Test boîte noire

Boîte noire ou test fonctionnel

Principes de base

- Basés sur la définition de ce qu'est la spécification d'un programme en opposition à sa structure.
- La notion de couverture complète peut aussi s'appliquer au test fonctionnel.
- Des spécifications rigoureuses ont un autre bénéfice, elles aident au test fonctionnel, e.g., catégorisent les données, obtiennent les sorties anticipées.
- Autrement dit, elles aident les générations de cas de tests et les tests oracles.

Plan

- Partitionnement en classes d'équivalence
- Analyse de la valeur limite
- Test catégorie-partition
- Tables de décision
- Graphes de cause-à-effet
- Fonctions logiques

Partitionnement par classe d'équivalence

Test par classes d'équivalence

- Motivation : nous aimerions obtenir un sens de test complet et nous souhaiterions éviter la redondance.
- Classe d'équivalence : partage de l'ensemble des données.
- Ensemble des données entier est couvert : complet.
- Classes séparées : éviter la redondance.
- Cas de tests : un élément de chacune des classes d'équivalence.
- Mais les classes d'équivalence doivent être choisies prudemment …
- Estimer le comportement probable du système sous-jacent …

Test par classes d'équivalence faible/fort

- Trois variables d'entrée de domaines A, B, C :
  - A = A1 ? A2 ? A3 où a_n ?An
  - B = B1 ? B2 ? B3 ? B4 où b_n ?Bn
  - C = C1 ? C2 où c_n ?Cn

- Test par classes d'équivalence faible : choisir une forme de variable d'entrée pour chacune des classes d'équivalences :
  - max(|A|, |B|, |C|) cas de test

- Test par classes d'équivalence fort : basé sur le produit Cartésien des sous-ensembles de partition (A ? B ? C), i.e., teste toutes les interactions de classes :
  - |A| x |B| x |C| cas de test

Cas de test WECT

Cas de test	A	B	C
WE1	A1	B1	C1
WE2	A2	B2	C2
WE3	A3	B3	C1
WE4	A1	B4	C2

Cas de test SECT

Cas de Test	a	b	c
SE1	a1	b1	c1
SE2	a1	b1	c2
SE3	a1	b2	c1
SE4	a1	b2	c2
SE5	a1	b3	c1
SE6	a1	b3	c2
SE7	a1	b4	c1
SE8	a1	b4	c2
SE9	a2	b1	c1
SE10	a2	b1	c2
SE11	a2	b2	c1
SE12	a2	b2	c2
SE13	a2	b3	c1
SE14	a2	b3	c2
SE15	a2	b4	c1
SE16	a2	b4	c2
SE17	a3	b1	c1
SE18	a3	b1	c2
SE19	a3	b2	c1
SE20	a3	b2	c2
SE21	a3	b3	c1
SE22	a3	b3	c2
SE23	a3	b4	c1
SE24	a3	b4	c2

Exemple `NextDate`

- *`NextDate`* est une fonction avec trois variables : `MOIS`, `JOUR`, `ANNEE`. Elle retourne la date du jour après la donnée date. Limites : 1812-2012.

- Sommaire du traitement :
  - si ce n'est pas le dernier jour du mois, la dernière fonction date augmente simplement la valeur `JOUR`.
  - À la fin du mois, le jour suivant est 1 et la valeur `MOIS` est augmentée.
  - À la fin d'une année, les deux valeurs `JOUR` et `MOIS` sont remis à 1, et la valeur `ANNEE` est augmentée.
  - Finalement, le problème de l'année bissextile rend intéressant

  La détermination du dernier jour d'un mois.

Classes d'équivalence `NextDate`

- M1 = { MOIS: MOIS à 30 jours}
- M2 = { MOIS: MOIS à 31 jours}
- M3 = { MOIS: MOIS est Fevrier}
- D1 = {JOUR: 1<= JOUR <= 28}
- D2 = {JOUR: JOUR = 29}
- D3 = {JOUR: JOUR = 30}
- D4 = {JOUR: JOUR = 31}
- Y1 = {ANNEE: ANNEE = 1900}
- Y2 = {ANNEE: 1812 <= ANNEE <= 2012 et (ANNEE != 1900) et (ANNEE mod 4 = 0)}
- Y3 = {ANNEE: (1812 <= ANNEE <= 2012 et ANNEE mod 4 != 0)}

Cas de test `NextDate` *(1)*

- WECT: 4 cas de tests – partition maximum (D)

Cas ID:	**MOIS,**	**JOUR,**	**ANNEE, Sortie**
WE1:	6,	14,	1900, 6/15/1900
WE2:	7,	29,	1912, 7/30/1912
WE3:	2,	30,	1913, Donnée invalide
WE4:	6,	31,	1900, Donnée invalide

- SECT: 36 cas de tests >> WECT

NextDate SECT: 36 cas de tests

Cas ID	Mois	Jour	An	Sortie anticipée
SE1	6	14	1900	6/15/1900
SE2	6	14	1912	6/15/1912
SE3	6	14	1913	6/15/1913
SE4	6	29	1900	6/30/1900
SE5	6	29	1912	6/30/1912
SE6	6	29	1913	6/30/1913
SE7	6	30	1900	7/1/1900
SE8	6	30	1912	7/1/1912
SE9	6	30	1913	7/1/1913
SE10	6	31	1900	ERREUR
SE11	6	31	1912	ERREUR
SE12	6	31	1913	ERREUR
SE13	7	14	1900	7/15/1900
SE14	7	14	1912	7/15/1912
SE15	7	14	1913	7/15/1913
SE16	7	29	1900	7/30/1900
SE17	7	29	1912	7/30/1912
SE18	7	29	1913	7/30/1913
SE19	7	30	1900	7/31/1900
SE20	7	30	1912	7/31/1912
SE21	7	30	1913	7/31/1913
SE22	7	31	1900	8/1/1900
SE23	7	31	1912	8/1/1912
SE24	7	31	1913	8/1/1913
SE25	2	14	1900	2/15/1900
SE26	2	14	1912	2/15/1912
SE27	2	14	1913	2/15/1913
SE28	2	29	1900	ERREUR
SE29	2	29	1912	3/1/1912
SE30	2	29	1913	ERREUR
SE31	2	30	1900	ERREUR
SE32	2	30	1912	ERREUR
SE33	2	30	1913	ERREUR
SE34	2	31	1900	ERREUR
SE35	2	31	1912	ERREUR
SE36	2	31	1913	ERREUR

Discussion

- Si des conditions d'erreurs sont une haute priorité, nous devrions étendre le test par classes d'équivalence fort en incluant les classes invalides.
- L'ECT est approprié quet les données d'entrée sont définies en termes de domaines et d'ensembles de valeurs discrètes.
- Le SECT fait la supposition que les variables sont indépendantes – les dépendances génèreront des "erreurs" de cas de test.
- Possiblement trop …
- Voir les sections suivantes : catégorie-partition et techniques de tables de décision …

Analyse des valeurs limites

Motivations

- Nous avons partitionné les domaines de données en classes appropriées, sous la supposition que le comportement du programme est “similaire”
- Quelques erreurs typiques de programmation se passent à la limite entre les différentes classes.
- C’est sur quoi le test de valeur limite se concentre.
- Plus simple mais complémentaire aux techniques précédentes.

