

# Méthodes formelles pour la sûreté de fonctionnement des logiciels

2 mars 2004

Yves-Marie Quemener

Le présent document contient des informations qui sont la propriété de France Télécom. L'acceptation de ce document par son destinataire implique, de la part de ce dernier, la reconnaissance du caractère confidentiel de son contenu et l'engagement de n'en faire aucune reproduction, aucune transmission à des tiers, aucune divulgation et aucune utilisation commerciale sans l'accord préalable écrit de France Télécom R&D



# Problème

- S Ubiquité du logiciel
- S De plus en plus d'activités dépendent du bon fonctionnement de systèmes logiciels
  - Q En anglais, *dependability* = qualité qui fait que l'on peut raisonnablement dépendre de tels systèmes
- S Dans certains cas, question de vie ou de mort
  - Q Aéronautique, ferroviaire, nucléaire...
- S Dans d'autres cas, «**S**eulement» souci financier
  - Q Cartes à puce, téléinformatique...



# Exemples

## S Tirés de [Rushby93], p. 137

Q Therac 25, machine médicale d'irradiation, six morts entre 1985 et 1987

Q Saturation nationale du réseau longue distance AT&T le 15 janvier 1990

Q Erreur d'un missile Patriot le 25 février 1991, 28 morts, tués par le Scud non-intercepté

Q Problèmes dans le logiciel de contrôle de vol de plusieurs avions

## S Autres exemples

Q En partie, appels longue distance France Télécom depuis Paris en 1997

Q Ariane V, le 4 juin 1996

Q Mars Climate Observer, le 23 septembre 1999



# Comment garantir les propriétés des logiciels ?

**S** En général, les propriétés des logiciels sont indécidables

**Q** Par exemple, la terminaison

```
syracuse (n : int) {  
  if n == 1 then 1  
  elseif impair(n) then syracuse(3*n + 1)  
  elseif pair(n) then syracuse (n/2) }
```

**S** Est-ce que cette fonction termine pour tout  $n$  ?



# Comment garantir les propriétés des logiciels ?

S En les testant...

s (2) ->s (1) ->1

s (3) ->s (10) ->s (5) ->s (16) ->s (8) ->s (4) ->s (2)  
->s (1) ->1

s (4) ... s (5) ... s (6) ->s (3) ...

s (7) ->s (22) ->s (11) ->s (34) ->s (17) ->s (52)  
->s (26) ->s (13) ->s (40) ->s (20) ->s (10) ...



# Comment garantir les propriétés des logiciels ?

**S** Mais...

**S** Le test ne peut que garantir que, pour les valeurs testées, il y a eu effectivement terminaison

**Q** En général, un test qui échoue garantit la présence d'erreurs mais toute une batterie de tests qui réussit ne garantit pas l'absence d'erreurs

**S** Le test doit être effectué en un temps fini

**Q** On ne peut pas garantir qu'un appel à la fonction `s` qui semble ne pas terminer ne va pas s'achever un peu plus tard. Pour la question de la terminaison, un test ne peut pas garantir que le programme ne termine pas.



# Comment garantir les propriétés des logiciels ?

**S** Cependant, même si les tests ne peuvent pas garantir l'absence d'erreurs dans un logiciel, ils sont indispensables

**Q** Est-ce que vous accepteriez de monter dans un avion qui n'a jamais été testé en vol mais dont les fabricants vous garantissent qu'ils ont «**Prouvé**» que tout allait bien se passer ?



# Comment garantir les propriétés des logiciels ?

- S** Au moins, regarder le processus de développement du logiciel
  - Q** Les bons logiciels sont ceux développés suivant les bonnes méthodes (?)
  - Q** Présence ou non d'un ensemble de documents associés au logiciel (spécifications, documentations utilisateurs, respect de règles de codage, rapports de tests, de relectures croisées des logiciels ...)
- S** Pour des niveaux d'assurance élevés, utiliser des analyses fondées sur les propriétés mathématiques des logiciels, dans les limites intrinsèques de décidabilité



# Plan

- S Concepts de la sûreté de fonctionnement
- S Évaluation de la sûreté de fonctionnement
- S Quatre techniques d'évaluation
  - Q Analyse statique
  - Q Analyse comportementale
  - Q Tests
  - Q Méthodes formelles et preuves
- S Problématique économique
- S Conclusion - Exemples d'études menées à FTR&D
- S Bibliographie - Outils



# Avertissement

- S** Vaste sujet => vulgarisation à approfondir par ailleurs
- S** Bibliographie sommaire à la fin => points d'entrée possibles

# Concepts de la sûreté de fonctionnement





# Sûreté de fonctionnement

- S** Sûreté de fonctionnement = propriété d'un système dans lequel on peut avoir *à juste titre confiance dans la délivrance* d'un service
- S** En anglais, *dependability*. Notion que l'on peut raisonnablement dépendre du service rendu par le système.
- S** Une notion d'évaluation est nécessairement présente. On aura *à juste titre* plus ou moins *confiance* suivant les résultats d'une évaluation plus ou moins détaillée.



# Attributs de la sûreté de fonctionnement (I)

- S** Lors d'une évaluation, accent mis sur tel ou tel attribut
- S Disponibilité** : être prêt à fournir le service
- S Fiabilité** : continuité du service
- S Sûreté** : non-occurrence de conséquences catastrophiques sur l'environnement
- S Confidentialité** : non-occurrence de dévoilements non autorisés d'informations

