au Manuel de Référence du Langage B.

Les messages du Contrôleur de types sont tous encadrés de :

*Type checking <machine/refinement/implementation><nom\_comp>*

...

*End of Type checking*

Pour chaque contrôle effectué, un message d'information est affiché. Ainsi le message :

*Checking operation lire*

indique à l'utilisateur que le Contrôleur de types vérifie l'opération *lire*. Les messages d'erreur ou d'avertissement qui le suivront feront donc référence au corps de l'opération *lire*. Ils préciseront toutefois l'extrait du code source où est localisée l'erreur. Notons à ce sujet que les expressions  $A < : B$  et  $A << : B$  sont normalisées en  $A : POW(B)$ . Les éventuels messages associés citeront alors l'expression normalisée.

Ce manuel est composé de quatre chapitres. Le premier définit les termes utilisés dans les explications des messages. Les trois suivants présentent respectivement les messages d'avertissement, les messages d'erreur et les messages d'erreur interne.

Les messages sont classés par ordre alphabétique. Les symboles - en dehors des chiffres - n'interviennent pas dans le classement. Ainsi le message :

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

est classé à la lettre A. Il est donc placé avant le message :

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

Chaque message est présenté dans un tableau de la manière suivante :

Exemple	Commentaire
<i>Component name &lt;ident&gt; should be an identifier</i>	<i>libellé du message</i>
Le nom d'un composant doit être un nom simple, c'est-à-dire un identificateur B correct.	<i>description de l'erreur commise</i>
/*Le nom de la machine ci-dessous n'est pas correct car il contient un point.*/ MACHINE M1.N2 END	<i>exemple de spécification générant le message</i>

## Chapitre 2

# Définitions

Ce chapitre précise certains termes utilisés dans la suite du manuel.

**constante** désigne indifféremment une constante concrète ou abstraite.

**composant** désigne indifféremment une machine, un raffinement ou une implémentation.

**identificateur B** chaîne de caractères vérifiant les règles suivantes :

- au moins deux caractères
- commençant par une lettre
- uniquement composé des caractères A-Z a-z 0-9 \_

**mot clef** identificateur ayant une signification particulière. La liste des mots clef du langage B est présentée dans le Manuel de Référence du Langage B. Il est nécessaire de la compléter avec la liste suivante, qui est la liste des identificateurs réservés aux outils : 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, blood, blvar, bmark, bmatch, bmodr, bnewv, bnlmap, bnmap, bnot, bnum, bpattern, bpop, bprintf, bproved, breade, breadf, brecompact, bresetcomp, bresult, brev, brule, bsearch, bsetmode, bshell, bslmap, bsmmap, bsparemem, bsrv, bstatistics, bstring, bsubfrm, btest, bunproved, bvrbr, bwritef, bwritem, trace.

**prédicat de typage** prédicat de la forme "*Identificateur op Expression*" où *op* est soit l'appartenance ( $\in$ ), soit l'inclusion ( $\subset$  ou  $\subseteq$ ), soit l'égalité ( $=$ ). Ces prédicats sont décrits plus précisément dans le Manuel de Référence du Langage B.

**variable** désigne indifféremment une variable concrète ou abstraite.



## Chapitre 3

# Messages d'avertissement

Les messages d'avertissement du Contrôleur de types sont précédés de :

*Warning :*

Ils permettent à l'utilisateur d'anticiper un futur message d'erreur du B0Checker. Ils indiquent également d'éventuels problèmes de lisibilité du code.

<b><i>Concrete constant &lt;ident_cst&gt; has not been valued</i></b>
---

Toutes les constantes concrètes définies au cours du raffinement doivent être valuées dans la clause VALUES de l'implémentation. Cet avertissement permet d'anticiper un message d'erreur du B0Checker.
---

<pre>IMPLEMENTATION M1_1 REFINES M1 CONCRETE_CONSTANTS   cc PROPERTIES   cc : INTEGER --&gt; BOOL END /* cc n'est pas valuée */</pre>
---

<b><i>Concrete constant &lt;ident_cst&gt; is not an implementable array</i></b>
---

La constante concrète <ident_cst> n'est pas implémentable en B0 : son domaine doit être un intervalle ou un ensemble énuméré.
---

<pre>MACHINE   M1 CONSTANTS   Sequence, Relation SETS   EE PROPERTIES   Sequence : seq(EE) &amp;   Relation : INT &lt;-&gt; INT   /* Sequence et Relation ne sont pas implémentables */ END</pre>
---

***Concrete constant <ident\_cst> may not be implementable***

Le Contrôleur de types n'est pour le moment pas capable de dire si la constante <ident\_cst> est implémentable. Cet avertissement peut apparaître après une erreur dans le calcul de type. Dans ce cas d'autres messages précisent le problème.

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

***Constant <ident\_cst> is not an implementable record : it uses a non implementable array***

La constante concrète <ident\_cst> n'est pas implémentable en B0 : l'un de ses champs est un tableau non implémentable (son domaine doit être un intervalle ou un ensemble énuméré).

```
MACHINE
  M1
CONSTANTS
  Record1, Recod2
SETS
  EE
PROPERTIES
  Record1 : struct(seq1 : seq(EE), bb ; BOOL) &
  Record2 : struct(re11 : INT <-> INT, xx : INT)
  /* Record1 et Record2 ne sont pas implémentables */
END
```

***Deferred set <ident\_set> has not been valued***

Tous les ensembles abstraits définis au cours du raffinement doivent être valués dans la clause VALUES de l'implémentation. Cet avertissement permet d'anticiper un message d'erreur du B0Checker.

```
IMPLEMENTATION M1_1
REFINES M1
SETS SS
END /* SS n'est pas valué */
```

***Identifieur <ident> is already used***

L'identificateur <ident> est utilisé plusieurs fois dans le composant analysé. Ces deux définitions ne risquent pas d'entrer en conflit et la spécification est juste. Cet avertissement met seulement en relief un problème potentiel de compréhension des sources à la lecture.

```
REFINEMENT M1_1
REFINES M1
OPERATIONS
  op =
  VAR vv IN
    vv : (vv : NAT & !vv.(vv : BOOL => 0 = 0))
    /* le deuxième vv n'entre pas en conflit avec le premier mais peut gêner la
    compréhension de l'opération */
  END
END
```

***Local variable <ident> may be read before being initialised***

Ce message apparaît pour une machine ou un raffinement. La variable locale <ident> est une variable définie dans une substitution VAR ou dans la liste des paramètres de sortie d'une opération. Elle est utilisée alors qu'elle n'a pas été complètement initialisée par une substitution à branche.

```
MACHINE M1
OPERATIONS
  ss, tt <-- op(ii) = PRE ii : NAT THEN
    IF ii > 1 THEN
      ss := 2
    END;
    tt := ss
  /* ss n'a pas été initialisée dans toutes les branches du IF */
  END
END
```

***Local variable <ident> may not be initialised***

La variable locale <ident> définie dans une substitution VAR n'est pas initialisée ou ne l'est que dans une partie des branches d'une substitution à branches.

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

***Local variables <list\_ident> may not be initialised***

Les variables locales <ident> définies dans une substitution VAR ne sont pas initialisées ou ne le sont que dans une partie des branches d'une substitution à branches.

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

***Output parameter <ident> may not be initialised***

Ce message apparaît pour une machine ou un raffinement. Le paramètre de sortie <ident> de l'opération en cours de type-check n'a pas été initialisé dans toutes les branches des substitutions à branches du corps de cette opération.

```
MACHINE M1
OPERATIONS
  ss <-- op(ii) = PRE ii : NAT THEN
    IF ii > 1 THEN
      ss := 2
    END
  END
END
END
/* ss n'a pas été initialisé dans toutes les branches du IF */
```

***Output parameters <list\_ident> may not be initialised***

Ce message apparaît pour une machine ou un raffinement. Les paramètres de sortie <list\_ident> de l'opération en cours de type-check n'ont pas été initialisés dans toutes les branches des substitutions à branches du corps de cette opération.

```
MACHINE M1
OPERATIONS
  ss, tt <-- op(ii) = PRE ii : NAT THEN
    IF ii > 1 THEN
      ss := 2
    ELSE
      tt := 3
    END
  END
END
END
/* ss et tt n'ont pas été typés dans toutes les branches du IF */
```

## Chapitre 4

# Messages d'erreur

Les messages d'erreur du Contrôleur de types sont tous précédés de :

*Error :*

Dans la mesure du possible, le Contrôleur de types ne s'arrête pas après une erreur. Dans les cas où il lui est toutefois impossible de continuer, le message final suivant indique que la vérification a été interrompue :

*TypeCheck aborted*

<b><i>\$0 is not allowed : &lt;ident&gt;\$0</i></b>
L'expression \$0 n'est autorisée que dans les substitutions "devient tel que" et "WHILE". Ce message est émis dans tous les autres cas.
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***

Les en-têtes des implémentations des opérations locales doivent être strictement identiques aux en-têtes de leur spécification : le nombre de paramètres en entrée et en sortie doit être préservé, les noms des paramètres doivent être les mêmes.

```

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 en trop*/;
  out <-- oper2 = BEGIN
    /*out en trop,
    manque xx*/
    out:=0
  END;
  out <-- oper3 = BEGIN
    /*out au lieu de res*/
    out := v1
  oper4 = skip
    /*manque res et xx*/
END

```

*Abstract and concrete headers of operation <ident\_op> differ*

Dans un raffinement ou une implémentation, les en-têtes des opérations raffinées doivent être strictement identiques aux en-têtes de la machine abstraite : le nombre de paramètres en entrée et en sortie doit être préservé, les noms des paramètres doivent être les mêmes. De même lorsque l'on raffine une opération par une opération promue, les entêtes doivent être identiques.

<pre> MACHINE   M1 VARIABLES   v1 INVARIANT   v1:NAT INITIALISATION   v1:=0 OPERATIONS   oper1 =     skip;   oper2(xx) = PRE     xx:NAT   THEN     v1:=xx   END;   res &lt;-- oper3 =     res := v1;   res &lt;-- oper4(xx) = PRE     xx:NAT   THEN     res:=xx+v1   END;   out &lt;-- opincluder(in) = PRE     in:NAT   THEN     out:=in+1   END; END         </pre>	<pre> IMPLEMENTATION   M1_1 REFINES   M1 IMPORTS   M2 PROMOTES   opincluder OPERATIONS   oper1(xx) = skip     /*xx en trop*/;   out &lt;-- oper2 = BEGIN     /*out en trop,     manque xx*/     out:=0   END;   out &lt;-- oper3 = skip;     /*out au lieu de res*/   oper4 = skip     /*manque res et xx*/ END         </pre>
---	--

<pre> MACHINE   M2 OPERATIONS   res &lt;-- opincluder(xx) =     /* res et xx au lieu de out et in */   PRE xx:NAT THEN res:=xx+1 END END         </pre>
---

***Abstract constant <ident\_cst> cannot be used in <ident\_mach> instantiation***

Les constantes abstraites d'une machine ou d'un raffinement M ne peuvent pas servir à instancier les machines référencées dans les clauses INCLUDES et EXTENDS de M.