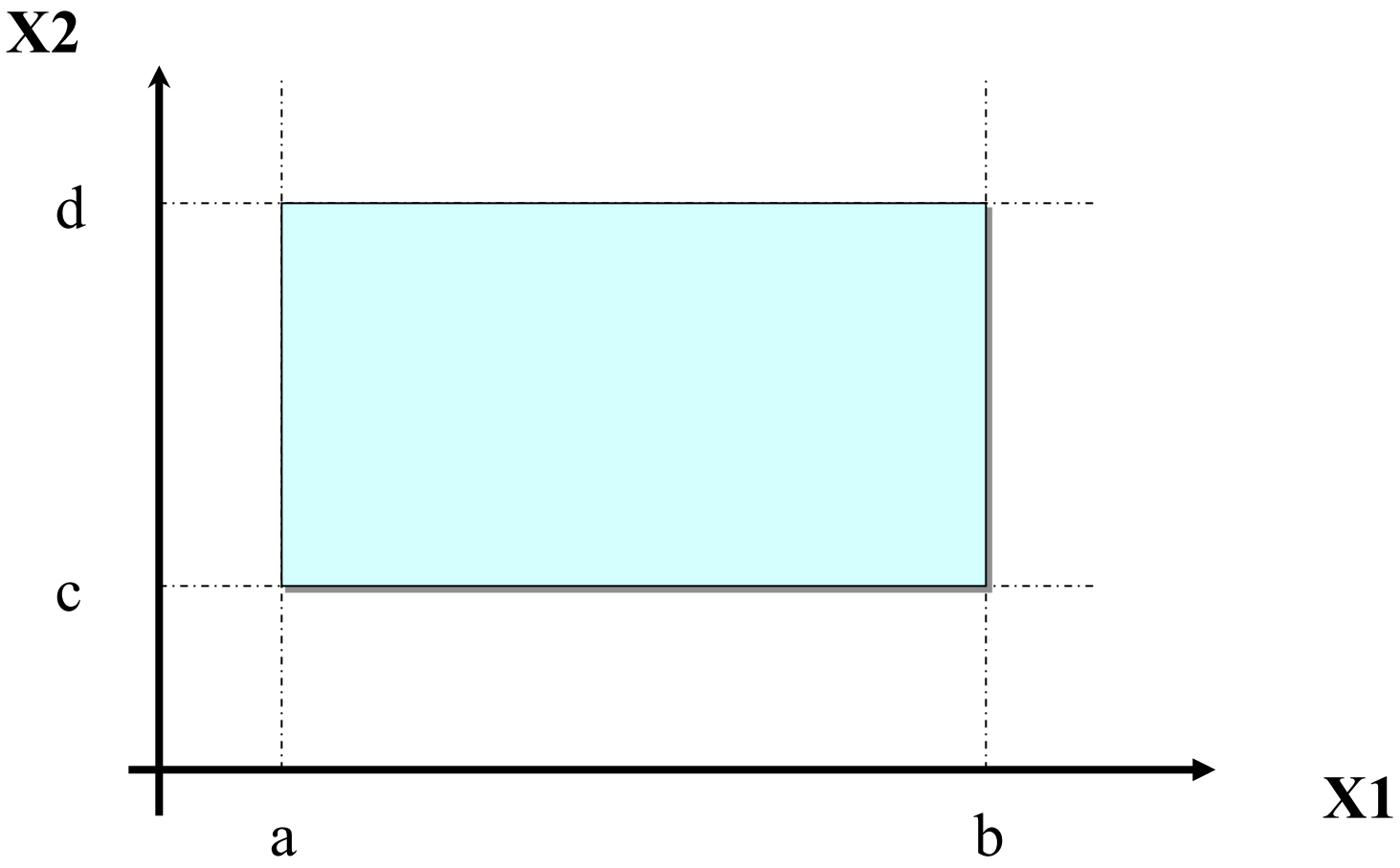
Analyses des valeurs limites

- Supposons une fonction F, avec deux variables x1 et x2.
- (Possiblement sans état) Limites :
  a <= x1 <= b, c <= x2 <= d.
- Dans quelques langages de programmation, un fort typage permet la spécification de tels intervalles.
- Se concentre sur les bornes de l'espace d'entée pour identifier les cas de test.
- Le raisonnement est que les erreurs tendent à se produire extrêmement près des valeurs des variables d'entée – ceci est supporté par quelques études.

Idées fondamentales

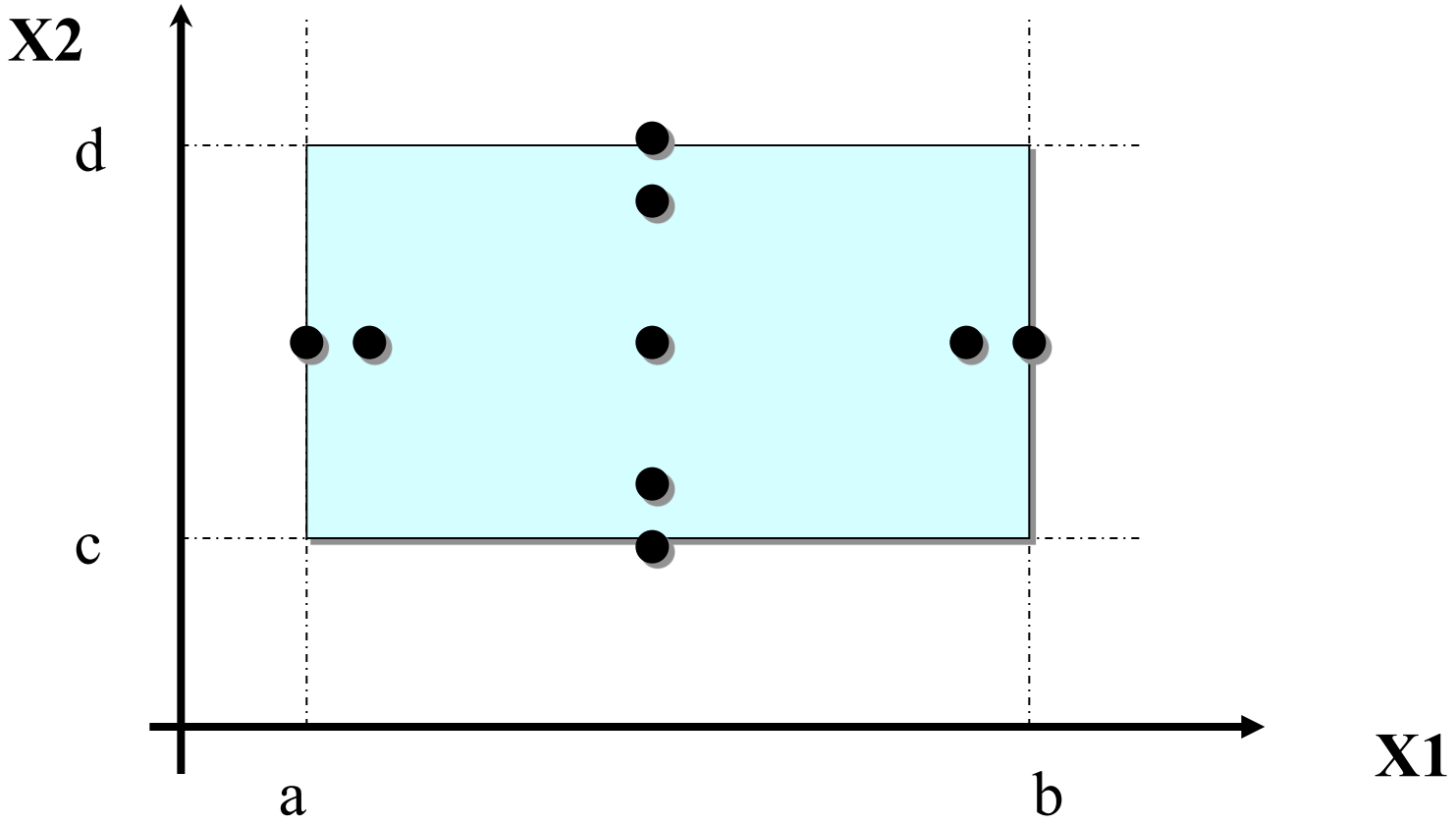
- Les valeurs des variables d'entrée à leur minimum, juste au-dessus du minimum, à une valeur nominale, juste en dessous de leur maximum, et à leur maximum.
- Convention : min, min+, nom, max-, max
- Garde les valeurs de tout sauf une seule variable à leurs valeurs nominales, laissant une seule variable suppose sa valeur extrême.

