Traduction de composants Scade/Lustre vers des Machines B

Florian THIBORD

M2-STL APR / UPMC

Année 2012-2013

Section 1

Introduction

Contexte

Stage réalisé au sein du projet CERCLES^2 : Certifier des **composants** réutilisables à l'aide de **méthodes formelles**.

Contexte

Stage réalisé au sein du projet CERCLES^2 : Certifier des **composants** réutilisables à l'aide de **méthodes formelles**.

- Composant : Programme + *Contrat*

Contexte

Stage réalisé au sein du projet CERCLES^2 : Certifier des **composants** réutilisables à l'aide de **méthodes formelles**.

- Composant : Programme + *Contrat*
- Méthode Formelle : raisonnement rigoureux sur un composant à l'aide d'une logique mathématique.

Description du travail

Composants développés avec **Scade**, développé par Esterel Technologies :

- Programme écrit avec des schémas-blocs
- Engendre du code pseudo-Lustre.

Description du travail

Composants développés avec **Scade**, développé par Esterel Technologies :

- Programme écrit avec des schémas-blocs
- Engendre du code pseudo-Lustre.

Méthode B, développée par J.R. Abrial, utilisée pour la validation des composants :

- Méthode basée sur le *raffinement* de *machines*
- Cadre du projet : **Machine abstraite** raffinée en **Machine implantation**

Description du travail

Composants développés avec **Scade**, développé par Esterel Technologies :

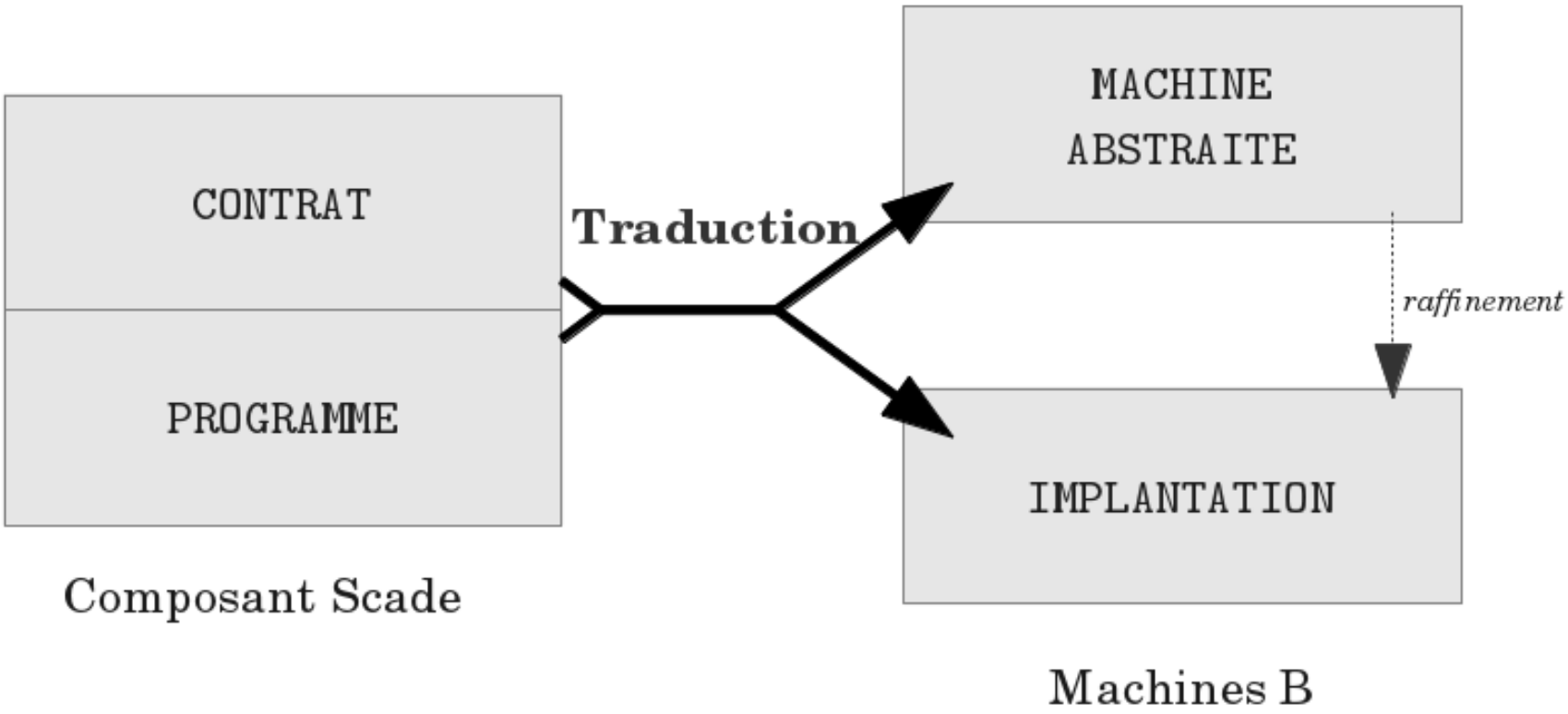
- Programme écrit avec des schémas-blocs
- Engendre du code pseudo-Lustre.

Méthode B, développée par J.R. Abrial, utilisée pour la validation des composants :

- Méthode basée sur le *raffinement* de *machines*
- Cadre du projet : **Machine abstraite** raffinée en **Machine implantation**

Ma tâche : développer un outil permettant de traduire un composant Scade en un couple de machines B.

