UNIVERSITÉ DE LORRAINE

TELECOM Nancy
Ingénieurs du numérique • Inspiring your digital future

Cours MALG & MOVEX

Modélisation synchrone

Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2023-2024

Plan

Sommaire

Current Summary

Horloges

- Toutes les expressions sont des streams.
- Les opérateurs d'horloge modifient la temporalité des streams et le résultat est stream :
  - `pre` s pour toute stream s.
  - $s1 -> s2$ s1 *suivi de* s2.
  - $s1$ `when` $s2$ *échantillonnage*
  - `current` s pour toute stream s.

Sémantique des horloges

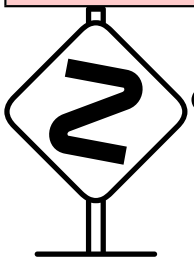
- $[\![\sigma]\!] \stackrel{def}{=} (\sigma_0, \sigma_1, \ldots)$
- $[\![\texttt{pre}(\sigma]\!]) \stackrel{def}{=} (\bot, \sigma_0, \sigma_1, \ldots)$
- $[\![\sigma - > \tau]\!]) \stackrel{def}{=} (\sigma_0, \tau_1, \tau_2, \ldots)$
- $[\![\sigma \texttt{ when } \varphi]\!]) \stackrel{def}{=} (\sigma_{t_0}, \tau_{t_1}, \tau_{t_2}, \ldots)$ où la suite des t_i est la suit des instants validant φ dans la suite σ.
- $[\![\texttt{current}(\sigma \texttt{ when } \varphi]\!])) \stackrel{def}{=}$
  $(\bot, \ldots, \bot, \sigma_{t_0}, \sigma_{t_0}, \sigma_{t_0}, \tau_{t_1}, \tau_{t_1}, \tau_{t_1}, \tau_{t_1}, \tau_{t_1}, \tau_{t_2}, \ldots)$

Exemple d'horloge

(Horloge)

Listing 1 – clock1.lus

```
node clock1(b: bool) returns (y: int);
var n:int;
var e:bool;
var f:int;
let
  n = 0 -> pre(n)+1;
  e = true ->  not pre(e);
  f = current ( n when e);
  y = current ( n when e) div 2;
tel
```



`current` n'est pas supporté.

Current Summary

4	4	4	...	4	...
x	x_0	x_1	...	x_n	...
y	y_0	y_1	...	y_n	...
x+y	x_0+y_0	x_1+y_1	...	x_n+y_n	...

x	x_0	x_1	...	x_n	...
pre x	NIL	x_0	...	x_{n-1}	...

x	x_0	x_1	...	x_n	...
y	y_0	y_1	...	y_n	...
x->y	x_0	y_1	...	y_n	...

```
nat = 0 -> (1 + pre nat)
```

h	**true**	**false**	**true**	**true**	**false**
x	x_0	x_1	...	x_n	...
x when h	x_0	$-$	x_2	x_3	$-$

Programmes LUSTRE

- **Un programme LUSTRE s'appelle un nœud ou NODE.**
- **Un programme LUSTRE dénote une suite infinie de valeurs comme suit :** $(x_0\ x_1\ x_2\ \ldots)$
- **Deux opérateurs sur les programmes :**
  - $\mathbf{pre}$
  - $\longrightarrow$
- $\forall n \geq 0. CUP_{n+1} = CUP_n + 1$ **s'écrira** $CUP = 0 \longrightarrow (1 + \mathbf{pre}(CUP))$
- **et produira la suite** $(0\ 1\ 2\ ...)$**.**
- $FIB = 0 \longrightarrow 1 \longrightarrow (\mathbf{pre}(FIB) + \mathbf{pre}(\mathbf{pre}(FIB)))$

Programmes LUSTRE

- $node \quad EDGE(X : bool) \quad returns \quad (Y : bool)$
  let
  $\quad Y = false \rightarrow \ X \ and \ not \ pre(X)$
  tel

désigne la suite $(false, x_1 \wedge \neg x_0, x_2 \wedge \neg x_1, \ ...)$

- **Counter**
  $C = 0 \longrightarrow \mathbf{pre}(C)+1$ **renvoie la suite des naturels**
  $C = 0 \longrightarrow if \ X \ then \ \mathbf{pre}(C)+1 \ else \ \mathbf{pre}(C)$
  compte le nombre d'occurrences de X **qui sont vraies.**
  On ignore la valeur initiale
  $PC = 0 \longrightarrow \mathbf{pre}(C)$
  $C = if \ X \ then \ PC+1 \ else \ PC$

