Bonus et réponses aux questions

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages
gerard.berry@college-de-france.fr

Cours 7, 13 avril 2016

Et séminaire de Chantal Keller (LRI Paris-Sud)

COLLÈGE DE FRANCE
1530

Agenda

1. Bonus 1 : le synchroniseur Esterel en Why (SMT)
2. Bonus 2 : la machine chimique (rappel de 2009)
3. Bonus 3 : exemple de vérification avec CADP
4. Réponse aux questions

Agenda

1. Bonus 1 : le synchroniseur Esterel en Why (SMT)
2. Bonus 2 : la machine chimique (rappel de 2009)
3. Bonus 3 : exemple de vérification avec CADP
4. Réponse aux questions

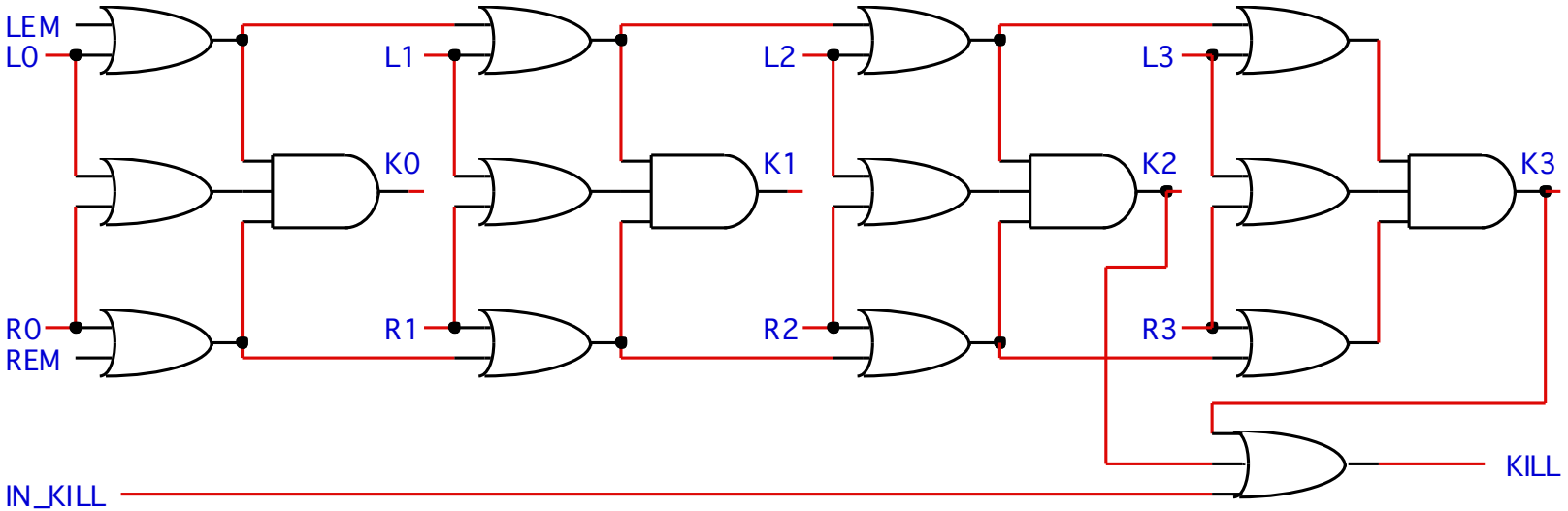
Numérotation des traps

```
trap T in
   trap U in
      nothing0
   ||
      pause1
   ||
      exit U2
   ||
      exit T3
   end trap
||
   exit T2
end trap
```

Si deux traps sont levés en même temps, seul le plus extérieur compte

Code de retour du parallèle
= max des codes des branches
(codage de Gonthier)

Esterel : Synchroniseur du parallèle (Gonthier) (cf cours 4 du 23 avril 2013)



Distribution de Max sur 2 ensembles

$$Max(X,Y) = \{\max(x,y) \mid x \in X, y \in Y\}$$
$$= \{z \in X \cup Y \mid z \geq \max(\min(X), \min(Y))\}$$

Codage en bitvecteurs :

$$Max(X,Y) = (X \mid Y) \,\&\, (X \mid -X) \,\&\, (Y \mid -Y)$$

X = 00101010 = {1,3,5}

Y = 01100100 = {2,5,6}

Max(X,Y) = 01101100 = {2,3,5,6}

X = 00101010

$-X$ = 11010110

$X \mid -X$ = 11111110 = $\{z \mid z \geq \min(X)\}$

$X \mid Y$ = 01101110 = $\{z \mid X \cup Y\}$

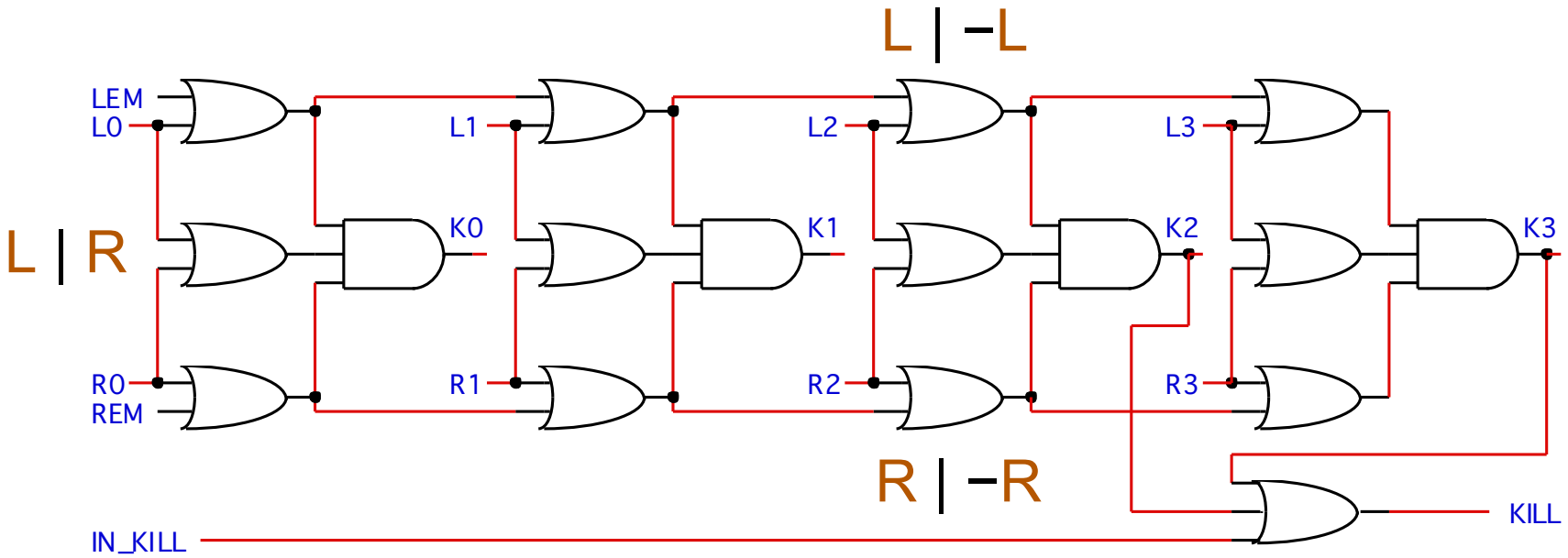
& $X \mid -X$ = 11111110 = $\{z \mid z \geq \min(Y)\}$

