Prouver des Programmes, Tester des Systèmes, Vérifier des Modèles…

Des outils et méthodes pour la sûreté de fonctionnement des systèmes informatiques

Marie-Claude Gaudel, Université de Paris-Sud, Orsay
Laboratoire de Recherche en Informatique
mcg@lri.fr

Plan

- *Quelques exemples de problèmes de logiciel très médiatisés*
- *Spécificités du logiciel*
- Programmes, Modèles, et Systèmes
- Méthodes statiques, indécidabilités
  - Analyses statiques ~~et preuves de programmes~~
- Méthodes dynamiques, incomplétude… et indécidabilités
  - Test de systèmes, couvertures de test, test aléatoire
- ~~Vérification de modèles: « model-checking »~~
- *Quelques histoires heureuses et peu médiatisées*

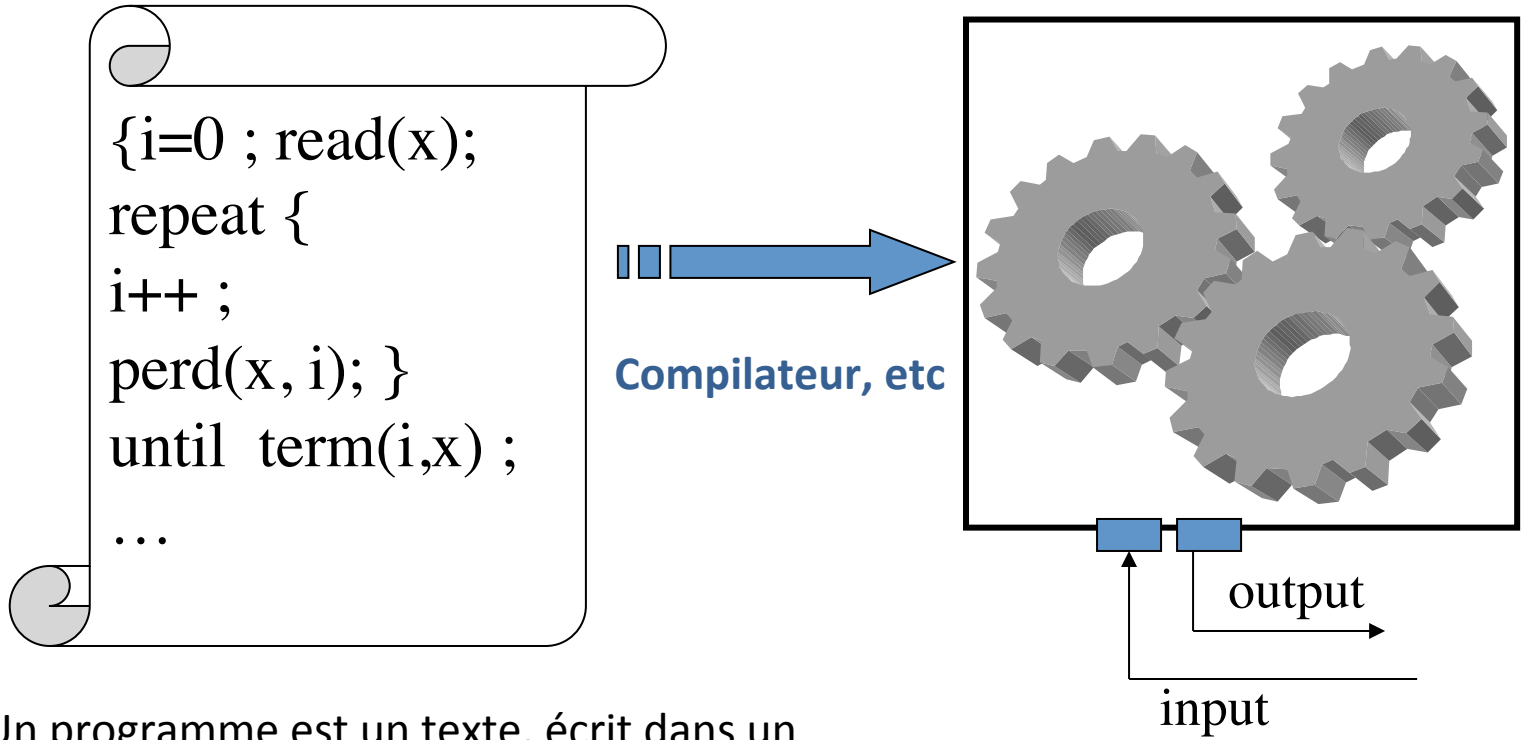
Quelques histoires malheureuses et très médiatisées

- Internet en est rempli… certaines n'ont rien à voir avec le logiciel!
- Quelques exemples phare (et authentiques) :
  - Therac-25, radiothérapie (1985-1987)
  - Premier vol d'Ariane 5 (1996)
  - Diviseur Pentium d'Intel (1994)
    - (c'était du hardware, mais le pb était algorithmique…)
  - Plantage de Gmail en février 2009
- Et bien sûr toutes vos contrariétés quand vous utilisez vos PC, téléphones, tablettes…

Pourquoi ?