Current Summary

Propriétés des programmes LUSTRE

()

Listing 2 – kindsession2/power2.lus

```
-- power2(_) contains all the square  numbers in increasing order
-- where 0^2 = 0
--       (n+1)^2 = n^2 +2*n + 1
node power2() returns (P: int);
var W: int;
let
  -- all the natural even numbers
  W = 1 -> (pre W) + 2;
  P = 0 -> (pre P) + (pre W);
tel
```

Propriétés des programmes LUSTRE

()

Listing 4 – kindsession2/power2.lus

```
-- power2(_) contains all the square  numbers in increasing order
-- where 0^2 = 0
--       (n+1)^2 = n^2 +2*n + 1
node power2() returns (P: int);
var W: int;
let
  -- all the natural even numbers
  W = 1 -> (pre W) + 2;
  P = 0 -> (pre P) + (pre W);
tel
```

(`--enable BMC --enable IND`)

Listing 5 – kindsession2/power2prop.lus

```
-- power2(_) contains all the square  numbers in increasing order
-- where 0^2 = 0
--       (n+1)^2 = n^2 +2*n + 1
include "power2.lus"
node power2prop() returns (P: int);
var P2,PP2,N:int;
var PROP:bool;
let
  PP2 = power2();
  N = 0 -> (pre N)+1;
  P2 = N*N;
  PROP  = P2 = PP2;
  check PROP;
tel
```

Propriétés des programmes LUSTRE

()

Listing 6 – kindsession2/greycounter.lus

```
node greycounter (reset: bool) returns (out: bool);
var a, b: bool;
let
  a = false -> (not reset and not pre b);
  b = false -> (not reset and pre a);
  out = a and b;

tel
```

()

Listing 7 – kindsession2/intcounter.lus

```
node intcounter (reset: bool; const max: int) returns (out: bool);
var t: int;
let
  t = 0 -> if reset or pre t = max then 0 else pre t + 1;
  out = t = 2;
tel
```

Propriétés des programmes LUSTRE

(Valid property)

Listing 8 – kindsession2/ex1.lus

```
/* include    */
include "greycounter.lus"

/* include    */
include "intcounter.lus"


node top (reset: bool) returns (OK, OK2: bool);
var b, d: bool;
let
  b = greycounter(reset);
  d = intcounter(reset, 3);
  OK = b = d;
  --%PROPERTY OK;
tel
```

Propriétés des programmes LUSTRE

(Invalid property)

Listing 9 – kindsession2/ex2.lus

```
/* include    */
include "greycounter.lus"

/* include    */
include "intcounter.lus"


node top (reset: bool) returns (OK, OK2: bool);
var b, d: bool;
let
  b = greycounter(reset);
  d = intcounter(reset, 3);
  OK = b = d;
  OK2 = not d;
  --  %PROPERTY OK;
  --%PROPERTY OK2;
tel
```