Schéma général

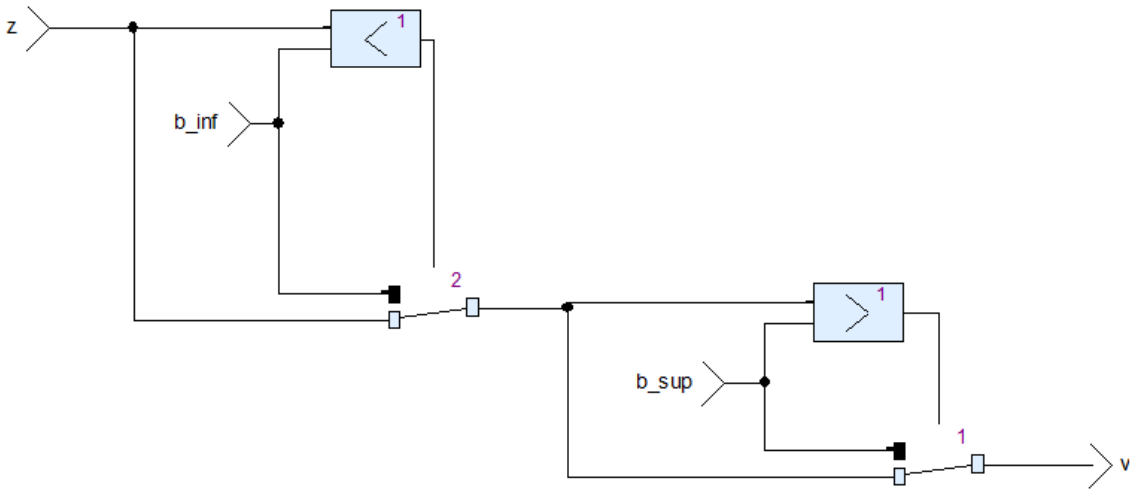


Section 2

Scade

Architecture d'un noeud Scade

Génération de code pseudo-Lustre.




```
node bound(b_sup : int; z : int; b_inf : int)
    returns (v : int)
var
    _L7 : int;
    _L6 : bool;
    _L5 : int;
    _L4 : int;
    _L3 : bool;
    _L2 : int;
    _L1 : int;
let
    v= _L5;
    _L1= z;
    _L2= b_inf;
    _L3= _L7 > _L4;
    _L4= b_sup;
    _L5= if _L3 then (_L4) else (_L7);
    _L6= _L1 < _L2;
    _L7= if _L6 then (_L2) else (_L1);
tel
```

Les équations sont atomisées.

Un fragment de Scade

Le code pseudo lustre est consitué d'un ensemble d'équations atomisées.

3 familles d'équations :

- $v = op_{base}(x_1, \ldots, x_n);$
- `v = if c then` x_1 `else` x_2`;`
- $v_1, \ldots, v_p = op_{appel}(x_1, \ldots, x_n);$

+

1 opérateur particulier : `fby`. Il prend 3 paramètres : une variable, un délai et une valeur d'initialisation.

instant	0	1	2	...
v	10	20	30	...
z	**0**	10	20	...

Figure: `z = fby(v, 1,` **`0`**`)`

Contrat avec Scade

Assertions écrites manuellement dans Scade. Elles sont de 2 type :

- `assume A : expr` où expr correspond à une condition sur une entrée
- `guarantee G : expr` où expr correspond à une condition sur une sortie

Contrat avec Scade

Assertions écrites manuellement dans Scade. Elles sont de 2 type :

- `assume A : expr` où expr correspond à une condition sur une entrée
- `guarantee G : expr` où expr correspond à une condition sur une sortie

Exemple sur le noeud `Bound` :

```
assume A_1 : b_inf <= 2000 and b_inf >= -2000;
assume A_2 : b_sup <= 2000 and b_sup >= -2000;
assume A_3 : z <= 2000 and z >= -2000;
guarantee G_1 : v <= 2000 and v >= -2000;
```

Section 3

Machines B

Les machines sont organisées en *clauses*.

MACHINE ABSTRAITE

```
MACHINE Ma_machine

OPERATIONS
...
END
```

IMPLANTATION

```
IMPLEMENTATION Ma_machine_i
REFINES Ma_machine
IMPORTS ...
CONCRETE_VARIABLES ...
INVARIANT
...
INITIALISATION
...
OPERATIONS
...
END
```

Les machines sont organisées en *clauses*.

MACHINE ABSTRAITE

MACHINE Ma_machine

OPERATIONS
...*Substitutions*
END

IMPLANTATION

IMPLEMENTATION Ma_machine_i
REFINES Ma_machine
IMPORTS ...*Déclarations machines*
CONCRETE_VARIABLES ...*Déclaration variables*
INVARIANT
...*Prédicat*
INITIALISATION
...*Substitutions*
OPERATIONS
...*Substitutions*
END

Le langage B est un langage ensembliste.
Les substitutions généralisées permettent de transformer les prédicats.

Exemple

```
MACHINE Integr

OPERATION
yy ← integr(xx) =
PRE
   xx ∈ INT & -256 <= xx & xx <= 255
THEN
   yy ∈: {ii | ii ∈ INT &
     -1024 <= ii & ii <= 1023}
END
```

```
IMPLEMENTATION Integr_i
REFINES Integr
IMPORTS Bound
CONCRETE_VARIABLES reg1
INVARIANT
  reg1 ∈ INT & -1024 <= reg1 & reg1 <= 1023
INITIALISATION
  reg1 := 0

OPERATIONS

yy ← integr(xx) =
VAR zz IN
   zz := xx + reg1;
   yy ← bound(-1024, zz, 1023);
   reg1 := yy
END
```

Exemple

```
MACHINE Integr

OPERATION
yy ← integr(xx) =
PRE
   xx ∈ INT & -256 <= xx & xx <= 255
THEN
   yy ∈: {ii | ii ∈ INT &
     -1024 <= ii & ii <= 1023}
END
```

```
IMPLEMENTATION Integr_i
REFINES Integr
IMPORTS Bound
CONCRETE_VARIABLES reg1
INVARIANT
  reg1 ∈ INT & -1024 <= reg1 & reg1 <= 1023
INITIALISATION
  reg1 := 0

OPERATIONS

yy ← integr(xx) =
VAR zz IN
   zz := xx + reg1;
   yy ← bound(-1024, zz, 1023);
   reg1 := yy
END
```

Chaque étape de raffinement engendre des obligations de preuves.

