Promela -> SPIN

Cours GLSD

Partie 2

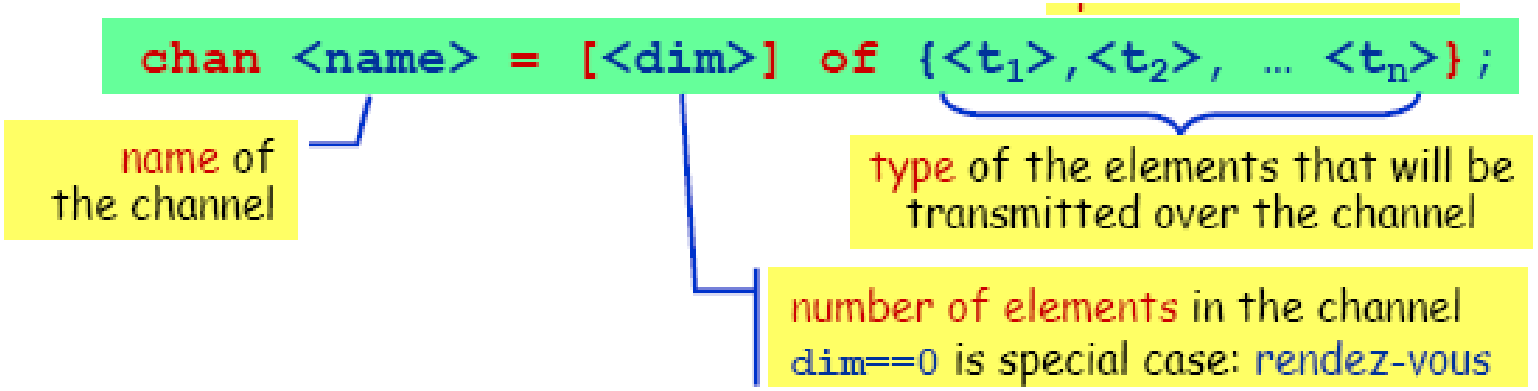
2015-2016

Communications

- Communication avec canaux
- Envois et réception de messages
- Rendez vous et synchronisation

Communication

- À travers de canaux:



- exemple

```
chan c        = [1] of {bit};
chan toR      = [2] of {mtype, bit};
chan line[2]  = [1] of {mtype, Record};
```

Passage de messages (1)

- Les processus échangent des **msg** via des canaux (**FIFO**):
  exemple:
  chan *qname = [16]* ***of { short }***
  chan *qname = [16]* ***of { byte, int, chan, byte }***

- **Envois** : **qname! expr**
- **Réception**: **qname? msg**

- Exemple :
  qname!expr1,expr2,expr3
  qname?var1,var2,var3
  qname!expr1(expr2,expr3)
  qname?var1(var2,var3)

- **len**(qname) : pour calculer le nombre de msgs dans un canal

Voir : exp_p30

Exemple (1)

```
chan ch = [0] of {bit, byte};
```

Processus 1: `ch ! 1, 3+7`

Processus 2: `ch ? 1, x`

?

Passage de messages (2)

- NB:

- L'envoi est **bloquant** si le canal est plein;
- La réception est **bloquante** si le canal est vide;

Exemple 2 & remarque

- Pour un canal:

chan qname = [16] of {short, byte, bool};

- Envoie des messages :

qname!exp1,exp2,exp3;

- exécuté seulement si le canal qname n'est pas plein; autrement, la tâche est bloquée

- Réception des messages :

qname?var1,var2,var3;

exécuté seulement si le canal qname n'est pas vide

- Notation équivalente:

qname!exp1(exp2,exp3);

qname?var1(var2,var3);

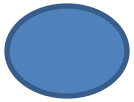
Communiquer les noms des canaux

- Il est **possible d'envoyer le nom de canal** comme paramètre à un processus qui peut l'utiliser par la suite pour faire des communication avec.