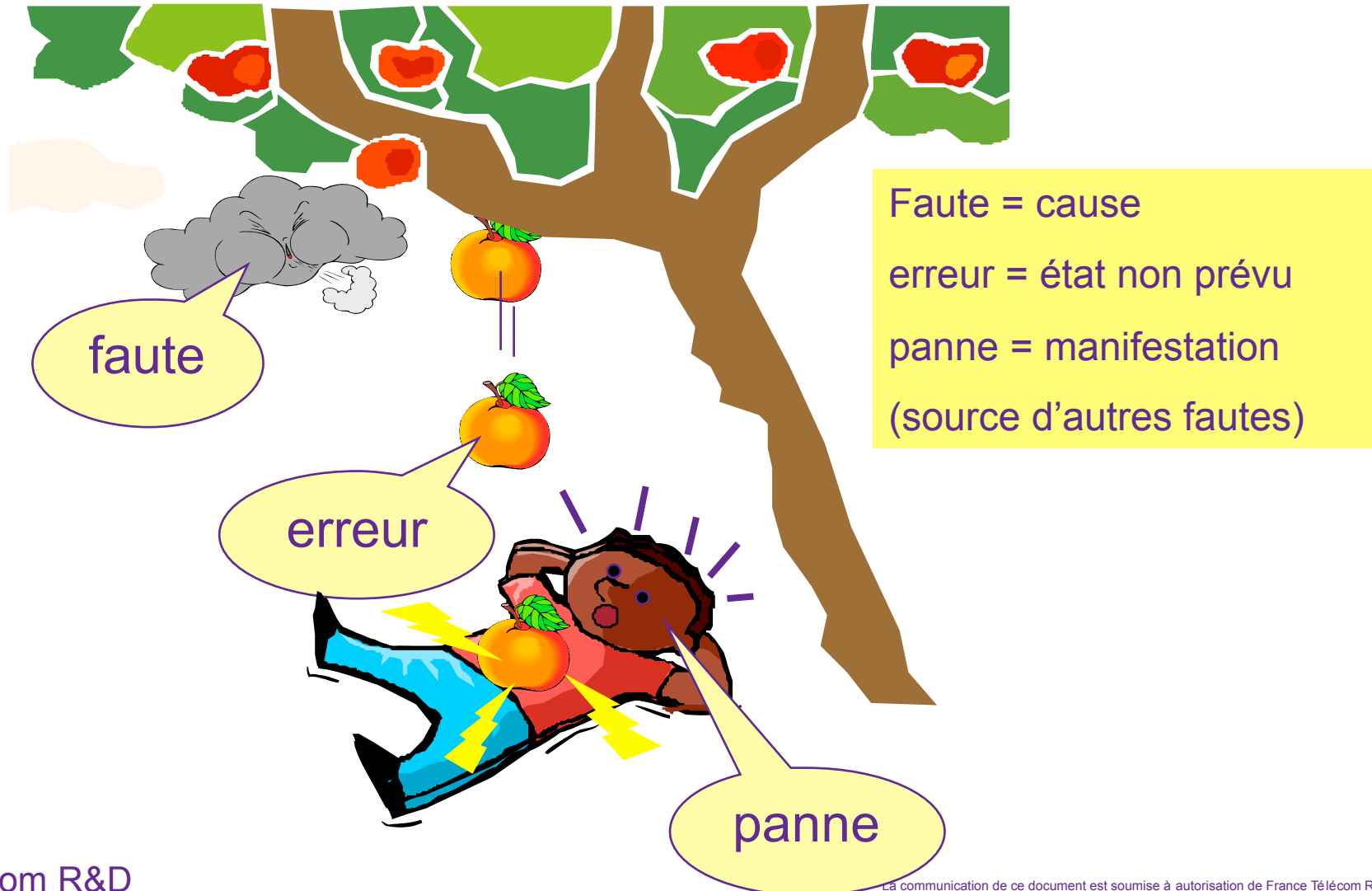


# Attributs de la sûreté de fonctionnement (II)

- S Intégrité** : non-occurrence de modifications incorrectes du système
- S Maintenabilité** : capacité à être réparé et à évoluer
- S Sécurité** = intégrité + confidentialité + disponibilité des actions autorisées



# Fautes – erreurs – défaillances





# Supprimer les défaillances

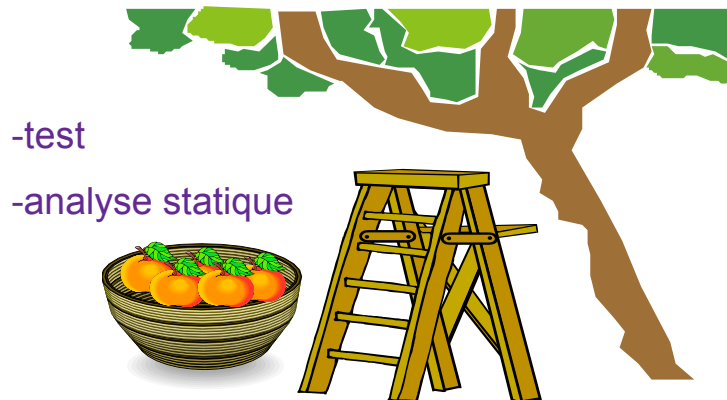


- méthodes formelles (qualité des produits)
- démarche qualité (qualité des processus)



Prévoir les fautes

Prévenir les fautes



Supprimer les fautes

Tolérer les fautes



# Sources

- S Terminologie commune à la communauté « Sûreté de Fonctionnement »
- S Présentée dans [Laprie94] ou dans le rapport final du projet Squale, *Dependability Assessment Criteria*

# Évaluation de la sûreté de fonctionnement





# Évaluation de la sûreté de fonctionnement – Projet Squale

- S** Il existe des normes proposant des critères d'évaluation de la SdF des systèmes informatiques soit :
  - Q** Pour des attributs particuliers : évaluation sécuritaire (dite Critères Communs)...
  - Q** Pour des domaines industriels particuliers : nucléaire, ferroviaire...
- S** Objectif du projet Squale : proposer des critères d'évaluation généraux pour la SdF de tous systèmes informatiques
- S** Et des liens des critères généraux vers les normes préexistantes