Section 4

Schémas de traduction

Traduction des conditions

Utilisation d'une substitution *Precondition* PRE **P** THEN **S** END :

- **P** prédicat reprenant les pré-conditions des `assumes`. Pour chaque entrée, un prédicat est constitué de l'information de type et de la condition associée à l'entrée.
- **S** substitution reprenant les post-conditions des `guarantees`. Pour chaque sortie, une substitution *Devient élément de* + la définition d'un ensemble en compréhension.

Traduction des conditions

Utilisation d'une substitution *Precondition* PRE **P** THEN **S** END :
Prenons un exemple.

```
assume A_1 : b_inf <= 2000 and b_inf >= -2000;
assume A_2 : b_sup <= 2000 and b_sup >= -2000;
assume A_3 : z <= 2000 and z >= -2000;
guarantee G_v : v <= 2000 and v >= -2000;
```

Est traduit par

```
PRE
   zz ∈ INT & zz <= 2000 & zz >= -2000 &
   b_sup ∈ INT & b_sup <= 2000 & b_sup >= -2000 &
   b_inf ∈ INT & b_inf <= 2000 & b_inf >= -2000
THEN
   vv ∈: {ii | ii ∈ INT & ii <= 2000 & ii >= -2000 }
END
```

Le cas des tableaux

Les tableaux sont traduits par des fonctions.

Soit une condition cond sur un tableau Tab de dimension m contenant des éléments de ENS.

$\texttt{Tab} \in \texttt{0 .. (m-1)} \rightarrow \texttt{ENS} \;\&\; \forall \texttt{jj}.\ (\texttt{jj} \in \texttt{0 .. m-1} \Rightarrow \textit{condition cond sur Tab(jj)})$

Le cas des tableaux

Les tableaux sont traduits par des fonctions.

Soit une condition cond sur un tableau Tab de dimension m contenant des éléments de ENS.

$Tab \in 0 \mathinner{..} (m-1) \rightarrow ENS \ \& \ \forall jj.\ (jj \in 0 \mathinner{..} m-1 \Rightarrow \textit{condition cond sur Tab(jj)})$

Exemple :

$assume\ A : Tab[0] > 0\ and\ Tab[0] < 10\ an\ Tab[1] > 0\ and\ Tab[1] < 10$

Le prédicat généré sera alors :

$Tab \in 0 \mathinner{..} 1 \rightarrow INT \ \& \ \forall jj.\ (jj \in 0 \mathinner{..} 1 \Rightarrow 0 < Tab(jj) \ \& \ Tab(jj) < 10)$

Schéma Général

```
node foo (in_1: in_1_type, ..., in_p: in_p_type)
   returns
   (out_1: out_1_type, ..., out_q: out_q_type);
var
   ...
let
   assume A_1 : pred_in_1;
   ...
   assume A_p : pred_in_p;

   liste d'equations

   guarantee G_1 : pred_out_1;
   ...
   guarantee G_q : pred_out_q;
tel;
```

```
MACHINE Foo

OPERATION
out_1, ..., out_q ← foo(in_1, ..., in_p) =
PRE
   in_1 ∈ in_1_type & pred_in_1 &
   ... &
   in_p ∈ in_p_type & pred_in_p &
THEN
   out_1 ∈: {ii | ii ∈ out_1_type & pred_out_1}||

   ... ||
   out_q ∈: {ii | ii ∈ out_q_type & pred_out_q}
  END
END
```

Subsection 2

Implantation

Traduction des équations

Opérateurs de base :

$$a = op_{base}(b_1, ..., b_n) \xrightarrow{traduction\ \ equations} a := op_{base}(b_1, ..., b_n)$$

Appel de noeud :

$$(a_1, ...a_n) = op_{appel}(b_1, ..., b_m) \xrightarrow{traduction\ \ equations} (a_1, ...a_n) \leftarrow op_{appel}(b_1, ..., b_n)$$

Alternative :

$$a = if\ \ cond\ \ then\ \ b1\ \ else\ \ b2 \xrightarrow{\textbf{traduction\ \ equation}}$$

$$IF\ \ cond = TRUE\ \ THEN\ \ a := b1\ \ ELSE\ \ a := b2\ \ END$$

Traduction des registres

Soit l'équation de registre suivante :
`reg = fby(a, 1, ini)`

Traduction des registres

Soit l'équation de registre suivante :
`reg = fby(a, 1, ini)`

```
IMPLEMENTATION ...
...
CONCRETE_VARIABLES ..., reg
INVARIANT
   ...& reg : t & P_reg
INITIALISATION
   ... ; reg := ini

OPERATION
... =
VAR ... IN
   ...;
   reg := a
END
```

Schéma Général

```
node foo (in_1: in_1_type, ..., in_p: in_p_type)
   returns
   (out_1: out_1_type, ..., out_q: out_q_type);
var
   v1 : v1_type;
   ...
   vn : vn_type;
   r1 : r1_type;
   ...
   rn : rn_type;
let
   assume in_1 : pred_in_1;
   ...
   assume in_p : pred_in_p;

   liste d'equations

   guarantee out_1 : pred_out_1;
   ...
   guarantee out_q : pred_out_q;
tel;
```

```
IMPLEMENTATION Foo_i
REFINES Foo
IMPORTS M_imp

CONCRETE_VARIABLES r1, ..., rn
INVARIANT
   r1 : r1_type &
   ... &
   rn : rn_type
INITIALISATION
   r1 := ...;
   ...;
   rn := ...;

OPERATION
out_1, ..., out_q ← foo(in_1, ..., in_p) =
VAR v1, ..., vn IN

   Sequence de substitutions

END
```

Schéma Général

```
node foo (in_1: in_1_type, ..., in_p: in_p_type)
   returns
   (out_1: out_1_type, ..., out_q: out_q_type);
var
   v1 : v1_type;
   ...
   vn : vn_type;
   r1 : r1_type;
   ...
   rn : rn_type;
let
   assume in_1 : pred_in_1;
   ...
   assume in_p : pred_in_p;

   liste d'equations

   guarantee out_1 : pred_out_1;
   ...
   guarantee out_q : pred_out_q;
tel;
```

```
IMPLEMENTATION Foo_i
REFINES Foo
IMPORTS M_imp

CONCRETE_VARIABLES r1, ..., rn
INVARIANT
   r1 : r1_type &
   ... &
   rn : rn_type
INITIALISATION
   r1 := ...;
   ...;
   rn := ...;

OPERATION
out_1, ..., out_q ← foo(in_1, ..., in_p) =
VAR v1, ..., vn IN

   Sequence de substitutions

END
```