& $Y \mid -Y$ = 11111100 = $\{z \mid z \geq \min(Y)\}$

Max(X,Y) = 01101100 = {2,3,5,6}

Preuve :
Why3
+ Alt-Ergo
+ CVC4

Esterel : Synchroniseur du parallèle (Gonthier) (cf cours 4 du 23 avril 2013)



$$K = Max(L, R)$$

Merci J-C. Filliâtre et l'équipe Why (Orsay)

```
module Esterel
use import int.Int
        (* integer arithmetic from Why3 Stdlib *)
use import int.MinMax
        (* min and max of integers from Why3 Stdlib *)
use import set.Fsetinit
        (* Finite sets of integers. Provides in particular:
                mem, union, inter
                min_elt  (unspecified for empty set)  *)
use import bv.BV64
        (* 64-bit bitvectors *)
```

```
type s = { bv : BV64.t ;
           (* a 64-bit bitvector *)
ghost mdl: set int;
           (* its interpretation as a set *) }
invariant { forall i  : int.
                  (0 <= i ≤ size /\ nth self.bv i) <-> mem i self.mdl }

let union (a b : s) : s      (* operator [a|b] *)
ensures { result.mdl = union b.mdl a.mdl }
   = { bv = bw_or a.bv b.bv ;
       mdl = union b.mdl a.mdl }

let intersection (a b : s) : s     (* operator [a&b] *)
ensures { result.mdl = inter a.mdl b.mdl }
   = { bv = bw_and a.bv b.bv;
       mdl = inter a.mdl b,mdl }
```

```
let unionUminus (a : s) : s (* operator [a|-a] *)
    requires { not is_empty a.mdl }
    ensures { result.mdl = interval (min_elt a.mdl) size }
  = let ghost p = min_elt a.mdl in
     let ghost p_bv = of_int p in
        assert { eq_sub_bv a.bv zeros zeros p_bv };
        let res = bw_or a.bv (neg a.bv) in
        assert { eq_sub_bv res zeros zeros p_bv };
        assert { eq_sub_bv res ones p_bv (sub size_bv p_bv) };
        { bv = res;
          mdl = interval p size }
```

```
let maxUnion (a b : s) : s  (* operator [(a|b)&(a|-a)&(b|-b)] *)
   requires { not is_empty a.mdl /\ not is_empty b.mdl }
   ensures { forall x. mem x result.mdl <->
                        (mem x (union a.mdl b.mdl)
                     /\ x >= max (min_elt a.mdl) (min_elt b.mdl)) }
   ensures { forall x. mem x result.mdl <->
                     exists y z. mem y a.mdl
                                 /\ mem z b.mdl
                                 /\ x = max y z }
    = intersection (union a b)
                   (intersection (unionUminus a)
                                 (unionUminus b))
end
```

Theory "esterel.Esterel": fully verified in 3.92 s

Obligations	Alt-Ergo (1.20.prv)	CVC4 (1.4)	Z3 (4.3.1)
VC for union	0.11	Timeout (5s)	Out Of Memory (5M)
VC for intersection	0.11	Timeout (5s)	Out Of Memory (5M)
VC for aboveMin	---	---	---
split_goal_wp			
1. assertion	0.16	Timeout (5s)	Timeout (5s)
2. assertion	Timeout (5s)	0.19	Timeout (5s)
3. assertion	Timeout (5s)	0.39	Timeout (5s)
4. type invariant	0.30	Timeout (5s)	Timeout (5s)
5. postcondition	0.02	0.01	0.00
VC for maxUnion	---	---	---
split_goal_wp			
1. precondition	0.02	0.02	0.08
2. precondition	0.02	0.03	0.08
3. postcondition	0.63	0.10	Out Of Memory (5M)
4. postcondition	1.65	Timeout (5s)	Out Of Memory (5M)

Agenda

1. Bonus 1 : le synchroniseur Esterel en Why (SMT)
2. Bonus 2 : la machine chimique (rappel de 2009)
3. Bonus 3 : exemple de vérification avec CADP
4. Réponse aux questions

Calculs de processus : CCS (Meije, Lotos,..)

- Actions $A = \{a,b,...\}$, $A^- = \{a^-,b^-,\ldots\}$, τ (action invisible)

$$\alpha \in A \cup A^-,\ \ \alpha^- : a \leftrightarrow a^- ,\ \mu \in A \cup A^- \cup \{\tau\}$$

- Identificateurs de processus x, y, z
- Processus p, q, r,… définis par

0	inaction
μ.p	action
p | q	parallélisme
p + q	choix non-déterministe
p \ a	restriction
p[b/a]	renommage d'action
x	identificateur de processus
rec x = p	définition récursive

Sémantique Opérationnelle Structurelle (SOS – G. Plotkin)

$$\mu.p \xrightarrow{\mu} p$$

$$\frac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'} \qquad \frac{q \xrightarrow{\mu} q'}{p+q \xrightarrow{\mu} q'}$$

$$\frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q} \qquad \frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'}$$

$$\frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\alpha^-} q'}{p|q \xrightarrow{\tau} p'|q'}$$

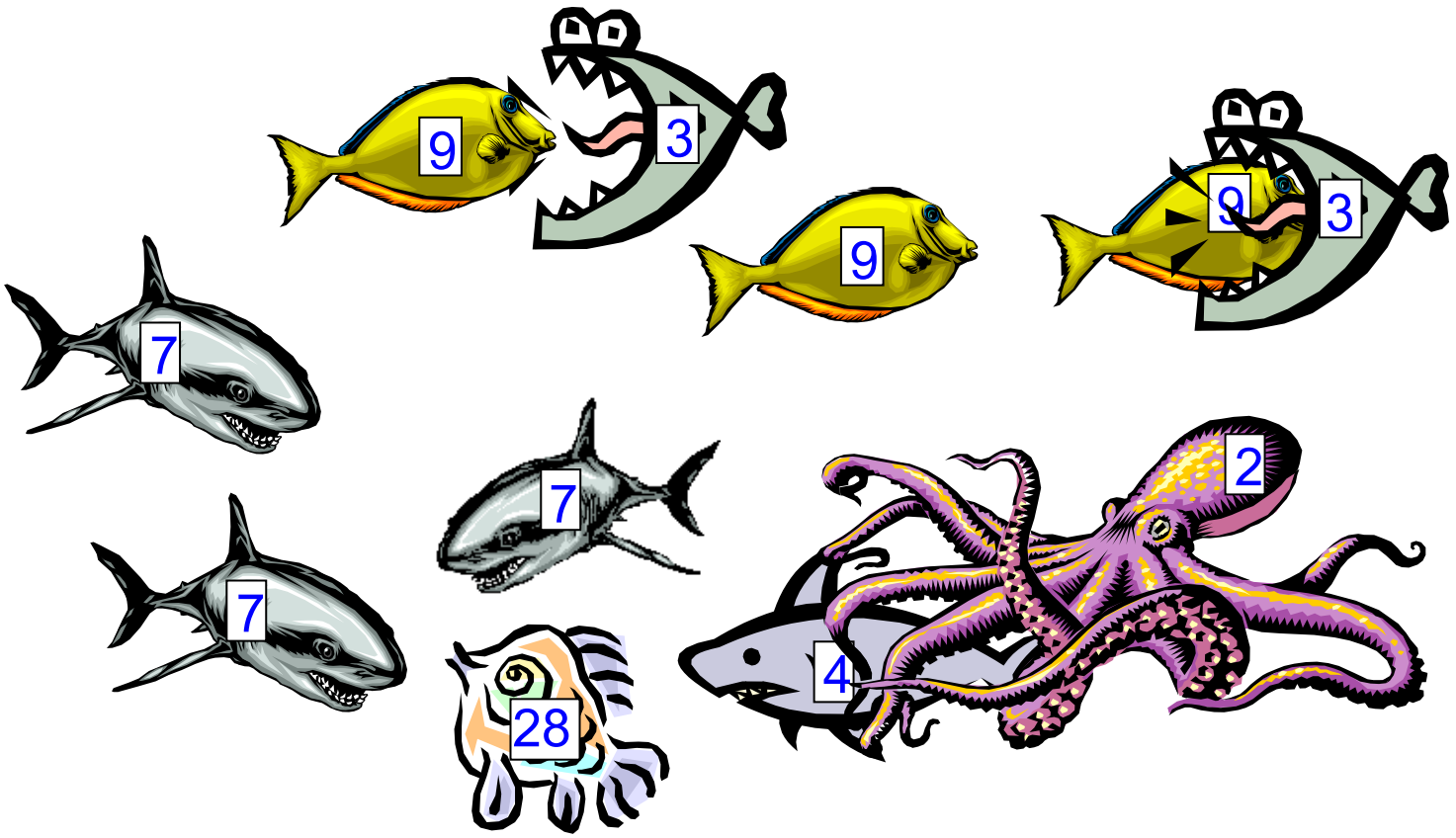
SOS: simple et précis, mais lourd

$$a.0 \mid (b.a^{-}.c.0 \mid b^{-}.0 \mid p)\backslash b \xrightarrow{\tau} a.0 \mid (a^{-}.c.0 \mid 0 \mid p)\backslash b \xrightarrow{\tau} 0 \mid (c.0 \mid 0 \mid p)\backslash b$$