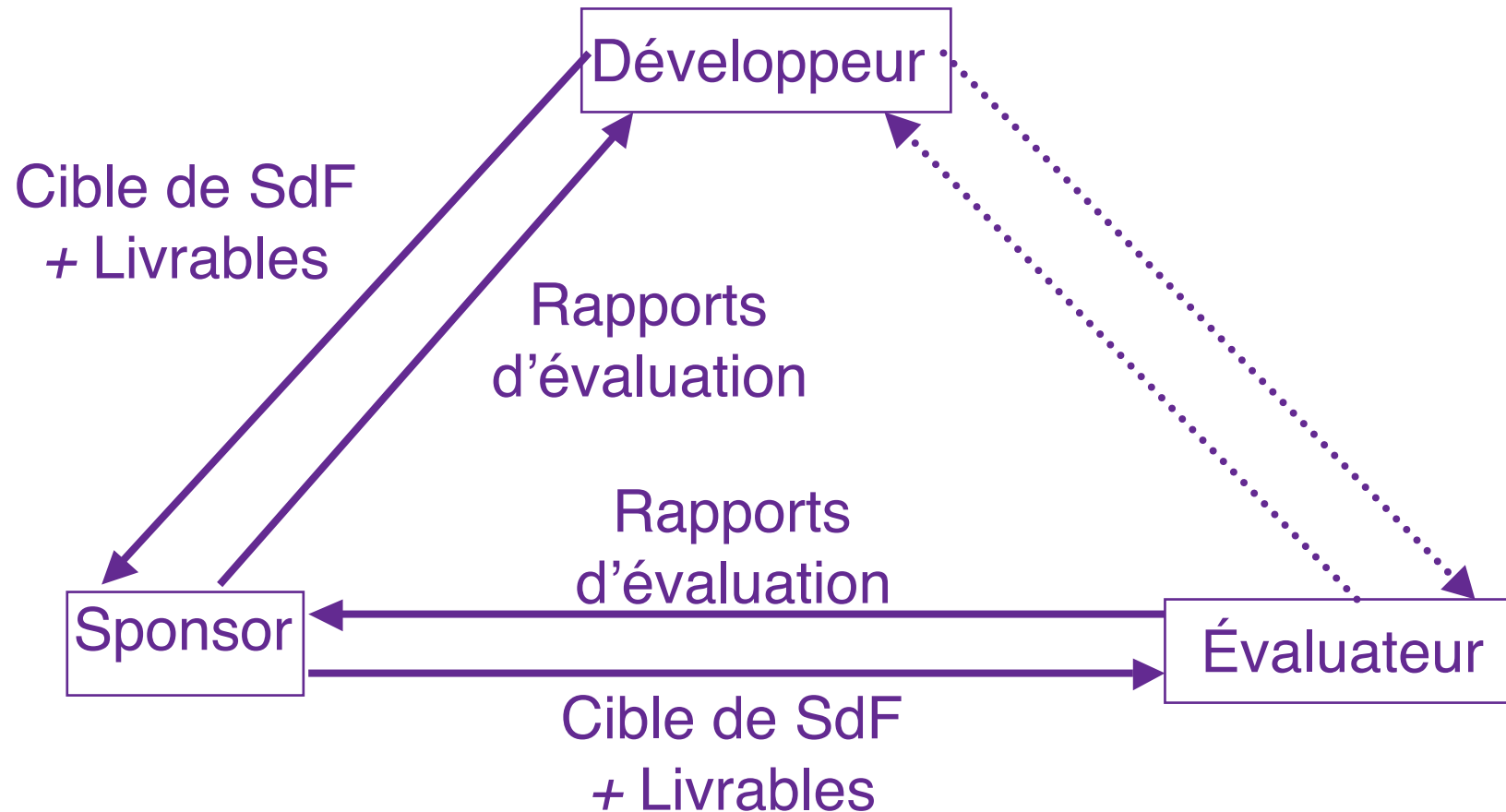


# Terminologie et avertissement

- S** Dans la suite, je traduis certains termes du rapport du projet Squale
- S** Pas de terminologie officielle en français à ma connaissance
- S** La suite présente une simplification du processus d'évaluation proposé par Squale – Se référer à l'original



# Différents rôles (I)





# Différents rôles (II)

- S** Le *sponsor* est l'organisation qui demande que soit réalisée une évaluation (vendeur, fournisseur ou utilisateur d'un système, ou autorité de certification)
- S** Pour cela il doit fournir à l'*évaluateur* :
  - Q** La Cible de Sûreté de Fonctionnement
  - Q** Les différents livrables nécessaires à l'évaluation
- S** Il peut s'appuyer pour cela sur le *développeur* responsable du développement du système évalué
- S** L'évaluateur doit être indépendant du sponsor et du développeur



# Notion de cible de SdF

- S *Dependability Target* = document spécifiant la cible que cherche à atteindre l'évaluation
- S Doit comprendre (parmi d'autres) les éléments suivants
  - Q Une description du système évalué
  - Q Une description des dangers (*hazards*) étudiés
  - Q Une description de l'environnement
  - Q Une description des *objectifs de sûreté de fonctionnement*
  - Q Une description du *profil de sûreté de fonctionnement*
  - Q Une identification des *activités augmentant la confiance*



# Une première difficulté...

- S** Contexte télécom = ouverture de plus en plus grande des terminaux et des réseaux
- S** Par exemple, l'opérateur n'a (n'aura) plus la maîtrise complète des logiciels qui s'exécutent sur les terminaux mobiles (*midlets*)
- S** Quelles hypothèses faire sur les dangers possibles?
  - Q** Si on suppose ouverture complète, impossible de garantir quoi que ce soit
  - Q** Si on suppose contrôle complet, pas réaliste
- S** Établir la cible de SdF dans un contexte ouvert est un problème non résolu



# Notion de *profil de sûreté de fonctionnement*

- S Pour *Dependability Profile*
- S En se basant sur l'analyse des dangers, on donne pour chaque attribut de la SdF un niveau de confiance qui doit être obtenu
- S Niveaux de 0 (pas de confiance) à 4
  
- S Un niveau peut être vu comme :
  - Q Un objectif à atteindre lors du développement d'un système en suivant les techniques appropriées
  - Q Le niveau effectivement atteint tel que déterminé par l'évaluation



# Comment obtenir les niveaux de confiance ?

- S Pour obtenir les niveaux de confiance, il faut montrer que certaines *activités augmentant la confiance* ont été réalisées
- S Ces activités sont regroupées dans des *processus augmentant la confiance*
- S *Confidence Providing Processes* (CPPs)
- S *Confidence Providing Activities* (CPAs)



# Comment obtenir les niveaux de confiance ?

- S** Chaque CPA est qualifiée sur une échelle de 1 à 3 suivant trois dimensions
- S** Rigueur
  - Q** Est-ce que l'activité est réalisée selon des méthodes et en utilisant des outils présentant plus ou moins de rigueur ?
- S** Détail
  - Q** Est-ce que l'activité est réalisée sur plus ou moins d'aspects du système évalué ?
- S** Indépendance
  - Q** Est-ce que l'activité est réalisée par des équipes plus ou moins indépendantes des développeurs du système ?



# Comment obtenir les niveaux de confiance ?

- S Suivant les niveaux atteints dans les 3 dimensions
- S Exemple pour plusieurs activités

Activité	Niveau de confiance			
	1	2	3	4
Ana. Stat.	R1,D1,I1	R2,D2,I1	R2,D3,I2	R3,D3,I3
Test	R1,D1,I1	R1,D2,I1	R2,D3,I2	R3,D3,I3
Méth. For.	--	R1,D1,I1	R2,D1,I2	R3,D2,I3

En général, indépendamment des attributs



# Quatre CPPs

## S Validation des exigences de SdF

Q Cherche à assurer que les objectifs de SdF décrits dans la *Dependability Target* couvrent bien les dangers/menaces identifiés

Q À faire avant que le système soit développé

## S Vérification de la correction

Q Cherche à assurer que la réalisation du système remplit bien les objectifs spécifiés

Q Si ses objectifs ne couvrent pas les dangers, tant pis !