Domaine d'entrée de la fonction F



Analyse de la limite des cas de test

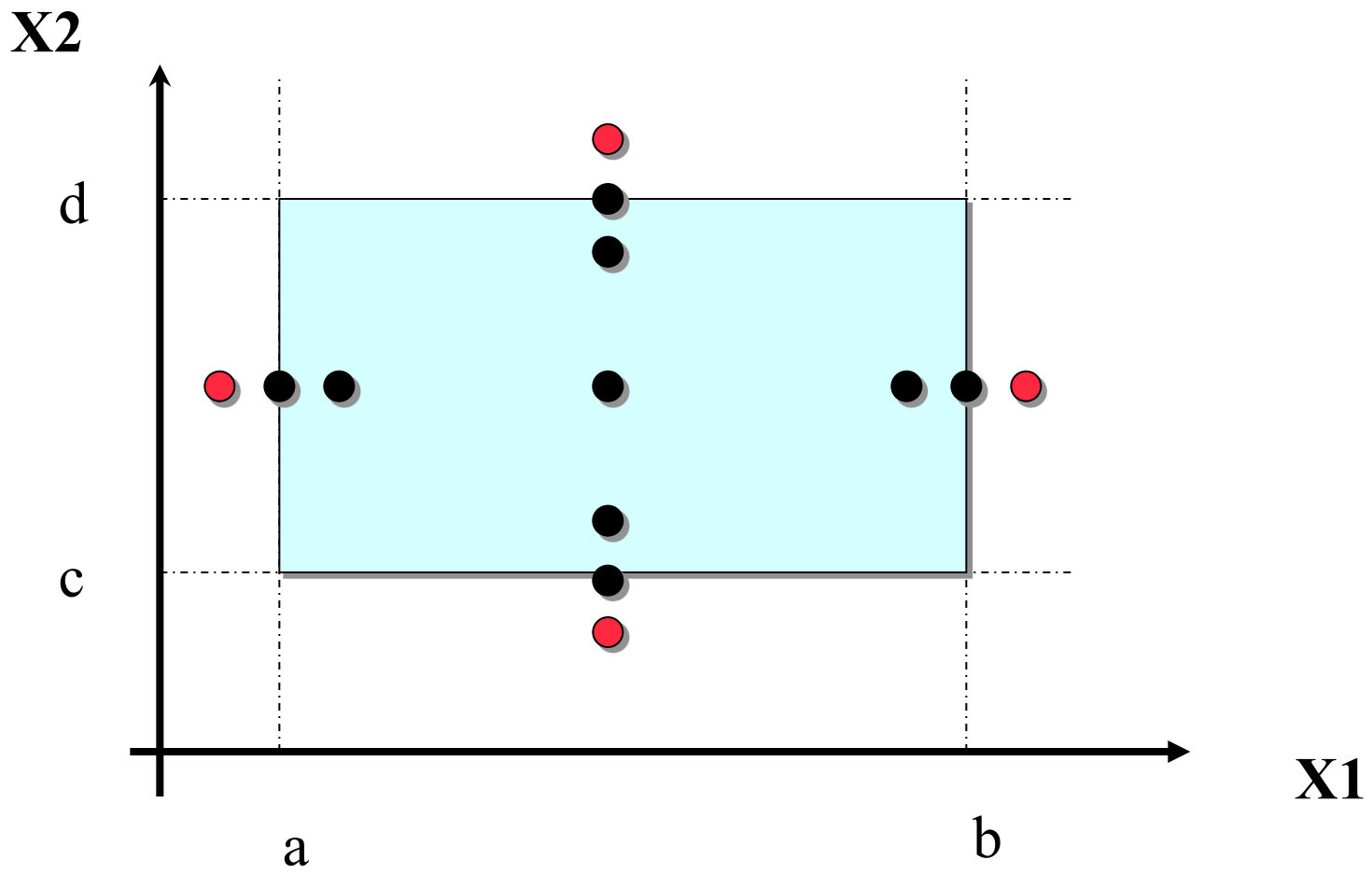
- Ensemble de tests ={<$x1_{nom}$, $x2_{min}$>, <$x1_{nom}$, $x2_{min+}$>, <$x1_{nom}$, $x2_{nom}$>, <$x1_{nom}$, $x2_{max-}$>, <$x1_{nom}$, $x2_{max}$>, <$x1_{min}$, $x2_{nom}$,>, <$x1_{min+}$, $x2_{nom}$,>, <$x1_{max-}$, $x2_{nom}$>, <$x1_{max}$, $x2_{nom}$>}



Cas général et limitations

- Une fonction avec n variables nécessitera 4n + 1 cas de tests.
- Fonctionne bien avec les variables qui représentent des quantités physiques limitées.
- Sans considération de la nature de la fonction et le sens des variables.
- Une technique rudimentaire qui est adéquate au test de robustesse.