- Tout objet manufacturé complexe doit être analysé et testé avant d'être mis en service.
- C'est le cas des systèmes informatiques à base de logiciel.
- Mais *spécificités du logiciel* :
  - Exclusivement des fautes de conception (pas de fautes de « fabrication », ou presque)
  - Pas de bons modèles de fautes (ou plutôt, beaucoup trop!)
  - TRÈS mauvaise visibilité, car ce qu'on teste est un *système où le logiciel est enfoui*…

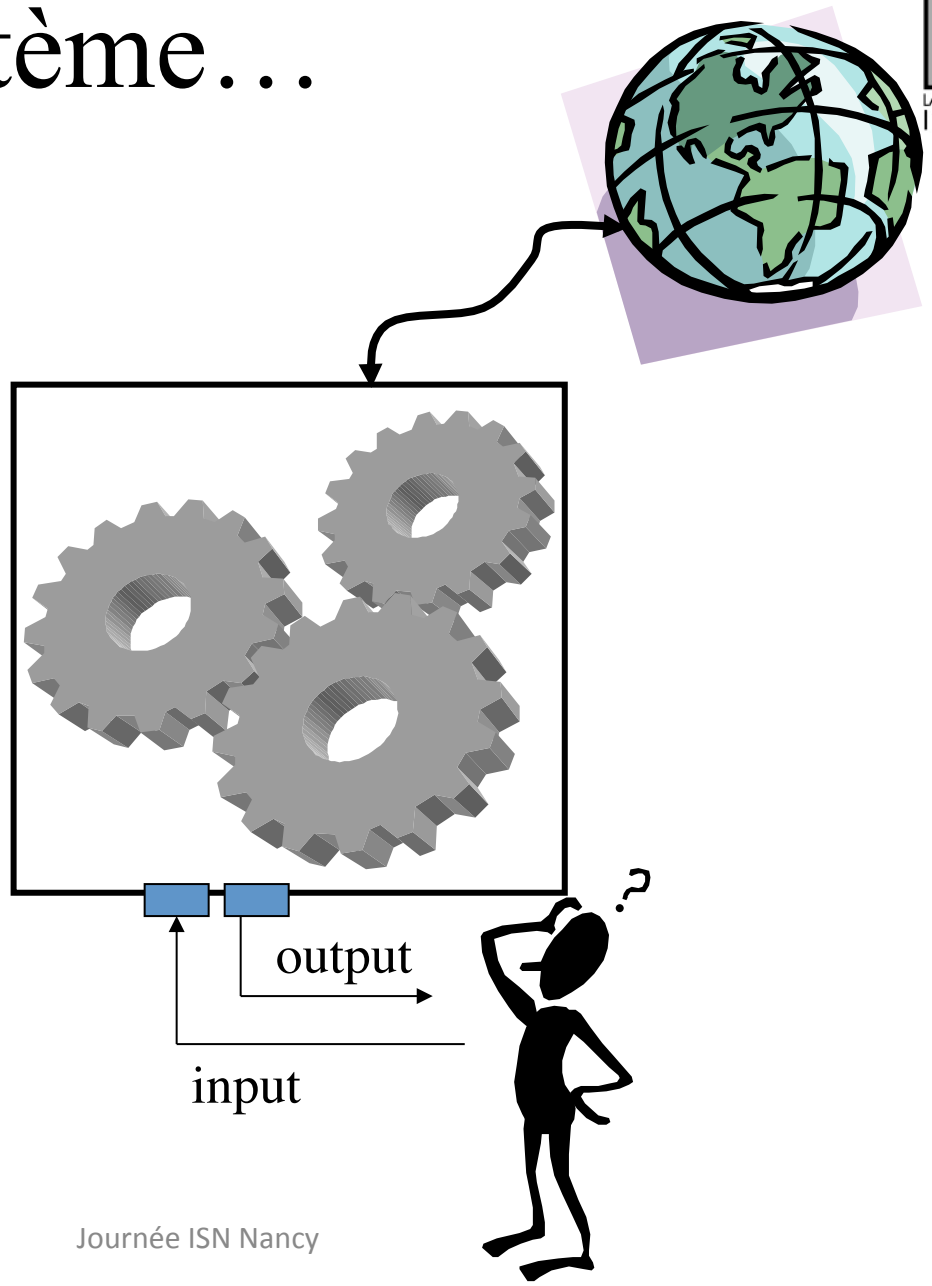
Programme/Système



Un programme est un texte, écrit dans un langage bien défini, avec syntaxe, sémantique, compilateurs et autres.