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

***Abstract constant <ident> has not been typed***

Toutes les constantes abstraites doivent être typées dans la clause PROPERTIES au moyen d'un prédicat de typage (cf définition chapitre 1).

```
/*Dans l'exemple ci-dessous les constantes valmin et valmed n'ont pas été
typées.*/
MACHINE MACH_CONST
ABSTRACT_CONSTANTS
  valmax,
  valmin,
  valmed
PROPERTIES
  valmax = 100 &
  valmin < valmax /* Cela ne type pas valmin */
END /* valmed non typée */
```

<p><i>Abstract constant &lt;ident_hcst&gt; has not the same type in &lt;ident_comp1&gt; and in &lt;ident_comp2&gt;</i></p>	
<p>&lt;ident_comp1&gt; désigne le composant raffiné par le composant analysé.                  &lt;ident_comp2&gt; désigne une machine requise directement par le composant analysé.                  Une constante abstraite &lt;ident_hcst&gt; de &lt;ident_comp1&gt; ne peut être implantée par homonymie par une constante abstraite ou concrète de type différent définie dans &lt;ident_comp2&gt;.</p>	
<pre> MACHINE   autre ABSTRACT_CONSTANTS   cc PROPERTIES   cc : BOOL END                 </pre>	<pre> MACHINE   M1 ABSTRACT_CONSTANTS   cc PROPERTIES   cc : NAT   /* remplacer cc : NAT par   cc : BOOL */ END                 </pre>
<pre> IMPLEMENTATION   M1_i REFINES   M1 IMPORTS   autre END                 </pre>	

<p><i>Abstraction and refinement have the same name</i></p>	
<p>Les noms des composants d'un développement vertical doivent tous être distincts. En général, le raffinement de rang n d'une machine M1 est nommé M1_n.</p>	
<pre> REFINEMENT   MACH /* interdit */ REFINES   MACH END                 </pre>	<pre> REFINEMENT   MACH_1 /* écriture conseillée */ REFINES   MACH END                 </pre>

<p><i>Abstract set name &lt;ident&gt; should be an identifier, or invalid list separator</i></p>	
<p>Un nom d'ensemble est obligatoirement un identificateur B (cf définition chapitre 1). Chaque définition d'ensemble doit être séparée par un point-virgule.</p>	
<pre> MACHINE   M1 SETS   2; "chaine"; nom-compose END                 </pre>	

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

L'expression <exp> déterminant le comportement de la substitution CASE et la constante <ident> énumérée dans une branche du CASE devraient avoir le même 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 et TRUE ne sont pas du type de vv */
END

```

*<ident\_op1> and <ident\_op2> of machine <ident\_mach> are called simultaneously*

Deux opérations incluses ne peuvent pas être appelées en parallèle.

<pre> MACHINE M1 VARIABLES   v1,v2 INVARIANT   v1:NAT &amp; v2:NAT &amp; v1&lt;=v2 INITIALISATION   v1:=0    v2:=0 OPERATIONS   increment = PRE     v1&lt;v2   THEN     v1:=v1+1   END ;   decrement = PRE     v1&lt;v2   THEN     v2:=v2-1   END END </pre>	<pre> MACHINE M0 INCLUDES   M1 OPERATIONS   op_errone = PRE     v1&lt;v2   THEN     increment    decrement     /* l'invariant est brisé */   END END </pre>
--	---

*A record element without label can not be used in <Expression>*

Deux éléments de record sans label ne peuvent être comparés. Un élément de record sans label a un typage générique.

```

MACHINE
  M1
ABSTRACT_VARIABLES
  xx,yy
INVARIANT
  xx : NAT &
  yy : BOOL &
  rec(xx,yy) = rec(2,TRUE)
  /* Expression incorrecte est : rec(xx,yy) = rec(2,TRUE) */
  /* xx = 2 & yy = TRUE est correcte */
INITIALISATION
  xx := 2 ||
  yy := TRUE
END

```

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

Les deux bornes d'un intervalle doivent être des entiers.

```

MACHINE
  M1
CONSTANTS
  cc, dd
PROPERTIES
  cc = TRUE..7 &      /* TRUE n'est pas un entier */
  dd = 2..Binconnue  /* Binconnue n'est pas un entier */
END

```

*<ident> can not be typed by {}*

Ce message est émis lorsqu'un identificateur <ident> est typé par l'ensemble vide.

```

MACHINE
  test
ABSTRACT_VARIABLES
  vv
INVARIANT
  vv = {} /* vv n'a pas été typé. Il faut écrire par exemple
          vv <: NAT &
          vv = {}
          */
INITIALISATION
  vv := {}
END

```

***Component name <ident> is a keyword***

L'identificateur <ident> est un mot réservé du langage (cf chapitre 1). Il est interdit de l'utiliser pour nommer un composant.

```
MACHINE
  MAXINT
END
```

***Component name <ident> should be an identifier***

Le nom d'un composant doit être un nom simple, c'est-à-dire un identificateur B correct (cf définition chapitre 1).

```
/*Le nom de la machine ci-dessous n'est pas correct car il contient un point.*/
MACHINE
  M1.N2
END
```

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

Dans une implémentation, une variable concrète peut être implémentée implicitement avec une variable de même nom issue d'une machine importée.

Dans le cas de ce message, la variable à implémenter et celle qui est importée n'ont pas le même type, ce qui est interdit.

```
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***

Toutes les constantes doivent être typées dans la clause PROPERTIES au moyen d'un prédicat de typage (cf définition chapitre 1).

```

/*Dans l'exemple ci-dessous les constantes valmin et valmed n'ont pas été
typées.*/
MACHINE MACH_CONST
CONSTANTS
  valmax,
  valmin,
  valmed
PROPERTIES
  valmax = 100 &
  valmin < valmax    /* Cela ne type pas valmin */
END                  /* valmed non typée */

```

***Constant <ident> is not an implementable array***

Ce message est émis en implémentation. Un tableau n'est pas implémentable en B0 si son domaine n'est pas un intervalle ou un ensemble énuméré.

```

IMPLEMENTATION M1_1
REFINES M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : INTEGER --> BOOL
VALUES
  cc = INTEGER * {TRUE}
  /* INTEGER n'est pas borné */
END

```

***Constants should be defined in the PROPERTIES clause***

Le composant analysé n'a pas de clause PROPERTIES alors qu'il contient des constantes.

```

MACHINE MACH_CONST
CONSTANTS
  valmin, valmax
END    /* il manque la clause PROPERTIES */

```

***<ident> declaration is not visible***

Le composant analysé fait référence à un objet de nom <ident> qui n'appartient pas à l'ensemble des objets visibles. Cette situation se produit soit lors d'une erreur de frappe, soit lorsque les contraintes de visibilité sont violées.

```
MACHINE M1
OPERATIONS
vv <-- op = vv := IdInconnu
    /* IdInconnu n'est pas un identificateur visible */
END
```

***Distinct definitions of enumerated set <ident\_set>***

En implémentation, un même ensemble énuméré peut être défini dans un des composants raffinés (ou dans l'implémentation) et dans une machine vue ou importée. Toutefois, les deux définitions doivent être identiques : même nombre d'éléments, même nom pour chaque élément, même ordre des éléments.

```
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 et Enum2 n'ont pas la même définition dans M1 et M2 */
END
```

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

L'opération de nom <ident> n'appartient pas à l'ensemble des opérations visibles. Cette situation se produit soit lors d'une erreur de frappe, soit lorsque les contraintes de visibilité sont violées.

```
/*L'opération opinconnue de la machine
suiivante n'appartient pas à la machine
incluse, il n'est donc pas possible de
la promouvoir :*/
MACHINE
  M1
INCLUDES
  M2
PROMOTES
  opinconnue
END
```

```
MACHINE
  M2
END
```

<i>Element &lt;ident_elt&gt; of set &lt;ident_set&gt; is already defined</i>	
Il s'agit d'un conflit d'identificateurs.	
<pre>MACHINE MACH SETS   COULEURS = { rouge, vert, bleu } ; VERT = { vert }      /* vert est en conflit */ END</pre>	
<i>Enumerated set name in definition &lt;enum_def&gt; should be an identifier</i>	
Un nom d'ensemble est obligatoirement un identificateur B (cf définition chapitre 1).	
<pre>MACHINE   M1 SETS   2 = {aa};   "chaine" = {bb};   nom-compose = {cc} END</pre>	
<i>&lt;ident_cst&gt; has not the same type in &lt;ident_mach1&gt; (or in an abstraction of &lt;ident_mach1&gt;) and in &lt;ident_mach2&gt;</i>	
La constante <ident_cst> est évaluée implicitement par une constante de même nom appartenant à une machine vue ou importée. Le type de <ident_cst> défini dans la clause PROPERTIES de l'abstraction du composant analysé et celui qui est défini dans la machine vue ou importée doivent donc être identiques.	
<pre>MACHINE M1 CONSTANTS   cst PROPERTIES   cst : NAT END</pre>	<pre>MACHINE M2 CONSTANTS   cst PROPERTIES   cst : BOOL END</pre>
<pre>IMPLEMENTATION M1_1 REFINES   M1 SEES   M2 /* valuation implicite de cst */ END</pre>	
<i>Identifieur &lt;ident&gt; is a keyword</i>	
L'identificateur <ident> est un mot réservé du langage (cf chapitre 1). Il ne peut pas servir à nommer une autre entité.	
<pre>MACHINE MACH(skip) END</pre>	

***Identifieur <ident> is already defined***

Ce message rappelle la présence d'un conflit d'identificateurs lors de l'analyse d'une clause particulière.

```
MACHINE
  MACH
SEES
  SEE01
CONSTANTS
  cst1
  /*conflit avec SEE01*/
PROPERTIES
  cst1 : NAT
END
```

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

***Identifieur <ident\_cst> is already valued***

Une constante ou un ensemble du composant analysé est valué deux fois, ce qui est interdit.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
VALUES
  val1 = 2 ;
  val1 = 2 /*val1 est valué 2 fois*/
END
```

***Identifieur <ident> is defined in <ident\_mach1> and in <ident\_mach2>***

Ce message signale un conflit d'identificateurs entre deux machines faisant l'objet d'une clause de visibilité. L'utilisation d'un préfixe de renommage peut vous permettre de résoudre ce conflit.