Current Summary

Treillis complet

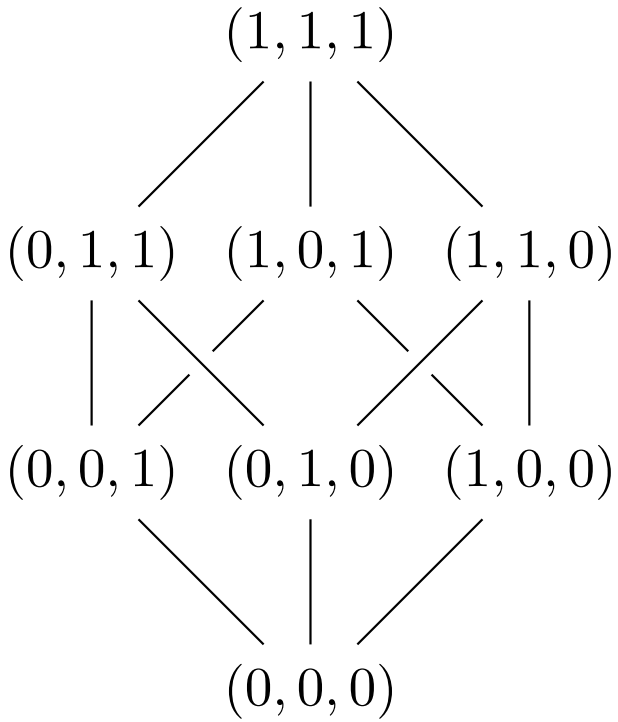
⊠ Definition

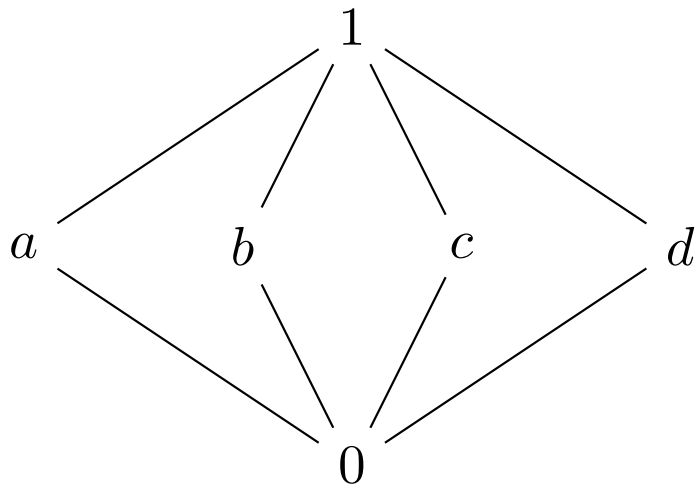
Un treillis complet $(L, \sqsubseteq, \bot, \sqcup, \top, \sqcap)$ est un treillis satisfaisant les propriétés suivantes :

1. Pour toute partie A de L, il existe une borne supérieure notée $\sqcup A$.
2. Pour toute partie A de L, il existe une borne inférieure notée $\sqcap A$.

- Un treillis complet $(L, \sqsubseteq,)$ peut être défini par les éléments suivants $(L, \sqsubseteq, \bot, \sqcup, \top, \sqcap)$
- Un treillis est une structure munie d'un ordre partiel et telle que deux éléments ont une borne supérieure et une inférieure dans le treillis : $(L, \sqsubseteq, \sqcup, \sqcap)$
  - $a \sqcup b$ existe et est la borne supérieure des deux éléments a et b.
  - $a \sqcap b$ existe et est la borne inférieure des deux éléments a et b.

Représentation par des diagrammes de Hasse





Pre/Post Points-fixes

Pour une fonction f définie sur un treillis $(L, \sqsubseteq, \bot, \sqcup, \top, \sqcap)$ et à valeurs dans $(L, \sqsubseteq, \bot, \sqcup, \top, \sqcap)$.

Définition

- on appelle pré-point-fixe de f, tout élément x de L satisfaisant la propriété $f(x) \sqsubseteq x$
- on appelle post-point-fixe de f, tout élément x de L satisfaisant $x \sqsubseteq f(x)$.

- $PostFIX(f) = \{x | x \in L \wedge x \sqsubseteq f(x)\}$: l'ensemble des post-points-fixes de f.
- $PreFIX(f) = \{x | x \in L \wedge f(x) \sqsubseteq x\}$: l'ensemble des pré-points-fixes de f.
- $FIX(f) = \{x | x \in L \wedge f(x) = x\}$: l'ensemble des points-fixes de f.

Théorème de Knaster-Tarski (I)

Soit f une fonction monotone croissante sur un treillis complet $(T, \bot, \top, \bigvee, \bigwedge)$. Alors il existe un plus petit point fixe et un plus grand point fixe pour f.

- μf désigne le plus petit point fixe de f.
- νf désigne le plus grand point fixe de f.
- μf vérifie les propriétés suivantes : $f(\mu f) = \mu f$ et $\forall x \in T. f(x) \sqsubseteq x \Rightarrow \mu f \sqsubseteq x$ (μf est la borne inférieure des prépoints fixes de f ou $\bigwedge(Pre(f))$).
- νf vérifie les propriétés suivantes : $f(\nu f) = \nu f$ et $\forall x \in T. x \sqsubseteq f(x) \Rightarrow x \sqsubseteq \nu f$ (νf est la borne supérieure des postpoints fixes de f ou $\bigvee(Post(f))$).

Preuve

Posons $y = \bigwedge\{x | f(x) \sqsubseteq x\}$ et montrons que y est un point fixe de f et que y est le plus petit point fixe de f.