- Voir l'exemple suivant

Exemple (3): processus & msg

```
1.  proctype A (chan q1)
2.      {       chan q2;
3.              q1?q2;
4.              q2!123
5.      }
6.
7.  proctype B(chan  qforb)
8.      {int x;
9.      qforb?x;
10.     printf("x = %d\n", x)
11.     }
12.
13. init {
14. chan qname = [1] of
        {chan};
15. chan qforb = [1] of { int };
16.     run A(qname);
17.     run B(qforb);
18.     qname!qforb
19.     }
```



Synchronisation

Synchronisation par rdv

- Se réalise **avec des canaux de taille 0**

- L'émetteur se bloque tant qu'il n'y a pas de récepteur et vice et versa

Exemple (4)

```
#define msgtype 33

chan name = [0] of {byte, byte};

proctype A()
{     name! msgtype, 124;
      name! msgtype, 121
}
```

```
proctype B()
{byte state;
   name? msgtype, state;
   printf("State: %d", state)
}

init
{atomic {run A();run B()}
}
```

Exemple (5): Dijkstra

```
1.  #define p       0
2.  #define v       1
3.
4.  chan sema = [0] of { bit };
5.
6.  proctype dijkstra()
7.      {       byte count = 1;
8.
9.      do
10.     :: (count == 1) ->
11.             sema!p; count = 0
12.     :: (count == 0) ->
13.             sema?v; count = 1
14.     od
15.     }
16.
```

```
17.  proctype user()
18.  {do
19.  :: sema?p;
20.                  /*    critical section */
21.  sema!v;
22.                  /* non-critical section */
23.  od
24.  }
25.
26.  init
27.  {run dijkstra();
28.  run user();
29.  run user();
30.  run user()
31.  }
```

Notions spécifiques

- Procédure & récursion
- Timeout

Procedure & récursion

```
1.  proctype fact(int n; chan p)
2.      {chan child = [1] of { int };
3.      int result;
4.      if
5.      :: (n <= 1) -> p!1
6.      :: (n >= 2) ->
7.        run fact(n-1, child);
8.        child?result;
9.        p!n*result
10.     fi
11.     }
12.
```

```
13. init
14. {chan child = [1] of {int};
15.       int result;
16.       run fact(7, child);
17.       child?result;
18. printf("result: %d\n", result)
19.     }
```

Timeout

- **timeout** se vérifie quand aucune instruction ne peut être exécutée dans le système

```
1.  proctype watchdog()
2.      {
3.          do
4.          :: timeout -> guard!reset
5.          od
6.      }
```

Ce processus va envoyer le message reset sur le canal guard quand aucune action n'est exécutable dans le système

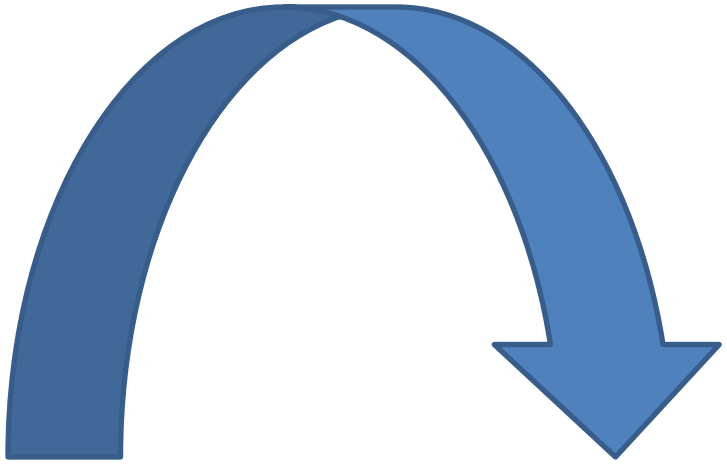
Promela -> SPIN

Cours GLSD

Partie 3

2014

Vérification



Promela **SPIN**

Objectifs de SPIN

1) **Simuler** une spécification Promela: une simulation aléatoire;

2) **Vérifier**: **assersion**, **end** state, **progress**, et **accept**;

3) **Vérifier des propriétés** : construire un programme en C (un vérificateur) qui permet de calculer l'espace d'état du système spécifié et vérifier certaines propriétés LTL.

Les assertions

- Une assertion est une **expression logique**, qu'on veut voir si elle est vérifiée ou non durant l'exécution d'un processus.

- Si l'assertion est non vérifiée l'exécution retourne runtime-error

assert(any_boolean_condition)

The *end* state

End: est utilisé comme un étiquète pour indiquer à SPIN que si le processus s'arrête sur l'instruction étiqueté par end, ceci représente une **fin normal** et non pas un **deadlock**.