```
MACHINE INC01
VARIABLES
  v_conflit /*conflit*/
INVARIANT
  v_conflit : NAT
INITIALISATION
  v_conflit := 0
END
```

```
MACHINE INC02
VARIABLES
  v_conflit /*conflit*/
INVARIANT
  v_conflit : BOOL
INITIALISATION
  v_conflit := FALSE
END
```

```
MACHINE GLOBAL
INCLUDES
  INC01, INC02
  /* une écriture correcte :
INCLUDES
  i1.INC01, i2.INC02 */
END
```

*Identifieur <ident> is defined in <ident\_mch1> and in an included renamed machine of <ident\_mch2>*

Ce message signale un conflit d'identificateurs entre deux machines faisant l'objet d'une clause de visibilité. L'utilisation d'un préfixe de renommage peut vous permettre de résoudre ce conflit.

*Identifieur <ident> is defined in <ident\_mch1> and in <ident\_mch2> (or in an abstraction of <ident\_mch2>)*

Ce message signale un conflit d'identificateurs entre deux machines faisant l'objet d'une clause de visibilité. L'utilisation d'un préfixe de renommage peut vous permettre de résoudre ce conflit.

<pre>MACHINE MACH INCLUDES   INC01 END</pre>	<pre>MACHINE INC01 VARIABLES   v_conflit /*conflit*/ INVARIANT   v_conflit : NAT INITIALISATION   v_conflit := 0 END</pre>
<pre>REFINEMENT MACH_1 REFINES MACH END</pre>	<pre>REFINEMENT MACH_2 REFINES MACH_1 CONCRETE_CONSTANTS   v_conflit /* conflit */ INVARIANT   v_conflit : BOOL INITIALISATION   v_conflit := FALSE /*v_conflit de INC01 est toujours visible, d'où le conflit*/ END</pre>

*Identifieur <ident> is defined in an included (possibly renamed) machine of <ident\_mch1> and in an included (possibly renamed) machine of <ident\_mch2>*

Ce message signale un conflit d'identificateurs entre deux machines faisant l'objet d'une clause de visibilité. L'utilisation d'un préfixe de renommage peut vous permettre de résoudre ce conflit.

*Identifieur <ident> is defined in an included renamed machine of <ident\_mch1> and in <ident\_mch2>*

Ce message signale un conflit d'identificateurs entre deux machines faisant l'objet d'une clause de visibilité. L'utilisation d'un préfixe de renommage peut vous permettre de résoudre ce conflit.

*Identifieur <ident> is defined in <ident\_mch1> (or in <ident\_mch1>'s abstractions) and in <ident\_mch2>*

Ce message signale un conflit d'identificateurs entre deux machines faisant l'objet d'une clause de visibilité. L'utilisation d'un préfixe de renommage peut vous permettre de résoudre ce conflit.

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

Ce message est émis lorsque l'on type une donnée avec un record dont tous les labels ne sont pas précisés.

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

***Incompatible types in <exp>***

La syntaxe de <exp> implique certaines conditions sur les types. Ce message indique une violation de ces conditions.

Par exemple, dans l'expression ff(xx), xx doit appartenir au domaine de départ de ff. De même dans la substitution vv := {aa, bb, cc}, les trois éléments aa, bb et cc doivent avoir le même type.

```
MACHINE
  M1
SETS
  SS; TT
CONSTANTS
  relation, ff
PROPERTIES
  relation : SS <-> TT &
  ff : INT --> SS
OPERATIONS
  vv <-- op1 = vv := relation[{1}];
  /* 1 n'appartient pas à SS */
  vv <-- op2 = vv := [1, 2, TRUE, 6];
  /* TRUE n'est pas du même type que 6 */
  vv <-- op3 = vv := {1, TRUE, 2};
  /* TRUE n'est pas du même type que 2 */
  vv <-- op4 = vv := ff(TRUE)
  /* TRUE n'est pas un entier */
END
```

***<exp1> in <exp2> has not been typed***

L'expression <exp1> contient un ou plusieurs identificateurs qui n'ont pas été typés avant leur utilisation dans <exp2>.

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

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

L'opérateur utilisé dans <exp> attend en argument un couple d'ensembles.

```

MACHINE
  M1
SETS
  EE
CONSTANTS
  cc, dd
PROPERTIES
  cc = prj1(EE) &      /* EE n'est pas une couple d'ensembles */
  dd = prj2(Inconnu) /* Inconnu n'est pas une couple d'ensembles */
END

```

*<exp1> in <exp> should be a function*

Dans une expression de la forme  $f(x)$ ,  $f$  doit avoir été définie comme une fonction.

```

MACHINE
  M1
CONSTANTS
  c1, c2
PROPERTIES
  c1 = TRUE(1) &      /* TRUE n'est pas une fonction */
  c2 = Inconnue(1) /* Inconnue n'est pas une fonction */
END

```

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

<exp1> doit être une liste d'identificateurs B distincts les uns des autres, séparés par des virgules. La définition d'un identificateur B est rappelée chapitre 1.

```

MACHINE
  M1
CONSTANTS
  cc
PROPERTIES
  !(xx, xx). (cc = xx) &
    /*xx apparaît deux fois*/
  cc = PI(xx; yy).(xx : NAT & yy : NAT | 1) &
    /*utilisation de ';' au lieu de ',' */
  cc = SIGMA(xx, _1).(xx : NAT | 1) &
    /* _1 n'est pas un identificateur */
  cc = UNION(xx, 1).(xx : NAT | {xx})
    /* 1 n'est pas un identificateur */
END

```

*<exp1> in <exp> should be an expression*

Ce message est émis pour une lambda expression : dans l'écriture %L.(P | E), E doit être une expression.

```

MACHINE
  M1
CONSTANTS
  cc, dd, ee
PROPERTIES
  cc = %(xx).(xx : NAT | skip) &
      /* skip n'est pas une expression */
  dd = %(xx).(xx : NAT | ExpInconnue)
      /* ExpInconnue n'est pas une expression */
  ee = %(xx).(xx : NAT | xx = 2)
      /* xx = 2 n'est pas une expression,
      bool(xx = 2) est correcte */
END

```

*<exp1> in <exp> should be an integer*

Les opérateurs utilisés dans <exp> nécessitent que <exp1> représente un entier.

```

MACHINE
  M1
CONSTANTS
  Relation
PROPERTIES
  Relation : INT <-> INT
OPERATIONS
  vv <-- op1 = vv := SIGMA(xx).(xx: 1..100 | bool(xx <= 20));
      /* bool(xx <= 20) est un booléen*/
  vv <-- op2 = vv := iterate(Relation, EntierInconnu)
      /* EntierInconnu n'a pas de type*/
END

```

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

L'opérateur utilisé dans <exp> nécessite que <exp1> représente un ensemble entier ou énuméré.

```

MACHINE
  M1
SETS
  AA
OPERATIONS
  vv <-- opMinAbst = vv := min(AA);          /*AA est un ensemble abstrait */
  vv <-- opMinScal = vv := min(3);          /*3 n'est pas un ensemble */
  vv <-- opMaxInc = vv := max(EnsInconnu)   /*EnsInconnu n'est pas un ensemble*/
END

```

*<exp1> in <exp> should be a relation*

L'opérateur utilisé dans <exp> nécessite que <exp1> représente une relation.

```

MACHINE
  M1
SETS
  SS
CONSTANTS
  cc, tt
PROPERTIES
  cc = ran(6) &                               /* 6 n'est pas une relation */
  tt : NAT
OPERATIONS
  vv <-- op1 = vv := rel(tt);                 /* tt n'est pas une relation */
  vv <-- op2 = vv := Inconnu~;                /* Inconnu n'est pas une relation*/
  vv <-- op3 = vv := fnc(SS)                  /* SS n'est pas une relation */
END

```

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

L'opérateur utilisé dans <exp> attend en argument une relation d'un ensemble sur lui-même.

```

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

```

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

L'opérateur utilisé dans <exp> attend en argument une suite de suites.

```

MACHINE
  M1
CONSTANTS
  Sequence
PROPERTIES
  Sequence : seq(INT)
OPERATIONS
  vv <-- opConc = vv := conc(Sequence);
  /* Sequence n'est pas une suite de suites */
  vv <-- opConc2 = vv := conc(SeqInconnue)
  /* SeqInconnue n'est pas une suite de suites */
END

```

*<exp1> in <exp> should be a set*

Les opérateurs utilisés dans <exp> nécessitent que <exp1> représente un ensemble.

```

MACHINE
  M1
CONSTANTS
  cc, dd, ee
SETS
  EE
PROPERTIES
  cc : EnsInconnu & /* EnsInconnu devrait être un ensemble */
  ee : NAT &
  dd /: ee          /* ee n'est pas un ensemble */
OPERATIONS
  vv <-- opInter = vv := INTER(xx).(xx : NAT | ee);
                          /* ee n'est pas un ensemble */
  vv <-- opCard = vv := card(EnsInconnu);
                          /* EnsInconnu n'est pas un ensemble */
  vv <-- opSeq = vv := seq(1)
                          /* 1 n'est pas un ensemble */
END

```

*<exp1> in <exp> should be a set of sets of same type*

Les opérateurs utilisés dans <exp> nécessitent que <exp1> représente un ensemble d'ensembles de même type.

```

MACHINE M1
CONSTANTS
  aa, bb
PROPERTIES
  aa = union(EnsInconnu) & /* EnsInconnu n'a pas de type */
  bb = inter({1, 2})      /* {1, 2} est un ensembles d'entiers */
END

```

*Internal name clash between identifier <ident> and a renamed identifier of the abstraction of <ident\_mach>*

Lorsqu'un composant renomme une machine avec le préfixe "pp", et que celle-ci possède un identificateur nommé "ident", le Générateur d'Obligations de Preuve et le Prouveur manipulent l'identificateur "ppident" et non "pp.ident". Si un identificateur "ppident" est également défini dans une machine non renommée ou dans le composant lui-même, un conflit apparaît.

Ce conflit est détecté afin de ne pas avoir d'obligations de preuve incorrectes, il est uniquement dû au fonctionnement interne de l'Atelier B.

<pre>MACHINE M1 INCLUDES   pp.M2 END</pre>	<pre>MACHINE M2 VARIABLES   var INVARIANT   var : NAT INITIALISATION   var := 0 END</pre>
--	---

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

*Invalid assignement for a record element in <Expression>*

Ce message est émis lorsque la syntaxe relative à l'affectation d'un élément de record est incorrecte.

```
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
        /* Cette syntaxe n'est pas permise.
          xx(1) := rec(TRUE,1) est correcte. */
END
```

***Invalid call of <ident\_op> : wrong number of input parameters***

Lors de l'appel d'une opération, le nombre de paramètres effectifs doit être égal au nombre de paramètres formels.

```
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***

Lors de l'appel d'une opération, le nombre de paramètres effectifs doit être égal au nombre de paramètres formels.

```
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***

Un ensemble énuméré constant ou une chaîne de caractère constante a été utilisé dans une branche d'une substitution CASE. Seuls des constantes numériques ou des identificateurs sont autorisés.

```

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 sans les accolades est correct*/
      OR "3,4,5" THEN uu:=1
        /* 3,4,5 sans les accolades est correct */
      OR _1 THEN uu:=2
        /* _1 n'est pas un identificateur*/
      ELSE uu:=3
    END
  END
END
END
    
```

***Invalid extended machine <ident\_mach>, it uses other machines***

Une machine effectuant un USES ne peut pas être référencée dans une clause IMPORTS. Elle ne peut donc pas apparaître dans la clause EXTENDS d'une implémentation, car cela revient à l'importer.

Notons que ce message n'apparaît qu'en implémentation. Dans une machine abstraite ou un raffinement, l'extension sous-entend une inclusion, elle reste donc autorisée.

<pre> MACHINE   M2 USES   M3 ENDD     </pre>	<pre> IMPLEMENTATION   M1_1 EXTENDS   M2 END     </pre>
--	---

***Invalid formula in VALUES clause***

Une erreur de syntaxe a été détectée dans la clause VALUES. Les différentes valuations doivent être séparées par des points virgules, chaque valuation est indiquée par un caractère '='.

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

***Invalid identifier or invalid list separator <ident>***

Une erreur syntaxique a été détectée dans une liste d'identificateurs. Il peut s'agir soit d'un identificateur B incorrect, soit de l'utilisation d'un autre caractère que la virgule pour séparer les éléments de la liste. La définition d'un identificateur B est rappelée chapitre 1.

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

***Invalid imported machine <ident\_mach>, it uses other machines***

Une machine effectuant un USES ne peut pas être référencée dans une clause IMPORTS.