Note : les machines IMPORTS doivent être ajoutés manuellement

Subsection 3

Le traducteur

Implémentation du traducteur

Traducteur écrit en OCaml (2700 lignes de code).

fichier.scade $\xrightarrow{\textit{OCamlLex/OcamlYacc}}$ AST_1

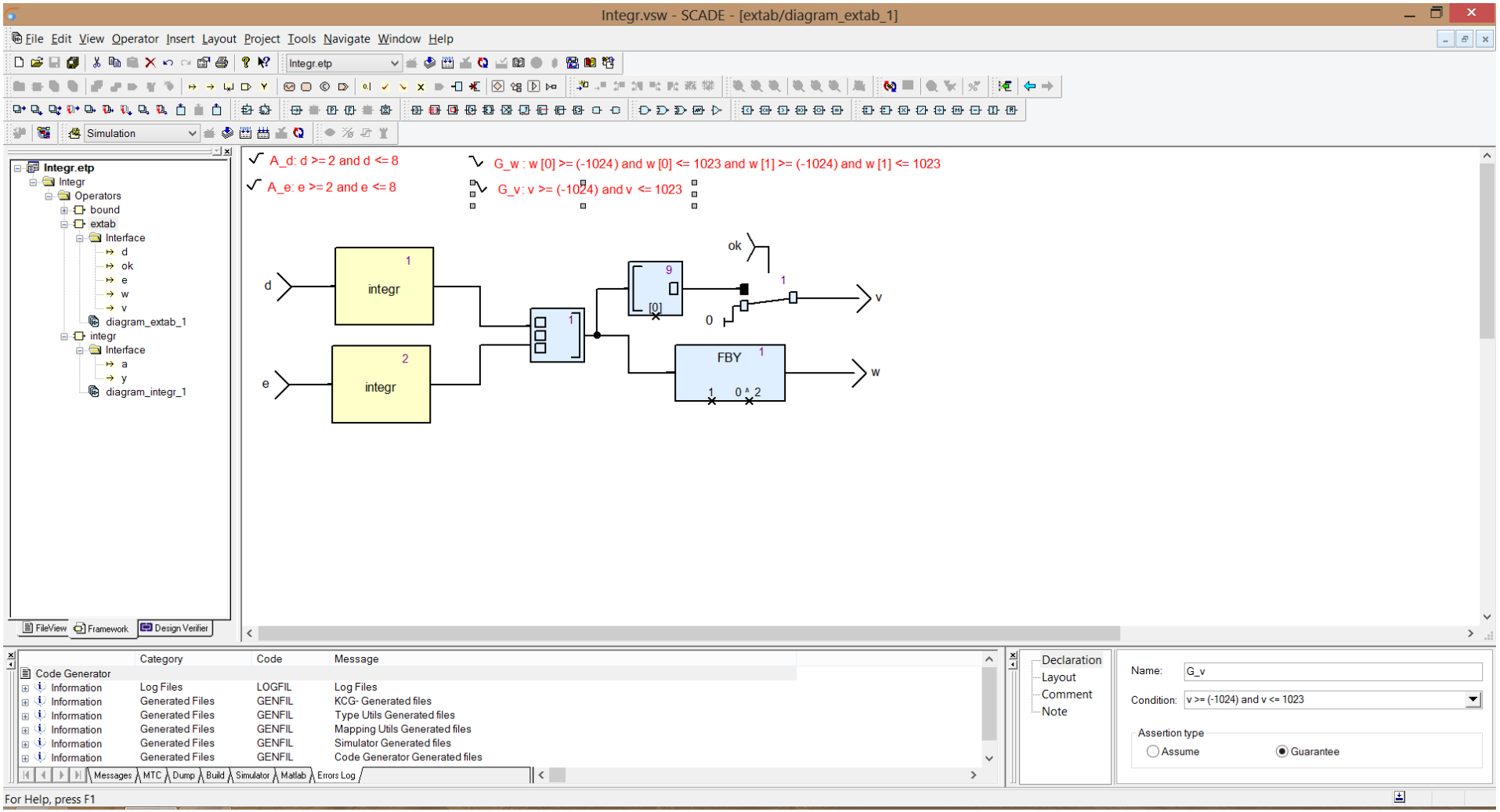
$\Downarrow$ *Manipulations AST*

AST_n $\xrightarrow{\textit{Pretty Printer}}$ fichier.mch, fichier_i.imp

Manipulations AST : séquencement équations, rennomage, ...

Section 5

Exemple


Integr.vsw - SCADE - [extab/diagram_extab_1]
File Edit View Operator Insert Layout Project Tools Navigate Window Help
Integr.etp
Simulation
Integr.etp
Integr
Operators
bound
extab
Interface
d
ok
e
w
v
diagram_extab_1
integr
Interface
a
y
diagram_integr_1
A_d: d >= 2 and d <= 8
A_e: e >= 2 and e <= 8
G_w : w [0] >= (-1024) and w [0] <= 1023 and w [1] >= (-1024) and w [1] <= 1023
G_v: v >= (-1024) and v <= 1023
integr
integr
FBY
ok
v
w
FileView
Framework
Design Verifier
Category
Code
Message
Code Generator
Information Log Files LOGFIL Log Files
Information Generated Files GENFIL KCG- Generated files
Information Generated Files GENFIL Type Utils Generated files
Information Generated Files GENFIL Mapping Utils Generated files
Information Generated Files GENFIL Simulator Generated files
Information Generated Files GENFIL Code Generator Generated files
Messages
MTC
Dump
Build
Simulator
Matlab
Errors Log
Declaration
Layout
Comment
Note
Name: G_v
Condition: v >= (-1024) and v <= 1023
Assertion type
Assume
Guarantee
For Help, press F1

