

Fiabilité Logicielle Test Boite Noire

- 1 Introduction
- 2 Spécifications
- 3 Techniques d'écritures de Tests
 - Partitionnement en Classe d'Equivalence
 - Test aux limites
 - Combinatoire
 - Graphe de causalité

① Test boîte noire.

La structure du programme est inconnue et le test est construit à partir des spécifications seules.

Spécification incomplète ou imprécises

② Test boîte blanche

Le programme est donné et le test est construit à partie de la structure du programme et des spécifications.

③ Test probabiliste

Les tests sont construits selon une loi statistique.

- 1 BN: bon pour détecter les omissions ou erreurs de spécification.
- 2 BB: bon pour détecter les erreurs de programmation.
- 3 Probabiliste: détecte rapidement des erreurs, efficacité tend à plafonner.

A propos de Spécifications

Test BN: nécessité d'une **spécification** du **comportement** du programme.

- Langue naturelle
- Logique
- Automates (expression régulières)
- ...

Langue naturelle

Avantages: simple, tout le monde comprend.

Inconvénients

- *la méthode retourne l'indice de l'élément maximal du tableau ambigu, omission,*
Quid de: tableau null, occurrence multiples?
⇒ ambiguïté, trous dans la spécification,..
- Description de normes (par ex consortium W3C): verbosité, difficulté à suivre,...
tout le monde comprend mais pas forcément la même chose

Langue naturelle

Avantages: simple, tout le monde comprend.

Inconvénients

- *la méthode retourne l'indice de l'élément maximal du tableau ambigu, omission,*
Quid de: tableau null, occurrence multiples?
⇒ ambiguïté, trous dans la spécification,..
- Description de normes (par ex consortium W3C): verbosité, difficulté à suivre,...
tout le monde comprend mais pas forcément la même chose

Logique

$$result = i \wedge \forall j \in [0, n - 1], a[j] \leq a[i]$$

- Expressivité: domaines usuels (Nat, Réels, ...) et construction utiles (tableaux, listes, ...), fonctions, ensembles nécessaires \implies indécidabilité de toutes les propriétés.
- Précision: taille des spécifications, difficile à comprendre, peu maniable, ... \implies nécessité d'outils automatiques

Notions très utiles en pratique: **précondition**, **poscondition**, **invariant**

Langages d'annotation

Langage de spécification et outil de preuves (CoQ, HOL,...)

Automates

Simple pour l'ingénieur, malléable (ajout de temps, entrée/sortie,..., méthodes de vérification automatique,...
Difficile à comprendre dès que la taille est importante.

Automate d'état fini

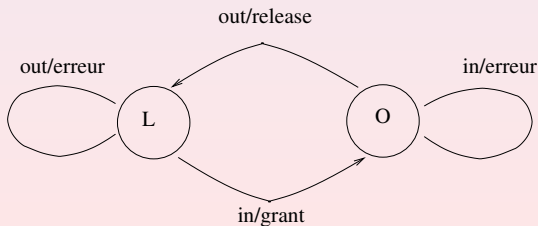
- Alphabet fini d'entrée E (évènements)
- Alphabet fini de sortie A (action)
- Etat (nb fini) Q
- Etat initial s_0
- Relation de transition: $\text{Etat} \times E \times \text{Ensemble d'action} \times \text{Etat}$
i.e. $s, e \rightarrow 2^A, s$

Exemple

Contrôle d'un accès par jeton:

- Evenements: in demande du jeton et out relache du jeton.
- Actions: grant (donner le jeton), release (reprendre le jeton), erreur
- Etat: jeton libre ou occupé.

Système:



Propriétés du système:

- Vérification de la complétude pour tout état, toute évènement une action
- Déterminisme du système: pour un état, une action une seule transition possible.
- Vérification de l'atteignabilité d'un état.

Les comportements sont réguliers.