Système…

Un système est une entité dynamique qui fait partie du monde physique.
Il est observable selon une interface ou des procédures limitées.
Il n'est pas toujours complétement contrôlable.



Prouver ou tester quoi?
Il faut un *modèle* de ce qu'on veut!

- Un terme TRÈS utilisé!
  - Pour un physicien, cela peut être une équation différentielle,
  - pour un biologiste des souris ou des grenouilles
- En informatique … cela dépend
  - UML, bien sûr, qui fournit tout un arsenal
  - Pour les programmes classiques *: préconditions et postconditions*
  - Pour les systèmes réactifs et les protocoles :
    - Automates, Machines à états finis, Systèmes de transitions, réseaux de Petri, etc

Un premier type de modèle: *Précondition, Postcondition*

- *Assertion*: formule logique qui doit être satisfaite à un certain point du programme par les variables du programmes:
  - Exemple : $x > 0 \wedge i \leq N$
- *Précondition*: assertion sur l'état initial du programme
- *Postcondition*: assertion sur l'état final, si le programme termine …
- *Invariant de boucle*: assertion qui est vraie avant l'entrée dans la boucle et qui reste vraie après chaque passage dans la boucle.

Exemple: pgcd

- On connaît 3 algorithmes (diviseurs, soustractions, Euclide)
- Ils satisfont tous le couple <précondition, postcondition>:

$< n>0 \wedge m>0 ,$

$d|n \wedge d|m \wedge ((q|n \wedge q|m) => q \leq d) >$

(*d* est un diviseur de *n* et de *m*, et c'est le plus grand ; où *n* et *m* sont les données positives; *p* est le résultat)

Autre exemple: racine carrée

- $x \geq 0$
- $|(rac \times rac) - x| < \varepsilon$

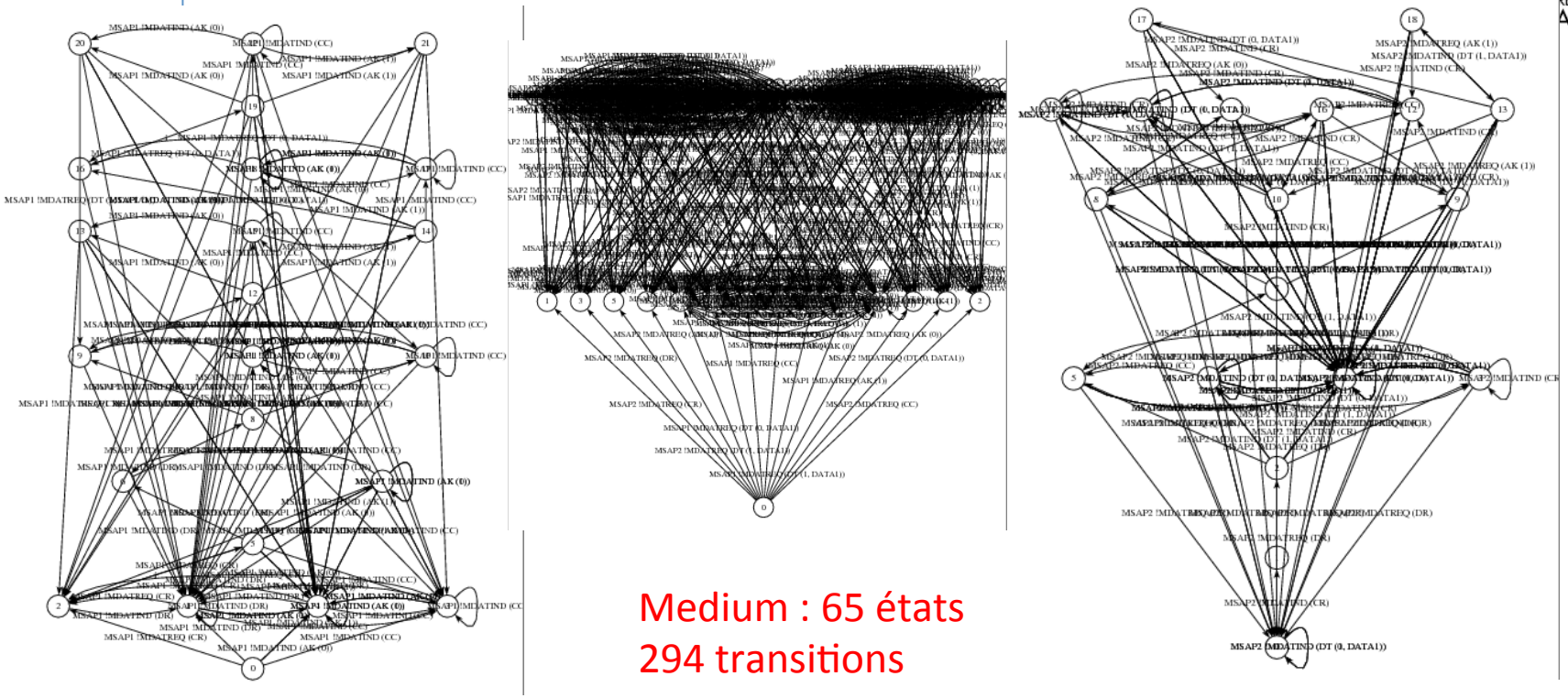
- Notation (triplet de Hoare)