## S Validation de la SdF

Q Vérification effective sur le système développé

## S Qualité des processus

Q Vérification tout le long du développement que les processus de développement sont bien respectés



# Quatre CPPs

- S Dans la suite, on ne s'intéresse qu'à la vérification de la correction



# CPAs de la vérification de la correction

- S Analyse statique
- S Analyse comportementale
- S Tests
- S Méthodes formelles et preuves de correction
  
- S Analyse de traçabilité = consiste à vérifier qu'à toutes les exigences de haut niveau correspondent une implémentation dans le système, et que le système ne fournisse pas de fonctions non-désirées



# Une confusion possible

- S** Méthodes formelles = une des CPAs qui consiste à utiliser les mathématiques discrètes pour prouver des propriétés des systèmes
- S** Méthodes formelles = techniques informatiques automatisées permettant d'obtenir des niveaux élevés de rigueur dans plusieurs CPAs
- S** On va s'intéresser au second sens dans la suite

# Quatre techniques d'évaluation





# Analyse statique

- S** Rechercher des fautes dans le système sans l'exécuter
  - Q** Revues de code, relectures croisées etc.
  
- S** Au niveau de rigueur 3
  - Q** «*Besides the requirements corresponding to Rigour Level 2, automatic static analysis tools shall be used*»
  
- S** Pour la suite, cf. [Cousot 2000]



# Analyse statique automatique par interprétation abstraite

- S** En général, les propriétés des programmes sont indécidables
- S** Interprétation abstraite = remplacer un élément précis d'un programme par une abstraction moins précise pour espérer arriver à calculer les propriétés des programmes
- S** L'abstraction doit garantir
  - Q** Abstraction(Programme) 'correct' => Programme 'correct'
- S** Mais on aura
  - Q** Abstraction(Programme) 'incorrect' : on ne sait pas conclure
- S** Abstraction = perte d'information « **Sûre** »



# Exemple d'abstraction

- S X variable entière
- S On veut garantir : X toujours inférieur à  $2^{32}$
  
- S Abstraction : entiers  $\Rightarrow$  intervalles d'entiers
- S Si  $X = [1, 10]$ , propriété garantie
- S Si  $X = [1, 2^{32} + 1]$ , on ne sait pas conclure
  - Q Peut-être que X restera toujours inférieur à  $2^{32}$ , mais pas de garantie
  - Q L'analyse automatique peut indiquer un problème potentiel à étudier de façon poussée



# Exemple d'analyse automatique

```
1:   x := 1;  
2:   while x < 10000 do  
3:       x := x + 1  
4:   od;
```

$$\left\{ \begin{array}{l} X1 = [1,1] \\ X2 = (X1 \sqcap X3) \sqcap [-, 9999] \\ X3 = X2 \oplus [1,1] \\ X4 = (X1 \sqcap X3) \sqcap [10000, + ] \end{array} \right.$$

La théorie de l'interprétation abstraite montre  
que les valeurs sont solutions d'une équation de point fixe



# Résolution par itération à partir du minimum

	0	1	2	3	4	5	6	7	8	...
$X_1$	$\emptyset$	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	...
$X_2$	$\emptyset$	$\emptyset$	[1,1]	[1,1]	[1,2]	[1,2]	[1,3]	[1,3]	[1,4]	...
$X_3$	$\emptyset$	$\emptyset$	$\emptyset$	[2,2]	[2,2]	[2,3]	[2,3]	[2,4]	[2,4]	...
$X_4$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	...



# Techniques d'accélération

		Accélération									
		0	1	2	3	4	5	6	7	8	
{	X1 =	∅	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	[1,1]	<i>Idem</i>
	X2 =	∅	∅	[1,1]	[1,1]	[1,2]	[1,+ ]	[1,+ ]	[1,9999]	[1,9999]	<i>Idem</i>
	X3 =	∅	∅	∅	[2,2]	[2,2]	[2,2]	[2,10000]	[2,10000]	[2,10000]	<i>Idem</i>
	X4 =	∅	∅	∅	∅	∅	∅	∅	∅	[10000,10000]	<i>Idem</i>

Le calcul s'arrête quand il n'y a plus d'évolution

# Analyse comportementale





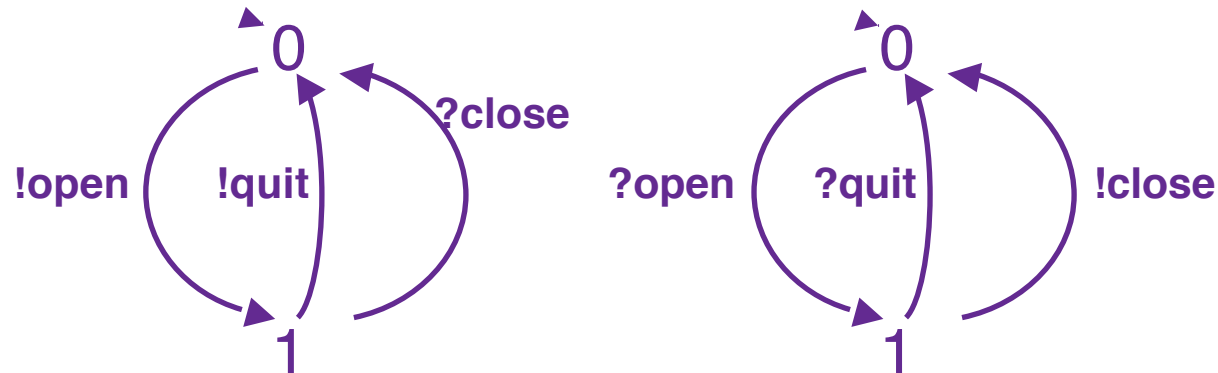
# Analyse comportementale

- S** Vérifier des propriétés du système en analysant un modèle « haut niveau » de son comportement
  - Q** Tables de décision, machines à états, réseaux de Petri...
  