emacs@florian-Inspiron-1520

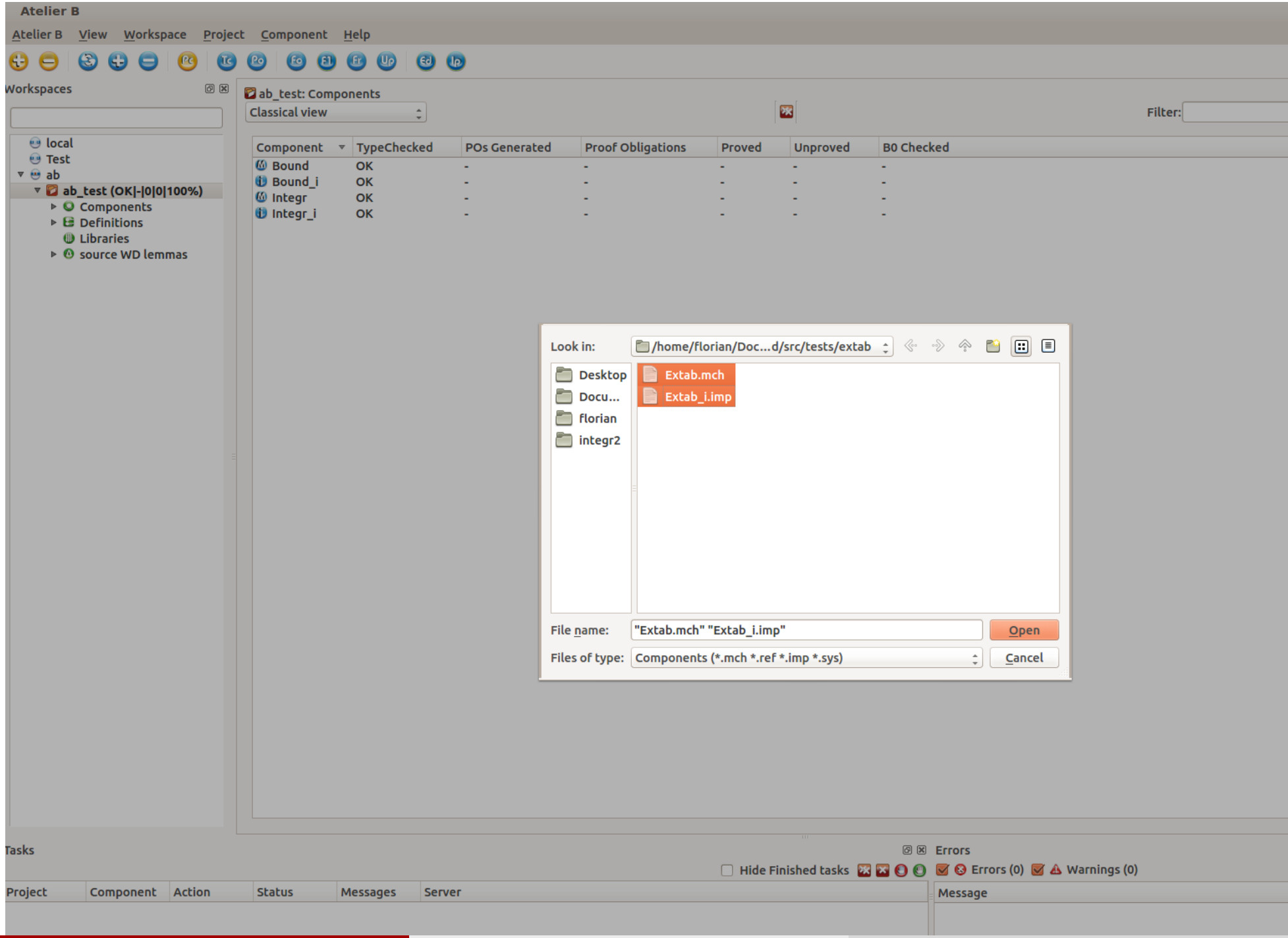
File Edit Options Buffers Tools Help

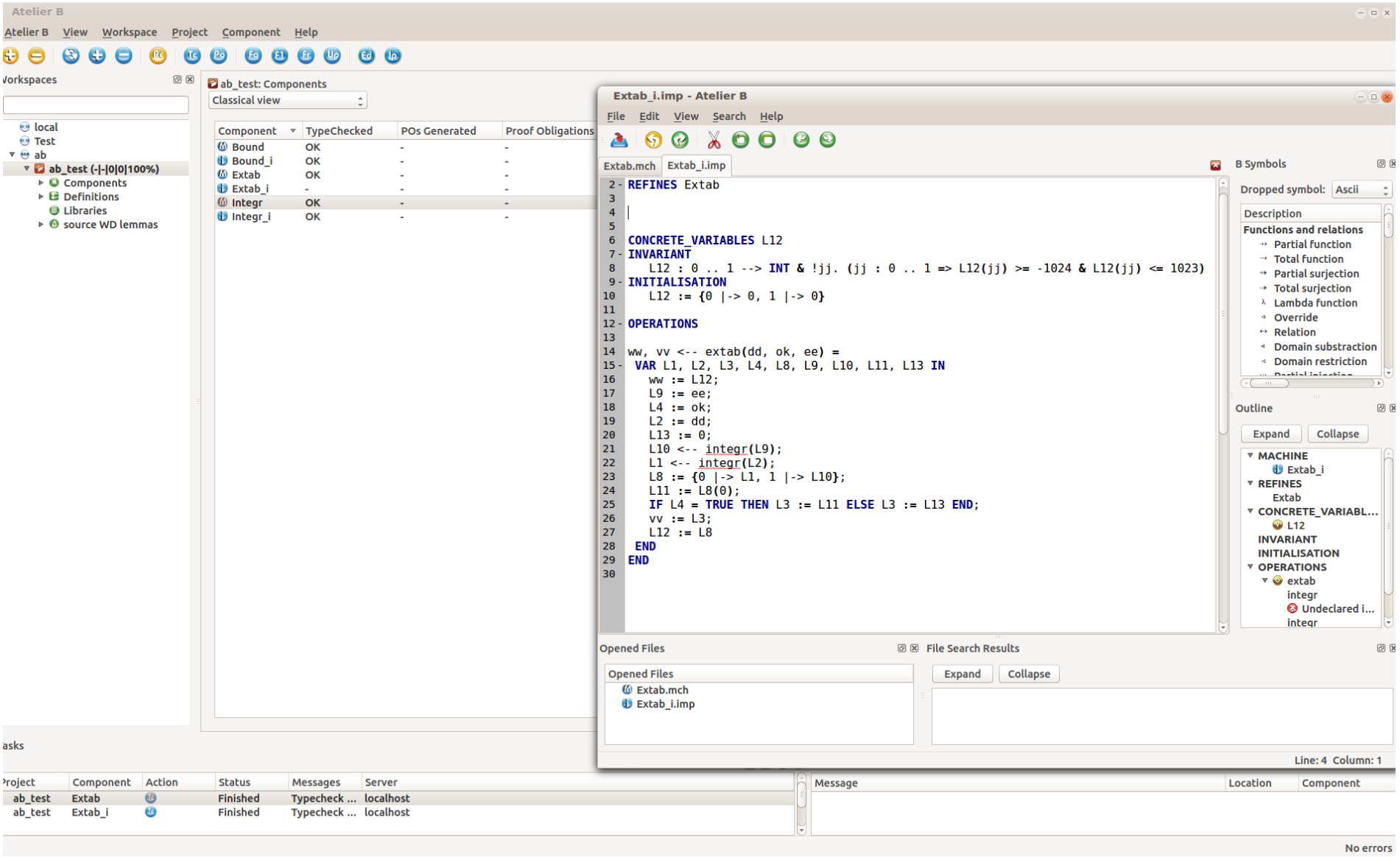
```
  guarantee G_v : v >= b_inf and v <= b_sup;
tel


/* xscade source: C:/Users/florian/Documents/Workspace/Integr/Operator2.xscade */
node integr(a : int) returns (y : int)
var
  _L7 : int;
  _L6 : int;
  _L5 : int;
  _L4 : int;
  _L3 : int;
  _L8 : int;
let
  _L3= 1023;
  _L4= a;
  y= _L8;
  _L5= - 1024;
  _L6= fby(_L8; 1; 0);
  _L7= _L4 + _L6;
  assume A_a : a >= 0 and a <= 10;
  guarantee G_y : - 1024 <= y and y <= 1023;
  _L8= #1 bound(_L3, _L7, _L5);
tel


/* xscade source: C:/Users/florian/Documents/Workspace/Integr/Operator3.xscade */