$$\{Pre\}\ Prog\ \{Post\}$$

- Logique de Hoare : axiome et règles de raisonnement sur les triplets.
  - Par exemple :

$$\{P[E/x]\}\ x := E\ \{P\} \qquad \frac{\{P\}S\{Q\}, \{Q\}T\{R\}}{\{P\}S;T\{R\}}$$

Un autre type de modèle…



Medium : 65 états
294 transitions

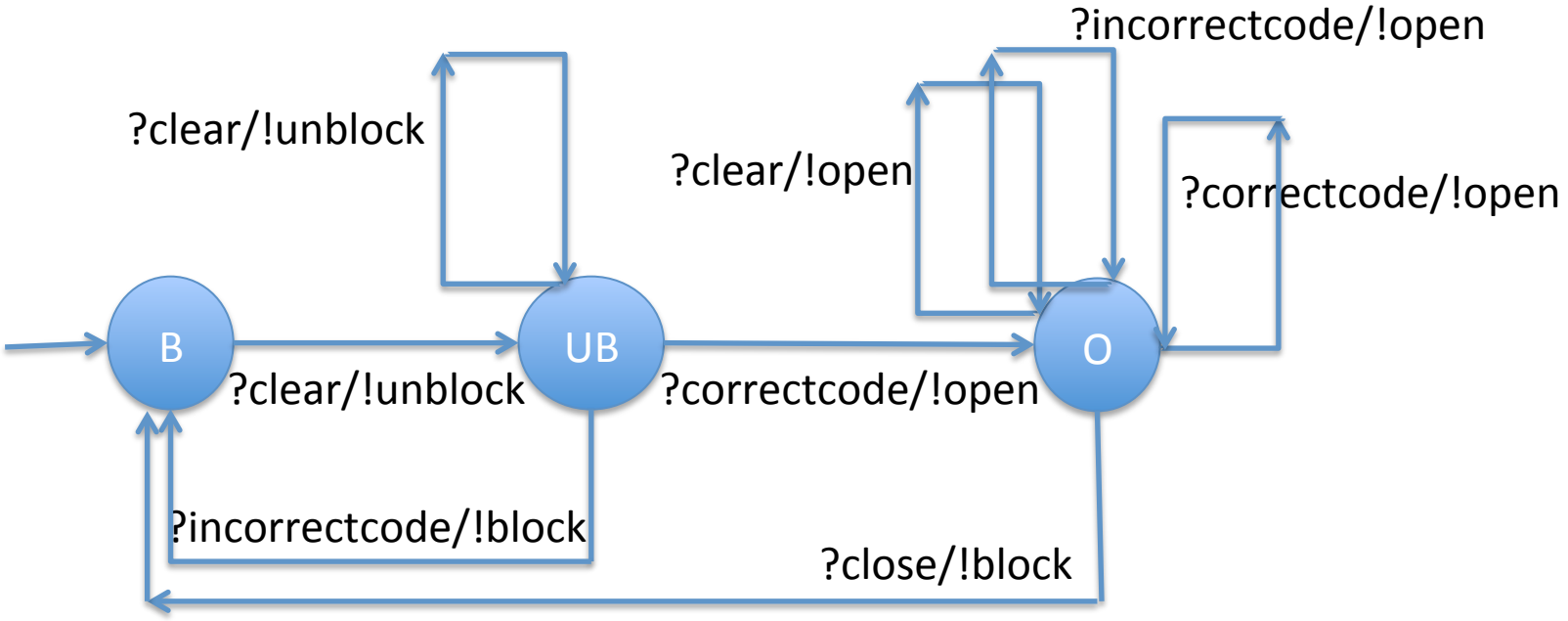
Initiateur: 34 états, 115 transitions

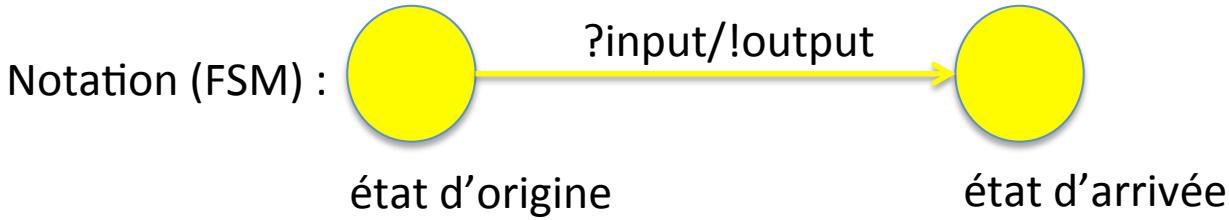
Receveur : 26 états, 83 transitions

Le protocole INRES permet d'assurer un service fiable de transfert de données d'un site informatique vers un autre.
Le protocole opère via un medium de connexion qui, lui, n'est pas fiable: des paquets peuvent être perdus.
Les modèles ci-dessus correspondent respectivement au logiciel du site de départ, au medium (dont les défaillances potentielles sont décrites) et au logiciel du site d'arrivée. Ces modèles se synchronisent 2 à 2 : certaines actions (transitions) du medium se font toujours avec une action du receveur, et c'est aussi le cas de certaines actions de l'initiateur. De ce fait, le système global comporte 981 états et 2552 transitions. En raison de sa taille raisonnable et de sa relative complexité cet exemple a été souvent utilisé, comme première expérience, dans les recherches sur la vérification de protocoles.

Image réalisée avec le logiciel CADP

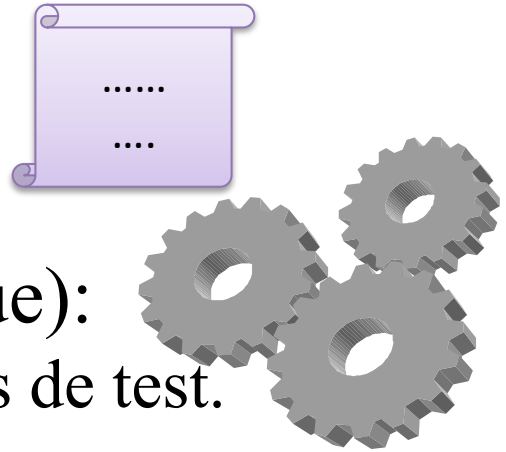