1. $f(y) \sqsubseteq y$
   - Pour tout x de $\{x | f(x) \sqsubseteq x\}$, $y \sqsubseteq x$
   - $f(y) \sqsubseteq f(x)$ (par monotonie de f).
   - $f(x) \sqsubseteq x$ (par définition de x).
   - $f(y) \sqsubseteq x$ (par déduction).
   - $f(y) \sqsubseteq \bigwedge\{x | f(x) \sqsubseteq x\}$ (par définition de la borne inférieure, $f(y)$ est un minorant).
   - $f(y) \sqsubseteq y$
2. $y \sqsubseteq f(y)$
   - $f(y) \sqsubseteq y$ (par le cas 1)
   - $f(f(y)) \sqsubseteq f(y)$ (par monotonie de f)
   - $f(y) \in \{x | f(x) \sqsubseteq x\}$
   - $y \sqsubseteq f(y)$ (par définition de la borne inférieure)
3. Conclusion : $f(y) = y$ ou $y \in FIX(f)$.

Preuve

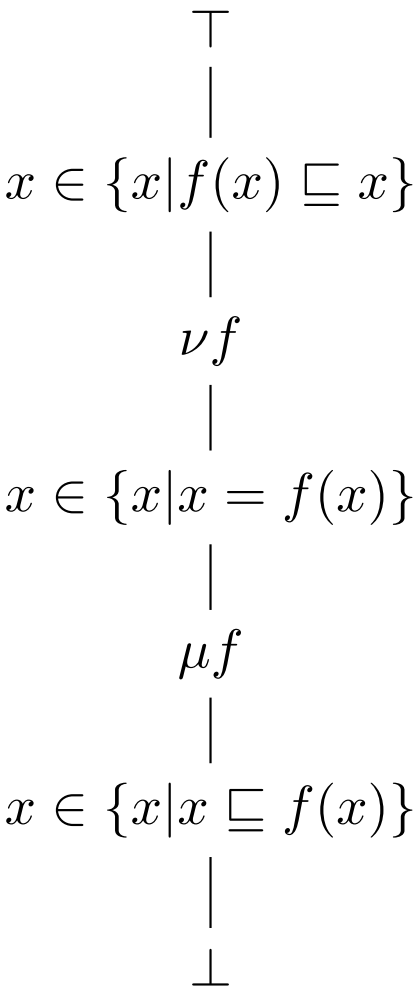
- $f(y) = y$ et z tel que $f(z) = z$
  - $f(z) = z$ (par hypothèse sur z)
  - $f(z) \sqsubseteq z$ (par affaiblissement de l'égalité)
  - $z \in \{x | f(x) \sqsubseteq x\}$ (par définition de cet ensemble)
  - $y \sqsubseteq z$ (par construction)
- y est le plus petit point fixe de f.

lfp(f) et gfp(f)

- $\mu f = lfp(f) = \bigwedge\{x | f(x) \sqsubseteq x\}$
- $\nu f = gfp(f) = \bigvee\{x | x \sqsubseteq f(x)\}$

- $lfp(f)$ signifie *least fixed-point*
- $gfp(f)$ signifie *greatest fixed-point*

Positionnement des éléments



- $\top = \sqcup\{x | f(x) \sqsubseteq x\}$
- $gfp(f) = \nu f = \sqcup\{x | x \sqsubseteq f(x)\}$
- $lfp(f) = \mu f = \sqcap\{x | f(x) \sqsubseteq x\}$
- $\bot = \sqcap\{x | x \sqsubseteq f(x)\}$

Preuve algébrique selon Hoare

- $x \in pre(f)$ **si, et seulement si,** $f(x) \leq x$ (définition)

Preuve algébrique selon Hoare

- $x \in pre(f)$ **si, et seulement si,** $f(x) \leq x$ (définition)
- $\forall a \in pre(f).\mu f \leq a$ (application de la définition de la borne inférieure d'un ensemble)

Preuve algébrique selon Hoare

- $x \in pre(f)$ **si, et seulement si,** $f(x) \leq x$ (définition)
- $\forall a \in pre(f).\mu f \leq a$ (application de la définition de la borne inférieure d'un ensemble)
- $\forall a \in pre(f).f(\mu f) \leq f(a) \leq a$ (croissance de f et propriété de a)