- S** Au niveau de rigueur 3
  - Q** « Behavioural analyses shall be supported by computer aided simulation tools »
  
- S** Vérifier automatiquement les propriétés d'un modèle = *model checking*
  
- S** Voir [Schnoebelen1999]



# Notion de modèle

- S S'abstraire de détails d'implémentation pour se concentrer sur les points délicats
- S En particulier, systèmes distribués difficiles à appréhender
- S Exemple : automates communiquant par messages

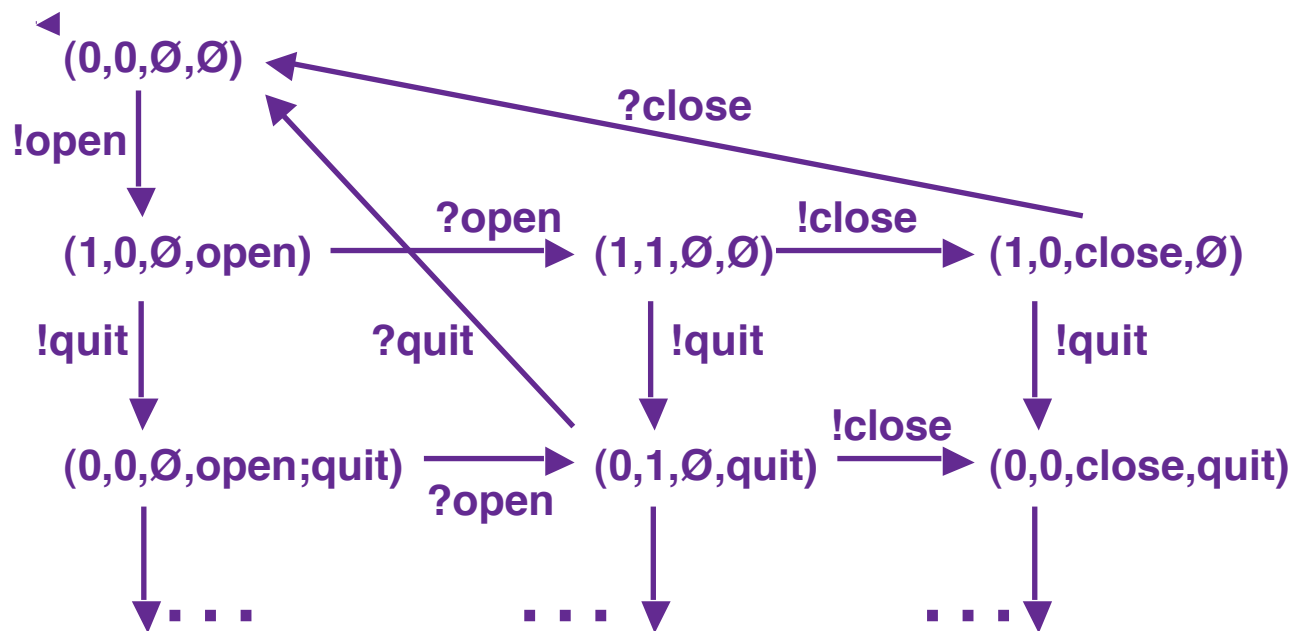


! = émission   ? = réception   ▶ = état initial



# Graphe d'atteignabilité

- S Explorer l'ensemble des comportements possibles
- S États globaux = états automates + contenus files





# Exprimer des propriétés sur les exécutions possibles (I)

- S Une *logique temporelle* va permettre de spécifier les propriétés attendues des exécutions du modèle
- S Comprend des propriétés atomiques des états, des connecteurs booléens et temporels
- S Cf. [Schnoebelen99] pour définition précise



# Exprimer des propriétés sur les exécutions possibles (II)

## S Propriétés d'atteignabilité

QEF ( $X = 2^{32}$ )

Q *Il est possible d'atteindre un état où X vaut  $2^{32}$*

## S Propriétés de sûreté

QA  $\square$  démarre W clef

Q *Le système ne démarre pas tant qu'on n'a pas mis la clé*

## S Propriétés de vivacité

QAG (requete  $\square$  AF satisfaite)

Q *Toute requête est satisfaite un jour*

QAG EF init

Q *Le système peut toujours revenir à l'état initial*

## S Dans l'exemple précédent, on n'a pas AG EF init



# Algorithme de *model checking*

- S** Un algorithme d'exploration permet de décider si tel ou tel état du graphe d'atteignabilité vérifie telle ou telle formule
- S** Explosion combinatoire : graphe trop gros ou infini
  - Q** Intégration de techniques d'abstraction
- S** Travail sur un modèle
  - Q** La présence d'une erreur à haut niveau d'abstraction dénote une erreur de conception
  - Q** L'absence d'erreur dans le modèle ne garantit pas l'absence d'erreur dans l'implémentation
- S** Progrès des techniques : *model checking* de code source (outil Bandera pour Java)

# Tests



# Tests



- S Technique de vérification dynamique
- S Faire fonctionner le système (ou un de ses composants) en lui fournissant des valeurs d'entrée
- S Choisir des valeurs d'entrée à la fois
  - Q Pas trop nombreuses car temps de test fini
  - Q Couvrant un maximum de cas différents pour obtenir une meilleure confiance
- S Au niveau de rigueur 3
  - Q « A mixed strategy for the generation of test patterns shall be used : statistical test patterns generation complemented with deterministic test cases »



# Génération automatique de tests

- S** Pas obligatoire au niveau 3 de rigueur mais des outils de génération automatique de tests peuvent aider à remplir ces besoins
- S** Deux problèmes plus ou moins distincts
  - Q** Système = fonction. Produire des jeux d'essais de données à fournir en entrée.
  - Q** Système = système réactif interagissant de manière continue avec l'environnement. Produire des scénarios d'interaction.
- S** Par la suite, second aspect
- S** Voir [Cote2001] comme introduction, y compris au premier aspect



# Formaliser le test de conformité

- S Test de conformité = test fonctionnel boîte noire
- S Spécifications et implémentations sont modélisées en automates communicants par messages
- S Exemple de relation de conformité
  - Q  $I R S \sqsubseteq \sqsubseteq s \sqsubseteq \text{trace}(S) \quad (s \sqsubseteq \text{trace}(I) \sqsubseteq \text{out}(s,I) \sqsubseteq \text{out}(s,S))$
  - Q  $\text{trace}(I)$  = ensemble des séquences d'exécution de I
  - Q  $\text{out}(s,I)$  = émissions de I après l'exécution de s