Exemple (9) : *end* state

```
1.  proctype dijkstra()
2.      {       byte count = 1;
3.
4.      end:    do
5.              :: (count == 1) ->
6.                      sema!p; count = 0
7.              :: (count == 0) ->
8.                      sema?v; count = 1
9.              od
10.     }
```

The *progress* state

progress indique à SPIN qu'une exécution normal d'une boucle doit passer toujours par l'instruction étiqueté par *progress*

Exemple (10) : *progress* state

```
1.  proctype dijkstra()
2.      {       byte count = 1;
3.
4.      end:    do
5.              :: (count == 1) ->
6.              progress:       sema!p; count = 0
7.              :: (count == 0) ->
8.                      sema?v; count = 1
9.              od
10.     }
```

The *accept* state

accept :

Si une instruction est étiquetée par ***accept*** ceci indique à SPIN qu'une boucle infinie **qui contient** cette instruction est une erreur.

Options d'exécution de simulation: pglrs

- ***spin spec.pml***: exécute aléatoirement la spécification spec.pml
- ***spin –p spec.pml:*** montre les états des processus pas à pas
- ***spin –g spec.pml:*** montre les valeurs des variables globales pas à pas
- ***spin –l spec.pml***: montre les valeurs des variables locales pas à pas
- ***spin –r spec.pml***: montre les réceptions de msgs
- ***spin –s spec.pml***: montre les envois de msgs

Les options d'exécution de l'analyse: amt

- ***spin –a spec.pml***: crée un analyseur **pan.c** (un pgm C) qu'on peut compiler : **gcc –o pan pan.c**. Cet analyseur fait une analyse de l'espace d'états du système.

- -m: les envois ne seront pas bloqués si le canal est complet;

- -t: permet d'analyser le fichier **pan.trail** (qui contient les traces des erreurs) qui contient toutes les erreurs détectées durant l'analyse. On **recrée** le chemin qui mène à l'erreur.

Concentrer sur un processus

$ spin -n100 -r spec | **grep** "proc 2"

montre les informations concernant proc 2 par exemple.

Exemple d'usage de SPIN(1)

```
bool demande[2];
bool tour;
proctype P(bool i)
        {demande[i] = 1;
        do
                :: (tour != i) ->
                        (!demande[1 - i]);
                        tour = i
                :: (tour == i) ->
                        break;
        od;
        skip; /* section critique */
        demande[i] = 0
        }
init {run P(0); run P(1)}
```

Exemple d'usage de SPIN(2)

Création de l'analyseur:

$ spin -a exp.pml

Compilation de l'analyseur

$ cc pan.c

Donc vous devez disposer de cc:

Sinon erreur: c:\mingw\bin\cc.exe pour jspin

Exécution de l'analyseur

$ a.out

Exemple d'usage de SPIN(3)

Résultats: (en anglais ce que doit afficher SPIN)

Full statespace search for:

never-claim - (none specified) // des propriétés LTL

assertion violations +

non-progress cycles - (not selected)

acceptance cycles - (not selected)

invalid endstates + // arrêts sur état invalide

Exemple d'usage de SPIN(4)

Résultats: (en anglais ce que doit afficher SPIN)

State-vector 20 byte, depth reached 19, errors: 0

56 states, **stored**

15 states, **matched**

71 transitions (= stored+matched)

0 atomic steps

Exemple d’usage de SPIN(5)

Résultats: (en anglais ce que doit afficher SPIN)

unreached in proctype P

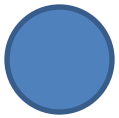
(0 of 12 states)

unreached in proctype :init:

(0 of 3 states)

Exemple d'usage de SPIN(5)

```
proctype P(bool i)
        {demande[i] = 1;
        do
                :: (tour != i) ->
                        (!demande[1 - i]);
                        tour = i
                :: (tour == i) ->
                        break;
        od;
        skip; /* section critique */
                nb++;
                assert(nb == 1);
                nb--;
        demande[i] = 0
        }
```



Exemple d'usage de SPIN(6)

$ spin -a expl.pml

$ cc pan.c

$ a.out

pan: **assertion violated** (nb==1) (at depth 15)

pan: wrote **expl.trail**

Le fichier expl.trail contient une trace qui montre la violation

Exemple d'usage de SPIN(7)