```
MACHINE M2
USES M3
END
```

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

***Invalid input format***

Le texte de la spécification contient un caractère mal placé. Il peut s'agir d'une chaîne de caractères non fermée.

```
MACHINE M1
CONSTANTS
  message
PROPERTIES /* cette chaine n'est pas fermée */
  message = "intitule du message.
END
```

***Invalid inputs in <op\_header>***

Les paramètres d'entrée d'une opération doivent être des identificateurs B, séparés par des virgules et distincts les uns des autres. La définition d'un identificateur B est rappelée chapitre 1.

```
MACHINE M1
OPERATIONS
  op1(_1) = ... /* _1 n'est pas un identificateur */
; vv <-- op2(b) = ... /* b n'est pas un identificateur */
; op3(vv, vv) = ... /* vv apparaît deux fois */
; op4(vv; ww) = ... /* le séparateur doit être une virgule*/
END
```

***Invalid label <ident\_label> in <ident\_elem\_rec>'<ident\_label>***

Ce message est émis lorsque le champ <ident\_label> de l'élément de record <ident\_elem\_rec> auquel on veut accéder est inexistant dans <ident\_elem\_rec>

```
MACHINE
  M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : struct(aa : BOOL, bb : BOOL, ee : NAT) &
  cc'dd = 3
    /* dd n'est pas un champ de cc.
    cc'ee = 3 est correct */
END
```

***Invalid label <ident\_label> in a record expression***

Ce message est émis dès que dans une expression de record un même label est utilisé plus d'une fois et que ce label n'est pas un identificateur au sens du langage B.

```
MACHINE
  M1
  CONCRETE_CONSTANTS
    cc
  PROPERTIES
    cc : struct(aa : BOOL, bb : BOOL, 2cc : NAT, 2cc : NAT)
      /* 2cc n'est pas un identificateur B.
         2cc apparaît deux fois dans la
         même expression de record */
  END
```

***Invalid list of identifiers in enumerated set definition <enum\_def>***

Un élément d'ensemble énuméré est obligatoirement un identificateur B (cf définition chapitre 1). Ces éléments doivent être tous distincts et séparés par des virgules.

```
MACHINE
  M1
  SETS
    ACTIONS = {ouvrir-porte, fermer-porte};
    E1 = {"chaine"};
    E2 = {1};
    E4 = {aa, aa};
    E5 = {aa; bb}
  END
```

***Invalid number of arguments for <subst>***

La substitution 'devient égal' est utilisée avec un mauvais nombre de paramètres : le nombre de variables est différent du nombre de valeurs à affecter.

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

***Invalid operation call for <ident> assignment***

L'appel d'opération ne peut pas être utilisé pour affecter ce type de variables.

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

L'appel d'opération ne peut pas être utilisé pour affecter ce type de variables.

***Invalid output parameter <exp>***

Le paramètre effectif de retour d'une opération appelée ne peut pas être de la forme  $f(x)$ . Il faut obligatoirement utiliser une variable intermédiaire.

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

***Invalid output parameters in <op\_header>***

Les paramètres de sortie d'une opération doivent être des identificateurs B, séparés par des virgules et distincts les uns des autres. La définition d'un identificateur B est rappelée chapitre 1.

```
MACHINE M1
OPERATIONS
  a <-- op1 = ...;          /* a n'est pas un identificateur */
  _1 <-- op2(ii) = ...;     /* _1 n'est pas un identificateur " */
  (tt, tt) <-- op3 = ...;  /* tt apparaît deux fois */
  (tt; uu) <-- op4 = ...; /* le séparateur doit être une virgule */
END
```

***Invalid predicate <pred>***

Le prédicat <pred> est syntaxiquement incorrect.

Ce message peut être émis lorsqu'une substitution ou une expression est utilisée alors qu'un prédicat est attendu. Par exemple il ne faut pas confondre le signe d'affectation ' := ' qui s'utilise dans les substitutions uniquement, et le signe d'égalité '=' réservé aux prédicats.

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

***Invalid seen machine <ident\_mach>, it uses other machines***

Une machine effectuant un USES ne peut pas être référencée dans une clause SEES.

```
MACHINE MAC02
USES
  UMAC01
END
```

```
MACHINE MACH
SEES
  MAC02
END
```

***Invalid sequence in <exp>***

L'opérateur utilisé dans <exp> attend en argument une suite.

```
MACHINE
  M1
CONSTANTS
  c1, c2
PROPERTIES
  c1 = size(TRUE) &          /* TRUE n'est pas une suite */
  c2 = first(SeqInconnue) /* SeqInconnue n'est pas une suite */
```

***Invalid substitution <subst>***

La substitution <subst> est syntaxiquement incorrecte. Dans le cas d'un appel d'une opération, le message peut être émis si l'opération est inexistante ou non visible (en particulier les opérations de modification issues d'une machine vue ne sont pas utilisables dans le composant "voyant").

```
MACHINE MACH
OPERATIONS
  op1 = BEGIN
    opinc(0) /* opinc : opération inconnue */
  END
; op2 = BEGIN
  MAXINT /* MAXINT n'est pas une substitution */
  END
; op3 = BEGIN
  v1 = v2 /* v1 := v2 est correct */
  END
END
```

***Invalid syntax for substitution CASE <subst>***

La substitution CASE du composant B analysé est syntaxiquement incorrecte. Ce message est émis lorsqu'une partie obligatoire de la substitution CASE manque.

```
/*Dans la substitution CASE suivante, le second THEN manque.*/
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>***

La substitution IF du composant analysé est syntaxiquement incorrecte. Ce message est émis lorsqu'une partie obligatoire du IF manque.

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

***Invalid syntax for substitution SELECT <subst>***

La substitution SELECT <subst> est syntaxiquement incorrecte. Ce message peut être émis lorsque une partie obligatoire du SELECT manque.

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

***Invalid syntax in operation definition <op>***

La définition d'opération <op> n'a pas pu être analysée. Il peut s'agir d'un problème syntaxique, ou bien d'un problème de priorité. Rappelons que deux opérations doivent être séparées par un point-virgule.

```
/* Dans la clause OPERATIONS suivante,
une erreur d'analyse est due à la
précédence du '|' par rapport au '=' */
MACHINE M1
OPERATIONS
  op1 = skip || skip
END
```

```
/* L'utilisation de BEGIN ... END
permet ici de résoudre le problème */
MACHINE M1
OPERATIONS
  op1 = BEGIN skip || skip END
END
```

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

<ident> désigne une donnée non typée.

<ident> ne peut pas être typée par un élément de record sans label.

```
MACHINE
  M1
  CONCRETE_CONSTANTS
    cc
  PROPERTIES
    cc=rec(2,3)    /* rec(2,3) ne peut typer cc. */
                  /* L'expression cc = rec(champ1:2,champ2:3) est correcte */
END
```

***Invalid use of a record element without label***

Deux éléments de record sans label ne peuvent être comparés. Un élément de record sans label a un typage générique.

```
MACHINE
  M1
  ABSTRACT_VARIABLES
    xx,yy
  INVARIANT
    xx:NAT &
    yy:BOOL &
    rec(xx,yy) = rec(2,TRUE)
    /* Expression incorrecte est : rec(xx,yy) = rec(2,TRUE) */
    /* xx = 2 || yy = TRUE est correcte */
  INITIALISATION
    xx := 2 ||
    yy := TRUE
END
```

***Invalid valuation of <ident\_const>***

Les règles permettant de valuer les ensembles et les constantes ont été violées. Les types des constantes formelles définis dans la clause PROPERTIES de l'abstraction et les types des valeurs affectées en implémentation doivent être identiques.

Notons de plus qu'un ensemble ne peut être valué par un autre ensemble du même composant.

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

```
IMPLEMENTATION MACH_imp
REFINES MACH
VALUES
  S1 = NAT    /* ok */
; S2 = S1    /* non */
; c1 = TRUE  /* non */
END
```

***<ident\_mach> is not a machine***

Une clause USES, SEES, INCLUDES, EXTENDS ou IMPORTS du composant analysé fait référence à <ident\_mach> qui est un raffinement ou une implémentation. Or seules les machines abstraites peuvent faire l'objet d'une clause de visibilité.

```
IMPLEMENTATION IMP_1
REFINES IMP
END
```

```
MACHINE MACH
SEES
  IMP_1
/*une implémentation ne peut pas
être vue*/
END
```

***<ident> is not an identifier***

L'identificateur <ident> viole les règles syntaxiques définissant les identificateurs B (cf chapitre 1).

```
MACHINE
  M1
CONSTANTS
  5, _1 /* 5 et _1 ne sont pas des identificateurs */
PROPERTIES
  5 : NAT &
  _1 : INT
END
```

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

Lors de l'utilisation d'une égalité, d'une inégalité, d'une affectation, etc..., les types des membres gauches et droits doivent être identiques.

Lors de l'utilisation d'opérateurs tels que  $, ><, / ;$ , etc..., certaines règles sur les types doivent être vérifiées. Par exemple, lors de la composition de deux relations :

relation1 ; relation2

telles que relation1 :  $A \leftrightarrow B$  et relation2 :  $C \leftrightarrow D$ , on doit avoir B et C identiques.

Si ce n'est pas le cas, le message d'erreur est émis.

```
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 & /* incompatibilité */
  v1/=v2 /* incompatibilité */
INITILISATION
  v1:=0 || v2:=TRUE || v3:=""
OPERATIONS
  op1 = v1:= v3 /* incompatibilité */
  vv <-- op2 = vv := 1..2 /\ BOOL; /* incompatibilité */
  vv <-- op5 = vv := relation1 |>> GG; /* incompatibilité */
  vv <-- op7 = vv := EE - FF /* incompatibilité */
END
```

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

Le membre gauche d'une valuation est obligatoirement un identificateur B (cf définition chapitre 1).