Preuve algébrique selon Hoare

- $x \in pre(f)$ **si, et seulement si,** $f(x) \leq x$ (définition)
- $\forall a \in pre(f).\mu f \leq a$ (application de la définition de la borne inférieure d'un ensemble)
- $\forall a \in pre(f).f(\mu f) \leq f(a) \leq a$ (croissance de f et propriété de a)
- $f(\mu f) \leq \mu f$ (application pour $a = \mu f$))

Preuve algébrique selon Hoare

- $x \in pre(f)$ **si, et seulement si,** $f(x) \leq x$ (définition)
- $\forall a \in pre(f).\mu f \leq a$ (application de la définition de la borne inférieure d'un ensemble)
- $\forall a \in pre(f).f(\mu f) \leq f(a) \leq a$ (croissance de f et propriété de a)
- $f(\mu f) \leq \mu f$ (application pour $a = \mu f$))
- $f(f(\mu f)) \leq f(\mu f)$ (croissance de f)

Preuve algébrique selon Hoare

- $x \in pre(f)$ **si, et seulement si,** $f(x) \leq x$ (définition)
- $\forall a \in pre(f).\mu f \leq a$ (application de la définition de la borne inférieure d'un ensemble)
- $\forall a \in pre(f).f(\mu f) \leq f(a) \leq a$ (croissance de f et propriété de a)
- $f(\mu f) \leq \mu f$ (application pour $a = \mu f$))
- $f(f(\mu f)) \leq f(\mu f)$ (croissance de f)
- $f(\mu f) \in pre(f)$ (définition des pré-points-fixes)

Preuve algébrique selon Hoare

- $x \in pre(f)$ **si, et seulement si,** $f(x) \leq x$ (définition)
- $\forall a \in pre(f).\mu f \leq a$ (application de la définition de la borne inférieure d'un ensemble)
- $\forall a \in pre(f).f(\mu f) \leq f(a) \leq a$ (croissance de f et propriété de a)
- $f(\mu f) \leq \mu f$ (application pour $a = \mu f$))
- $f(f(\mu f)) \leq f(\mu f)$ (croissance de f)
- $f(\mu f) \in pre(f)$ (définition des pré-points-fixes)
- $\mu f \leq f(\mu f)$ (définition de μf)
- $\mu f = f(\mu f)$ (définition de μf et propriété précédente)

Version constructive du théorème de Knaster-Tarski

Soit f une fonction monotone croissante sur un treillis complet $(T, \bot, \top, \bigvee, \bigwedge)$. Alors

1. La structure formée des points fixes de f sur T, $(fp(f), \sqsubseteq)$ est un treillis complet non-vide.
   $(fp(f) = \{x \in T : f(x) = x\}$
2. $lfp(f) \stackrel{def}{=} \bigvee_\alpha f^\alpha$ est le plus petit point fixe de f où :
$$\begin{cases} & f^0 & \stackrel{def}{=} & \bot \\ \alpha \textit{ ordinal successeur} & f^{\alpha+1} & \stackrel{def}{=} & f(f^\alpha) \\ \alpha \textit{ ordinal limite} & f^\alpha & \stackrel{def}{=} & \bigvee_{\beta<\alpha} f^\beta \end{cases}$$
3. $gfp(f) \stackrel{def}{=} \bigwedge_\alpha f_\alpha$ est le plus grand point fixe de f où
$$\begin{cases} & f_0 & \stackrel{def}{=} & \top \\ \alpha \textit{ ordinal successeur} & f_{\alpha+1} & \stackrel{def}{=} & f(f_\alpha) \\ \alpha \textit{ ordinal limite} & f_\alpha & \stackrel{def}{=} & \bigwedge_{\beta<\alpha} f_\beta \end{cases}$$

Computing the least fixed-point over a finite lattice

INPUT $F \in T \longrightarrow T$
OUTPUT $result = \mu.F$
VARIABLES $x, y \in T, i \in \mathbb{N}$
$\ell_0 : \{x, y \in T\}$
$x := \bot;$
$y := \bot;$
$i := 0;$
$\ell_{11} : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T) \wedge i = 0\};$
WHILE $i \leq Card(T)$
$\ell_1 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T)\};$
$x := F(x);$
$\ell_2 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T)\};$
$y := x \sqcup y;$
$\ell_3 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0;k=i+1} F^k \wedge i \leq Card(T)\};$
$i := i+1;$
$\ell_4 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T)+1\};$
OD;
$\ell_5 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T)+1\};$
$result := y;$
$\ell_6 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T)+1 \wedge result = y\};$

Verification in action

- Identify the safety property S to check.
- Run the algorithm for computing μF.
- Check that $\mu F \subseteq S$ or $\overline{S} \cap \mu F = \varnothing$.
- Check that $BUG \cap \mu F = \varnothing$, when BUG is a set of states that you identify as *bad states*.