$$\dfrac{\dfrac{\dfrac{\dfrac{b.a^{-}.c.0 \xrightarrow{b} a^{-}.c.0 \qquad b^{-}.0 \xrightarrow{b^{-}} 0}{b.a^{-}.c.0 \mid b^{-}.0 \xrightarrow{\tau} a^{-}.c.0 \mid 0}}{b.a^{-}.c.0 \mid b^{-}.0 \mid p \xrightarrow{\tau} a^{-}.c.0 \mid 0 \mid p}}{(b.a^{-}.c.0 \mid b^{-}.0 \mid p)\backslash b \xrightarrow{\tau} (a^{-}.c.0 \mid 0 \mid p)\backslash b}}{\dfrac{\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots}{\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots}}$$

$$\dfrac{a.0 \xrightarrow{a} 0 \qquad (a^{-}.c.0 \mid 0 \mid p)\backslash b \xrightarrow{a^{-}} (c.0 \mid 0 \mid p)\backslash b}{a.0 \mid (a^{-}.c.0 \mid 0 \mid p)\backslash b \xrightarrow{\tau} 0 \mid (c.0 \mid 0 \mid p)\backslash b}$$

Le crible de Darwin : p, kp → p

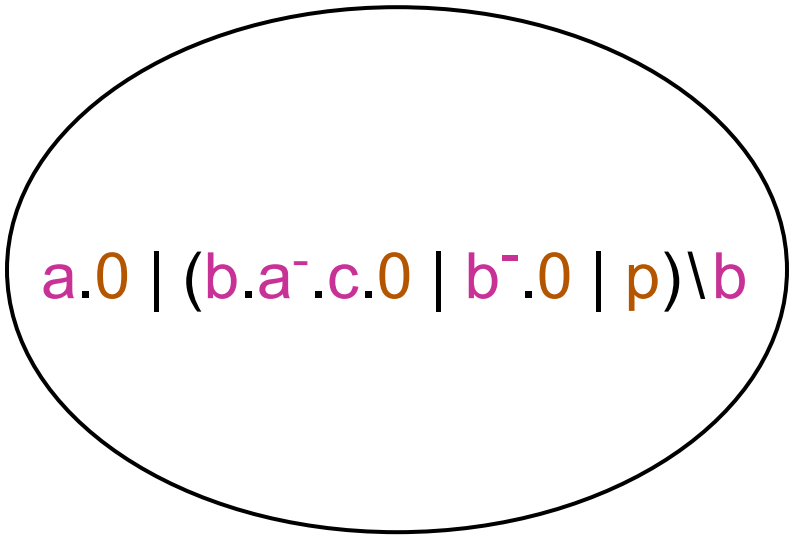


Banâtre - Le Métayer : GAMMA

Berry - Boudol : CHAM

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

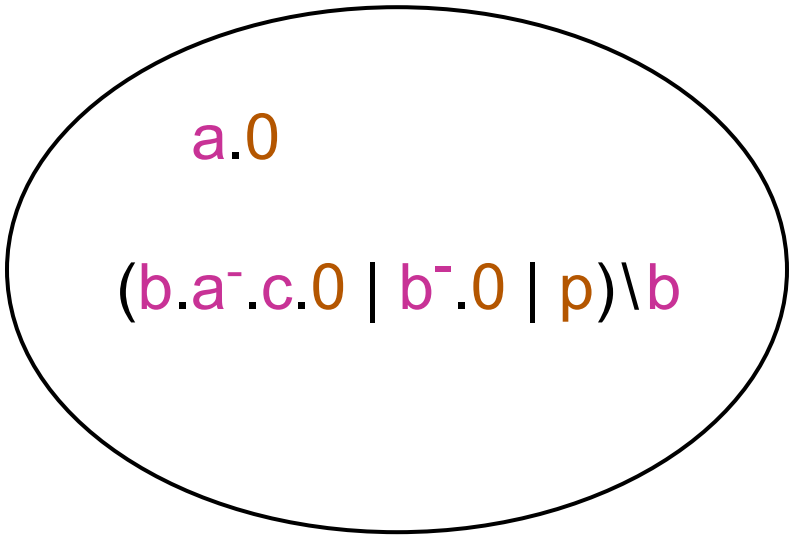


a.0 | (b.a^-.c.0 | b^-.0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution

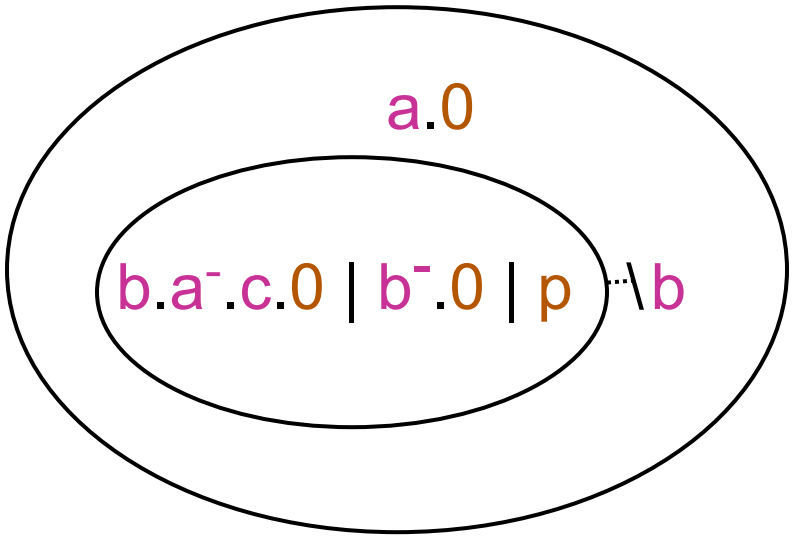


$a.0 \mid (b.a^-.c.0 \mid b^-.0 \mid p)\backslash b$

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane

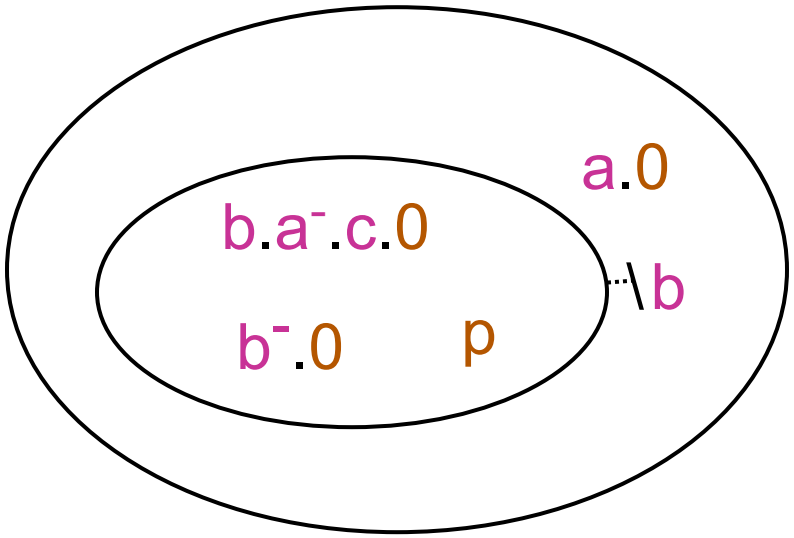


a.0 | (b.a^-.c.0 | b^-.0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution