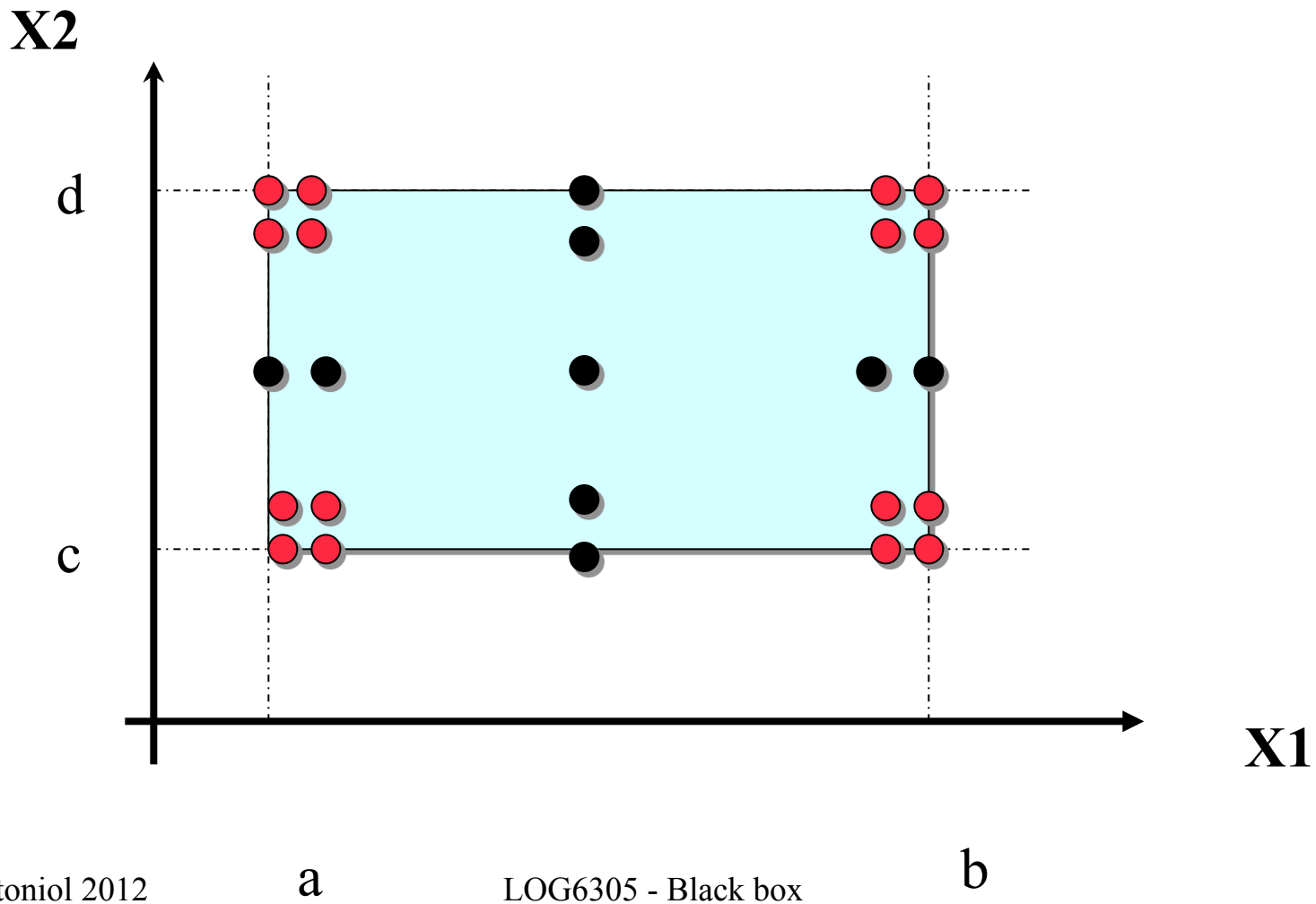
Test de robustesse



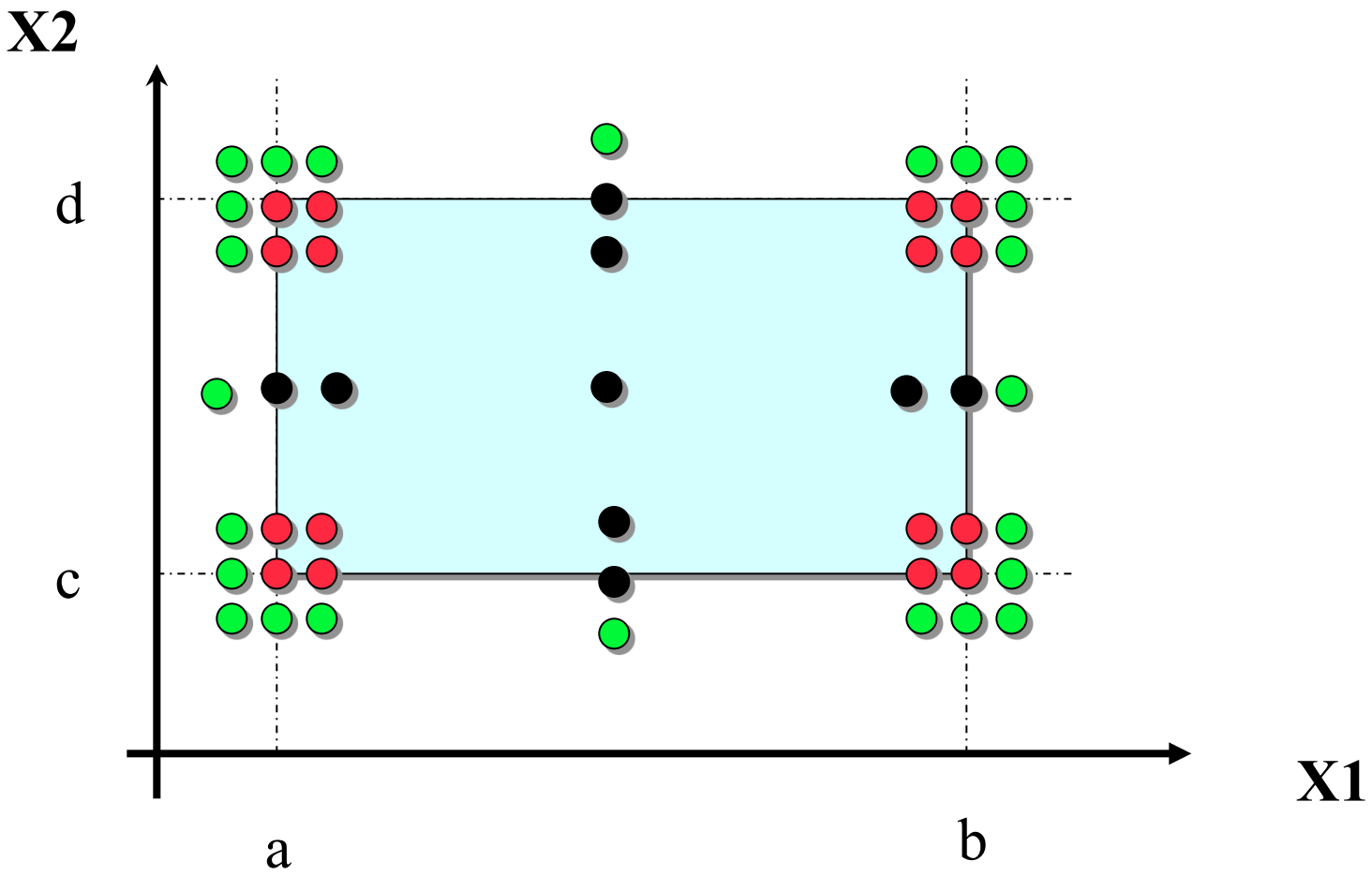
Test des pires cas (WCT)

- La valeur limite fait la supposition commune que les défaillances, la plupart du temps, proviennent d'une faute.
- Qu'arrive-t-il quet plus d'une variable ont une valeur extrême ?
- Une idée provient d'électroniques dans l'analyse de circuit.
- Produit Cartésien de {min, min+, nom, max-, max}.
- Clairement plus minutieusement que l'analyse de la valeur de limite mais beaucoup plus d'effort : 5^n cas de test.
- Une bonne stratégie quet des variables physiques ont de nombreuses interactions et où la défaillance est coûteuse.
- Encore plus loin : test robuste de pire cas (RWCT).

WCT pour 2 variables



WCT robuste pour 2 variables



Test catégorie-partition

Étapes

- Le système est divisé en “fonctions” individuelles qui peuvent être testées individuellement.
- La méthode identifie les *paramètres* de chaque “fonction” et, pour chaque paramètre, identifie des *catégories* distinctes.
- En plus des paramètres, les *caractéristiques environnementales,* sous lesquelles la fonction opère (caractéristiques de l’état du système), peuvent aussi être considérées.
- Les *catégories* sont des propriétés majeures ou caractéristiques.
- De plus, les *catégories* sont subdivisées en *choix* de la même manière que le partitionnement par classe d’équivalence qui est appliqué (“valeurs” possibles).

Petit exemple

- Fonction : tri d'une matrice.
- Caractéristiques :
  - longueur de la matrice (Len),
  - type d'éléments,
  - valeur max,
  - valeur min,
  - position de la valeur max (Max pos),
  - position de la valeur min.
- Choix pour Max pos : {1, 2..Len-1, Len}.

Étapes II

- Les *contraintes* opérant entre les choix sont alors identifiées, i.e., comment l'occurrence d'un choix peut affecter l'existence d'un autre.
  - E.g., dans un exemple de tri de tableau, si Len = 0, alors le reste n'a pas d'importance.
- Les *trames de test* "*test frames"* sont générés, ce qui consiste en combinaisons admissibles de choix dans les catégories (test de spécifications).
- Les trames de test sont alors converties en données de test.

Contraintes

- Les *propriétés, les sélecteurs* associés avec les choix.

Catégorie A :

Choix A1 [propriété X, Y, Z],

Choix A2.

Catégorie B :

Choix B1,

Choix B2 [si X et Z].

Annotation spéciale : [Erreur], [Simple]

Exemple complet

- Spécification :
  - Le programme demetant à l'usager de taper un nombre entier positif dans un champ de 1 à 20 et une chaîne de caractères de cette longueur.
  - Le programme, alors, demete de taper un caractère et retourne la position dans la chaîne où le caractère a été trouvé en premier ou un message indiquant que ce caractère n'était pas présent dans la chaîne.
  - L'usager a l'option de chercher d'autres caractères.

Paramètres et catégories

- Trois paramètres : nombre entier x (longueur), la chaîne a et le caractère c.
- Pour x, les catégories sont dans la séquence “in-range” (1-20) ou hors de la séquence “out-of-range”.
- Les catégories pour a : minimal, maximal, longueur intermédiaire.
- Les catégories pour c : le caractère apparaît au commencement, au milieu, à la fin de la chaîne, ou ne se produit pas dans la chaîne.

Choix

- Nombre entier x, hors de la séquence (out-of-range) : 0, 21.
- Nombre entier x, dans la séquence (in-range) : 1, 2-19, 20.
- Chaîne a : 1, 2-19, 20.
- Caractère c : premier, milieu, dernier, ne se produit pas.
- Remarque : parfois, il n'y a qu'un choix dans la catégorie.

Spécifications du test formel

x:

x1)	0	[erreur]
x2)	1	[propriété chaineok, longueur1]
x3)	2-19	[propriété chaineok, midlongueur]
x4)	20	[propriété chaineok, longueur20]
x5)	21	[erreur]

a:

a1)	longueur 1	[si chaineok et longueur1]
a2)	longueur 2-19	[si chaineok et midlongueur]
a3)	longueur 20	[si chaineok et longueur20]

c:

c1)	à la premiere first position dans la chaine [si chaineok]
c2)	à la derniere position dans la chaine [si chaineok et not longueur1]
c3)	au milieu de la chaine [si chaineok et not longueur1]
c4)	pas dans la chaine [si chaineok]

Trames de test et de cas de test

x 1	x = 0
x 2a1c1	x = 1, a = 'A' , c = 'A'
x 2a1c4	x = 1, a = 'A' , c = 'B'
x 3a2c1	x = 7, a = 'ABCDEFG' , c = 'A'
x 3a2c2	x = 7, a = 'ABCDEFG' , c = 'G'
x 3a2c3	x = 7, a = 'ABCDEFG' , c = 'D'
x 3a2c4	x = 7, a = 'ABCDEFG' , c = 'X'
x 4a3c1	x = 20, a = 'ABCDEFGHIJKLMNOPQRST' , c = 'A'
x 4a3c2	x = 20, a = 'ABCDEFGHIJKLMNOPQRST' , c = 'T'
x 4a3c3	x = 20, a = 'ABCDEFGHIJKLMNOPQRST' , c = 'J'
x 4a3c4	x = 20, a = 'ABCDEFGHIJKLMNOPQRST' , c = 'X'
x 5	x = 21

Critères utilisant les choix

- Toutes les combinaisons (AC) : c'est ce qui était montré dans l'exemple précédent, ce qui est typiquement fait et utilisé dans catégorie-partition. Une valeur pour chaque choix de chaque paramètre doit être utilisée avec une valeur de chaque choix (possible) de chaque autre catégorie.

- Chaque choix (EC) : c'est un critère plus faible. Une valeur de chaque choix pour chaque catégorie doit être utilisée dans un cas de test.

- Choix de base (BC) : ce critère est un compromis. Un choix de base est choisi pour chaque catégorie, et un premier test de base est formé en utilisant le choix de base pour chaque catégorie. Des tests ultérieurs sont choisis en retenant tout sauf un choix de base constant (i.e., nous sélectionnons un choix non-de-base pour une catégorie) et formant des combinaisons de choix en couvrant tous les choix non-de-bases de la catégorie sélectionnée. Cette procédure est répétée pour chaque catégorie.

- Le choix de base peut être le plus simple, le plus petit, le premier dans une liste selon un certain ordre, le plus vraisemblable selon un point de vue d'utilisateur final, e.g., dans l'exemple précédent, le caractère c se produit dans le milieu de la chaîne, la longueur x est entre 2-19.