# Cas de test et suite de tests

- S** Cas de test TC = automate communicant
  - Q**Émissions et réceptions inversées par rapport à la spécification
  - Q**États de verdicts : Pass, Fail, Inconclusive
  - Q**Contrôlable : au plus une émission (i.e. réception de S) à chaque état
  - Q**Complet : toutes les réceptions (i.e. émissions de S) sont possibles à chaque état
  
- S** Exécuter un cas de test = composition parallèle de I et de TC (notée TC  $\square\square$ )
  
- S** Suite de tests = ensemble de cas de tests



# Signification des verdicts

- S** Fail : une différence entre S et I, invalidant la relation de conformité, a été détectée
- S** Pass : « l'objectif de test » est rempli
- S** Inconclusive : « l'objectif de test » n'est pas rempli, mais aucune différence entre S et I n'a été détectée



# Exemple

- S** Spécification : « Une boîte noire, surmontée d'un bouton et d'ampoules. Quand on appuie sur le bouton, une lampe verte ou une lampe rouge s'allume. »
- S** Objectif de test : « Je veux voir s'allumer la lampe rouge. »
- S** Cas de test :
  - Q** Appuyer sur le bouton
    - Si lampe rouge -> Pass
    - Si lampe verte -> Inconclusive
    - Si lampe bleue ou pas de réaction en 5 secondes -> Fail



# Correction et exhaustivité

- S** Une suite de tests TS est correcte ssi  
 $Q(\text{TC} \neq \text{TS}, \text{TC} \neq \text{atteint Fail}) \neq \text{IRS}$
- S** Une suite de tests TS est exhaustive ssi  
 $Q(\text{TC} \neq \text{TS}, \text{TC} \neq \text{atteint Pass}) \neq \text{IRS}$
- S** Correction facile (pas de Fail), mais exhaustivité...



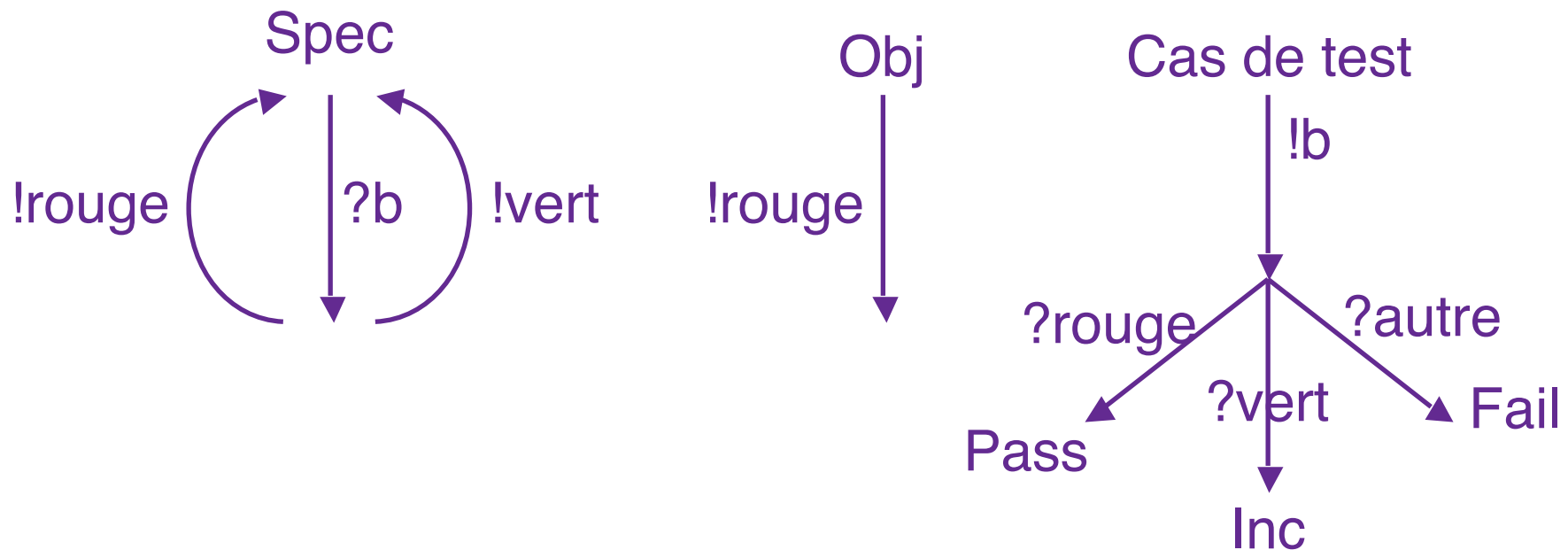
# Un algorithme de génération de tests

- S Outil TorX (Jan Tretmans, Univ. Nijmegen)
- S Cas de test  $t(S)$ , où  $S$  est un ensemble d'états (initialement,  $S = \{s_0\}$ )
- S Appliquer aléatoirement une des trois opérations
  1. Fin du cas de test, verdict Pass
  2. Fournir une entrée  $a$  et continuer :  $t(S \text{ après } ?a)$
  3. Observer une sortie : si interdite par la conformité, verdict Fail, sinon  $t(S \text{ après } !x)$
  
- S Clairement correct ; exhaustif si infinité de tests, et spécification finie



# Un autre algorithme

- S Outil TGV (Fernandez-Jéron, Verimag-Irisa)
- S Produit synchronisé de la spécification et de l'objectif de test donne un cas de test



# Méthodes formelles et preuves de correction





# Méthodes formelles et preuves de correction

- S Utilisation des mathématiques discrètes (fondement de la modélisation des logiciels) pour prouver des propriétés des programmes
  
- S Au niveau de rigueur 3
  - Q « Use of fully formal specification languages with comprehensive support environments including mechanised theorem proving or proof checking »



# Principe de base

- S** Définir une logique dédiée permettant de raisonner sur les logiciels
- S** Par exemple, la logique de Hoare (1969) décrit comment établir les post-conditions d'instructions d'un langage impératif à partir des pré-conditions
- S** Raisonnement sur des assertions relatives à la valeur des variables
- S** Permet de prouver des théorèmes relatifs aux programmes
- S** Voir [Monin2000]



# Deux types d'outils (en fait, mélangés)

## S Assistant de preuves (*proof checker*)

Q Programme qui vérifie les étapes d'une preuve fournies par un humain

## S Prouveur de théorèmes (*theorem prover*)

Q Programme qui essaie de réaliser automatiquement des preuves de propriétés

## S *Model checking* et interprétation abstraite □

techniques de preuve automatique dans un cadre restreint

Q Pour des logiques moins expressives, pour des systèmes plus simples => décidabilité => automatique