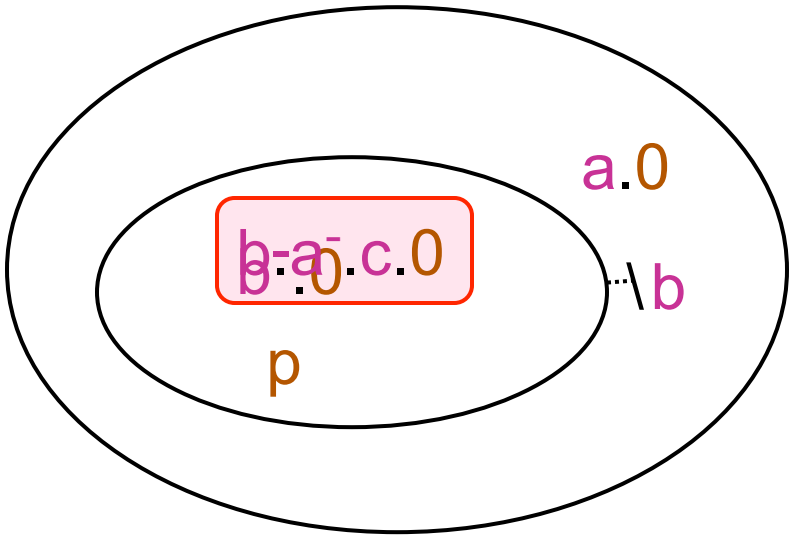


a.0 | (b.a^-.c.0 | b^-.0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre

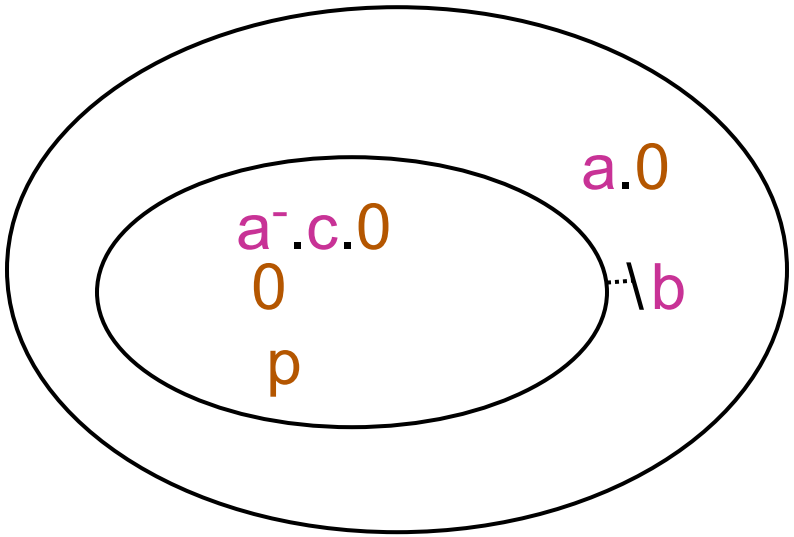


$a.0 \mid (b.a^-.c.0 \mid b^-.0 \mid p)\backslash b$

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre
réaction

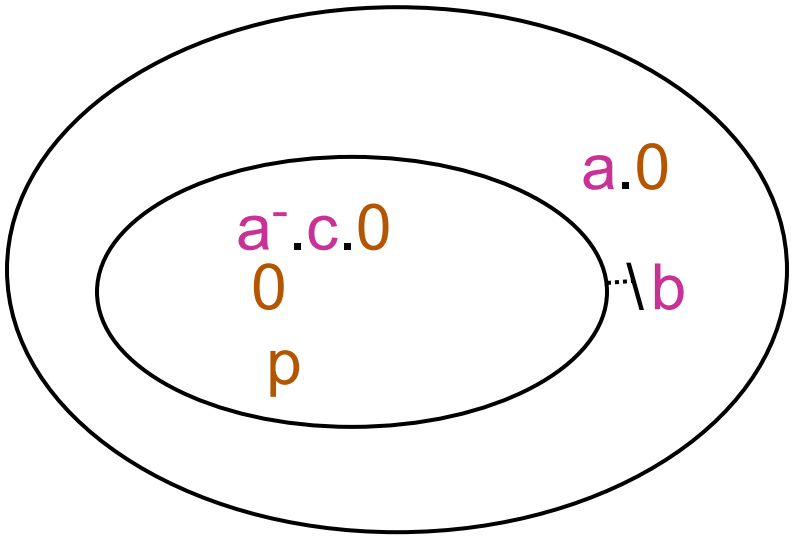


a.0 | (a^-.c.0 | 0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre
réaction
évaporation

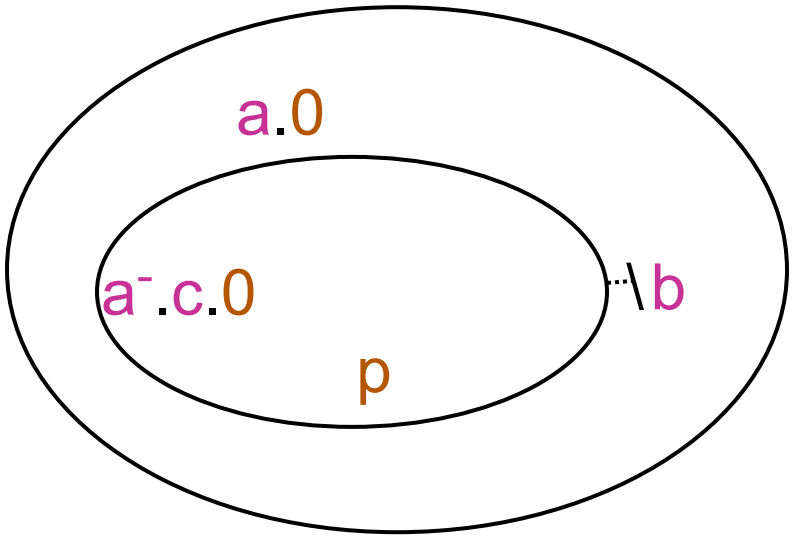


a.0 | (a^-.c.0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre
réaction
évaporation

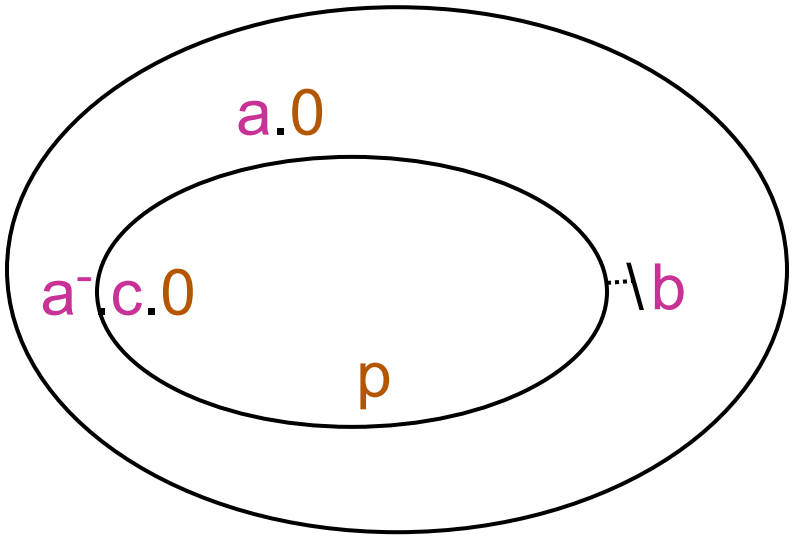


a.0 | (a^-.c.0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre
réaction
évaporation



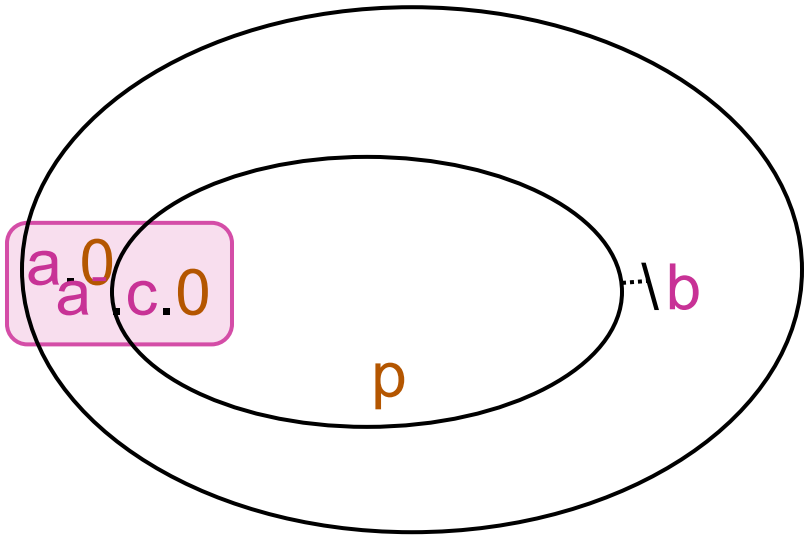
extrusion

a.0 | (a^-.c.0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre
réaction
évaporation