Certains systèmes ne sont pas réguliers

consommateur/producteur avec une seule étape de production,
suivie de consommation: langage $p^n c^m$ avec $n \geq m$

Propriétés du système:

- Vérification de la complétude pour tout état, toute évènement une action
- Déterminisme du système: pour un état, une action une seule transition possible.
- Vérification de l'atteignabilité d'un état.

Les comportements sont réguliers.

Certains systèmes ne sont pas réguliers

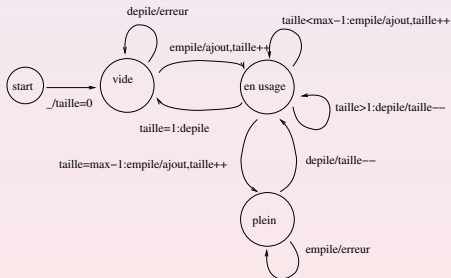
consommateur/producteur avec une seule étape de production,
suivie de consommation: langage $p^n c^m$ avec $n \geq m$

Extensions

- Ajout de variables: paramètres interne du système, paramètres extérieurs (par ex temps)
- Possibilité de garde (conditions) $taille < max:empiler/...$

Modélisation du comportement de la classe pile (bornée)

Un modele de pile bornee de $\text{max} > 1$ elements
par automate (avec garde)



garde:evenement/ action(s)

Utilisation: analyser le comportement du système. (voir cours Méthodes Formelles)

Utilisation du modèle pour définir des scénarios d'utilisations (et donc des tests):

- état initial
- état(s) final(aux) à atteindre
- Scénario d'utilisation: chemin de l'état initial vers un état final.

Exercice

Modéliser le distributeur automatique du RdC.

Comment écrire les **suites de Tests**

Pas de méthode rigoureuse Typologie de situations

- Classe d'équivalence
- Cas limites
- Combinatoire
- Graphe de causalité

Principe de la méthode

Objectif: réduire l'ensemble des DT.

- 1 Partitionner les données en classe d'équivalence tq le programme a le même comportement pour toutes les valeurs d'une classe
- 2 Tester une valeur par classe.

⇒ économies de DT.

Exemple: *abs* lit une valeur entière x et renvoie la valeur absolue de x .

Classes:	négatif < 0	égal à 0	positif > 0
DT:	-2	0	2

Difficulté: construire les classes d'équivalence.

Principe de la méthode

Objectif: réduire l'ensemble des DT.

- 1 Partitionner les données en classe d'équivalence tq le programme a le même comportement pour toutes les valeurs d'une classe
- 2 Tester une valeur par classe.

⇒ économies de DT.

Exemple: *abs* lit une valeur entière x et renvoie la valeur absolue de x .

Classes:	négatif < 0	égal à 0	positif > 0
DT:	-2	0	2

Difficulté: construire les classes d'équivalence.

Principes du Partitionnement

Analyse de la spécification \implies Partitionnement des données:

- 1 Selon relation entre entrée sorties.
- 2 Selon conformité des entrées:
 - Classe valide: données acceptables par le programme.
 - Classe invalide: données refusées par le programme.

Quelques pistes pour déterminer les classes.

- Condition spécifie un intervalle de valeur (compteur entre 1 et 99) alors 1 classe valide et 2 invalides.
- Condition spécifie un nombre de valeur (Ex: chacun des 3 articles peut être choisi), identifier 1 classe valide et deux invalides (0 articles, +de 3)
- Condition spécifie un ensemble de valeur (étudiants, étudiants boursier, professeur,...) et que le programme se conduit différemment sur chacune autant de classe que d'élément et une invalide
- Condition nécessaire ("tout nom commence par une lettre") induit 2 classes 1 valide, une invalide

Choix du Jeu de test:

- tant qu'une classe valide n'est pas couverte écrire un test la couvrant
- pour chaque classe non valide écrire un test qui ne couvre que celle-ci

Raffiner avec les cas limites

Principe: erreurs souvent liées aux cas limites donc utiliser ceux-ci comme DT.

Retourner la valeur absolue de x : $] -\infty, 0[$, 0 et $]0, +\infty[$.