Conclusions

- L'identification des paramètres et des conditions d'environnement, et des catégories dépend fortement de l'expérience du testeur.
- Rend le test des décisions explicite (e.g., contraintes), prêt pour l'évaluation.
- Combine l'analyse des valeurs limites, le test de robustesse et le partitionnement par classes d'équivalences.
- Un fois que la première étape est complétée, la technique est simple et peut être automatisée.
- La technique pour la réduction de cas de test le rend utile pour le test pratique.

Tables de décision

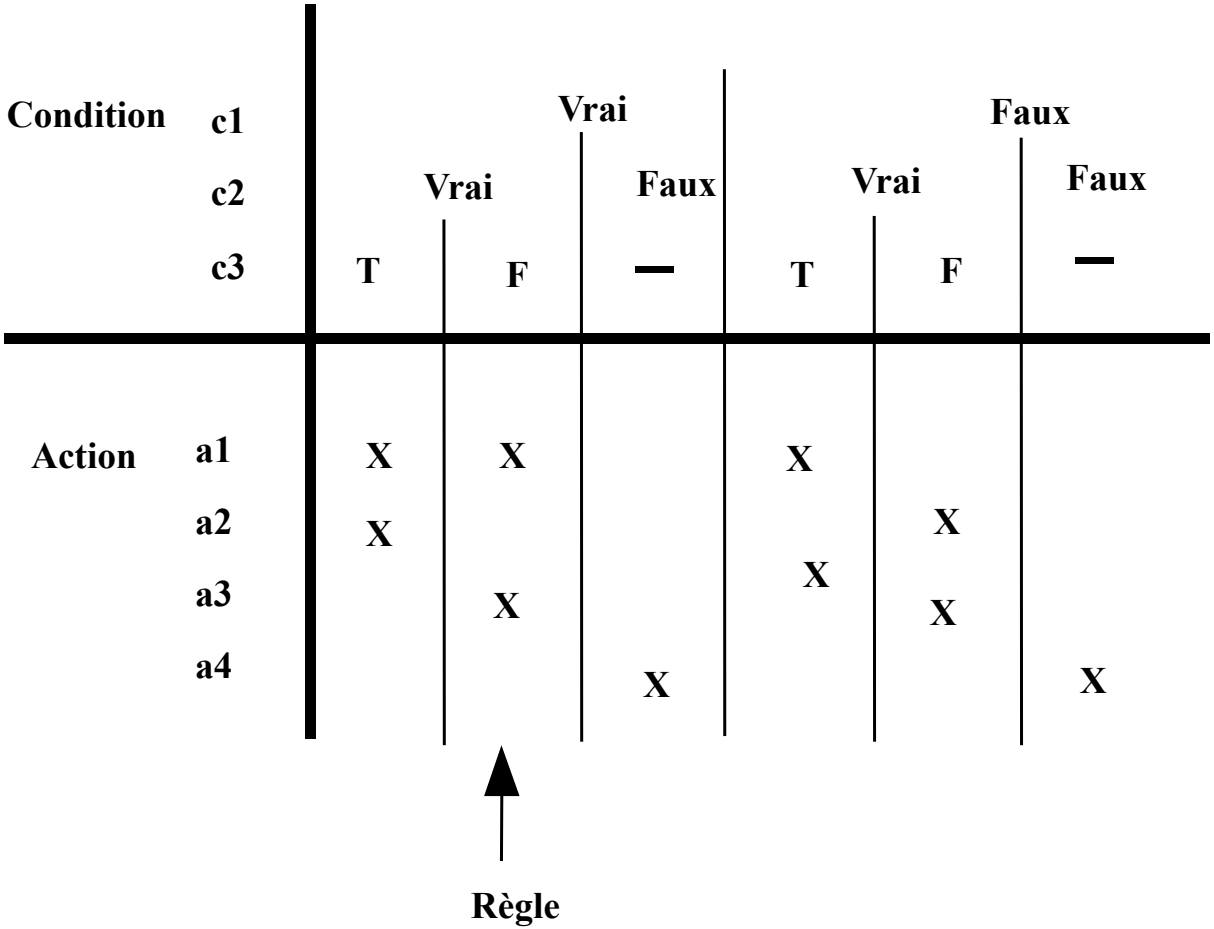
Motivations

- Aide à exprimer les spécification de test directement dans une forme utilisable.
- Facile à comprendre et supporte la dérivation systématique des tests.
- Supporte la génération des cas de test automatiques ou manuelles.
- Une réponse particulière ou un sous-ensemble de réponse doit être choisie en évaluant plusieurs conditions reliées.
- Idéal pour décrire les situations dans lesquelles un nombre de combinaisons d'actions sont prises selon des ensembles variables de conditions , e.g., systèmes de contrôle.

Structure

- La section condition liste les *conditions* et les combinaisons correspondantes
- La condition exprime l'association entre les *variables de décision.*
- La section action liste les *réponses* à être produites quand les combinaisons correspondantes des conditions sont vraies.
- Limitations : les actions résultantes sont déterminées par les valeurs courantes des variables de décision!
- Les actions sont *indépendantes* de l'ordre de des entrées et de l'ordre dans lequel les conditions sont évaluées.
- Les actions peuvent apparaître plus d'une fois mais chaque combinaison de conditions est unique.

Table de structure



Exemple de table

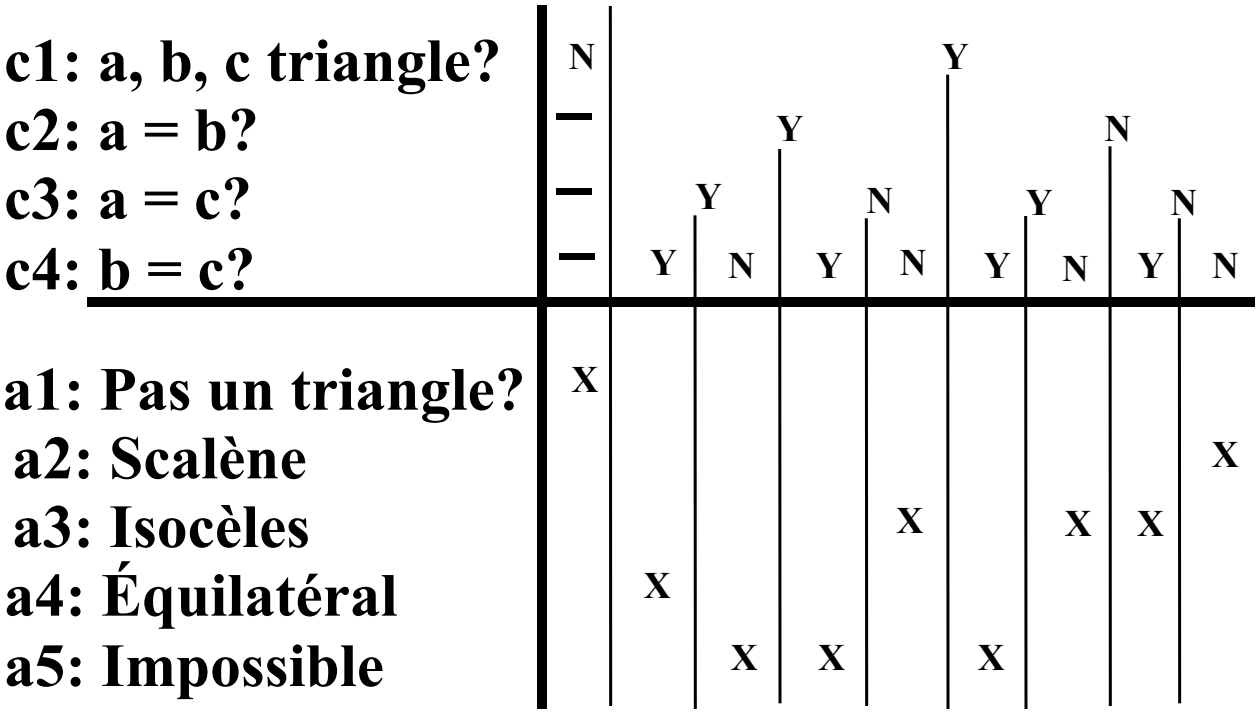


Table de vérité

conditions											
c1: a < b + c?	F	T	T	T	T	T	T	T	T	T	T
c2: b < a + c?	-	F	T	T	T	T	T	T	T	T	T
c3: c < a + b?	-	-	F	T	T	T	T	T	T	T	T
c4: a = b?	-	-	-	T	T	T	T	F	F	F	F
c5: a = c?	-	-	-	T	T	F	F	T	T	F	F
c6: b = c?	-	-	-	T	F	T	F	T	F	T	F
a1: Pas un triangle	X	X	X								
a2: Scalène											X
a3: Isocèles							X		X	X	
a4: Équilatéral				X							
a5: Impossible					X	X		X			

Cas de test

Cas ID	a	b	c	Sortie anticipée
TC1	4	1	2	Pas un triangle
TC 2	1	4	2	Pas un triangle
TC 3	1	2	4	Pas un triangle
TC 4	5	5	5	Équilatéral
TC 5	?	?	?	Impossible
TC 6	?	?	?	Impossible
TC 7	2	2	3	Isocèles
TC 8	?	?	?	Impossible
TC 9	2	3	2	Isocèles
TC 10	3	2	2	Isocèles
TC 11	3	4	5	Scalène

Conditions d'utilisation idéale

- Une de plusieurs réponses distinctes sera sélectionnée en fonction des cas distincts de variables d'entrée.
- Ces cas peuvent être modelés par des expressions de booléennes mutuellement exclusives utilisant les variables d'entrées.
- La réponse qui est produite ne dépend pas de l'ordre dans lequel les variables d'entrée sont arrangées ou évaluées (e.g., les événements sont reçus).
- La réponse ne dépend pas des entrées et sorties prioritaires.