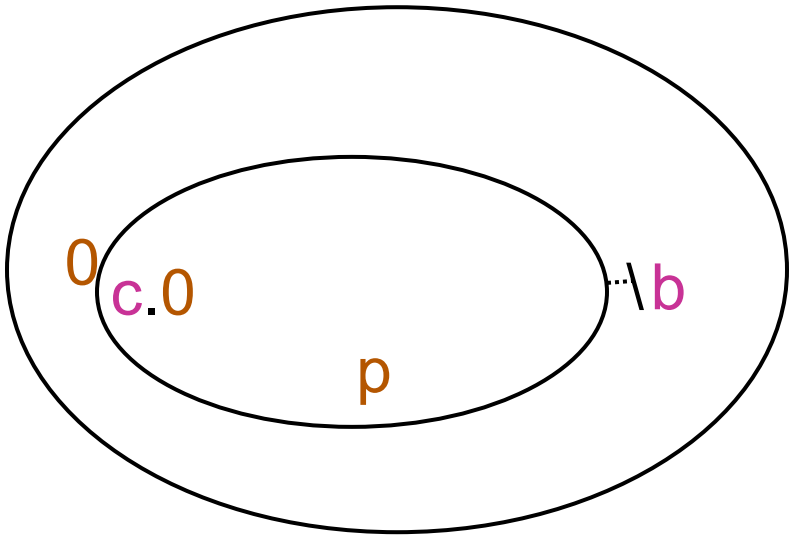
extrusion
rencontre

$a.0 \mid (a^-.c.0 \mid p)\backslash b$

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre
réaction
évaporation



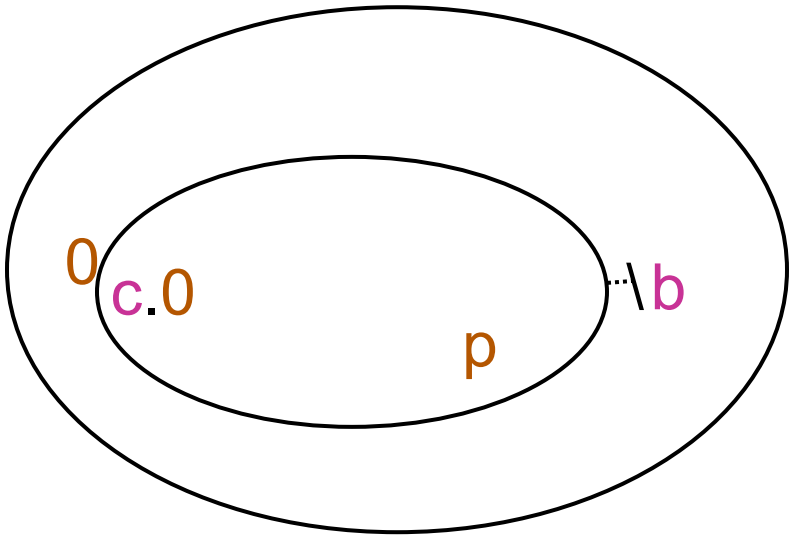
extrusion
rencontre
réaction

0 | (c.0 | p)\b

La machine chimique

- Idées centrales :
  - objets = molécules flottant dans une soupe chimique
  - a, a^- = valences des molécules
  - hiérarchisation => membranes perméables aux valences

dissolution
membrane
dissolution
rencontre
réaction
évaporation



extrusion
rencontre
réaction
évaporation

(c.0 | p)\b

Références

- G. Berry et G. Boudol

  The Chemical Abstract Machine

  Theoretical Computer Science, vol. 96 (1992) 217-248

- C. Fournet et G. Gonthier

  The Join-Calculus, a Language for Distributed Mobile Programming

  Applied Semantics, Springer-Verlag LNCS 2395, pp. 268-332

- Voir aussi le langage JoCaml, http://jocaml.inria.fr

  The Join-Calculus, a Language for Distributed Mobile Programming

Agenda

1. Bonus 1 : le synchroniseur Esterel en Why (SMT)
2. Bonus 2 : la machine chimique (rappel de 2009)
3. Bonus 3 : exemple de vérification avec CADP
4. Réponse aux questions

Exemple de vérification avec CADP

- Un système de perçage, qui sert d'exemple commun à beaucoup de vérifieurs
- Une utilisation des expressions régulières non présentée dans les cours précédents

Stephan Merz, Nicolas Navet et Radu Mateescu
Specification and Analysis of Asynchronous Systems using CADP
Dans « Modeling and Verification of Asynchronous Systems using CADP » Wiley, 2010
http://onlinelibrary.wiley.com/doi/10.1002/9780470611012.ch5/summary