Problem

- The general case is either infinite or large . . . approximations of μF.
- Computing over abstract finite domain
- How to compute when it is not decidable ?
- Develop a framework for defining sound abstractions of software systems under analysis.

Expression du problème à résoudre

- S is the set of states
- $(\mathcal{P}(S \times S), \subseteq)$ is a complete lattice.
- $R \subseteq S \times S$ is a binary relation over S simulating the computation or the transition as $Next(x, x')$.
- $F \in \mathcal{P}(S \times S) \rightarrow \mathcal{P}(S \times S)$ is defined by the following expression :
  - $X \subseteq S \times S$
  - $I = \{(s, s) | s \in S\}$
  - $F(X) = I \cup R; X$

Transitive closure of R

- F is a monotonous function.
- F has a least-fixed point denoted μF.
- $\mu F = R^\star$
- $\forall n \in \mathbb{N} : F^{n+1}(\varnothing) = \bigcup_{i=0}^{n}$
- $\bigcup_{n \geq 0} F^n = R^\star$

Checking safety

- $G \subseteq S$ is the set of Good states.
- $I \subseteq S$ is the set of initial states.
- $\forall s_0, s \in S : s_0 \in I \wedge R^\star(s_0, s) \Rightarrow s \in G$ (Expression of safety of G)
- $\forall s \in S : (\exists s_0 . s_0 \in I \wedge R^\star(s_0, s)) \Rightarrow s \in G$
- $R^\star[I] \subset G$
- $R^\star[I] = \bigcup_{n \geq 0} F^n[I] = \bigcup_{n \geq 0} G^n$
- $\bigcup_{n \geq 0} G^n \subseteq G$ (Expression de la safety)

Techniques fo checking

$\bigcup_{n \geq 0} G^n \subseteq G$ is checked using at least two possible techniques :

- Bounded Model Checking
- k-Induction
- ...

Current Summary

Exemple de contrats

(simple contrat)

Listing 10 – kindsession2/contract0.lus

```
node g () returns ();
(*@contract
  assume true;
  guarantee true;
*)
const n = 7;
var t : int;
let
        t = n;

tel;
```

Exemple de contrats

(simple contrat)

Listing 11 – kindsession2/contract1.lus

```
contract countSpec(trigger: bool; val: int) returns (count: int ; error: bool) ;

let
  assume val >= 0;
  var initVal: int = val -> pre(initVal);
  var once: bool = trigger or (false -> pre once) ;
  guarantee count >= 0 ;

mode still_zero (
    require not once ;
    ensure count = initVal ;
  ) ;

mode gt (
    require not ::still_zero ;
    ensure count > 0 ;
  ) ;
tel
```

Contrats

assumes

An assumption over a node n is a constraint one must respect in order to use n legally. It cannot depend on outputs of n in the current state, but referring to outputs under a pre is fine.
The idea is that it does not make sense to ask the caller to respect some constraints over the outputs of n, as the caller has no control over them other than the inputs it feeds n with.
The assumption may however depend on previous values of the outputs produced by n. Assumptions are given with the assume keyword, followed by any legal Boolean expression :

guarantees

Unlike assumptions, guarantees do not have any restrictions on the streams they can depend on. They typically mention the outputs in the current state since they express the behavior of the node they specified under the assumptions of this node.
Guarantees are given with the guarantee keyword, followed by any legal Boolean expression :

modes

A mode (R,E) is a set of requires R and a set of ensures E.
Modes are named to ease traceability and improve feedback.

(mode)

Listing 12 – kindsession2/mode.lus

```
mode <id> (
  [require <expr> ;]*
  [ensure  <expr> ;]*
);
```

Current Summary

Sommaire sur la modélisation synchrone

- Modèle de calcul simplifié
- Modélisation des systèmes réactifs : systèmes qui réagissent à des stimuli ou des signaux de l'environnement.
- Méthodologie de développement fondée sur le triptyque système,environnement,observation.
- Outils pour simuler, vérifier et transformer :
  - Compilateur V6 lv6
  - Kind2