Q Intégration entre différentes techniques à réaliser



# Logique de Hoare (I)

**S** Relie les instructions à des assertions sur les variables -  $\{P\} s \{P'\}$  :  $s$  instruction ;  $P, P'$  assertions

**S** Instruction vide  
 $\{P\} \text{skip} \{P\}$

**S** Séquencement  
$$\frac{\{P_1\} s \{P_2\} \quad \{P_2\} s' \{P_3\}}{\{P_1\} s ; s' \{P_3\}}$$

**S** Alternative  
$$\frac{\{P \wedge C\} s_1 \{Q\} \quad \{P \wedge \neg C\} s_2 \{Q\}}{\{P\} \text{if } C \text{ then } s_1 \text{ else } s_2 \{Q\}}$$



# Logique de Hoare (II)

## S Boucle

$$\frac{(I \sqsubseteq C) \sqsubseteq v \sqsubseteq N \quad \{I \sqsubseteq C \sqsubseteq v = V\} S \{I \sqsubseteq v < V\}}{\{I\} \mathbf{while} C \mathbf{do} S \{I \sqsubseteq \neg C\}}$$

Le variant  $v$  sert à établir la terminaison

## S Affectation

$$\{[x := E]P\} \mathbf{x} := \mathbf{E} \{P\}$$

## S Renforcement et relâchement

$$\frac{P' \sqsubseteq P \quad \{P\} S \{Q\} \quad Q \sqsubseteq Q'}{\{P'\} S \{Q'\}}$$



# Exemple

```
x := 1;  
while x < 10000 do  
    x := x + 1  
od;
```

- S On veut montrer que, à la fin,  $x = 10000$
- S On a :  $C \sqcup x < 10000$  soit  $\neg C \sqcup x \geq 10000$
- S On prend :  $I \sqcup x \leq 10000$  et  $v = 10000 - x$



# Raisonnement (I)

**S** Application de la règle de l'affectation

$$\{x + 1 \leq 10000\} \mathbf{x} := \mathbf{x} + 1 \{x \leq 10000\}$$

$$\{x < 10000\} \mathbf{x} := \mathbf{x} + 1 \{x \leq 10000\}$$

$$\{I \leq C\} \mathbf{x} := \mathbf{x} + 1 \{I\}$$

**S** Et aussi

$$\{9999 - x < V\} \mathbf{x} := \mathbf{x} + 1 \{10000 - x < V\}$$

$$\{10000 - x \leq V\} \mathbf{x} := \mathbf{x} + 1 \{10000 - x < V\}$$

**S** Par renforcement

$$\{10000 - x = V\} \mathbf{x} := \mathbf{x} + 1 \{10000 - x < V\}$$

$$\{v = V\} \mathbf{x} := \mathbf{x} + 1 \{v < V\}$$



# Raisonnement (II)

- S** On a aussi  
 $x < 10000 \wedge 10000 - x \geq 0$   
 $(I \wedge C) \wedge \forall v \in N$
- S** Les prémisses pour appliquer la règle de raisonnement de la boucle sont satisfaites
- S** L'invariant  $I \wedge x \leq 10000$  est vrai en entrée de boucle (après  $x := 1$ )
- S** Donc, en sortie de boucle  $I \wedge \neg C$  est vrai, c'est-à-dire  $x \leq 10000 \wedge x \geq 10000$ , soit  $x = 10000$



# Remarques

- S On obtient le même résultat que par l'analyse statique automatique
  - Q Mais parfois l'analyse automatique n'aboutit pas
- S Technique non automatique : il faut choisir à bon escient  $I$  et  $v$ , et les étapes du raisonnement
- S Un assistant de preuve permet d'être sûr de la correction du raisonnement
- S Autre démarche possible : utiliser une logique dite constructive pour prouver des propriétés de spécifications, et en extraire un programme
  - Q Par exemple, prouver *de manière constructive* que deux entiers ont toujours un PGCD ; le programme extrait est certifié correct

# Problématique économique





# Intérêt de la certification de la SdF

## S Certains domaines imposent la certification

Q Domaines industriels précis : IEC 1508 *electrical/electronic/programmable electronic systems* ; EN 50126 *railway applications* ; EN 50128 *railway control and protection systems* ; IEC 880 *safety systems of nuclear power stations* ; RTCA DO-178B *avionics*

Q Attributs précis de la SdF : ITSEC/Critères communs pour la sécurité

## S Pas le cas dans les télécoms en général

Q Domaine trop vaste + environnement ouvert

Q Intérêt naissant pour la certification des aspects sécurité (certification « Critères Communs »)



# Certification sécuritaire (I)

- S** D'après un article interne France Télécom d'Eric Wiatrowski (respons. Sécurité Transpac)
- S** Sécurité de l'information autour de quatre critères
  - Q** Confidentialité
  - Q** Intégrité
  - Q** Disponibilité
  - Q** Traçabilité
- S** Deux objectifs
  - Q** Gagner des clients
  - Q** Améliorer la rentabilité



# Certification sécuritaire (II)

- S Gagner des clients
- S Pour Transpac, objectif = convaincre des entreprises d'externaliser
- S « La sécurité ne se mesure pas, elle s'évalue. »
- S Externaliser nécessite une confiance qui peut reposer sur la certification par des organismes indépendants
  
- S Certification = argument marketing



# Certification sécuritaire (III)

- S** Améliorer la rentabilité
- S** Non-qualité en amont = surcoûts en aval pour résoudre les problèmes
- S** Accroissement de la criminalité informatique = sécurité doit être traitée en amont plutôt que d'assumer les coûts
- S** La certification permet
  - Q** De détecter des failles
  - Q** D'imposer des solutions
- S** Certification = contrainte interne de qualité
  - Q** Transpac a fait certifier son service de VPN IP au niveau 1 (évaluation des méthodes d'exploitation et d'administration des routeurs IP)



# Coûts/bénéfices des méthodes formelles (I)

## S Coûts de mise en œuvre

Q Techniques difficiles/peu répandues => besoin de disposer d'experts, ou de les former

Q Techniques prenant du temps, même si plus ou moins automatisées

Q Limites intrinsèques des techniques : explosion combinatoire + indécidabilités

## S Bénéfices de mise en œuvre

Q Marketing + contrainte interne de qualité

Q Image de fiabilité des télécoms à préserver

Q Plus grande confiance vis-à-vis des développements de tiers

Q Coût de récupération/mise à jour de logiciels embarqués

Q Coût éventuel de primes d'assurance à la baisse



# Coûts/bénéfices des méthodes formelles (II)

- S** Analyse coûts/bénéfices à faire
- S** Très hauts niveaux de garantie pour des petits logiciels critiques
  - Q** Paiement, cryptographie, cartes bancaires...
- S** Analyses automatiques moins poussées pour des logiciels moins critiques
  - Q** Midlets développées par des tiers, protocoles de communication...