Échelle

- Pour n conditions, il y a peut-être au plus 2^n *variantes* (combinaisons uniques de conditions et actions).
- Mais, heureusement, il y a d'habitude beaucoup moins de variantes *explicites* …
- Les valeurs "Don't care" dans les tables de décision aide à réduire le nombre de variantes.
- "Don't care" peut correspondre à plusieurs cas :
  - les données sont nécessaires mais n'ont pas d'effet ;
  - les données peuvent être omises ;
  - les cas mutuellement exclusifs (exclusions type-sécurité).

Cas spéciaux

- La condition "*can't happen*" reflète quelques suppositions que plusieurs entrées sont mutuellement exclusives, ou qui ne peuvent pas être produites dans l'environnement.
- Une source chronique de bogues, e.g., Ariane 5.
- "*can't happen*" se produit à cause des erreurs de programmation et des effets de changements inattendus.
- La condition "*don't know*" reflète un modèle incomplet, e.g., dû à une documentation incomplète.
- La plupart du temps, ils sont des bogues de spécification.

Graphes de cause-effet

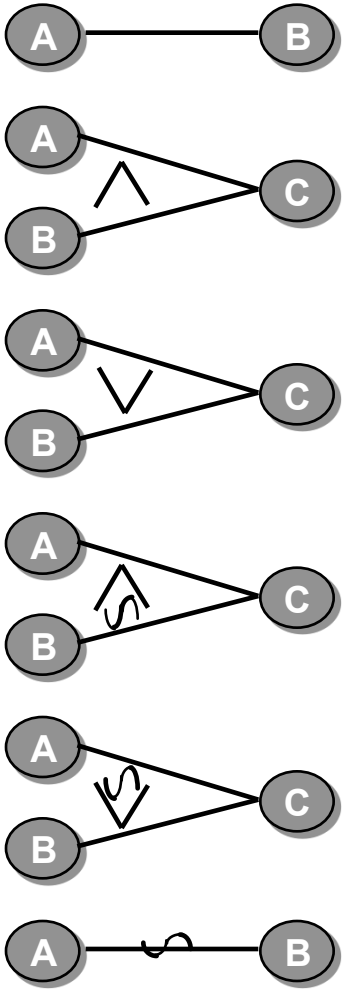
Définition

- Une technique graphique qui aide à dériver les tables de décision.
- Vise à supporter l'interaction avec les experts de domaine et l'ingénierie inverse des spécifications, dans le but de tester.
- Identifie les causes (conditions de entrées, impulsions) et les effets (sorties, changements dans l'état du système).
- Les causes doivent être formulées d'une manière pour être soit vraies ou fausses (expression booléenne).
- Précise explicitement les contraintes (environnementales, externes) des causes et des effets.
- Aide à sélectionner des sous ensembles de combinaisons de sous-ensembles "significatifs" d'entrées-sorties et à construire de plus petites tables de décision.

Structure des graphes de cause-effect

- Un nœud est tiré pour chaque cause et chaque effet.
- Les nœuds sont placés sur les côtés opposés de la feuille.
- Une ligne de la cause à l'effet indique que la cause est une condition nécessaire pour l'effet.
- Si un simple effet a deux causes ou plus, le rapport logique des causes est annoté par des symboles pour un *et* logique (^) et *ou* logique (?) placés entre les lignes.
- Une cause dont la négation est nécessaire est désignée par un *non* logique (~).
- Une cause simple peut être nécessaire pour plusieurs effets; un effet simple peut avoir plusieurs causes nécessaires.
- Les nœuds intermédiaires peuvent être utilisés pour simplifier le graphe et sa construction.