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

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

Le membre gauche de <exp> n'a pas été typé. Ce message peut être émis lorsque les prédicats de typage sont placés après la propriété <exp>. La définition d'un prédicat de typage est rappelée chapitre 1.

```

MACHINE
  M1(pp)
CONSTRAINTS
  pp <= 1 & /* pp n'a pas encore été typé*/
  pp : NAT
CONSTANTS
  cc
PROPERTIES
  cc < 2 & /* cc n'a pas encore été typé*/
  cc : NAT
VARIABLES
  vv
INVARIANT
  vv >= 3 & /* vv n'a pas encore été typé*/
  vv : NAT
INITIALISATION
  vv := 0
OPERATIONS
  op(ii) = PRE ii >4 & ii : NAT THEN skip END
          /* ii n'a pas encore été typé */
END
/* Pour corriger cette spécification, il suffit d'inverser les prédicats */

```

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

Une comparaison se fait obligatoirement entre des entiers.

```

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

```

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

Le membre gauche de <exp> n'a pas été typé. Ce message peut être émis lorsque les prédicats de typage sont placés après la propriété <exp>. La définition d'un prédicat de typage est rappelée chapitre 1.

```

REFINEMENT
  M1
CONSTANTS
  PP
PROPERTIES
  pp /= 1 & /* pp n'a pas encore été typé*/
  pp : NAT
OPERATIONS
  uu, vv <-- op = BEGIN
    uu := vv; /* vv n'a pas encore été typé */
    vv := 1
  END
END

```

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

L'opérateur utilisé dans <exp> attend un entier en membre gauche.

```

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

```

***Left hand side of <exp> should be a relation***

L'opérateur utilisé dans <exp> attend une relation en membre gauche.

```

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 n'est pas une relation */
  v2 <-- op2 = v2 := (Rinconnue >< relation)
    /* Rinconnue n'est pas une relation */
END

```

***Left hand side of <exp> should be a sequence***

L'opérateur utilisé dans &lt;exp&gt; attend une suite en membre gauche.

```

MACHINE
  M1
CONSTANTS
  sequence
PROPERTIES
  sequence : seq(INT)
OPERATIONS
  vv <-- op1 = vv := 2 ^ sequence; /*2 n'est pas une suite*/
  vv <-- op2 = vv := SeqInconnue <- 2 /*SeqInconnue n'est pas une suite*/
END

```

***Left hand side of <exp> should be a set***

L'opérateur utilisé dans &lt;exp&gt; attend un ensemble en membre gauche.

```

MACHINE
  M1
SETS
  SS; TT
VARIABLES
  relation,
  relation2
INVARIANT
  relation : SS <-> TT &
  relation2 : 2 <-> SS /*2 n'est pas un ensemble*/
INITIALISATION
  relation :: SS <-> TT ||
  relation2 :: EnsInconnu <-> SS
/* EnsInconnu n'est pas un ensemble*/
OPERATIONS
  vv <-- op1 = vv := 3 \ / 1..2;
/*3 n'est pas un ensemble*/
  vv <-- op2 = vv := (5 <| relation);
/*5 n'est pas un ensemble*/
  vv <-- op4 = vv := TRUE * SS
/*TRUE n'est pas un ensemble*/
END

```

***Local operation <ident\_op> has not been implemented***

Dans une implémentation, toutes les opérations locales définies dans la clause LOCAL\_OPERATIONS doivent être implémentées dans la clause OPERATIONS.

```
IMPLEMENTATION
  M1_1
REFINES
  M1
LOCAL_OPERATIONS
  op = skip
END
/*op devrait être implémentée*/
```

***Local variable <ident> is read before being initialised***

Ce message n'apparaît qu'en implémentation. La variable locale <ident> est une variable définie dans une substitution VAR ou dans la liste des paramètres de sortie d'une opération. Elle est utilisée alors qu'elle n'a pas été initialisée par une substitution.

```
IMPLEMENTATION M1_1
REFINES M1
OPERATIONS
  ss, tt <-- op(ii) =
    IF ii > 1 THEN
      ss := 2
    END;
  tt := ss
  /* ss n'a pas été initialisée dans toutes les branches du IF */
END
```

***Machine <ident\_mach> can not be refined, it uses other machines***

Une machine effectuant un USES ne peut pas être raffinée, c'est un module abstrait.

```
MACHINE M1
USES
  M2
END
```

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

<p><i>Machine &lt;ident_mach1&gt; should be included in &lt;ident_mach2&gt; : it has been included in the abstraction of &lt;ident_mach2&gt;</i></p>	
<p>La machine &lt;ident_mach1&gt; est incluse dans un composant qui raffine &lt;ident_mach2&gt;. Or &lt;ident_mach2&gt; n'inclut pas &lt;ident_mach1&gt;, par contre certaines de ses abstractions le font. Ceci est interdit.</p> <p>En effet si un composant Mi inclut une machine N, alors :</p> <ul style="list-style-type: none"> <li>- soit aucun de ses raffinements n'inclut ni n'importe N</li> <li>- soit un de ses raffinements Mj inclut ou importe N, et dans ce cas TOUS les composants de la chaîne de raffinement entre Mi et Mj doivent inclure N.</li> </ul> <p>Cette contrainte permet d'éviter certains conflits d'identificateurs.</p>	
<pre>MACHINE M1 INCLUDES M2 END</pre>	<pre>REFINEMENT M1_1 REFINES M1 END</pre>
<pre>REFINEMENT M1_2 REFINES M1_1 INCLUDES M2 /* interdit si M1_1 n'inclut pas M2*/ END</pre>	

***Machine <ident\_mach1> should be seen by <ident\_mach2>***

Le composant analysé M1 inclut deux machines M2 et M4 telles que M2 utilise M4. L'invariant de collage (ou gluing invariant) liant les variables de M2 et M4 est défini dans la clause INVARIANT de M2 mais doit être prouvé au niveau de M1.

Dans le cas de figure qui génère ce message, M2 voit une machine M3 et les variables de M3 interviennent dans l'invariant de collage liant M2 et M4. Or le composant M1 ne voit pas M3, il ne connaît donc rien sur ses variables. La preuve est vouée à l'échec. Il faut donc rajouter M3 dans la clause SEES de 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 &
  /*invariant de collage M2/M4 : */
  v2 < ss.v3 + uu.v4
INITIALISATION
  v2 :: NAT
END
```

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

***Machine <ident\_mach1> should be seen by <ident\_mach2> (it is seen by <ident\_mach3>)***

Si une machine est vue par un composant, elle doit continuer à l'être par tous les composants qui le suivent dans la chaîne de raffinement. Ce message est donc émis si un composant M est raffiné par un composant N tel que la machine S apparaît dans la clause SEES de M mais pas dans celle de N.

```
MACHINE M1
SEES M2
END
```

```
REFINEMENT M1_1
REFINES M1
/* il manque :
SEES M2 */
END
```

<i>Machine &lt;ident_mach&gt; should have parameters</i>
Le composant analysé contient une clause CONSTRAINTS alors qu'il n'a pas de paramètres. Pourtant, cette clause permet seulement de définir les propriétés des paramètres du composant.
<pre> MACHINE   M1 CONSTANTS   c1 CONSTRAINTS   c1 : NAT /* prédicat à mettre dans une clause PROPERTIES */ END         </pre>

<i>Machine &lt;ident_mach1&gt; uses &lt;ident_mach2&gt; which is neither included nor extended</i>		
Lorsqu'une machine qui effectue un USES est incluse, toutes les machines utilisées doivent également être incluses. Par exemple, si M1 utilise M2 qui utilise M3, alors si M2 est incluse, il faut également inclure M1 et M3.		
<table border="0" style="width: 100%;"> <tr> <td style="width: 50%; vertical-align: top;"> <pre> MACHINE MAC02 USES   UMAC01 END         </pre> </td> <td style="width: 50%; vertical-align: top;"> <pre> MACHINE MACH INCLUDES   MAC02   /* il faut aussi inclure UMAC01 */ END         </pre> </td> </tr> </table>	<pre> MACHINE MAC02 USES   UMAC01 END         </pre>	<pre> MACHINE MACH INCLUDES   MAC02   /* il faut aussi inclure UMAC01 */ END         </pre>
<pre> MACHINE MAC02 USES   UMAC01 END         </pre>	<pre> MACHINE MACH INCLUDES   MAC02   /* il faut aussi inclure UMAC01 */ END         </pre>	

<i>Missing symbol =&gt; in predicate &lt;pred&gt;</i>
Ce message concerne les expressions de la forme !X.A. Il est émis lorsque A n'est pas de la forme (P => Q). Le prédicat P doit contenir les prédicats de typage des variables de X. La définition d'un prédicat de typage est rappelée au chapitre 1.
<pre> MACHINE   M1 CONSTANTS   vv PROPERTIES   vv : 1..10 &amp;   !xx.(xx : NAT &amp; xx &gt;5 &amp; xx &gt; vv)   /* écriture correcte :   !xx.(xx : NAT &amp; xx &gt; 5 =&gt; xx &gt; vv)   */ END         </pre>

***Multiple assignment of <ident\_var> in parallel substitutions***

Une même variable ne peut pas être affectée plusieurs fois dans une substitution multiple ou dans un retour d'opération.

```
/*La machine suivante tente de donner à
la variable v1, la valeur 0 et la valeur
1 en parallèle. Elle est incorrecte.*/
MACHINE
  MACH
VARIABLES
  v1
INVARIANT
  v1:NAT
INITIALISATION
  v1,v1:=0,1
END
```

```
/*La machine suivante propose
plusieurs solutions pour exprimer
correctement l'idée intuitive qui
était implémentée ci-contre,
c'est-à-dire que v1 devient égal
soit à 0, soit à 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
sont également possibles*/
END
```

***Multiple assignment of <ident> when calling local operation <ident\_op>***

L'opération locale modifie la variable <ident>. Lors de l'appel, l'un de ses paramètres de sortie effectifs est également la variable <ident>. Cet appel est donc 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
/* une écriture correcte serait :
  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>)***

L'identificateur <ident> est défini à la fois dans le composant analysé et dans un composant visible. Ce conflit peut être dû à la transitivité de la clause INCLUDES.

***Multiple definition of identifier <ident> in <ident\_mach>***

Le composant analysé contient un conflit interne d'identificateurs.

```
MACHINE MACH
CONSTANTS
  cst1,cst2,cst2    /* cst2 apparaît deux fois */
PROPERTIES
  cst1 : NAT & cst2 : NAT
END
```

***Multiple promotion of operation <ident\_op>***

Chaque opération promue ne doit être mentionnée qu'une seule fois.

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

***Multiple reference of machine <ident\_mach>***

Une même machine ne doit apparaître qu'une seule fois dans les clauses INCLUDES, IMPORTS, EXTENDS, SEES, USES d'un même composant.

```
MACHINE M1
INCLUDES M2
SEES M2
/* problème car M2 apparaît deux fois */
END
```

***Multiple use of constant <ident\_cst> in branches of CASE***

La même constante <ident\_cst> apparaît plusieurs fois dans les branches d'une substitution CASE, alors que les différents cas d'une telle substitution doivent être exclusifs.

```
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 apparaît à nouveau */
    END
  END
END
END
```

***Multiple use of identifier <ident> in branches of CASE***

Le même identificateur apparaît plusieurs fois dans les branches d'une substitution CASE, alors que les différents cas d'une telle substitution doivent être exclusifs.

```

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 apparaît à nouveau */
      ELSE out := 3
    END
  END
END
END

```

***Multiple use of label <ident\_label> in a record expression***

Les labels contenus dans les ensembles de records ou les éléments de record doivent être deux à deux distincts.

