Promela -> SPIN
(… Suite… Vérification de propriétés)

GLSD 2015-2016

L. Kahloul

Plan de l’exposé

1. Introduction
2. Propriétés : **liveness**, **safety**, **fairness**.
3. Formalisation LTL: **never claims**
4. Les assertions de traces

Introduction

- Dans la partie 3 du cours, on a examiné: les **assertions**, les **end**, **progress** et **accept** labels.

- Dans cette partie, on verra la vérification de propriétés LTL spécifiées explicitement hors la spécification Promela.

Assertion (rappel)

```
byte x = 0;
proctype P() {
        do
                :: x >= 10 -> break
                :: x = x+1
        od;
        assert(x==10)
        // something else
}
init {
run P();
}
```

End-state

- Verification du **deadlock-freedom**:

1. Est ce que tous les processus ont atteint leurs propres end-states?
2. Distinguer les propres end-states des end-states illégales

Exemple:

1. Tout accolade fermant est un propre end-state
2. On peut ajouter d'autres end-state avec le label end (ex. end1, end_p, end_loop etc.)

End-state (exemple)

```
#define p 0
#define v 1
chan sema = [0] of {bit};
proctype semaphore() {
        byte count = 1;
        do
            :: (count == 1) -> end1: sema!p; count = 0
            :: (count == 0) -> end2: sema?v; count = 1
        od
}
```

Progress vs accept

- Progress-state: **error** when there is a **cycle** **not going through** a progress state;

- Accept-state: **error** when a **cycle** is **going through** an accept state;

Progress vs accept

```
#define p 0
#define v 1
chan sema = [0] of {bit};
proctype semaphore() {
        byte count = 1;
        do
          :: (count == 1) -> progress: sema!p; count = 0
          :: (count == 0) -> sema?v; count = 1
        od
}
```

Propriétés Majeurs

- 2 classes de proprietés (1977):

(1) **Liveness**: ”something good will eventually happen“. Un contre-exemple: un **deadlock**, où jamais le bien n’arrivera.

(2) **Safety**: “something bad will never happen"

Remarque: dans un système composé de plusieurs processus, le fait d’exécuter **un seul processus toujours ou souvent** ne touche pas à la vivacité (liveness) mais rend le système non **équitable** (**unfair**), d’où il est important d’avoir la propriété de fairness (équitabilité) pour avoir des systèmes vivace et équitable aussi.

Fairness

- Example: if the system allocates a shared resource among several users, only those paths along which no user keeps the resource forever should be considered.

Frairness (équitablitié)

- **Weak fairness :**

*If a process P reaches a point where it has **an executable statement**, and the **executability** of that statement never changes, then P should eventually proceed by executing the statement.*

- **Strong fairness :**

*If a process P reaches a point where it has an **executable statement**, and the **executability** of that statement **occurs infinitely** often from there on, then P should eventually proceed by executing the statement.*

Vérification: safety

- Si un **“Assert statement”** échoue,

- Si **une exécution finie** se **termine** dans un **état** qui **viole** une certaine condition de terminaison (e.g., with all processes in a valid end-state, and all message channels empty).

Vérification: Liveness

Si une **exécution infinie** :

1) Contient un nombre fini de passages sur un “***progress* states”**,

ou

2) Contient un nombre infini de passages sur un ”***accept* states”**.

Tout ça en SPIN ?

- Safety (State properties)
    - Assertions
    - Invalid Endstates
- Liveness
    - Non-Progress Cycles
    - Acceptance Cycles
    - With Weak Fairness

Formalisation LTL: Syntaxe

- La Syntaxe des opérateurs est comme suit:

– [] - “**always**”
– <> - “**eventually**”
– U - “**until**”
– | | - “**or**”
– & - “**and**”
– ~ - “**not**”

- Un atom peut être défini comme suit:

- #define p (x < 4 || x > 8)
- #define q (len(c) < 4)

- L’usage de **next** n’est pas possible;
- SPIN convertit une formule LTL: φ vers un automate de Büchi $\mathbf{A}_{\neg \varphi}$.
  exemple: $ **spin** -f ‘’[]<>(p || q)’’

Formalisation LTL: **never claims**

Les propriétés LTL seront introduites sous forme de: **never claims** (veut dire ne dois jamais arriver) et ces never claims seront ajoutés dans la spécification.

Exemple:

la propriété LTL: **Gp** est transformée en:

```
never{
    do
        :: !p -> break
        :: else
    od
}
```

Formalisation LTL:
never claims (exemple)

*Every system state in which p is true **eventually leads** to a system state in which q holds, and in between p must remain true.*

LTL: [](*p → pUq)*

Never claim:

```
never{
S0 : do
        :: p && !q -> break
        :: true
     od
S1 :
accept: do
        :: !q
        :: !(p || q) -> break
        od
}
```

Formalisation LTL: vérification des **never claims**

- Un «**never claim**» est vérifiée dans chaque étape d'exécution du programme Promela;

- Si le **never claims** est satisfait donc la propriété LTL n'est pas vérifiée, et le SPIN affiche la trace d'exécution qui satisfait ce **never claim**

Question

Pourquoi la satisfaction du **never claim** signifie que la propriété LTL n'est pas satisfaite

???

Les assertions de traces (1)

- *Ils sont utilisés pour vérifier la validité ou l'invalidité **d'une séquence d'opérations** exécutées par les processus sur les "***message channels"**
- Tous les "**channel names**" utilisés dans une assertion de trace doivent être **globales**.
- On peut utiliser des structures de control, mais pas de variables globales /locales, pas d'assertions, ni de conditions booléennes.
- Les traces qui concernent des "**rendezvous channels**" traitent **seulement** des actions: **receive**.
- Les actions "**receive**" et les "**rendezvous channels**" peuvent être étudiés seulement avec **les assertion des traces** mais pas avec les "**never claims**".

Exemple (trace)

```
trace {
        do
        :: sem?p; sem?v
        od
}
```

Remarque:

- **trace assertions** formulate properties over **events**
- L**TL-formulae** formulate properties over **states**

Les assertions de traces (2)

- Une déclaration de trace permet de **spécifier un comportement correct du système**

Exemple 31

```
trace{
    do
    :: q1!a; q2?b
    od
}
```

- *La trace d'événements déclarée spécifie que les opérations d'envoi sur le canal q1 alternent avec les opérations de réception sur le canal q2.*

- *De plus tous les envois sur q1 sont des messages de valeur a et toutes les réceptions sur q2 sont des messages de valeur b.*

- De manière similaire, la primitive **notrace** décrit les séquences d'exécution invalides.