Un autre exemple + simple : ☺ modèle d'un digicode





Méthodes statiques ou dynamiques ?

- *Analyse Statique* de programmes: méthodes utilisées pour obtenir des informations sur le comportement d'un programme sans réellement l'exécuter.
  - Un *compilateur* fait de l'analyse statique



- *Analyse Dynamique* (ou *test* dynamique):
  - on exécute le programme sur des données de test.

- Dans les deux cas, références au modèle!
  - Et nécessité de *valider le modèle* ("model-checker")

Interlude

```
{i=0 ; read(x);
repeat {
i++ ;
perd(x, i); }
until  term(i,x) ;
…
```

CORRECT!

Interlude (suite)

- Le texte du programme *n’est pas le système*
- On ne peut faire confiance ni au compilateur, ni au système d’exploitation, ni à l’environnement d’exécution

Que faire?

Combiner méthodes statiques et dynamiques, bien sûr

Comment?

Analyse statique

- Du plus facile : contrôles syntaxiques
  - *Mesures de la qualité du code*
  - Exemples : Sonar, et plein d'autres
- Au plus difficile : preuve de programmes
  - *{précondition} code {postcondition}*
  - exemples : Frama-C/CAVEAT (CEA puis start-up), Boogie/ VCC et HAVOC (Microsoft Research, puis github), …
- En passant par des propriétés spécifiques :
  - *flot des données, débordement de tableaux ou de pile, division par zéro, overflow, WCET, ...*
  - Exemples : Astrée (CNRS/ENS puis AbsInt), Polyspace (MathWorks), …