Beaucoup d'autres exemples CADP à l'adresse
http://cadp.inria.fr/case-studies/

Le système mécanique

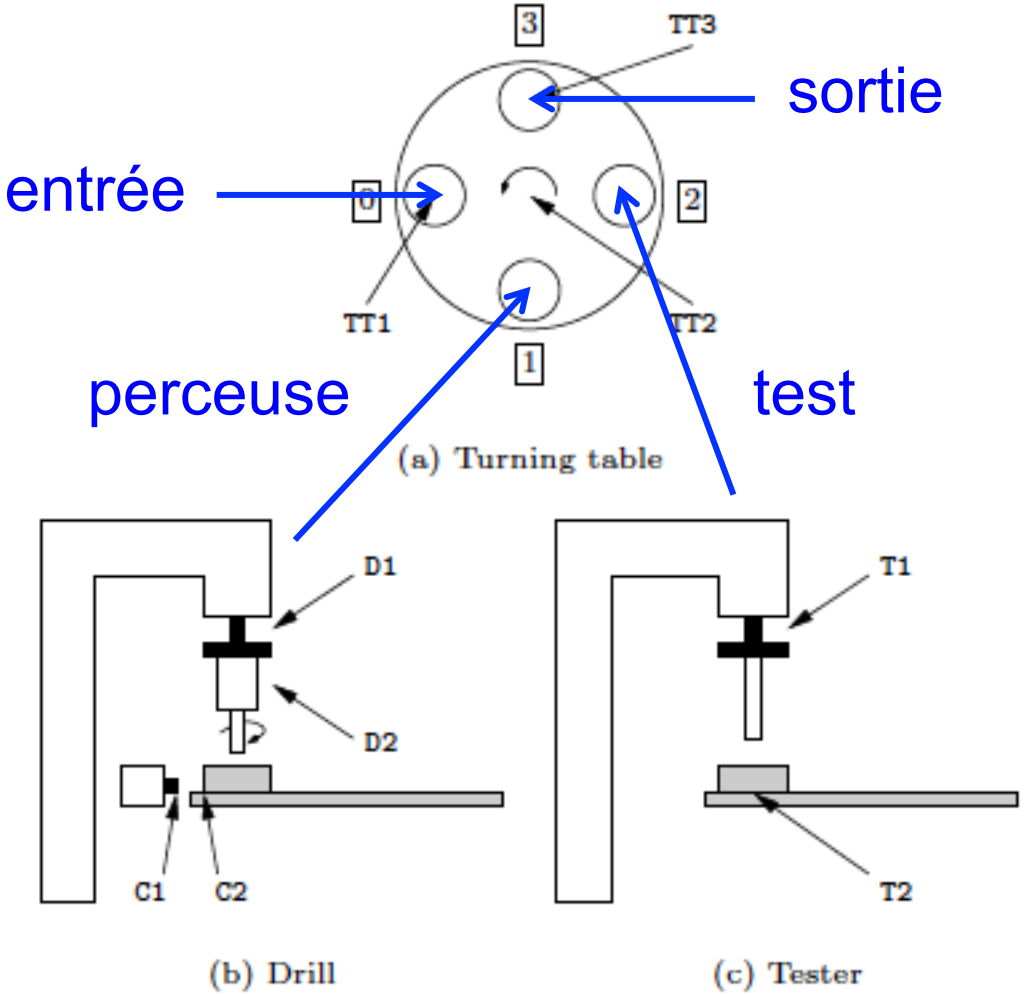


Fig. 1. Drilling unit for metallic products

L'interface physique

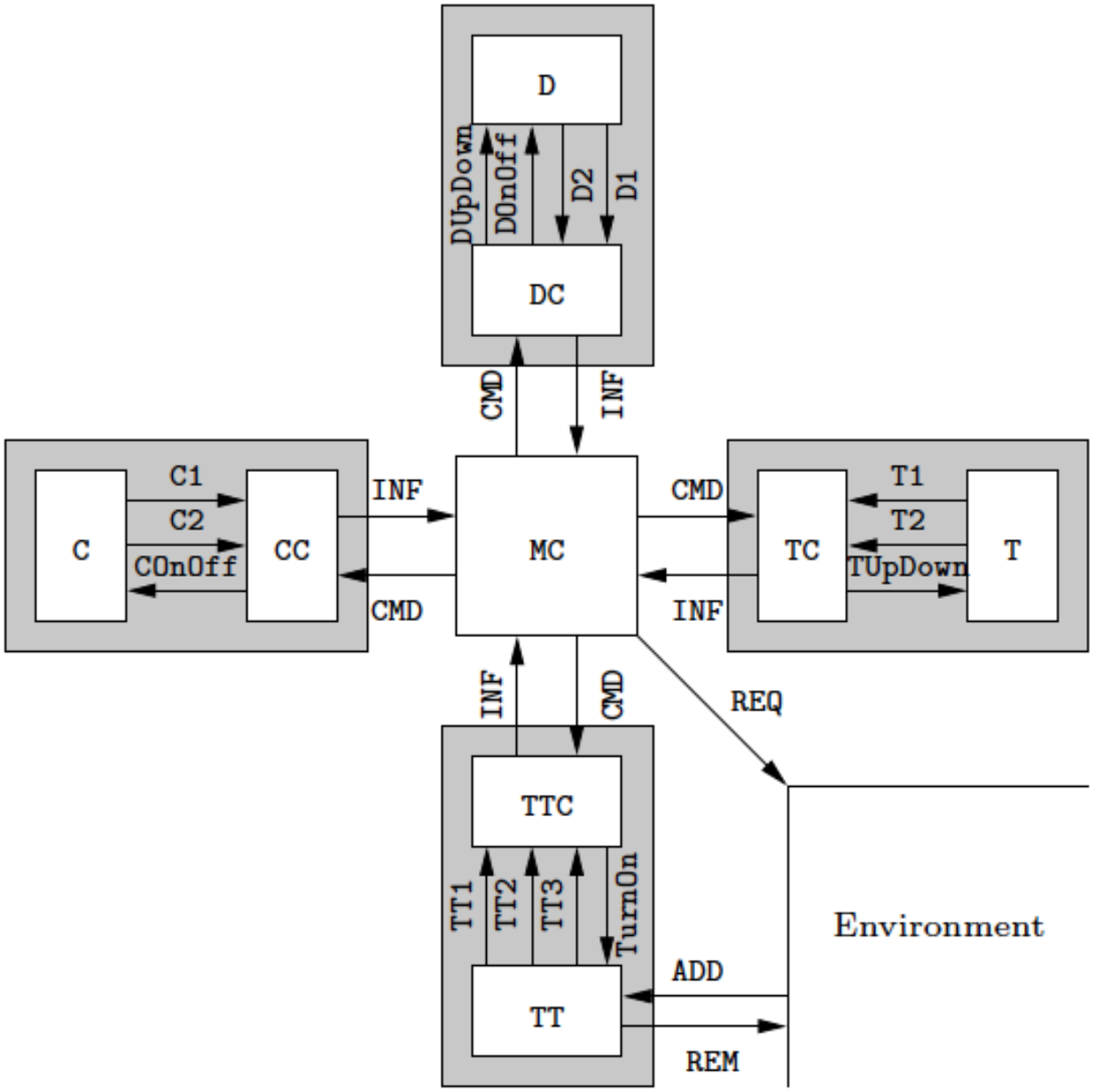
Gates for sending commands to actuators		
Device	Gate	Function
Table	`TurnOn`	Starts a 90 ° rotation
Clamp	`COnOff`	Blocks or releases the clamp
Drill	`DOnOff`	Starts or stops the engine of the drill
	`DUpDown`	Starts the ascending or descending movement
Tester	`TUpDown`	Starts the ascending or descending movement

Gates for receiving signals from sensors		
Device	Gate	Function
Table	`TT1`	Product present at position 0
	`TT2`	Rotation of 90 ° completed
	`TT3`	Product absent at position 3
Clamp	`C1`	Clamp released
	`C2`	Clamp blocked
Drill	`D1`	Drill in upper position
	`D2`	Drill in lower position
Tester	`T1`	Tester in upper position
	`T2`	Tester in lower position

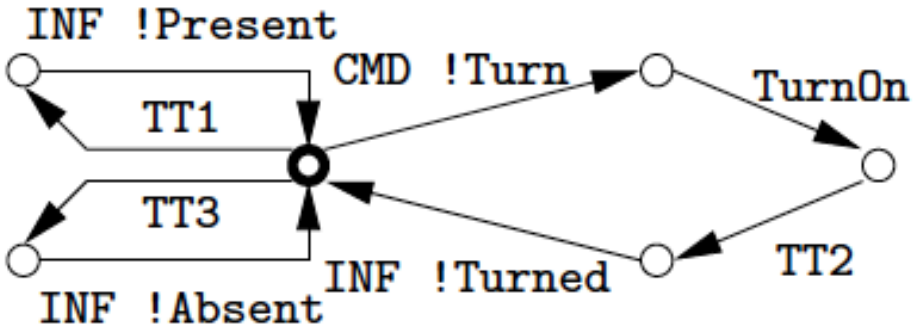
L'interface avec le contrôleur

Gate	Signal	Meaning
CMD	Turn	Rotation of 90 ° of the table
	Drill	Drilling of the product at position 1
	Lock	Blocking of the clamp
	Unlock	Release of the clamp
	Test	Test of the product at position 2
INF	Turned	Rotation of 90 ° of the table completed
	Present	Product present at position 0
	Drilled	Drilling of the product at position 1 completed
	Locked	Clamp blocked
	Unlocked	Clamp released
	Tested	Product at position 2 tested
	Absent	Product absent at position 3
REQ	Add	Input of a product at position 0
	Remove	Output of a product at position 3