node extab(d : int; ok : bool; e : int) returns (w : int^2; v : int)
var
  _L1 : int;
  _L2 : int;
  _L3 : int;
  _L4 : bool;
  _L8 : int^2;
  _L9 : int;
  _L10 : int;
  _L11 : int;
  _L12 : int^2;
  _L13 : int;
let
  _L1= #1 integr(_L2);
  _L2= d;
  _L3= if _L4 then (_L11) else (_L13);
  _L4= ok;
  _L8= [_L1, _L10];
  _L9= e;
  w= _L12;
  assume A_d : d >= 2 and d <= 8;
  assume A_e : e >= 2 and e <= 8;
  guarantee G_w : w[0] >= - 1024 and w[0] <= 1023 and w[1] >= - 1024 and
  w[1] <= 1023;
  _L10= #2 integr(_L9);
  _L11= _L8[0];
  v= _L3;
  _L12= fby(_L8; 1; (0)^2);
  guarantee G_v : v >= - 1024 and v <= 1023;
  _L13= 0;
tel
```

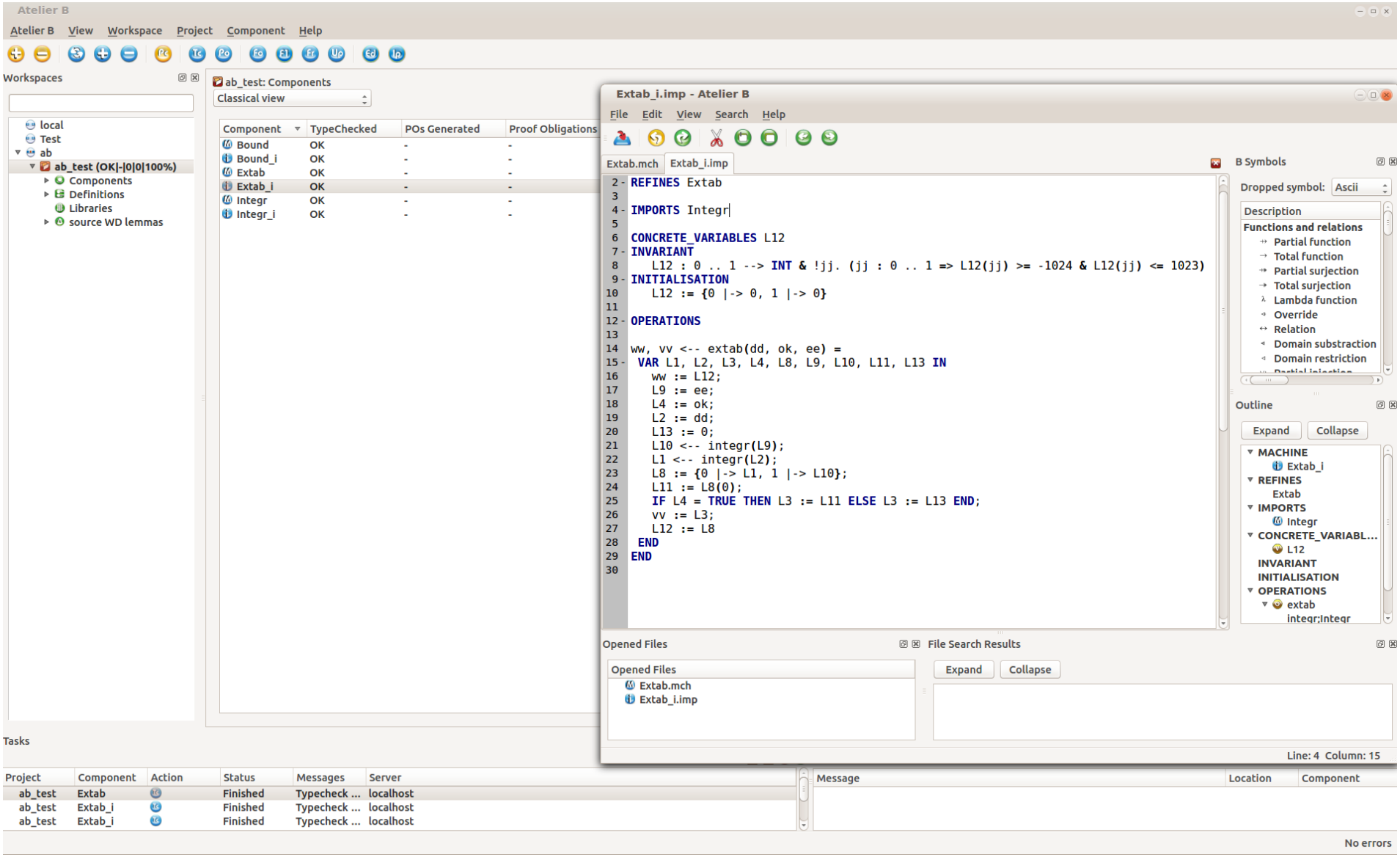
florian@florian-Inspiron-1520: ~/Documents/Cercles/Trad/src/tests/extab