```

MACHINE
  M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : struct(aa:NAT,bb:BOOL,aa:0..9)
      /* Remplacer la dernière utilisation du label
         aa par : ee */
END

```

***Object <ident> cannot be valued***

L'entité <ident> est évaluée, alors qu'elle est non valable ou inconnue. Il peut s'agir d'une erreur de frappe ou d'un problème de visibilité.

<pre> MACHINE M1 SETS   S1; S2 CONSTANTS   c1 PROPERTIES   c1 = 1 END </pre>	<pre> IMPLEMENTATION M1_1 REFINES M1 VALUES   S1 = NAT ; S2 = NAT1 ; c1 = 1 ; c2 = 1 /* c2 inconnu */ END </pre>
--	--

<p><i>&lt;ident_op&gt; of machine &lt;ident_mch&gt; is called simultaneously with a modification of variable &lt;ident_var&gt;</i></p>	
<p>Une opération locale peut modifier directement une variable importée. Ce message est émis lorsqu'on modifie une variable importée en parallèle avec un appel d'une opération de la même machine importée.</p>	
<pre> IMPLEMENTATION M1_1 REFINES M1 IMPORTS MO LOCAL_OPERATIONS   loc_op = BEGIN increment    v2 := v2-1   /* l'invariant de MO est brisé */ END         </pre>	<pre> MACHINE MO VARIABLES   v1,v2 INVARIANT ENDv1:NAT &amp; v2:NAT &amp; v1&lt;=v2 INITIALISATION   v1:=0    v2:=0 OPERATIONS   increment = PRE     v1&lt;v2   THEN     v1:=v1+1   END END         </pre>

<p><i>Only one ABSTRACT_CONSTANTS clause is allowed</i></p>	
<p>Ce message est émis lorsqu'une clause ABSTRACT_CONSTANTS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses ABSTRACT_CONSTANTS dans le même composant, ou une clause ABSTRACT_CONSTANTS et une clause HIDDEN_CONSTANTS. En effet ces deux mots clefs ont la même signification.</p>	
<pre> MACHINE M1 ABSTRACT_CONSTANTS   cst1 HIDDEN_CONSTANTS   cst2 PROPERTIES   cst1 : NAT &amp; cst2 : NAT END         </pre>	

***Only one ABSTRACT\_VARIABLES clause is allowed***

Ce message est émis lorsqu'une clause ABSTRACT\_VARIABLES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses ABSTRACT\_VARIABLES dans la même machine, ou une clause ABSTRACT\_VARIABLES et une clause VARIABLES ou HIDDEN\_VARIABLES. En effet ces trois mots clefs ont la même signification.

```

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

```

***Only one ASSERTIONS clause is allowed***

Ce message est émis lorsqu'une clause ASSERTIONS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses ASSERTIONS.

```

MACHINE MACH
ASSERTIONS
  TRUE
ASSERTIONS
  TRUE
END

```

***Only one component can be refined : <ident\_mach> is chosen for the Type Check continuation***

La clause REFINES du raffinement ou de l'implémentation analysé fait référence à plusieurs machines. C'est illégal, puisque deux composants ne peuvent pas être raffinés en même temps.

Dans ce cas le contrôle continue avec comme composant raffiné le dernier de la liste. C'est le nom de celui-ci qui apparaît dans le message d'erreur.

```

REFINEMENT M1_1
REFINES
  M1a, M1b
END

```

***Only one CONCRETE\_CONSTANTS clause is allowed***

Ce message est émis lorsqu'une clause CONCRETE\_CONSTANTS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses CONCRETE\_CONSTANTS dans le même composant, ou une clause CONCRETE\_CONSTANTS et une clause VISIBLE\_CONSTANTS ou CONSTANTS. En effet ces trois mots clefs ont la même signification.

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

***Only one CONCRETE\_VARIABLES clause is allowed***

Ce message est émis lorsqu'une clause CONCRETE\_VARIABLES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses CONCRETE\_VARIABLES, ou une clause CONCRETE\_VARIABLES et une clause VISIBLE\_VARIABLES. En effet ces deux mots clefs ont la même signification.

```
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***

Ce message est émis lorsqu'une clause CONSTANTS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses CONSTANTS dans le même composant, ou une clause CONSTANTS et une clause VISIBLE\_CONSTANTS ou CONCRETE\_CONSTANTS. En effet ces trois mots clefs ont la même signification.

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

***Only one CONSTRAINTS clause is allowed***

Ce message est émis lorsqu'une clause CONSTRAINTS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses CONSTRAINTS dans le même composant.

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

***Only one EXTENDS clause is allowed***

Ce message est émis lorsqu'une clause EXTENDS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses EXTENDS dans le même composant.

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

***Only one HIDDEN\_CONSTANTS clause is allowed***

Ce message est émis lorsqu'une clause HIDDEN\_CONSTANTS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses HIDDEN\_CONSTANTS dans le même composant, ou une clause HIDDEN\_CONSTANTS et une clause ABSTRACT\_CONSTANTS. En effet ces deux mots clefs ont la même signification.

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

***Only one HIDDEN\_VARIABLES clause is allowed***

Ce message est émis lorsqu'une clause HIDDEN\_VARIABLES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses HIDDEN\_VARIABLES dans la même machine, ou une clause HIDDEN\_VARIABLES et une clause VARIABLES ou ABSTRACT\_VARIABLES. En effet ces trois mots clefs ont la même signification.

```
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***

Ce message est émis lorsqu'une clause IMPORTS n'a pas sa place dans l'implémentation analysée. En particulier il interdit d'avoir deux clauses IMPORTS dans la même implémentation.

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

***Only one INCLUDES clause is allowed***

Ce message est émis lorsqu'une clause INCLUDES n'a pas sa place dans le composant analysé. En particulier, il est interdit d'avoir deux clauses INCLUDES dans le même composant.

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

***Only one INITIALISATION clause is allowed***

Ce message est émis lorsqu'une clause INITIALISATION n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses INITIALISATION dans le même composant.

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

***Only one INVARIANT clause is allowed***

Ce message est émis lorsqu'une clause INVARIANT n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses INVARIANT dans le même composant.

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

***Only one LOCAL\_OPERATIONS clause is allowed***

Ce message est émis lorsqu'une clause LOCAL\_OPERATIONS n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses LOCAL\_OPERATIONS dans le même composant.

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

***Only one OPERATIONS clause is allowed***

Ce message est émis lorsqu'une clause OPERATIONS n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses OPERATIONS dans le même composant.

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

***Only one PROMOTES clause is allowed***

Ce message est émis lorsqu'une clause PROMOTES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses PROMOTES dans le même composant. Toutes les opérations promues doivent figurer dans la même clause PROMOTES, même si elles proviennent de machines distinctes.

```
MACHINE MACH
INCLUDES MAC01(10), MAC02(1..1000, BOOL)
PROMOTES
  op_01
PROMOTES
  op_02 /* non correct */
END
```

***Only one PROPERTIES clause is allowed***

Ce message est émis lorsqu'une clause PROPERTIES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses PROPERTIES dans le même composant.

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

***Only one REFINES clause is allowed***

Ce message est émis lorsqu'une clause REFINES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses REFINES dans le même composant. C'est illégal, puisque deux composants ne peuvent pas être raffinés en même temps.

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

***Only one SEES clause is allowed***

Ce message est émis lorsqu'une clause SEES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses SEES dans le même composant.

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

***Only one SETS clause is allowed***

Ce message est émis lorsqu'une clause SETS n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses SETS dans le même composant.

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

***Only one USES clause is allowed***

Ce message est émis lorsqu'une clause USES n'a pas sa place dans la machine abstraite analysée. En particulier il est interdit d'avoir deux clauses USES dans la même machine.

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

***Only one VALUES clause is allowed***

Ce message est émis lorsqu'une clause VALUES n'a pas sa place dans l'implémentation analysée. En particulier il est interdit d'avoir deux clauses VALUES dans la même implémentation.

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

***Only one VARIABLES clause is allowed***

Ce message est émis lorsqu'une clause VARIABLES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses VARIABLES dans le même composant, ou une clause VARIABLES et une clause HIDDEN\_VARIABLES ou ABSTRACT\_VARIABLES. En effet ces trois mots clefs ont la même signification.

```
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***

Ce message est émis lorsqu'une clause VISIBLE\_CONSTANTS n'a pas sa place dans le composant analysé. En particulier on ne peut pas avoir deux clauses VISIBLE\_CONSTANTS dans le même composant, ou une clause VISIBLE\_CONSTANTS et une clause CONSTANTS ou CONCRETE\_CONSTANTS. En effet ces trois mots clefs ont la même signification.

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

***Only one VISIBLE\_VARIABLES clause is allowed***

Ce message est émis lorsqu'une clause VISIBLE\_VARIABLES n'a pas sa place dans le composant analysé. En particulier il est interdit d'avoir deux clauses VISIBLE\_VARIABLES, ou une clause VISIBLE\_VARIABLES et une clause CONCRETE\_VARIABLES. En effet ces deux mots clefs ont la même signification.

```
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>***

L'opération <ident\_op> apparaît dans la clause PROMOTES du composant analysé, mais n'est pas définie dans l'abstraction de celui-ci.

Lorsqu'une opération est promue, elle est considérée comme ayant été écrite dans le composant lui-même. Or, dans un raffinement, les opérations locales ne peuvent être que des raffinements des opérations de la machine abstraite, avec exactement la même 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 produit un message d'erreur car il ne correspond à
aucune opération de la machine M1 */
END
```

<b><i>Operation &lt;ident_op&gt; does not exist in abstraction</i></b>	
Les opérations locales d'un raffinement ou d'une implémentation doivent obligatoirement être spécifiées dans la machine abstraite. Vous ne pouvez pas définir une nouvelle opération dans un raffinement.	
<pre> MACHINE M1 OPERATIONS   res &lt;-- op1 (xx,yy) =   PRE     xx:1..100 &amp; yy:1..100   THEN     res :: BOOL   END END         </pre>	<pre> REFINEMENT M1_1 REFINES M1 OPERATIONS   op2 = skip   /*op2 n'existe pas dans M1*/ END         </pre>

<b><i>Operation &lt;ident_op&gt; has not been implemented</i></b>	
Dans une implémentation, toutes les opérations définies dans la machine abstraite doivent être implémentées.	
<pre> MACHINE   M1 OPERATIONS   op = skip END         </pre>	<pre> IMPLEMENTATION   M1_1 REFINES   M1 END /*op devrait être implémentée*/         </pre>

<b><i>Operation name &lt;ident_op&gt; in &lt;op_header&gt; is a keyword</i></b>	
<ident_op> est un mot réservé du langage (cf chapitre 1) : il ne peut pas servir à nommer une opération.	
<pre> MACHINE M1 OPERATIONS   MAXINT(xx) = ... /* MAXINT : mot réservé */ ; res &lt;-- skip = ... /* skip : mot réservé */ END         </pre>	

<b><i>Operation name &lt;ident_op&gt; in &lt;op_header&gt; should be an identifier</i></b>	
Le nom des opérations doit être un nom simple, c'est-à-dire un identificateur B (cf définition chapitre 1).	
<pre> MACHINE M1 OPERATIONS   _1 &lt;-- val = ... /* _1 n'est pas un identificateur */ ; res &lt;-- f(x) = ... /* f n'est pas un identificateur */ END         </pre>	

***Output parameter <ident> has not been initialised***

Ce message n'apparaît qu'en implémentation. Le paramètre de sortie <ident> de l'opération en cours de type-check n'a pas été initialisé par le corps de cette opération.

```
IMPLEMENTATION M1_1
REFINES M1
OPERATIONS
  ss <-- op(ii) =
    IF ii > 1 THEN
      ss := 2
    END
  END
END
/* ss n'a pas été initialisé dans toutes les branches du IF */
```