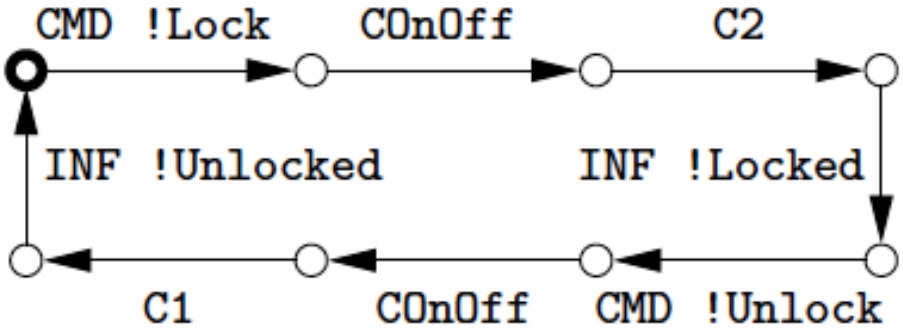
Architecture du programme LOTOS



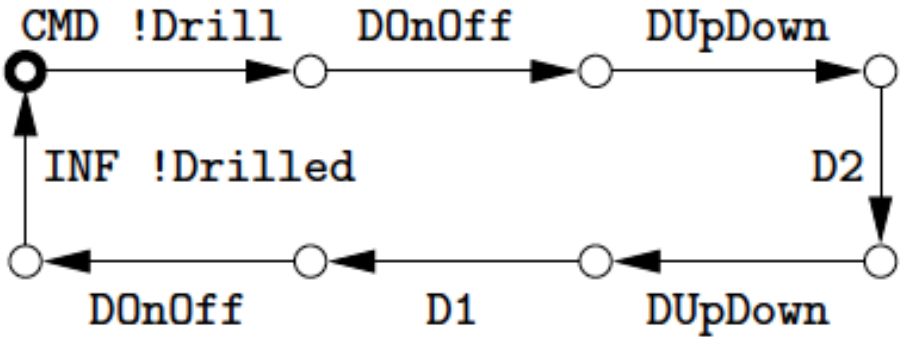
Contrôleurs locaux



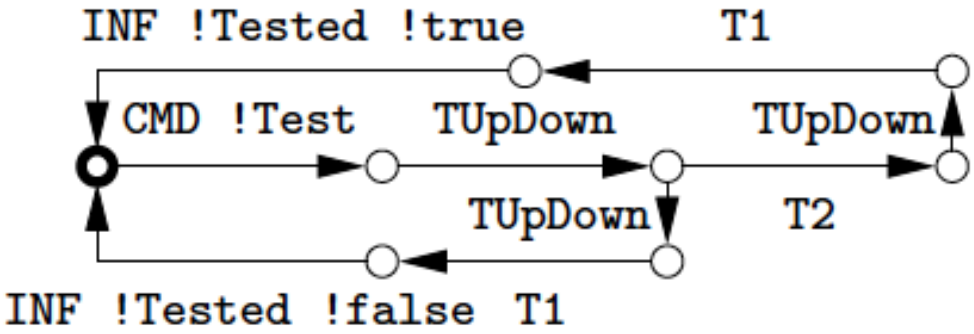
(process TTC)



(process CC)

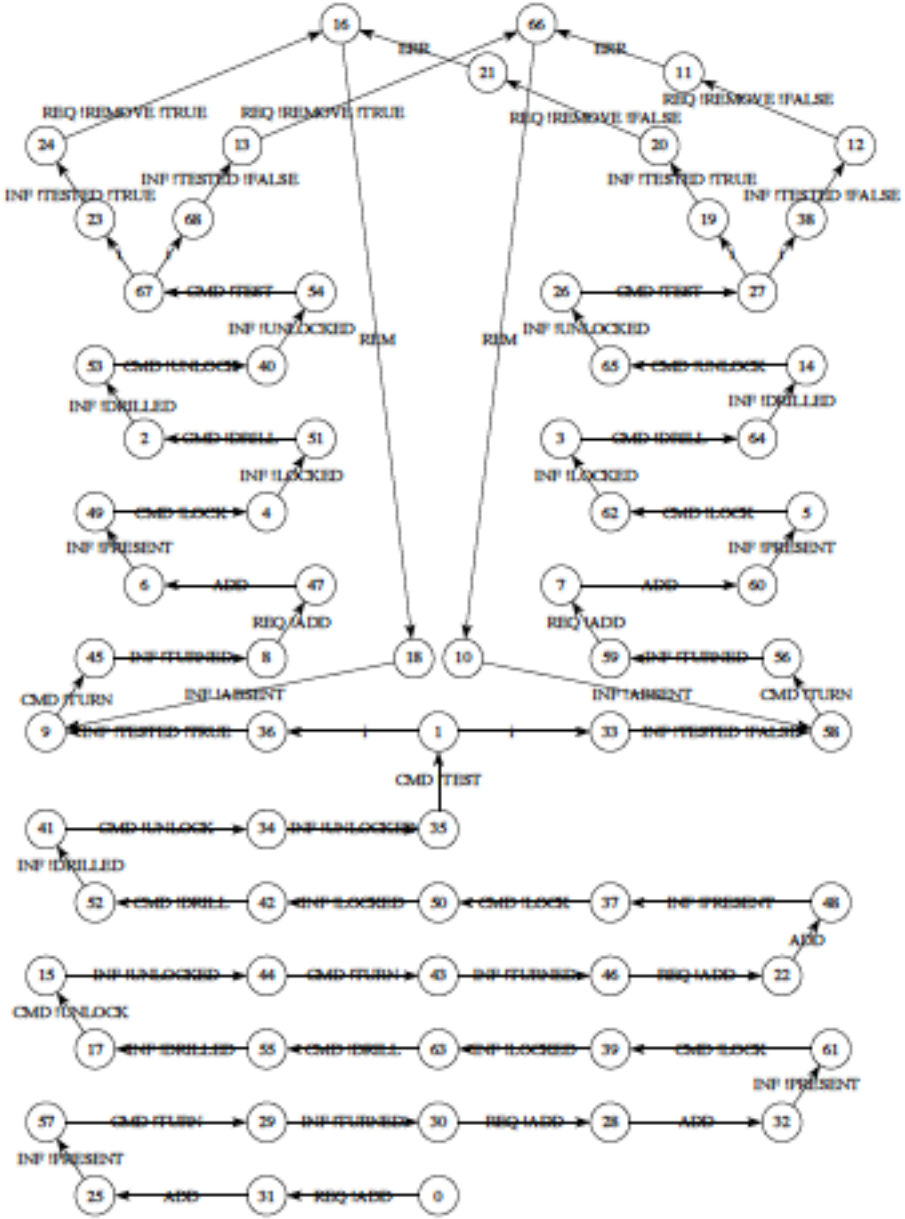


(process DC)



(process TC)

Contrôleur global purement séquentiel



Le contrôleur parallèle n'est pas dessinable !

Sûreté par expressions régulières

No.	Formula	Description
P_1	[true* . "INF !PRESENT" . (not "INF !TURNED")* . "INF !TURNED" . (not "INF !LOCKED")* . "CMD !DRILL"] false	After the input of a product and a rotation of the table, the main controller cannot command a drilling before the clamp has been blocked.
P_2	[true* . "INF !DRILLED" . (not 'INF !UNLOCKED.*')* . "CMD !TURN"] false	After the drilling of a product, the main controller cannot command a rotation before the clamp has been released.
P_3	[true* . "INF !UNLOCKED" . (not "INF !TURNED")* . "INF !TURNED" . (not 'INF !TESTED.*')* . "CMD !TURN"] false	After the release of the clamp on a product and a rotation of the table, the main controller cannot command another rotation before the product has been tested.
P_4	[true* . 'INF !TESTED.*' . (not "INF !TURNED")* . "INF !TURNED" . (not "INF !ABSENT")* . "CMD !TURN"] false	After the test of a product and a rotation of the table, the main controller cannot command another rotation before the product has been removed.

Sûreté par expressions régulières

P_5	[true* . "INF !ABSENT" . (not "INF !TURNED")*. "INF !TURNED'' . (not "INF !PRESENT")*. "CMD !TURN"] false	After a product remove and a rotation of the table, the main controller cannot command another rotation before a new product has been supplied.
P_6	[true* . "INF !TESTED !TRUE" . (not "INF !TURNED")* . "INF !TURNED" . (not "INF !TURNED")* . "ERR"] false	Every time the tester detects a correctly drilled product, no error will be signaled during the next processing cycle.
P_7	[true* . "INF !PRESENT" . true* . "INF !PRESENT" . true* . "INF !PRESENT" . true* . "CMD !TEST" . (not "INF !TURNED")* . "CMD !DRILL"] false	After the testing and drilling positions of the table have been occupied, the main controller cannot command a test before commanding a drill.

Vivacité par expressions régulières

No.	Formula	Description
P_8	inev (not "CMD !TURN", "REQ !ADD", inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", "REQ !ADD", inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", "REQ !ADD", inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", "REQ !ADD", true)))))))	Initially, the main controller eventually commands the insertion of products in all the slots of the turning table.
P_9	[true* . "INF !PRESENT"] inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", "CMD !LOCK", inev (not "CMD !TURN", "CMD !DRILL", inev (not "CMD !TURN", "CMD !UNLOCK", true))))	Each product inserted will be drilled after the next rotation of the table.
P_{10}	[true* . "INF !UNLOCKED"] inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", "CMD !TEST", true))	Each product drilled will be tested after the next rotation of the table.
P_{11}	[true* . 'INF !TESTED.*'] inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", 'REQ !REMOVE.*', true))	Each product tested will be removed after the next rotation of the table.