Remarque



Si A alors B

"*et*" : Si (A et B) alors C

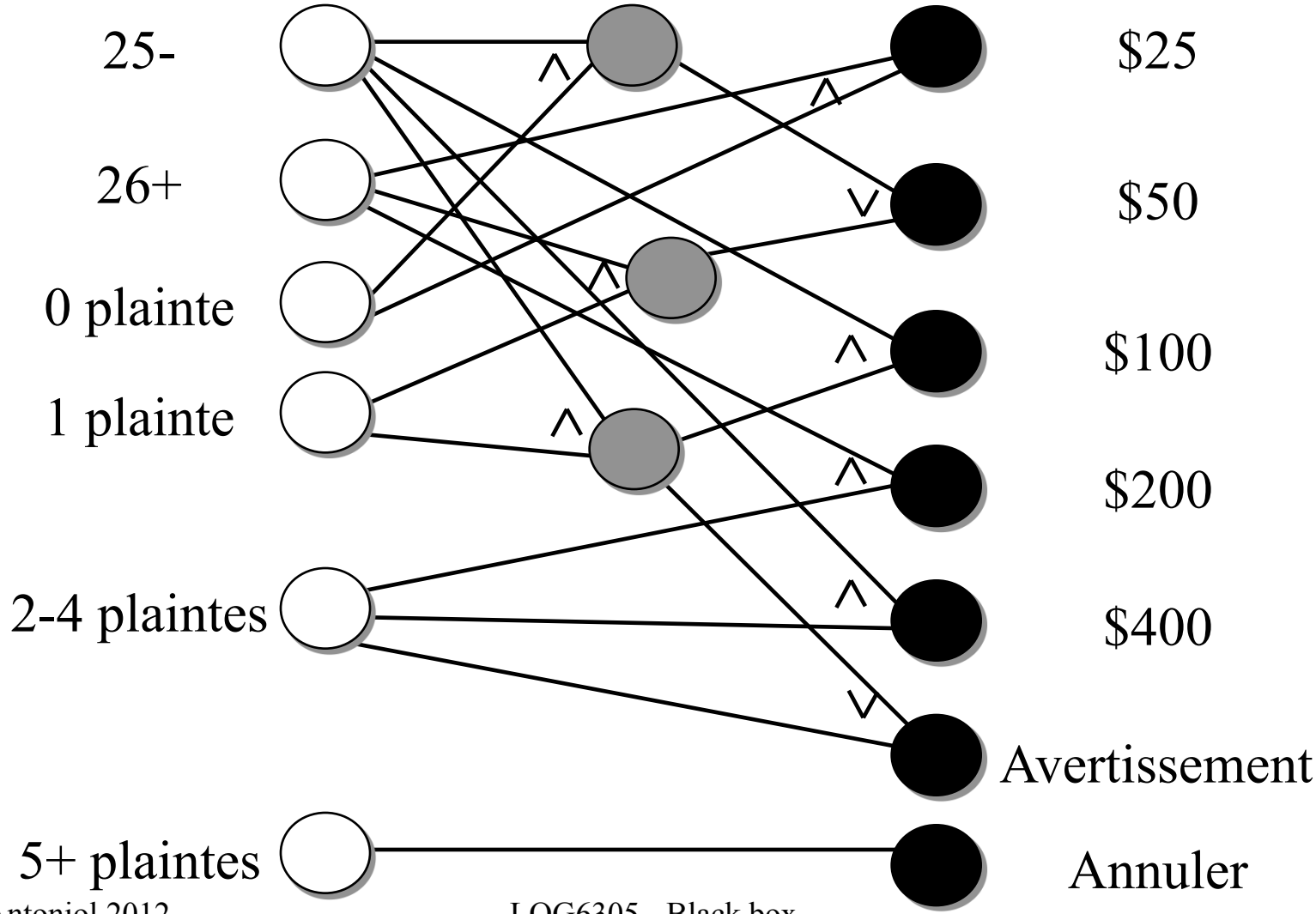
"*OU*" : Si (A ou B) alors C

"*Net*" : Si pas (A et B) alors C

"*NOU*" : Si (ni A ou B) alors C

"*NON*" : Si (pas A) alors B

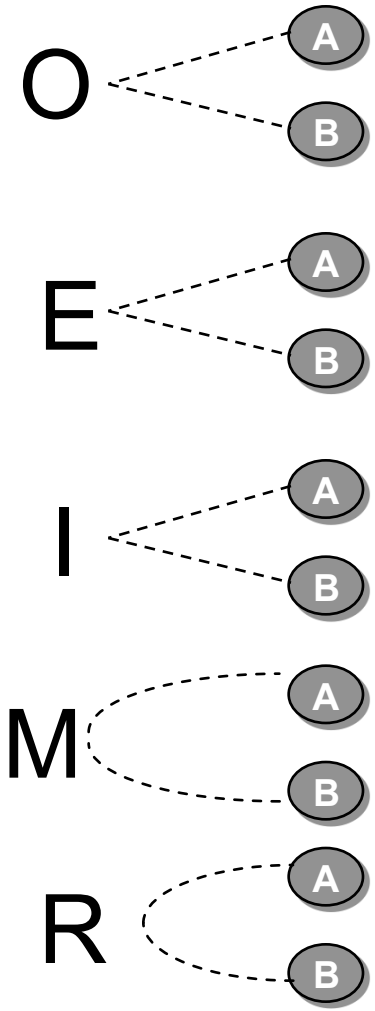
Exemple de renouvellement d'assurance



Autre exemple : Table de renouvellement d'assurance

	Section condition		Section action		
Variable	**Plaintes**	**Âge**	**Prime augmentée $**	**Envoie d'un avertissement**	**Annulation**
1	0	25-	50	Non	Non
2	0	26+	25	Non	Non
3	1	25-	100	Oui	Non
4	1	26+	50	Non	Non
5	2 à 4	25-	400	Oui	Non
6	2 à 4	26+	200	Oui	Non
7	5+	Tout	0	Non	Oui

Contraintes additionnelles



"*EXACEMENT UN*" de A et B doit être vrai.

"*AU PLUSUN*" de A et B doit être vrai.

"AU MOINS UN" de A et B peut être vrai.

"*A MASQUE B*", I.e., A => PAS B

"*A `REQUIERE B*", I.e., A => B

Autre exemple

- Donnée : La syntaxe de la fonction est "LEVEL(A,B)" où A est la hauteur en mètre de l'eau derrière le barrage et B est le nombre de centimètres en pluie dans la dernière période de 24 heures.
- Processus : La fonction calcule si le niveau de l'eau est (1) à sa mesure normale, (2) trop haute, (3) trop basse.
- Sorties : un des messages suivants :
  - LEVEL = SAFE (for normal et low)
  - LEVEL = HIGH
  - INVALID SYNTAX

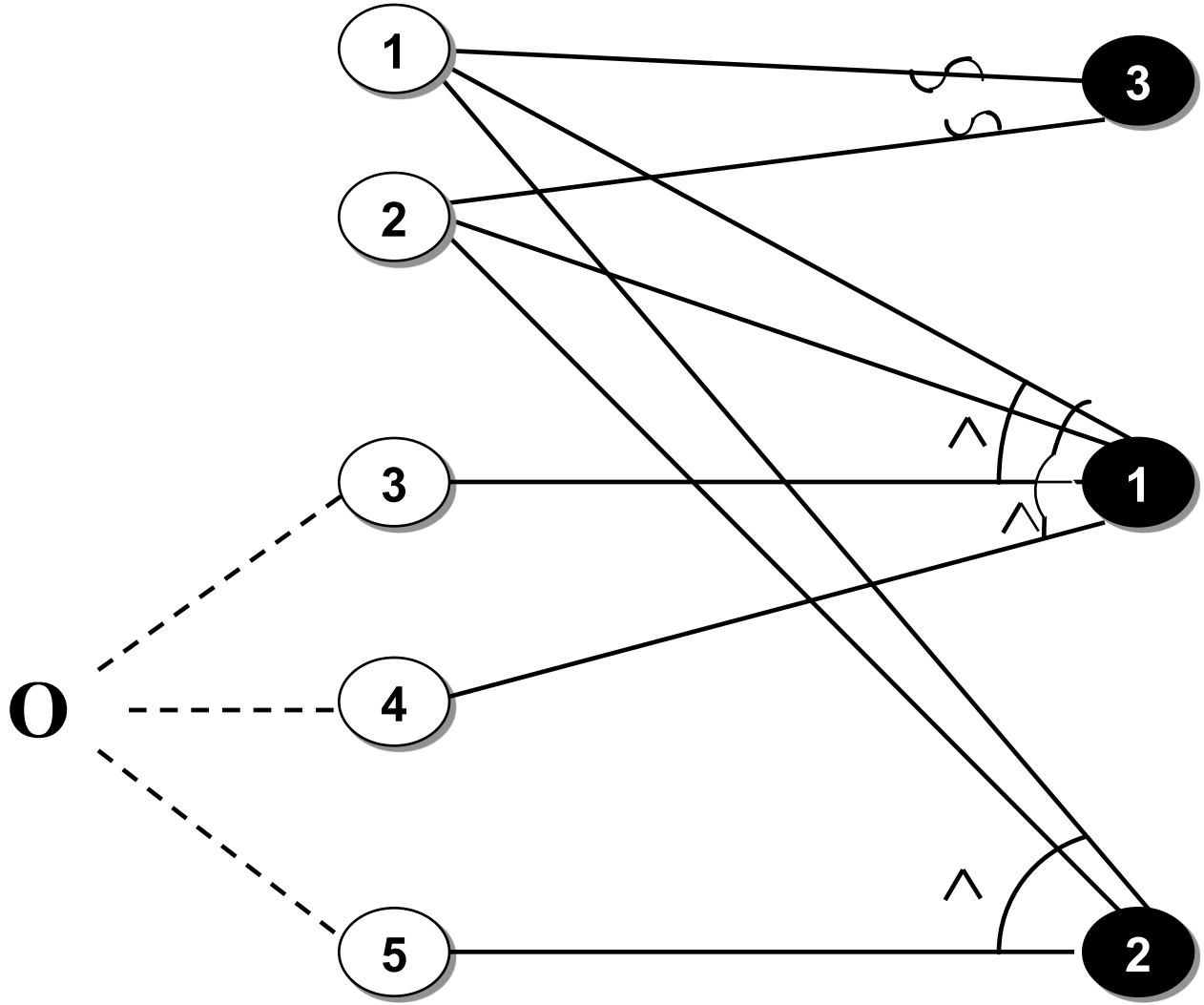
Identifier les causes

1. Les cinq premiers caractères de la commande : “LEVEL”.
2. Les deux paramètres séparés par une virgule et entourés par des parenthèses.
3. Les paramètres A et B sont des nombres réels de telle manière que le niveau de l’eau est calculé pour être “LOW”.
4. Les paramètres A et B sont des nombres réels de telle manière que le niveau de l’eau est calculé pour être “NORMAL”.
5. Les paramètre A et B sont des nombres réels de telle manière que le niveau de l’eau est calculé pour être “HIGH”.

Identifier les effets

1. “LEVEL = SAFE” est affiché à l’écran.
2. “LEVEL = HIGH” est étalé sur l’écran.
3. “INVALID SYNTAX” est imprimé.

Graphe cause-effect pour LEVEL



Dériver une table de décision

- Une ligne pour chaque cause ou effet.
- Les colonnes correspondent aux cas de tests (variables).
- Nous définissons les colonnes en examinant chaque effet et en listant toutes les combinaisons (conjonctions) des causes qui peuvent mener à cet effet.
- E.g., deux lignes séparées mène vers l'effet E3, chacune correspondant à un cas de test, quatre lignes mènent vers E1 mais ne correspondent qu'à deux combinaisons seulement.

Niveau de la table de décision

Table de décision pour un graphe de cause-et-effet

	Test 1	Test 2	Test 3	Test 4	Test 5
Cause 1	**T**	**T**	**T**	**F**	**T**
Cause 2	**T**	**T**	**T**	**-**	**F**
Cause 3	**T**	**F**	**F**	**-**	**-**
Cause 4	**F**	**T**	**F**	**-**	**-**
Cause 5	**F**	**F**	**T**	**-**	**-**
Effet 1	**P**	**P**	**A**	**A**	**A**
Effet 2	**A**	**A**	**P**	**A**	**A**
Effet 3	**A**	**A**	**A**	**P**	**P**

Discussion

- Le graphe cause-effet peut être utilisé pour générer toutes les combinaisons *possibles* de causes et vérifier si l'effet correspond à la spécification.
- Il fournit un test oracle et spécifie les contraintes sur les sorties (effets), aidant à détecter les mauvais états du système et les combinaisons d'action.
- Si le graphe est trop large, pour chaque combinaison admissible d'effets, on trouve quelques combinaison de causes qui déclenchent les combinaisons d'effets en remontant vers le graphe.
- À cause de contraintes additionnelles sur le graphe, on peut être plus restrictif que les tables de décision classiques.