# Conclusion





# Quelques études (ou projets) menées à FTR&D

- S Vérification de protocoles cryptographiques
- S Génération de tests à partir de calcul de séquences discriminantes
- S Certification Critères Communs niveau 7 d'algorithmes d'authentification
- S Plate-forme logicielle d'agents mobiles pour la tolérance aux fautes
- S Analyses statiques pour Java J2ME
- S Génération de tests d'IHMs
- S ...



# Bibliographie sommaire

## S Sur la problématique générale

**Q** *Introduction aux méthodes formelles*. Jean-François Monin, 2000. Hermès, collection technique et scientifique des télécommunications.

**Q** *Formal Methods and the Certification of Critical Systems*. John Rushby, 1993. Rapport technique du SRI, à destination de l'autorité de certification de l'aviation civile américaine. Sur le Web : [www.csl.sri.com/papers/csl-93-7/](http://www.csl.sri.com/papers/csl-93-7/)

**Q** *SdF des systèmes informatiques : concepts de base et terminologie*. J.C. Laprie, 1992. Rapport du LAAS 94448.

**Q** *SQUALE Dependability Assessment Criteria*. Squale Consortium, 1999. Rapport final du projet IST Squale. Sur le Web : [www.newcastle.research.ec.org/squale/SQUALE4.pdf](http://www.newcastle.research.ec.org/squale/SQUALE4.pdf)



# Bibliographie sommaire

## S Sur le model-checking

Q *Vérification de logiciels, techniques et outils du model-checking*. Coll. coordonné par Philippe Schnoebelen, 1999. Vuibert.

## S Sur l'interprétation abstraite

Q *Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. P. Cousot et R. Cousot, 1977. In POPL'77. Article fondateur. Sur le Web:

[www.di.ens.fr/~cousot/COUSOTpapers/POPL77.shtml](http://www.di.ens.fr/~cousot/COUSOTpapers/POPL77.shtml)

Q *Interprétation abstraite*. P. Cousot, 2000. In TSI, vol. 19, nb. 1-2-3. Très bon article de vulgarisation. Sur le Web : [www.di.ens.fr/~cousot/COUSOTpapers/TSI00.shtml](http://www.di.ens.fr/~cousot/COUSOTpapers/TSI00.shtml)

Q Et les autres articles de Cousot disponibles sur son site



# Bibliographie sommaire

## S Sur les assistants de preuve

Q Livre de Jean-François Monin ou rapport de Rushby déjà cités, plus documentation des différents outils

## S Sur la génération automatique de tests

Q *État de l'art sur la synthèse (génération automatique) de tests*. Collectif, 2001. Rapport du projet RNTL Cote. Disponible sur le Web : [www.irisa.fr/cote/doc/L22.pdf](http://www.irisa.fr/cote/doc/L22.pdf)

Q *A Formal Approach to Conformance Testing*. Jan Tretmans, 1992. Thèse de l'université de Twente. Disponible sur le Web : [fmt.cs.utwente.nl/publications/tretmans.pap.html](http://fmt.cs.utwente.nl/publications/tretmans.pap.html)

Q *TGV : théorie, principes et algorithmes*. Thierry Jéron, 2002. In *Techniques et Sciences Informatiques*, numéro spécial « Test de logiciel » (vol. 21).



# Quelques outils

## S Model-checking, simulation

QTelelogic : Tau SDL Suite, Tau UML Suite. Outil industriel pour UML 2.0. Voir : [www.telelogic.com](http://www.telelogic.com)

QSpin. Outil académique développé aux Bell Labs. Voir : [spinroot.com/spin/whatispin.html](http://spinroot.com/spin/whatispin.html)

QIF. Langage de description de systèmes distribués servant de pivot à l'étude de systèmes décrits en UML, SDL, etc. Outil académique développé à Verimag. Voir : [www-verimag.imag.fr/DIST\\_SYS/IF/](http://www-verimag.imag.fr/DIST_SYS/IF/)

QBandera. Outil académique développé à la Kansas State University. Model checking de code Java. Voir : [bandera.projects.cis.ksu.edu/](http://bandera.projects.cis.ksu.edu/)



# Quelques outils

## S Analyse statique par interprétation abstraite

Q PolySpace. Détection d'erreurs *run-time* dans des codes source Ada et C. Détection de code mort. Outil industriel utilisé dans l'aérospatiale : [www.polyspace.com](http://www.polyspace.com)

Q AbsInt. Divers outils industriels d'analyse statique pour des codes embarqués. [www.absint.com](http://www.absint.com)

Q Projet RNTL Astrée. Détection d'erreurs *run-time* dans du code C. Vérification du contrôleur de vol des A340 et A380 : [www.astree.ens.fr](http://www.astree.ens.fr)



# Quelques outils

## S Assistants de preuve

QCoq. Outil académique développé à l'Inria. Aide à l'obtention de preuves, vérification automatique des preuves, extraction de programmes certifiés. Voir : [coq.inria.fr](http://coq.inria.fr)

QPVS. Outil académique développé au Stanford Research Institute. Accent mis plutôt sur la dérivation automatique de preuves. Voir : [pvs.csl.sri.com](http://pvs.csl.sri.com)



# Quelques outils

## S Génération automatique de tests

**QTGV.** Outil académique développé à l'Irisa. Sert de moteur à l'outil industriel TestComposer de Telelogic (n'est plus maintenu). Une version travaille sur des spécifications IF, disponible par ce biais.

**TorX.** Outil académique développé à l'université de Twente. Disponible sur : [www.purl.org/net/torx](http://www.purl.org/net/torx)

**BZ Testing Tools.** Outil académique développé à l'université de Besançon. Génération de données de tests et de scénarios à partir de spécifications B et Z. Industrialisation prévue via une start-up. Voir : [lifc.univ-fcomte.fr/RECHERCHE/TFC/Page\\_BZ-TT.html](http://lifc.univ-fcomte.fr/RECHERCHE/TFC/Page_BZ-TT.html)