***Output parameters <list\_ident> have not been initialised***

Ce message n'apparaît qu'en implémentation. Les paramètres de sortie <list\_ident> de l'opération en cours de type-check n'ont pas été initialisés par le corps de cette opération.

```
IMPLEMENTATION M1_1
REFINES M1
OPERATIONS
  ss, tt <-- op(ii) =
    IF ii > 1 THEN
      ss := 2
    ELSE
      tt := 3
    END
  END
END
/* ss et tt n'ont pas été typés dans toutes les branches du IF */
```

***Parameter <ident> has not been typed***

Tous les paramètres scalaires doivent être typés dans la clause CONSTRAINTS au moyen d'un prédicat de typage (cf définition chapitre 1).

```
MACHINE MACH(par1,par2,par3)
CONSTRAINTS
  par1 : NAT &
  par2 < par1 /* par2 non typé */
END
/* par3 non typé */
```

<b><i>Parameter &lt;ident&gt; of &lt;ident_op&gt; is already defined in &lt;ident_mach&gt;</i></b>	
Un conflit entre les paramètres entrées/sorties de l'opération promue <ident_op> et un identificateur visible de la machine <ident_mach> a été détecté.	
<pre> MACHINE M2 OPERATIONS   op1(xx) = PRE xx:NAT THEN     skip   END END         </pre>	<pre> MACHINE M1 INCLUDES M2 PROMOTES   op1 VARIABLES   xx /* conflit avec xx de op1 */ INVARIANT   xx : NAT INITIALISATION   xx :: NAT END         </pre>

<b><i>Parameters of abstraction &lt;ident_mch1&gt; and refinement &lt;ident_mch2&gt; differ</i></b>	
Tous les raffinements d'un développement vertical doivent avoir les mêmes paramètres que la machine abstraite (le nombre et les noms des paramètres doivent être identiques).	
<pre> MACHINE MACH(var1,var2,ENS) CONSTRAINTS   var1 : ENS &amp;   var2 : ENS END         </pre>	<pre> REFINEMENT MACH_1(var,ENS) /* var en trop ;    var1 et var2 manquent */ REFINES   MACH END         </pre>

<b><i>Prefix &lt;ident1&gt; in &lt;ident1&gt;.&lt;ident2&gt; is a keyword</i></b>	
Le préfixe <ident1> est un mot réservé du langage (cf chapitre 1). Il ne peut pas servir à préfixer une machine.	
<pre> MACHINE M1 SEES skip.M0 END         </pre>	

<b><i>Prefix in &lt;ident&gt; should be an identifier</i></b>	
Un préfixe de renommage est obligatoirement un identificateur B correct (cf définition chapitre 1).	
<pre> MACHINE MACH INCLUDES   1.MAC1 ,   #10x.MAC1 ,   &lt;&gt;.MAC1 END         </pre>	

***Prefix <ident> is used twice***

Au sein d'un même composant chaque préfixe de renommage ne peut être utilisé qu'une seule fois, même s'il renomme des machines distinctes.

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

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

L'opérateur utilisé dans <exp> attend en argument une fonction dont l'ensemble de départ est un ensemble d'ensembles.

```
MACHINE
  M1
SETS
  SS; TT
CONSTANTS
  fonction,
  relation
PROPERTIES
  fonction : SS --> TT &
  relation = rel(fonction)
  /* TT devrait être une ensemble d'ensembles */
END
```

***Read only or unknown left hand side <ident>***

Ce message d'erreur est émis lorsque la substitution "devient égal" ou "appel d'opération" tente de modifier une entité qui ne peut pas l'être. Les tables de visibilité indiquent quelles sont les entités accessibles en écriture, et celles qui ne le sont pas, en fonction de la clause considérée.

```
MACHINE M1
CONSTANTS
  c1
PROPERTIES
  c1 : NAT
OPERATIONS
  ini = (c1, IdInconnu := 0, 0)
  /* c1 constante non modifiable,
  IdInconnu identificateur inconnu */
END
```

<b><i>Refined component &lt;ident&gt; cannot be renamed</i></b>	
Le nom du composant apparaissant dans la clause REFINES est précédé d'un préfixe de renommage. Ceci est interdit.	
<pre> /* Raffinement faux : */ REFINEMENT M1_1 REFINES pp.M1 END         </pre>	<pre> /* Raffinement correct : */ REFINEMENT M1_1 REFINES M1 END         </pre>

<b><i>Right hand side of comparison &lt;exp&gt; has not been typed</i></b>	
Le membre droit de <exp> n'a pas été typé. Ce message peut être émis lorsque les prédicats de typage sont placés après la propriété <exp>. La définition d'un prédicat de typage est rappelée chapitre 1.	
<pre> MACHINE   M1(pp) CONSTRAINTS   1 &lt; pp &amp; /* pp n'a pas encore été typé*/   pp : NAT CONSTANTS   cc PROPERTIES   2 &lt;= cc &amp; /* cc n'a pas encore été typé*/   cc : NAT VARIABLES   vv INVARIANT   3 &gt; vv &amp; /* vv n'a pas encore été typé*/   vv : NAT INITIALISATION   vv := 0 OPERATIONS   op(ii) = PRE 4 &gt;= ii &amp; ii : NAT THEN skip END           /* ii n'a pas encore été typé*/ END /* Pour corriger cette spécification, il suffit d'inverser les prédicats */         </pre>	

<b><i>Right hand side of comparison &lt;exp&gt; should be an integer</i></b>	
Une comparaison se fait obligatoirement entre des entiers.	
<pre> MACHINE   M1 CONSTANTS   cc PROPERTIES   cc : BOOL &amp;   2 &lt;= cc END         </pre>	

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

Le membre droit de <exp> n'a pas été typé. Ce message peut être émis lorsque les prédicats de typage sont placés après la propriété <exp>. La définition d'un prédicat de typage est rappelée chapitre 1.

```

REFINEMENT
  M1
VARIABLES
  pp
INVARIANT
  1 /= pp & /* pp n'a pas encore été typé*/
  pp : NAT
INITIALISATION
  pp := 4
OPERATIONS
  uu, vv <-- op = BEGIN
    uu := 1;
    IF vv = 1 THEN /* vv n'a pas encore été typé */
      vv := 2
    END
  END
END
END

```

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

L'opérateur utilisé dans <exp> attend un entier en membre droit.

```

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

```

***Right hand side of <exp> should be a relation***

L'opérateur utilisé dans &lt;exp&gt; attend une relation en membre droit.

```

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 n'est pas une relation */
  v2 <-- op2 = v2 := (relation >< Rinconnue)
    /* Rinconnue n'est pas une relation */
END

```

***Right hand side of <exp> should be a sequence***

L'opérateur utilisé dans &lt;exp&gt; attend une suite en membre droit.

```

MACHINE
  M1
PROPERTIES
  sequence : seq(INT)
OPERATIONS
  vv <-- op1 = vv := sequence ^ 2;
    /* 2 n'est pas une suite */
  vv <-- op2 = vv := a1 -> SeqInconnue
    /* SeqInconnue n'est pas une suite */
END

```

***Right hand side of <exp> should be a set***

L'opérateur utilisé dans &lt;exp&gt; attend un ensemble en membre droit.

```

MACHINE
  M1
SETS
  SS; TT
VARIABLES
  relation1, relation2
INVARIANT
  relation1 : SS <-> TT
INITIALISATION
  relation1 :: SS <-> TT
OPERATIONS
  vv <-- op2 = vv := 1..2 /\ EnsInconnu;
                                     /* EnsInconnu n'est pas un ensemble */
  vv <-- op3 = (vv :: SS --> 5);      /*5 n'est pas un ensemble */
  vv <-- op4 = vv := SS - TRUE      /*TRUE n'est pas un ensemble*/
END

```

***Seen machine <ident\_mach> cannot be instanciated***

Seules les machines référencées dans les clauses INCLUDES, IMPORTS et EXTENDS peuvent être instanciées.

```

MACHINE MACH
SEES
  MCH01(NAT)
END

```

***Sequence in <exp> should not be empty***

L'opérateur utilisé dans &lt;exp&gt; attend en argument une suite non vide.

```

MACHINE
  M1
CONSTANTS
  c1
PROPERTIES
  c1 = first(<>) /* first attend en argument une suite non vide */
END

```

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

Ce message est émis lorsque la substitution séquençement ";" est utilisée dans la spécification d'une opération locale. Or cette substitution n'est pas autorisée en spécification. Par contre la substitution simultanée "||" est recommandée.

```

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
/*écriture correcte: v1:=0 || v2 := 0 ou v1,v2 := 0,0 */
  END
OPERATIONS
  op = BEGIN
    v1 := 0; v2 := 0
  END
/* écriture correcte ici */
END

```

*Sequencing substitution is forbidden in a machine : <subst>*

Ce message est émis lorsque la substitution séquençement ";" est utilisée dans une machine abstraite. Or cette substitution n'est autorisée qu'en raffinement et en implémentation. Par contre la substitution simultanée "||" est recommandée en spécification.

```

MACHINE MACH
VARIABLES
  v1, v2
INVARIANT
  v1 : NAT & v2 : NAT
INITIALISATION
  v1 := 0; v2 := 0
/*écriture correcte: v1:=0 || v2 := 0 ou v1,v2 := 0,0 */
END

```

***Set <ident\_set> is already defined***

Un conflit d'identificateurs faisant intervenir l'ensemble <ident\_set> a été détecté.

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

***The ABSTRACT\_CONSTANTS clause is not allowed in an implementation***

La clause ABSTRACT\_CONSTANTS ne peut pas être utilisée dans une implémentation. Dans ce cas il est préférable d'utiliser la clause VISIBLE\_CONSTANTS.

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

***The ABSTRACT\_VARIABLES clause is not allowed in an implementation***

La clause ABSTRACT\_VARIABLES ne peut pas être utilisée dans une implémentation. Dans ce cas il est préférable d'utiliser la clause CONCRETE\_VARIABLES.

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

***The component <ident\_mach> cannot be referenced by itself***

Un composant B ne peut pas se référencer lui-même dans une de ses clauses SEES, INCLUDES, EXTENDS ou USES.

```
MACHINE MACH(XX)
CONSTRAINTS
  card(XX)=5
INCLUDES
  MACH(1..5) /* tentative illégale de récursivité */
END
```

***The CONSTRAINTS clause is only allowed in a machine***

Le composant analysé ne devrait pas contenir une clause CONSTRAINTS. Ce message est émis dans un raffinement ou dans une implémentation lorsque l'on tente de spécifier les contraintes des paramètres. Ces contraintes doivent être spécifiées exclusivement dans la machine abstraite.

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

***The HIDDEN\_CONSTANTS clause is not allowed in an implementation***

La clause HIDDEN\_CONSTANTS ne peut pas être utilisée dans une implémentation. Dans ce cas il est préférable d'utiliser la clause VISIBLE\_CONSTANTS.

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

***The HIDDEN\_VARIABLES clause is not allowed in an implementation***

La clause HIDDEN\_VARIABLES ne peut pas être utilisée dans une implémentation. Dans ce cas il est préférable d'utiliser la clause CONCRETE\_VARIABLES.

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

***The implementation <ident\_mach> cannot be refined***

Le composant analysé raffine une implémentation. Or, seuls les machines abstraites et les raffinements peuvent être raffinés. L'implémentation est l'étape finale d'un développement vertical (développement par raffinements successifs).

```
IMPLEMENTATION IMP
REFINES MACH
END
```

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

***The IMPORTS clause is only allowed in an implementation***

Ce message est émis lorsqu'une machine abstraite ou un raffinement contient une clause IMPORTS. Celle-ci est exclusivement réservée à l'implémentation. Par contre la clause INCLUDES peut être utilisée.