Indécidabilités et méthodes statiques

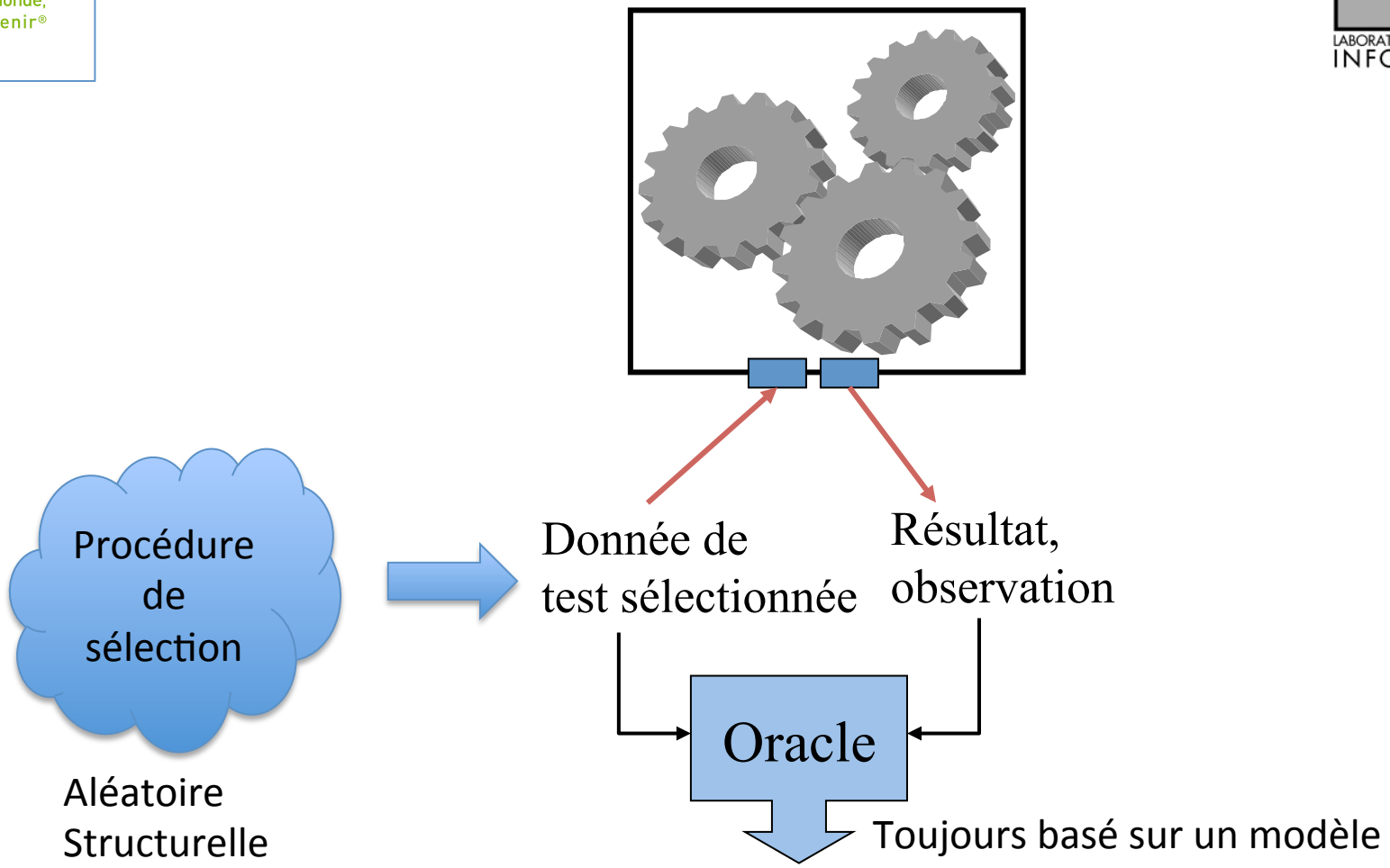
- La référence en indécidabilité : le problème de l'arrêt d'une machine de Turing (1936)
- PB : les langages de programmation sont presque tous Turing-complet
- Théorème de Rice (1953) : *« toute propriété non triviale d'un langage reconnu par une machine de Turing est indécidable »*
- S'applique aux 2 derniers items principaux de la page précédente

Contourner l'indécidable

- Comment ?
  - en se ramenant à des sous-problème décidables en fixant des paramètres ou en acceptant des approximations.
- Tout outil *automatique* qui considère des *propriétés intéressantes* sur des *programmes réalistes* repose soit sur des *cas* très particuliers, soit sur des approximations
- Selon les cas on va avoir
  - des fausses alarmes (sur-approximation): « warning »
  - des réponses inconclusives (sous-approximation)
  - la nécessité d'interactions avec l'utilisateur

Analyse dynamique, test

- Choix des données de test
  - Probabiliste,
  - structurel,
  - boîte noire (basé sur un modèle),
  - guidé par des modèles de fautes (mutations)
- Soumission : problème de *contrôle*
- Verdicts (oracle) :
  - Toujours basé sur une spécification ou un modèle
  - Problème d'*observabilité* des comportement et des résultats
  - Indécidable en général ☹, mais le plus souvent décidable pour les types de données élémentaires ☺


Procédure de sélection
Donnée de test sélectionnée
Résultat, observation
Oracle
Aléatoire
Structurelle
Model-based
…
Toujours basé sur un modèle

Incomplétude des méthodes dynamiques

- « Tester un programme peut démontrer la présence de bugs, jamais leur absence. »
- Edsger Dijkstra, 1972
  - Un anathème qui a freiné les travaux sur le test de logiciel