Vivacité par expressions régulières

P_{12}	[true* . "INF !ABSENT"] inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", "REQ !ADD", true))	Each product removal will be followed by the insertion of a new product after the next rotation of the table.
P_{13}	[true*] (["REQ !ADD"] inev (not "INF !TURNED", "INF !PRESENT", true) and ["CMD !LOCK"] inev (not "INF !TURNED", "INF !LOCKED", true) and ["CMD !DRILL"] inev (not "INF !TURNED", "INF !DRILLED", true) and ["CMD !UNLOCK"] inev (not "INF !TURNED", "INF !UNLOCKED", true) and ["CMD !TEST"] inev (not "INF !TURNED", 'INF !TESTED.*', true) and ['REQ !REMOVE.*'] inev (not "INF !TURNED", "INF !ABSENT", true) and ["CMD !TURN"] inev (not "INF !TURNED", "INF !TURNED", true))	Each command (resp. request) sent by the main controller to the physical devices (resp. to the environment) will be eventually followed by its acknowledgement before the next rotation of the table.
P_{14}	[true* . "INF !TESTED !FALSE"] inev (not "CMD !TURN", "CMD !TURN", inev (not "CMD !TURN", "ERR", true))	Every time the tester detects an incorrectly drilled product, an error will be eventually signaled during the next processing cycle.

Agenda

1. Bonus 1 : le synchroniseur Esterel en Why (SMT)
2. Bonus 2 : la machine chimique (rappel de 2009)
3. Bonus 3 : exemple de vérification avec CADP
4. Réponse aux questions

Question 1

Beaucoup de progrès en vérification, mais les systèmes très asynchrones communicant par de très grands canaux (télécom par fibres optiques transocéaniques par exemple) posent des défis sérieux à la vérification. Les progrès qui s'appliquent par exemple aux circuits ne semblent pas s'appliquer ici.

Peut-on concevoir des systèmes grandement distribués et sûrs, ou aussi des systèmes « tolérants les fautes », sortes d'hybrides entre preuves et sûreté de fonctionnement ?

Question 1 − Eléments de réponse

- La taille des buffers est un problème pour le model-checking, mais pas pour les systèmes qui savent faire de la récurrence : TLA+, Isabelle, Coq, etc.
- L'hybridation mentionnée est en fait déjà à l'œuvre. Sur Internet, on utilise en grand la tolérance aux fautes avec des algos probabilistes pour le routage, le pair-à-pair, etc., et la preuve pour la sécurité par exemple. Mais les acteurs ne se parlent pas encore beaucoup, il est vrai.
- A noter que les techniques de vérification pour la correction et la tolérance aux fautes ne sont pas forcément différentes, cf UPPAAL statistique, le travail de Patricia Bouyer sur la vérification de robustesse dans les automates temporisés, ou le travail de Marta Kiatowska sur la vérification probabiliste dont je n'ai pas parlé.

Question 1 − Eléments de réponse

- De toutes façons, pour faire des systèmes qui marchent, il faut arrêter de pisser du code, mais travailler à partir de composants individuellement vérifiés
- Et mettre l'attention sur la preuve des parties vraiment dures, celles qu'il faut vérifier formellement

  (cf cours 5 du 212 mars014, base de donnée temporelle distribuée et répliquée Spanner de Google)

Question 2

Sur les protocoles : il y a de nombreuse initiatives pour de nouveaux standards. Comment faire pour y exploiter les avancées théoriques présentées dans le cours ?

Question 2 − Eléments de réponse

- Le problème n'est pas spécifique aux protocoles, et ce qui a été présenté n'est pas que théorique :
  - Le model-checking est en fait né des besoins de vérification en protocoles de télécommunication. Il est tout à fait utilisé avant, pendant, ou après leur normalisation (SPIN a été fait aux Bell Labs, donc chez AT&T)
  - Les protocoles de sécurité sont de plus en plus vérifiés (ou attaqués) formellement
  - En avionique, la norme DO-178C intègre la vérification formelle comme outil important de vérification
  - La vérification indépendante par l'IRSN des logiciels de sécurité des centrales nucléaires utilise plusieurs vérifieurs formels sophistiqués
  - idem en signalisation ferroviaire, etc.
  - Donc la perméabilité s'améliore doucement....

Question 3

Sur les optimisations de circuits, peut-on capitaliser sur les opérations déjà effectuées, par exemple quand on retrouve des sous-expressions logiques vues ailleurs?

Peut-on enregistrer les optimisations déjà réalisées dans des formats adaptés ?

Question 3 − Eléments de réponse

- En rège générale, les optimisations se font sur des modules et pas sur le système complet. Quand elles sont faites, on peut effectivement stocker l'optimisé sous le même format que le source (ex. le langage Verilog)
- Le plus souvent, l'optimisation d'une sous-expression dépend très fortement de son contexte. Donc, descendre plus bas que le module n'apporterait pas beaucoup, car c'est le reste du module qui constitue le gros du contexte (plus des hypothèses sur les entrées)
- Cf séminaire Madre-Vuillod le 9 mars, optimisation par don't cares

Question 4 :

1. Sur les langages (cf. cours du 4 novembre), peut-on imaginer une synthèse des multiples facettes sous une syntaxe et une sémantique unifiée?
2. Un méta-langage est-il envisageable ?
3. Une bibliothèque de module certifiés est-elle envisageable ?

Question 4.1 − Eléments de réponse

1. Hmmm, ca ne semble pas être vraiment possible : beaucoup de cultures différentes, de points de vues cohérents et justifiés mais distincts et peu compatibles, des besoins en constant changement (ex. Web → Javascript, science → Python, etc.), et une recherche encore foisonnante qui n'a aucune envie d'être gelée.

   Mais des tentatives d'unification partielles pertinentes existent. Un bon exemple ambitieux est SCALA de Martin Odersky, langage fonctionnel, objet, parallèle, propre, et qui a pas mal de succès dans l'industrie

Question 4.2 − Eléments de réponse

2. Méta-langage : techniquement, cette expression signifie plutôt « langage parlant des langages ». En ce sens, nous avons plusieurs formalismes adaptés, par exemple SOS, signle pour Stuctural Operational Semantics utilisée dans le transparent 16 pour CCS et dans les cours pour Esterel

Mais pas encore grand-chose à espérer pour l'analogue du langage mathématique, qui est lui vraiment universel au moins dans son cœur de base....

Question 4.3 − Eléments de réponse

3. Bibliothèques de modules certifiés : c'est une excellente façon de faire, déjà à l'œuvre dans plusieurs domaines :
   - les calculs mathématiques, avec de nombreuses librairies
   - les SoCs (Systems on Chips), systématiquement construits par assemblages de modules très vérifiés
   - les calculs géométriques, cf. la librairie CGAL du projet Inria GEOMETRICA, dirigé par Jean-Daniel Boissonnat, le prochain processeur sur la chaire Inria / Collège de France « Informatique et sciences numériques »
   - et bien d'autres domaines je pense...

SCOOP !

Question 5

Des problèmes sur des systèmes embarqués critiques (Apollo IV) ou des incompréhensions entre participants à des congrès ont été dus à des problèmes d'unités hétérogènes et de notation des nombres non normalisées (virgules, points, etc.). Y peut-on quelque chose ?

Question 5 – Eléments de réponse

C'est tristement exact !

J'ai déjà mentionné Mars Climate Orbiter, détruit par une correction de trajectoire envoyée en pieds et pouces au lieu de mètres

Voir aussi l'extraordinaire cas du Boeing Gimli Glider, tombé en panne d'essence au milieu du Canada à 10 000m à cause d'une mauvaise conversion litres / gallons !

Il y a malheureusement peu de travail sur l'intégration des dimensions et unités dans les langages de programmation principaux.

Mais voir un travail intéressant d'Eric Goubault au CEA, et surtout le vrai type-checking physique dans F# chez Microsoft (aussi dans le langage Nemerle, que ne je connais pas... et des bibliothèques pour Java et Python par exemple).