17 Spécifications de propriétés à l'aide de la logique temporelle

17.1 Propriétés d'un système réactif: sécurité et vivacité

Description LOTOS= description des séquences possibles d'actions = description *opérationnelle*
Impossible, en LOTOS, de décrire des *propriétés* de ces séquences d'actions

17.1.1 Les deux grandes classes de propriété

1. Sécurité (*safety*): un comportement non désiré est certain de ne pas survenir
   (*nothing bad will ever happen*).
2. Vivacité (*liveness*): un comportement désiré est certain de survenir
   (*something good will eventually happen*).

Sécurité = invariant
`INVARIANT ~prop`

Vivacité = contrainte sur le futur
`prop` sera satisfaite "*dans le futur*"

17.1.2 Logique modale et temporelle

Logique temporelle = logique qui permet de spécifier des modalités sur les actions et sur l'écoulement du temps

Selon "le petit Robert": Gram. *Adv. de modalité*, qui modifie le sens d'une phrase entière (et non d'un mot isolé).

17.1.2.1 Modalités pour les actions

Une modalité sur une action permet d'indiquer que cette action, dans l'état actuel, est soit *possible*, soit *requise*.

Formules valides (syntaxe)

- `true`, `false`
- `not` ϕ
- ϕ_1 `and` ϕ_2
- ϕ_1 `or` ϕ_2
- `<`A`>`ϕ (où A est un ensemble d'actions)
- `[`A`]`ϕ (où A est un ensemble d'actions)

Interprétation des formules (sémantique)

- e $\models$ `true`
- e $\not\models$ `false`
- e $\models$ `not` ϕ ssi e $\not\models$ ϕ
- e $\models \phi_1$ `and` ϕ_2 ssi e $\models \phi_1$ et e $\models \phi_2$
- e $\models \phi_1$ `or` ϕ_2 ssi e $\models \phi_1$ ou e $\models \phi_2$
- e $\models$ `<`A`>`ϕ ssi il existe `e'` et `a` $\in A$ tel que e $\xrightarrow{a}$ `e'` et `e'` $\models \phi$
- e $\models$ `[`A`]`ϕ ssi pour tout `e'` et `a` $\in A$ tel que e $\xrightarrow{a}$ `e'`, on a que `e'` $\models \phi$

Exemple Voyons quelques exemples simples de formules modales:

- e $\models$ `<`a`>true`
- e $\models$ `[`a`]false`
- e $\models$ `[`a`][`b`]false`
- e $\models$ `[`a`,` b`]<`c`,` d`>true`
- e $\models$ `<*>true`
- e $\models$ `[*]false`

17.1.2.2 Modalités temporelles

Permettent de formuler des propriétés *sur les différents futurs possibles*

- e $\models$ `POT`(ϕ): Il existe un futur possible où la propriété *deviendra* vraie (potentiellement)
- e $\models$ `INEV`(ϕ): Dans tous les futurs possibles, la propriété *deviendra* vraie (inévitablement)
- e $\models$ `ALL`(ϕ): Dès maintenant et dans tous les futurs possibles, la propriété est et sera vraie

Autre façon d'exprimer:

- e $\models$ `INEV`(ϕ) ssi e satisfait la propriété X définie récursivement comme suit:
  $X = \phi$ `or (<*>true and [*]`X`)`
- e $\models$ `ALL`(ϕ) ssi e satisfait la propriété X définie récursivement comme suit:
  $X = \phi$ `and [*]`X
- e $\models$ `POT`(ϕ) ssi on a la propriété suivante
  `SOME`(ϕ) = `not (INEV(not` ϕ`))`.
  e $\models$ `not (ALL(not` ϕ`))`
  En d'autres mots, une propriété ϕ est potentiellement vraie s'il n'est pas possible qu'elle soit toujours fausse.

17.1.3 Quelques exemples

Exemple Soit le processus décrivant la machine distributrice de biscuits et muffins:

1. Dans on état initial, la machine accepte tant une pièce de 0.25$ que de 1.00$.

```
<p25> true and <p100> true
```

2. Après avoir mis une pièce d'un dollar, il n'est plus possible de mettre une autre pièce.

```
[p100]([p100] false and [p25] false)
```

3. Après avoir mis trois pièces de 0.25$, on peut soit mettre une autre pièce de 0.25$, soit sélectionner un biscuit; il n'est toutefois pas possible de sélectionner un muffin.

```
[p25][p25][p25](<p25, b_bisc> true and [b_muff] false)
```

4. Après avoir sélectionné un item, l'item doit nécessairement être livré avant que d'autres pièces puissent être insérées.

```
POT(
   <b_bisc, b_muff> true
   and
   [b_bisc, b_muff]<bisc, muff><p25, p100> true
)
```