Exécution symbolique

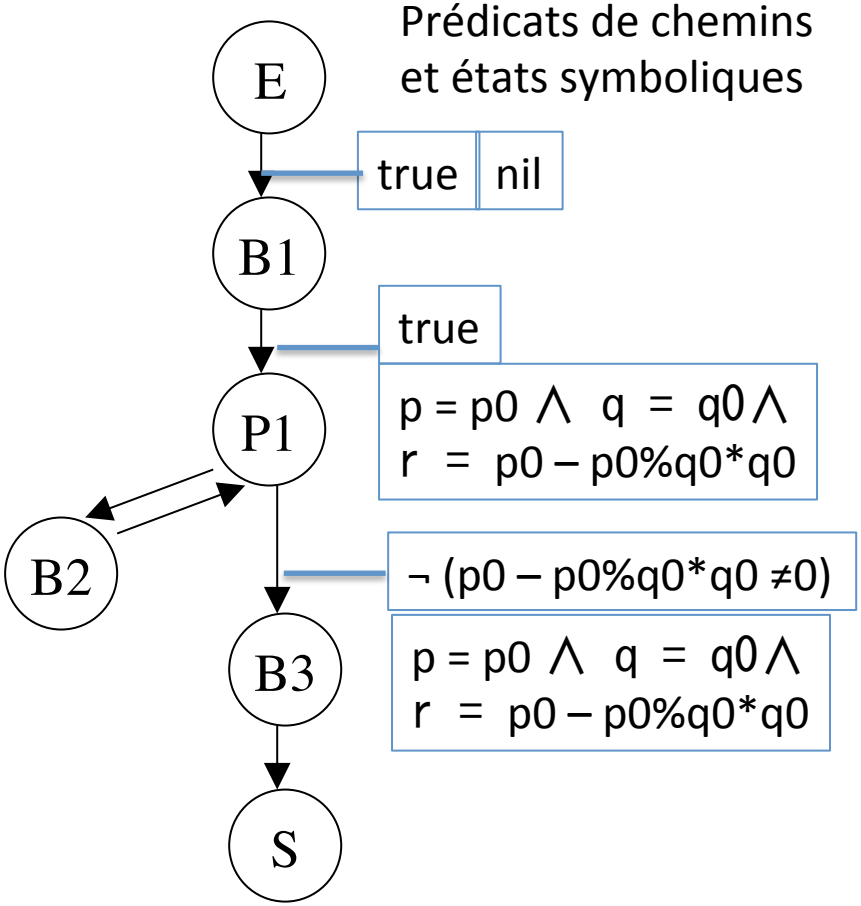
- Méthode statique… très liée au test dynamique
- On remplace les données par des symboles
- A partir de ces symboles on construit les expressions calculées et les conditions rencontrée le long des chemins du programme.
- Très vieille idée qui est devenue applicable récemment et se combine bien avec le test

Exemple: le bon vieux PGCD

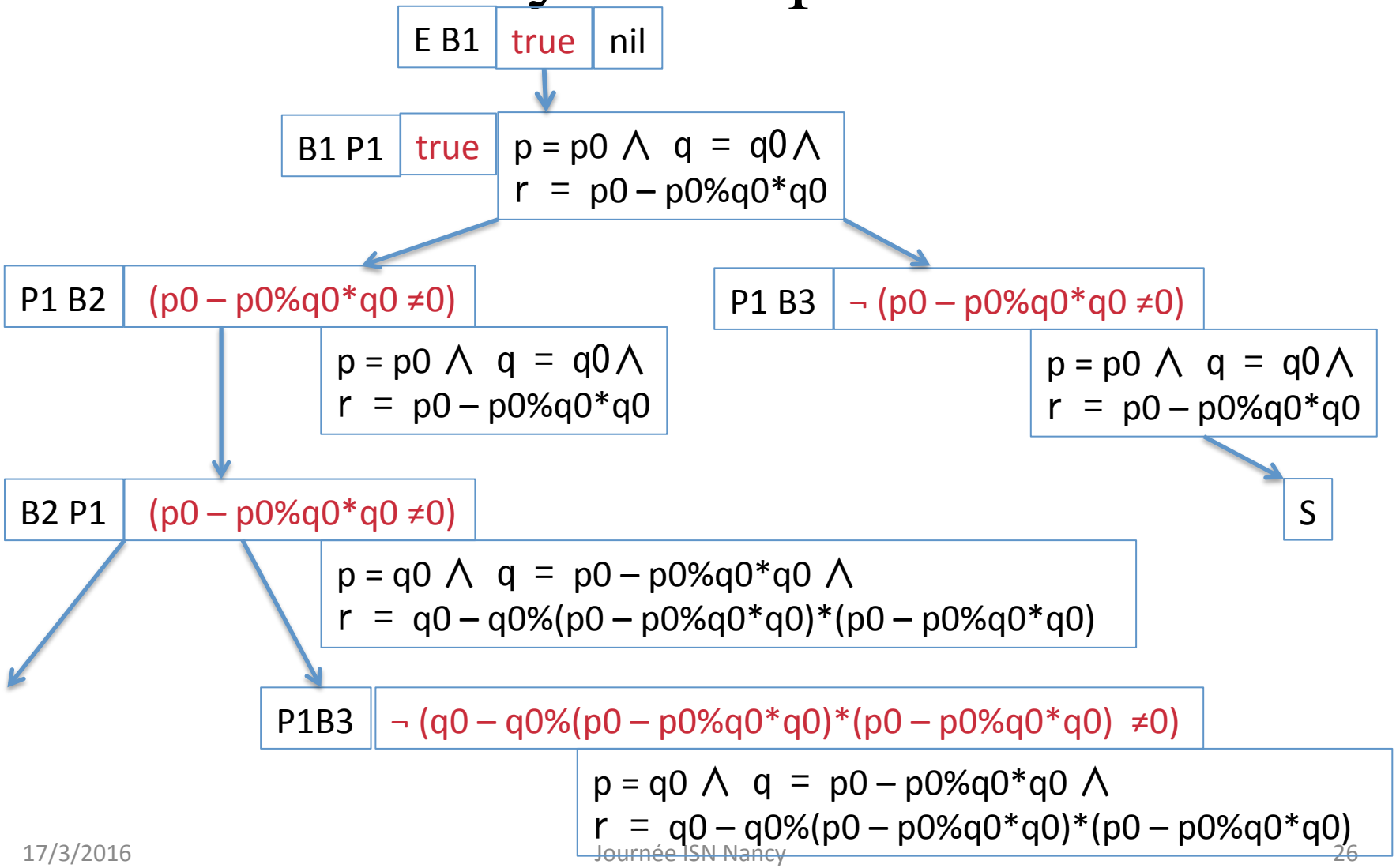
```
begin
read(p, q) ;                    B1
r := p - p%q*q ;
while  r≠0  do begin    P1
p := q ; q:=r ;                 B2
r := p - p%q*q ;
end ;
write(q) ;      B3
end
```