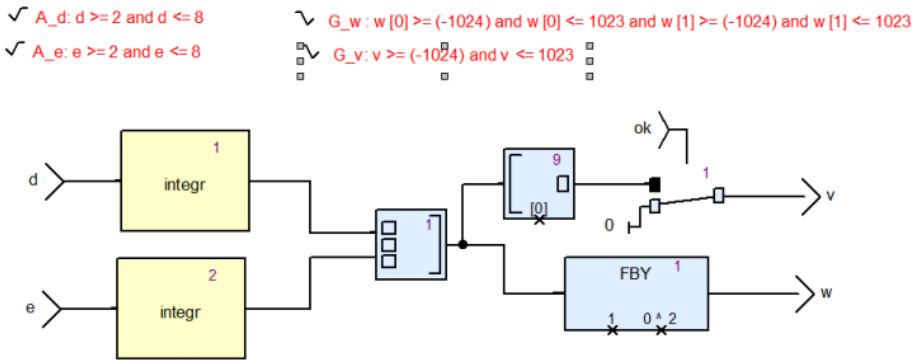
File Edit View Search Terminal Help

```
florian@florian-Inspiron-1520:~/Documents/Cercles/Trad/src/tests/extab$ ls
kcg_xml_filter_out.scade
florian@florian-Inspiron-1520:~/Documents/Cercles/Trad/src/tests/extab$ ../../cercles kcg_xml_filter_out.scade extab
florian@florian-Inspiron-1520:~/Documents/Cercles/Trad/src/tests/extab$ ls
Extab_i.imp  Extab.mch  kcg_xml_filter_out.scade
florian@florian-Inspiron-1520:~/Documents/Cercles/Trad/src/tests/extab$
```


Atelier B
Atelier B View Workspace Project Component Help
Workspaces
local
Test
ab
ab_test (OK|-|0|0|100%)
Components
Definitions
Libraries
source WD lemmas
ab_test: Components
Classical view
Filter:
Component TypeChecked POs Generated Proof Obligations Proved Unproved B0 Checked
Bound OK - - - - -
Bound_i OK - - - - -
Integr OK - - - - -
Integr_i OK - - - - -
Look in: /home/florian/Doc...d/src/tests/extab
Desktop
Docu...
florian
integr2
Extab.mch
Extab_i.imp
File name: "Extab.mch" "Extab_i.imp"
Open
Files of type: Components (*.mch *.ref *.imp *.sys)
Cancel
Tasks
Errors
Hide Finished tasks
Errors (0) Warnings (0)
Project Component Action Status Messages Server
Message


Atelier B
Atelier B View Workspace Project Component Help
Workspaces
local
Test
ab
ab_test (-|-|0|0|100%)
Components
Definitions
Libraries
source WD lemmas
ab_test: Components
Classical view
Component TypeChecked POs Generated Proof Obligations
Bound OK - -
Bound_i OK - -
Extab OK - -
Extab_i - - -
Integr OK - -
Integr_i OK - -
Extab_i.imp - Atelier B
File Edit View Search Help
Extab.mch
Extab_i.imp
2 - REFINES Extab
3
4
5
6 CONCRETE_VARIABLES L12
7 - INVARIANT
8 L12 : 0 .. 1 --> INT & !jj. (jj : 0 .. 1 => L12(jj) >= -1024 & L12(jj) <= 1023)
9 - INITIALISATION
10 L12 := {0 |-> 0, 1 |-> 0}
11
12 - OPERATIONS
13
14 ww, vv <-- extab(dd, ok, ee) =
15 - VAR L1, L2, L3, L4, L8, L9, L10, L11, L13 IN
16 ww := L12;
17 L9 := ee;
18 L4 := ok;
19 L2 := dd;
20 L13 := 0;
21 L10 <-- integr(L9);
22 L1 <-- integr(L2);
23 L8 := {0 |-> L1, 1 |-> L10};
24 L11 := L8(0);
25 IF L4 = TRUE THEN L3 := L11 ELSE L3 := L13 END;
26 vv := L3;
27 L12 := L8
28 END
29 END
30
B Symbols
Dropped symbol: Ascii
Description
Functions and relations
Partial function
Total function
Partial surjection
Total surjection
Lambda function
Override
Relation
Domain substraction
Domain restriction
Outline
Expand Collapse
MACHINE
Extab_i
REFINES
Extab
CONCRETE_VARIABL...
L12
INVARIANT
INITIALISATION
OPERATIONS
extab
integr
Undeclared i...
integr
Opened Files
File Search Results
Opened Files
Extab.mch
Extab_i.imp
Expand Collapse
Line: 4 Column: 1
asks
Project Component Action Status Messages Server
ab_test Extab Finished Typecheck ... localhost
ab_test Extab_i Finished Typecheck ... localhost
Message Location Component
No errors