5. En tout temps, la machine peut continuer à fonctionner, sans jamais arriver à un blocage (*deadlock*).

```
ALL( <*> true )
```

6. La machine effectue un cycle sans fin de livraisons d'items. En d'autres mots, de façon continue, la machine pourra potentiellement livrer des biscuits ou muffins.

```
ALL( POT( <bisc, muff> true )  )
```

Exemple Soit le processus décrivant la machine distributrice de café et bouillon:

1. Il n'est pas possible d'insérer 4 pièces de suite.

```
[p25][p25][p25][p25] false
```

2. Après avoir inséré une pièce, et uniquement à partir du moment où une première pièce a été insérée, on peut presser sur le bouton d'annulation.

```
[b_annul] false
and
[p25]<b_annul> true
and
[p25][p25]<b_annul> true
and
[p25][p25][p25]<b_annul> true
```

Exemple Soit le processus suivant:

```
PROCESS Horloge[tic]: NOEXIT :=
  tic; Horloge[tic]
ENDPROC
```

En tout temps, `Horloge` va émettre le signal `tic`:

```
ALL ( <tic> true )
```

Façon récursive d'exprimer cette propriété:

```
X = <tic> true and [*] X
```

Autre propriété: Le processus ne peut rien faire d'autre que des `tic`s

```
ALL ( [*-tic] false )
```

Soit maintenant l'horloge suivante:

```
PROCESS Horloge[tic, tac]: NOEXIT :=
  tic; tac; Horloge[tic, tac]
ENDPROC
```

```
ALL (
  [tic]<tac> true and [tac]<tic> true
  and
  ( <tic> true or <tac> true )
  and
  [*-tic, tac] false
)
```

Exemple

```
SPECIFICATION Semaphore[a1, a2, b1, b2] : NOEXIT
  BEHAVIOUR
   HIDE p, v IN
    ( Proc[a1, a2, p, v] ||| Proc[b1, b2, p, v] )
    |[p, v]|
    Semaphore[p, v]
   WHERE
     PROCESS Proc[ac1, ac2, p, v] : NOEXIT :=
       p; ac1; ac2; v; Proc[ac1, ac2, p, v]
     ENDPROC

     PROCESS Semaphore[p, v] : NOEXIT :=
       p; v; Semaphore[p, v]
     ENDPROC
ENDSPEC
```

1. Sans fin, une action a1 sera immédiatement suivie d'une action a2; idem pour b1 et b2.

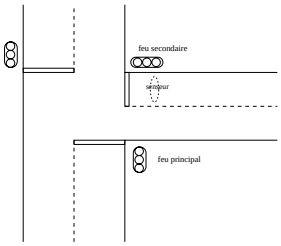
```
ALL(
   POT( <a1><a2> true ) and [a1]<a2> true
   and
   POT( <b1><b2> true ) and [b1]<b2> true
 )
```

2. On n'aura jamais une série d'actions où les a et b sont entrelacées.

```
not POT ( <a1><b1> true or <a1><b2> true
          or
          <b1><a1> true or <b1><a2> true )
```

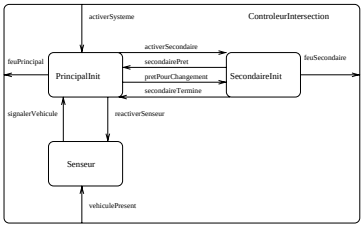
17.2 Système de contrôle de feux de circulation

17.2.1 Description du problème



17.2.2 Description détaillée des signaux et ports d'interaction

- `activerSysteme`
- `vehiculePresent`
- `feuPrincipal`
  Information transmise: `rouge`, `vert` ou `orange` et `allumer` ou `eteindre`
- `feuSecondaire`: idem



17.2.4 Propriétés requises (sécurité et vivacité)

- Sécurité: Le feu de la route principal et le feu de la route secondaire ne doivent jamais être verts en même temps.
- Vivacité: Si un véhicule arrive à l'intersection de la route secondaire, alors éventuellement le feu de la route secondaire devra devenir vert pour laisser passer ce véhicule.
- Vivacité: Éventuellement, le feu de la route principal devra redevenir vert.

1. Sécurité: Le feu de la route principal et le feu de la route secondaire ne sont jamais verts en même temps.

```
ALL (
   not POT (
     <feuPrincipal !vert !allumer> true
     and
     <feuSecondaire !vert !allumer> true
   )
)
```

2. Vivacité: Si un véhicule arrive à l'intersection de la route secondaire, alors éventuellement le feu va devenir vert pour laisser passer le véhicule.

```
ALL ( INEV ( <vehiculepresent> true
      and
      [vehiculepresent]
        (POT (<feusecondaire !vert !allumer> true)) ) )
```

3. Vivacité: Éventuellement, le feu de la route principale va redevenir vert.

```
ALL (
      INEV (<feuprincipal !vert !allumer> true)
)
```