Graphe de contrôle

Prédicats de chemins et états symboliques



Arbre d'exécution symbolique



Pro et cons…

- Un *prédicat de chemin* caractérise les tests pour couvrir ce chemin => on résout le prédicat et on a un test qui couvre le chemin ☺
- Problèmes ☹
  - Ces prédicats sont très longs
  - Il y a beaucoup de chemins et beaucoup sont infaisables
  - En présence de boucles, l'exécution symbolique ne termine pas
- Néanmoins très utilisée, soit en combinaison avec des techniques de Model-Checking pour
  - Vérifier statiquement des impossibilités (états non atteignable)
  - Générer des tests
- Soit pour faire du test « concolique », qui combine statique et dynamique

Le test concolique

- Exécution symbolique dynamique
  - Objectif : couvrir automatiquement tous les chemins faisables jusqu'à une certaine longueur
- Principe :
  - On exécute le programme sur une donnée arbitraire
  - Le programme est instrumenté pour fournir l'enregistrement de l'exécution symbolique correspondante
  - On remonte dans le chemin symbolique jusqu'au dernier choix, et on construit (facilement) le prédicat de l'autre alternative* ; on résout ce prédicat
    - Si on obtient une solution, on teste le programme avec cette donnée
    - Sinon, on remonte un coup de plus

*si elle n'a pas encore été couverte, sinon on remonte un coup de plus

Outils de test concolique

- PathCrawler (CEA)
  - Intégré dans Frama-C
  - Disponible on-line: **pathcrawler**-online.com
- Pex et CodeDigger (Microsoft Research)
  - Intégré dans Visual Studio
  - Disponible on-line: **www.pexforfun.com**
- Autres …
- Super sympas,
  - Qualité et défauts du solveur de contraintes utilisé
  - Ne terminent pas toujours…

Bonnes Pratiques …en 2016

- Ecrire les *préconditions* et les *postconditions* avant de programmer
  - Faire un peu de logique de Hoare ??? En tout cas, raisonner sur le programme par exécution symbolique
- Une fois le programme écrit *concevoir des tests* — qui satisfont la précondition — selon un critère de couverture (*exemple* : chemins qui passent 0, 1, n fois dans les boucles)
  - *Vérifier* les résultats des tests en se basant sur la postcondition

Quelques histoires heureuses et peu médiatisées

- On ne parle que de ce qui va mal !
- Mais vous pouvez essayer :
  - "software success stories" avec Google (ou un autre moteur de recherche…), c'est copieux (mais parfois commercial...)
- Des exemples précis :
  - Développement de CompCert (un compilateur C vérifié)
  - seL4 : Unix-like OS kernel, complétement vérifié.
- Des critères et des procédures de certification :
  - Critères communs (ISO15408) : niveaux (de 1 à 7),
  - Autorités de certification (ANSSI en France),
  - Centres agréés pour le processus de certification (CESTI en France).

Conclusion: on progresse!