Atelier B
Atelier B View Workspace Project Component Help
Workspaces
local
Test
ab
ab_test (OK|-|0|0|100%)
Components
Definitions
Libraries
source WD lemmas
ab_test: Components
Classical view
Component TypeChecked POs Generated Proof Obligations
Bound OK - -
Bound_i OK - -
Extab OK - -
Extab_i OK - -
Integr OK - -
Integr_i OK - -
Extab_i.imp - Atelier B
File Edit View Search Help
Extab.mch Extab_i.imp
2 REFINES Extab
3
4 IMPORTS Integr
5
6 CONCRETE_VARIABLES L12
7 INVARIANT
8 L12 : 0 .. 1 --> INT & !jj. (jj : 0 .. 1 => L12(jj) >= -1024 & L12(jj) <= 1023)
9 INITIALISATION
10 L12 := {0 |-> 0, 1 |-> 0}
11
12 OPERATIONS
13
14 ww, vv <-- extab(dd, ok, ee) =
15 VAR L1, L2, L3, L4, L8, L9, L10, L11, L13 IN
16 ww := L12;
17 L9 := ee;
18 L4 := ok;
19 L2 := dd;
20 L13 := 0;
21 L10 <-- integr(L9);
22 L1 <-- integr(L2);
23 L8 := {0 |-> L1, 1 |-> L10};
24 L11 := L8(0);
25 IF L4 = TRUE THEN L3 := L11 ELSE L3 := L13 END;
26 vv := L3;
27 L12 := L8
28 END
29 END
30
B Symbols
Dropped symbol: Ascii
Description
Functions and relations
Partial function
Total function
Partial surjection
Total surjection
Lambda function
Override
Relation
Domain substraction
Domain restriction
Outline
Expand Collapse
MACHINE
Extab_i
REFINES
Extab
IMPORTS
Integr
CONCRETE_VARIABL...
L12
INVARIANT
INITIALISATION
OPERATIONS
extab
integr:Integr
Opened Files
File Search Results
Opened Files
Extab.mch
Extab_i.imp
Expand Collapse
Line: 4 Column: 15
Tasks
Project Component Action Status Messages Server
ab_test Extab Finished Typecheck ... localhost
ab_test Extab_i Finished Typecheck ... localhost
ab_test Extab_i Finished Typecheck ... localhost
Message Location Component
No errors




```
node extab(d : int; ok : bool; e : int) returns (w : int^2; v : int)
var
  _L1 : int;
  _L2 : int;
  _L3 : int;
  _L4 : bool;
  _L8 : int^2;
  _L9 : int;
  _L10 : int;
  _L11 : int;
  _L12 : int^2;
  _L13 : int;
let
  _L1= #1 integr(_L2);
  _L2= d;
  _L3= if _L4 then (_L11) else (_L13);
  _L4= ok;
  _L8= [_L1, _L10];
  _L9= e;
  w= _L12;
  assume A_d : d >= 2 and d <= 8;
  assume A_e : e >= 2 and e <= 8;
  guarantee G_w : w[0] >= - 1024 and w[0] <= 1023 and w[1] >= - 1024 and
  w[1] <= 1023;
  _L10= #2 integr(_L9);
  _L11= _L8[0];
  v= _L3;
  _L12= fby(_L8; 1; (0)^2);
  guarantee G_v : v >= - 1024 and v <= 1023;
  _L13= 0;
tel
```

```
MACHINE Extab

OPERATIONS

ww, vv <-- extab(dd, ok, ee) =
 PRE
   ok : BOOL &
   ee : INT & ee >= 2 & ee <= 8 &
   dd : INT & dd >= 2 & dd <= 8
 THEN
   vv :: { ii | ii : INT & ii >= -1024 & ii <= 1023 }||
   ww :: { ii | ii : 0 .. 1 --> INT &
       !jj. (jj : 0 .. 1 => ii(jj) >= -1024 & ii(jj) <= 1023)}
 END
END
```

```
IMPLEMENTATION Extab_i
REFINES Extab

IMPORTS Integr

CONCRETE_VARIABLES L12
INVARIANT
   L12 : 0 .. 1 --> INT & !jj. (jj : 0 .. 1 => L12(jj) >= -1024 & L12(jj) <= 1023)
INITIALISATION
   L12 := {0 |-> 0, 1 |-> 0}

OPERATIONS

ww, vv <-- extab(dd, ok, ee) =
 VAR L1, L2, L3, L4, L8, L9, L10, L11, L13 IN
   ww := L12;
   L9 := ee;
   L4 := ok;
   L2 := dd;
   L13 := 0;
   L10 <-- integr(L9);
   L1 <-- integr(L2);
   L8 := {0 |-> L1, 1 |-> L10};
   L11 := L8(0);
   IF L4 = TRUE THEN L3 := L11 ELSE L3 := L13 END;
   vv := L3;
   L12 := L8
 END
END
```

Section 6

Conclusion

Traducteur fonctionnel, respecte les schémas de traduction.

Traducteur fonctionnel, respecte les schémas de traduction.

Fragment de Scade. Possibilités d'extensions **?**

Traducteur fonctionnel, respecte les schémas de traduction.

Fragment de Scade. Possibilités d'extensions **?**

Preuve de correction de la traduction à faire.

Traducteur fonctionnel, respecte les schémas de traduction.

Fragment de Scade. Possibilités d'extensions **?**

Preuve de correction de la traduction à faire.

Remplacement des tests de composants par une étape de validation par méthode formelle **? ?**

Merci de votre attention.