$ spin -t -p expl.pml

1: proc 0 (:init:) line 21 "expl" (state 1) [(run P(0))]
2: proc 0 (:init:) line 21 "expl" (state 2) [(run P(1))]
3: proc 2 (P) line 6 "expl" (state 1) [demande[i] = 1]
4: proc 2 (P) line 8 "expl" (state 2) [((tour!=i))]
5: proc 2 (P) line 9 "expl" (state 3) [(!(demande[(1-i)]))]
6: proc 1 (P) line 6 "expl" (state 1) [demande[i] = 1]
7: proc 1 (P) line 11 "expl" (state 5) [((tour==i))]
8: proc 1 (P) line 14 "expl" (state 10) [(1)]
9: proc 2 (P) line 10 "expl" (state 4) [tour = i]
10: proc 2 (P) line 11 "expl" (state 5) [((tour==i))]
11: proc 2 (P) line 14 "expl" (state 10) [(1)]
12: proc 2 (P) line 15 "expl" (state 11) [nb = (nb+1)]
13: proc 2 (P) line 16 "expl" (state 12) [assert((nb==1))]
14: proc 1 (P) line 15 "expl" (state 11) [nb = (nb+1)]
spin: line 16 "expl", **Error: assertion violated**
15: proc 1 (P) line 16 "expl" (state 12) [assert((nb==1))]

spin: trail ends after 15 steps

Exemple (4)

```
#define a 1
 #define b 2
 chan ch = [1] of { byte };

 proctype A()
 {      ch!a
 }
 proctype B()
 {      ch!b
 }

 proctype C()
 {      if
        :: ch?a
        :: ch?b
        fi
 }
```

Exemple (5): *do & case*

```
1.  #define N    128
2.  #define size  16

3.  chan in    = [size] of { short };
4.  chan large = [size] of { short };
5.  chan small = [size] of { short };
```

Exemple (5: suite): *do & case*

```
1.  proctype split()
2.      {       short cargo;
3.
4.              do
5.              :: in?cargo ->
6.                      if
7.                      :: (cargo >= N) ->
8.                              large!cargo
9.                      :: (cargo <  N) ->
10.                             small!cargo
11.                     fi
12.             od
13.     }
14.
15.     init
16.     {       run split()
17.     }
```

Exemple (5:suite): *do & case*

```
1.  proctype merge()
2.      {       short cargo;
3.
4.              do
5.              ::      if
6.                      :: large?cargo
7.                      :: small?cargo
8.                      fi;
9.                      in!cargo
10.             od
11.     }
```

Exemple (5: suite): *do & case*

```
1. init
2.     {       in!345; in!12; in!6777;
3.             in!32;  in!0;
4.             run split();
5.             run merge()
6.     }
```

Options pour la vérification

```
-a  find acceptance cycles
-A  ignore assert() violations
-b  consider it an error to exceed the depth-limit
-cN stop at Nth error (defaults to -c1)
-D  print state tables in dot-format and stop
-d  print state tables and stop
-e  create trails for all errors
-E  ignore invalid end states
-f  add weak fairness (to -a or -l)
-hN use different hash-seed N:1..32
-i  search for shortest path to error
-I  like -i, but approximate and faster
-J  reverse eval order of nested unlesses
-l  find non-progress cycles -> disabled, requires compilation with -DNP
-mN max depth N steps (default=10k)
-n  no listing of unreached states
-QN set time-limit on execution of N minutes
-q  require empty chans in valid end states
-r  read and execute trail - can add -v,-n,-PN,-g,-C
-rN read and execute N-th error trail
```

Options pour la vérification

```
-PN read and execute trail - restrict trail output to proc N
-g  read and execute trail + msc gui support
-S  silent replay: only user defined printfs show
-T  create trail files in read-only mode
-tsuf replace .trail with .suf on trailfiles
-V  print SPIN version number
-v  verbose -- filenames in unreached state listing
-wN hashtable of 2^N entries (defaults to -w19)
-x  do not overwrite an existing trail file
options -r, -C, -PN, -g, and -S can optionally be followed by
a filename argument, as in '-r filename', naming the trailfile
```

Fin

Questions?