Cas limites: -1 pour la première classe, 1 pour la troisième.

Exemples:

- 1 N entier maximal: N-1, N, N+1
- 2 Fichier: vide, taille maximale, trop grand
- 3 Indice d'un tableau: 0, indice maximal (indices valides)

Exercice: prise en compte de la taille maximale pour un entier.

Test exhaustif inapplicable: somme de 2 entiers sur 32 bits donne 2^{64} cas distincts.

Cas particuliers: entrées nombreuses mais domaines restreints (booleéens, domaine entier fini,...).

- commande avec options
- option de configurations (système, type de codage,...)
- interface

Déterminer des données de test permettant de tester toutes les paires de données.

- Nb de paires \ll Nb de combinaisons en général.
- Couvrir plusieurs paires avec une seule DT

Exemple: n variables booléennes

- Nb de combinaison possibles: 2^n
- Nb de paires possibles: $4n * (n - 1)/2$
- Une DT peut couvrir plusieurs paires

Exercice $n = 3$, variables X, Y, Z .

- 1 Donner les paires possibles.
- 2 Combien de paires couvre la DT $X = 1, Y = 1, Z = 1$?

Grphe de Causalité

Déterminer les DT à partir d'un graphe donnant les relations de causalité entre entités du problème.

- 1 Construire le graphe à partir des spécifications
- 2 Simplifier le graphe (si nécessaire)
- 3 Donner des DT permettant de couvrir le graphe (en limitant l'explosion combinatoire).

Un exemple

Une compagnie d'assurance a la politique suivante de renouvellement annuel de contrat:

- 0 accident et age ≤ 25 ans: augmenter de 50 euros
- 0 accident et age > 25 ans: augmenter de 20 euros
- 1 accident et age ≤ 25 ans: augmenter de 100 euros envoyer une mise en garde
- 1 accident et age > 25 ans: augmenter de 50 euros envoyer une mise en garde
- 2 à 4 accidents et age ≤ 25 ans: augmenter de 500 euros envoyer une mise en garde
- 2 à 4 accidents et age > 25 ans: augmenter de 250 euros envoyer une mise en garde
- Plus de 5 accidents: résiliation.

Graphe de causalité?

Déterminer **Causes**: avoir 1 accident

Déterminer **Effets**: augmenter de 250 euros

Causes:

- 1 avoir 0 accident, 1 accident, 2 à 4 accidents, > 5 accidents
- 2 avoir ≤ 25 ans, > 25 ans

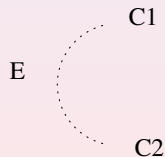
Effets:

- 1 augmenter de 20 euros, 50 euros, 250 euros, 500 euros,
- 2 envoyer une mise en garde,
- 3 résilier le contrat

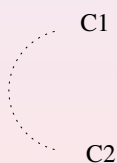
Conditions

Ajouter des liens entre conditions pour exprimer des contraintes.

Au plus un



Un et un seul



Implique

$C1 \dashrightarrow C2$

$C1=1$ alors $C2=1$

Exclut

$C1 \not\rightarrow C2$

$C1=1$ alors $C2=0$

Utilisation

- 1 Trouver les CT correspondant à un effet en remontant de l'effet aux causes (en suivant les arcs en AR).
- 2 Limitation de l'explosion combinatoire par utilisation de règles de choix.
- 3 Tableau cause/effet pour obtenir une suite de test.

Construction du tableau

Choisir un effet E et créer une colonne du tableau.

C1		...
.		
Cn		..
E1		
.		..
E	1	
.		..
Ep		..

Construction du tableau

Déterminer les conditions pour obtenir cet effet.

C1		...
Ci	1	
.		
Ck	1	
.		
Cn		...
<hr/>		
E1		
.		...
E	1	
.		...
Ep		...

Construction du tableau

Rajouter les autres effets induits par les conditions choisies.

C1		...
Ci	0	
.		
Ck	1	
.		
Cn		...
<hr/>		
E1		
.		...
E	1	
.		...
E'	1	
Ep		...