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

***The INCLUDES clause is not allowed in an implementation***

Ce message est émis lorsqu'une implémentation contient une clause INCLUDES. Celle-ci est uniquement autorisée dans les machines abstraites et les raffinements. Par contre la clause IMPORTS, dédiée à l'implémentation, peut être utilisée.

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

***The LOCAL\_OPERATIONS clause is only allowed in an implementation***

Ce message est émis lorsqu'une machine abstraite ou un raffinement contient une clause LOCAL\_OPERATIONS. Celle-ci est exclusivement réservée à l'implémentation.

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

***The refined machine <ident\_mach> cannot be required***

La machine abstraite raffinée par le composant analysé ne peut apparaître dans aucune des clauses de visibilité de celui-ci.

```
REFINEMENT MAC02
REFINES MACH
INCLUDES
  MACH /* MACH ne peut pas être incluse */
END
```

***The REFINES clause is not allowed in a machine***

Ce message est émis lorsqu'une clause REFINES apparaît dans une machine abstraite, alors qu'une machine abstraite ne peut pas raffiner un composant B. Seuls un raffinement (identifié par le premier mot du source REFINEMENT) et une implémentation (identifiée par le premier mot du source IMPLEMENTATION) peuvent (et doivent) contenir une clause REFINES.

```
MACHINE M0
REFINES
  M1
END
```

***The REFINES clause missing***

Le raffinement ou l'implémentation analysé n'a pas de clause REFINES. Cette clause est obligatoire.

```
REFINEMENT REF_1
END
```

***The USES clause is only allowed in a machine***

Ce message est émis lorsqu'un raffinement ou une implémentation contient une clause USES. Cette clause n'est autorisée que dans une machine abstraite.

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

***The VALUES clause is only allowed in an implementation***

Ce message est émis lorsqu'une machine abstraite ou un raffinement contient une clause VALUES. Les valuations des constantes et des ensembles ne sont possibles qu'en implémentation.

La clause PROPERTIES peut éventuellement contraindre une constante à prendre une valeur donnée. Mais elle devra quand même être évaluée, avec la même valeur, dans l'implémentation.

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

***The VARIABLES clause is not allowed in an implementation***

Ce message est émis lorsqu'une implémentation contient une clause VARIABLES. En effet, celle-ci est équivalente à la clause HIDDEN\_VARIABLES, elle ne peut donc pas être utilisée dans une implémentation. Il est préférable d'utiliser dans ce cas la clause CONCRETE\_VARIABLES.

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

***Unknown renamed identifier : <ident1>.<ident2>***

La forme <ident1>.<ident2> est un renommage : elle désigne l'identificateur <ident2> défini dans une machine requise renommée grâce au préfixe <ident1>. Ce message est émis lorsque l'identificateur <ident2> n'est visible dans aucune des machines renommées avec le préfixe <ident1>. Il peut s'agir d'une erreur de frappe ou d'une violation des contraintes de visibilité.

```
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***

Seules les machines référencées dans les clauses INCLUDES, IMPORTS et EXTENDS peuvent être instanciées.

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

***Use of non implementable arrays in <exp>***

Ce message est émis en implémentation. Un tableau n'est pas implémentable en B0 si son domaine n'est pas un intervalle ou un ensemble énuméré.

```
IMPLEMENTATION M1_1
REFINES M1
CONCRETE_CONSTANTS
  cc
PROPERTIES
  cc : INTEGER --> BOOL
CONCRETE_VARIABLES
  vv
INVARIANT
  vv : INTEGER --> BOOL
INITIALISATION
  vv := cc /* cc n'a pas un ensemble d'indices fini */
END
```

***Variable <ident\_var> has not been typed***

Toutes les variables doivent être typées dans la clause INVARIANT au moyen d'un prédicat de typage (cf définition chapitre 1).

```
MACHINE MACH
VARIABLES
  var1, var2, var3
INVARIANT
  var1 : NAT &
  var2 < var1 /* var2 doit être typée */
              /* var3 doit être typée */
INITIALISATION
  var1, var2, var3 := 5, 6, 7
END
```

***Variable <ident> is not an implementable array***

Ce message est émis en implémentation. Un tableau n'est pas implémentable en B0 si son domaine n'est pas un intervalle ou un ensemble énuméré.

```
IMPLEMENTATION M1_1
REFINES M1
CONCRETE_VARIABLES vv
INVARIANT
  vv : INTEGER --> BOOL
INITIALISATION
  vv := INTEGER * {TRUE} /* INTEGER n'est pas borné */
END
```

<b><i>Variable &lt;ident&gt; should be initialised</i></b>
Toutes les variables définies dans un composant doivent être initialisées dans la clause INITIALISATION.
<pre> MACHINE MACH VARIABLES   xx,yy INVARIANT   xx:NAT &amp; yy:NAT INITIALISATION   xx:=0    /* yy doit être initialisée */ END </pre>
<b><i>Variant &lt;exp&gt; should designate a natural</i></b>
Dans une boucle WHILE, le variant doit être une expression désignant un entier naturel.
<pre> IMPLEMENTATION M1_1 REFINES M1 OPERATIONS   opM1 = BEGIN     WHILE 12 &lt;0 DO skip INVARIANT 6 : NAT VARIANT "chaîne" END;     /* "chaîne" n'est pas un entier naturel */     WHILE 12 &lt;0 DO skip INVARIANT 6 : NAT VARIANT ident_inconnu END     /* le type de ident_inconnu est inconnu */   END END </pre>
<b><i>VAR substitution is forbidden in a local operation specification : &lt;subst&gt;</i></b>
La substitution VAR est une substitution de programmation réservée au raffinement et à l'implémentation. Dans une spécification d'opération locale, il faut utiliser les substitution LET ou ANY.
<pre> IMPLEMENTATION MM_1 REFINES MM LOCAL_OPERATIONS   op = VAR vv IN vv := 2 END   /* spécification incorrecte     LET vv BE vv = 2 IN skip END est correcte */ OPERATIONS   op = VAR vv IN vv := 2 END   /* implantation correcte */ END </pre>

***VAR substitution is forbidden in a machine : <subst>***

La substitution VAR est une substitution de programmation réservée au raffinement et à l'implémentation. Dans une machine, il faut utiliser les substitution LET ou ANY.

```
/* Machine incorrecte : */
MACHINE M1
OPERATIONS
  op = VAR vv IN vv := 2 END
END
```

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

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

Ce message est émis lorsqu'une boucle WHILE est utilisée dans la spécification d'une opération locale. Or cette instruction n'est pas une substitution de spécification.

```
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
    /* écriture interdite */
    END
OPERATIONS
  opWhile =
    WHILE vv > 10
    DO skip
    INVARIANT vv := NAT
    VARIANT vv
    END
    /* écriture autorisée */
END
```

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

Ce message est émis lorsqu'une boucle WHILE est utilisée dans une machine abstraite ou un raffinement. Or cette substitution n'est autorisée qu'en implémentation.

```

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>***

Lors d'une inclusion avec instanciation, vous devez instancier tous les paramètres de la machine incluse.

```

MACHINE
  M1
INCLUDES
  M2(TRUE)
  /* il manque la valeur de p2 */
END

```

```

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

```

***Wrong type for actual input parameters of called operation <ident\_op>***

Les paramètres formels d'entrée de l'opération <ident\_op> et les paramètres effectifs ne sont pas de même type. Les types des paramètres formels de l'opération et les types des valeurs passées en argument lors d'un appel doivent être identiques.

```

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

```

```

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

```

<b><i>Wrong type for actual output parameters of called operation &lt;ident_op&gt;</i></b>	
<p>Les paramètres formels de sortie de l'opération &lt;ident_op&gt; et les paramètres effectifs ne sont pas de même type. Les types des paramètres formels de l'opération et les types des variables recevant la valeur de retour lors de l'appel doivent être identiques.</p>	
<pre> MACHINE MACH INCLUDES   MAC01 VARIABLES   ww INVARIANT   ww : BOOL INITIALISATION   ww := TRUE OPERATIONS   op = ww &lt;-- oper01   /* ww est un booléen */ END </pre>	<pre> MACHINE MAC01 OPERATIONS   vv &lt;-- oper01 =     vv := 2   /* vv est un entier */ END </pre>

<b><i>Wrong type for actual parameter &lt;ident_param&gt; of machine &lt;ident_mach&gt;</i></b>	
<p>Le paramètre effectif utilisé lors de l'instanciation d'une machine incluse n'est pas du bon type. En effet, lors d'une inclusion avec instanciation, les types des paramètres formels de la machine incluse et les types des paramètres effectifs doivent être identiques. Il peut également s'agir d'une erreur syntaxique (&lt;ident&gt; est-il un identificateur B correct ?) ou d'une erreur de visibilité (l'objet &lt;ident&gt; est-il bien visible ?).</p>	
<pre> MACHINE   M2(p1, p2, p3) CONSTRAINTS   p1 : NAT &amp; p2 : BOOL &amp; p3 : INT END </pre>	<pre> MACHINE M1 INCLUDES   M2(ParamInconnu, 67, _1) /*ParamInconnu est inconnu,   67 n'est pas du bon type,   _1 n'est pas un ident B*/ END </pre>

***Wrong type for expression <exp> in a CASE substitution***

L'expression devant déterminer le comportement de la substitution CASE a un type qui n'est pas autorisé. Cette expression doit être soit de type entier, soit de type booléen, soit élément d'ensemble abstrait ou soit élément d'ensemble énuméré.

```
MACHINE
  M1
VARIABLES
  SS
INVARIANT
  SS <: NAT
INITIALISATION
  SS :: POW(NAT)
OPERATIONS
op1 =
  CASE "chaine" OF
  EITHER 1 THEN skip
  ELSE skip
  END
  END;
op2 =
  CASE ExpInconnue OF
  EITHER 1 THEN skip
  ELSE skip
  END
  END;
op3 =
  CASE SS OF /*SS est une partie de NAT*/
  EITHER 1 THEN skip
  ELSE skip
  END
  END
END
```

## Chapitre 5

# Messages d'erreur interne

Les messages présentés dans ce chapitre n'apparaissent que dans le cas d'une utilisation interdite de l'Atelier B - par exemple la modification manuelle des fichiers de la Base de Donnée Projet. Il est alors nécessaire de relancer le Contrôleur de types sur le composant signalé dans le message.

<b><i>Bad magic number for &lt;ident_mach&gt;.nf</i></b>
--

Le fichier .nf associé au composant <ident_mach> n'a pas été généré avec la même version du Contrôleur de types. Il est donc inutilisable par celui-ci. Il est nécessaire de relancer le Contrôleur de types sur <ident_mach>.
--

<b><i>Cannot load information file of component &lt;ident_mach&gt;</i></b>
--

Le composant analysé fait référence au composant <ident_mach> dont le fichier .nf n'existe pas ou est vide.
---

<b><i>Wrong Normal Form format for the refined structure.</i></b>
---

Le fichier .nf relatif au composant raffiné a été modifié par une action étrangère à l'Atelier B.
---