Fonctions logiques

Définitions

- Un prédicat est une expression qui évalue une valeur booléenne.
- Les prédicats peuvent contenir des variables booléennes, des variables non booléennes qui sont comparées avec les opérateurs comparateurs {>, <, =, …}, et les appels de fonction (retournent une valeur booléenne).
- La structure interne du prédicat est créée par les *opérateurs logiques* {non, et, ou, …}.
- Une *clause* est un prédicat qui ne contient aucun des opérateurs logiques, e.g., (a<b).
- Les prédicats peuvent être écrits de différentes manières logiques équivalentes (algèbre booléenne).

Définitions II

- Une fonction logique relie n variables d'entrée booléennes (clauses) à une seule variable de sortie booléenne.
- Pour rendre les expressions plus simple à lire, nous utiliserons la contiguïté (AB) pour l'opérateur *et*, + pour (A+B) l'opérateur *ou* et un ~ pour l'opérateur de négation.
- Exemple :
  mettre en état ou hors état l'ignition d'une chaudière se basant sur quatre variable d'entrée :
  - NormalPressure (A) : la pression dans une limite d'opération sécuritaire ?
  - CallForHeat (B) : la température ambiante sous le point de consigne ?
  - DamperShut (C) : le conduit du tuyau d'échappement est fermé ?
  - ManualMode (D) : sélection du manuel d'opération ?
- Fonction logique : Z = A(B~C+D) ~> Table de vérité.

Table de vérité de la chaudière

Numéro du vecteur d' entrées	Normalpressure	CallForHeat	DamperShut	ManualMode	Ignition
	A	B	C	D	Z
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0

Table de vérité de la chaudière II

Numéro du vecteur d'enrées	NormalPressure	CallForHeat	DamperShut	ManualMode	Ignition
	A	B	C	D	Z
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Éléments d'expressions booléennes

- *L'espace booléen* :
  L'espace n-dimensionnel formé par les variables d'entrées.
- *Le terme produit ou clause conjonctive* :
  Chaîne de clauses reliées par l'opérateur *et.*
- *La somme de produits ou forme normale disjonctive(DNF)* :
  Termes produit reliées par l'opérateur *ou.*
- *L'implicant* :
  Chaque terme de l'expression somme de produit – condition suffisante pour nécéssaire pour la sortie *True* de cette expression.
- *Les implicants premiers* :
  Un implicant ou aucun sous-ensemble (sous-terme propre) ne soit aussi un implicant.
- *La minimisation de la logique* :
  dérivation expressions booléennes compactes (non redondante) mais équivalentes en utilisant l'algèbre booléenne.

Exemple de la chaudière

- La fonction logique : Z=A(B~C+D).
- La forme somme de produits (DNF) :

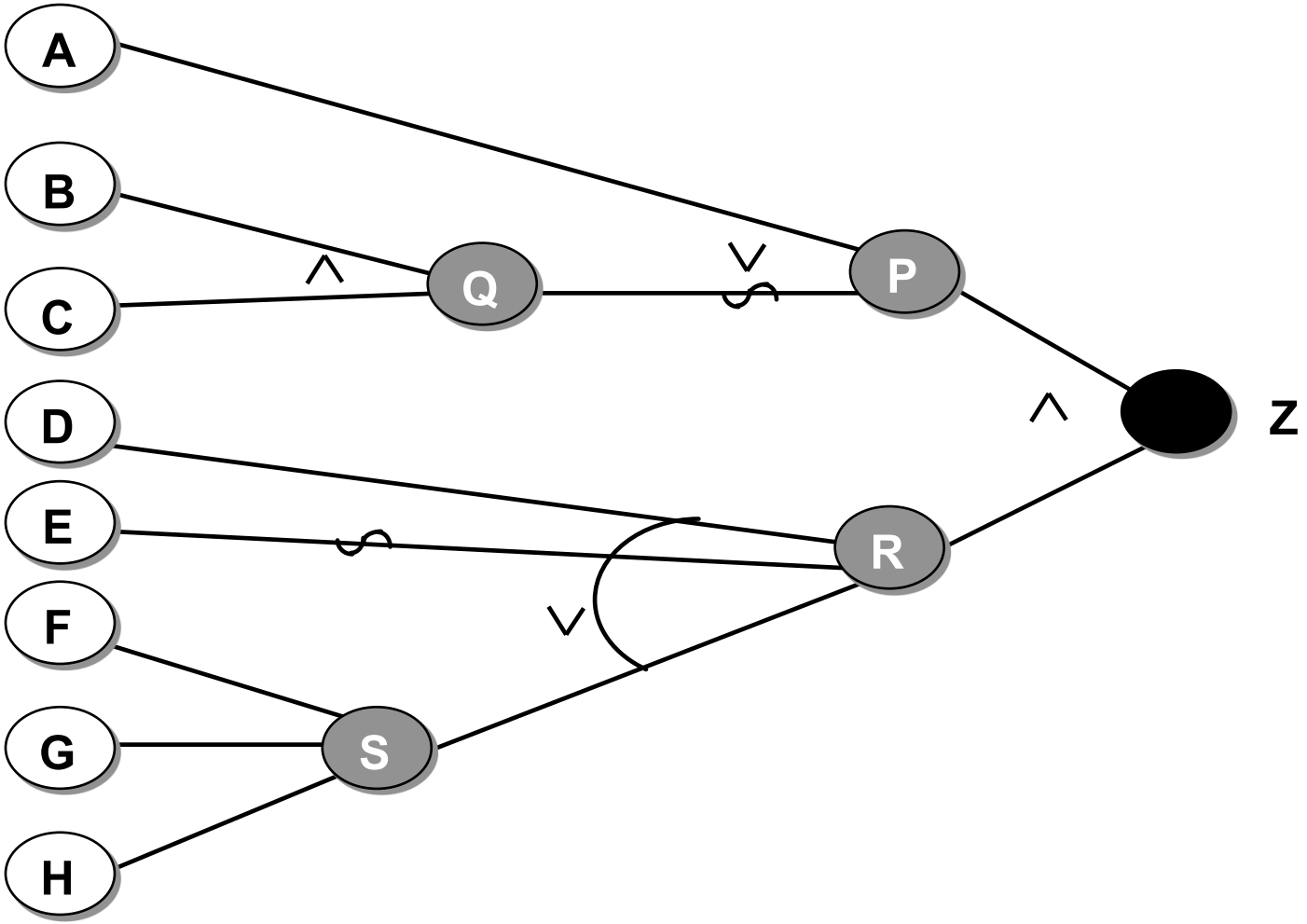
  Z=A(B~C+D) = AB~C+AD.

- Les implicants : AB~C, AD.
- Les implicants premiers : AB~C = TTFx = {TTFT, TTFF}, AD=TxxT={TFFT, TFTT, TTFT, TTTT} => les deux termes sont des implicants premiers.

D'un graphe à une fonction logique

- Une fois le graphe cause-effet est révisé et considéré correct, nous voulons dériver une fonction logique dans le but de dériver les spécifications du test (dans la forme d'une table de décision).
- Une fonction (prédicat, table de vérité) existe pour chaque effet (variable de sortie).
- Si plusieurs effets sont présents, alors la table de décision résultante est un composite de plusieurs tables de vérité qui partagent la décision / les variables d'entrée et les actions / les effets.
- Il est plus facile de dériver séparément une fonction pour chaque effet.
- Dérive une fonction booléenne à partir d'un graphe d'une manière systématique.

Exemple



Générer une fonction logique

- Générer une fonction initiale :
  - commencer à partir du nœud de l'effet,
  - retourner en arrière à travers le graphe,
  - substituer les clauses de niveau supérieur avec celles du niveau inférieur ainsi que les expressions booléennes, jusqu'à ce que vous atteigniez les nœuds de la cause.
- Transformer en une minimale, forme DNF :
  - utiliser la loi de l'algèbre booléenne pour réduire les expressions booléennes ;
  - Ré-exprimer sous forme de somme-de-produits (forme normale disjonctive),
  - il existe des algorithmes pour faire cela automatiquement.

Exemple

- Z = PR (effet)
- P = A + ~Q (intermédiaire)
- Q = BC (intermédiaire)
- R = D + ~E + S (intermédiaire)
- S = F + G + H (intermédiaire)
- Z = (A + ~(BC)) (D + ~E + (F+G+H)) (substitution)
- Z=(A+~B+~C)(D+~E+F+G+H) (Loi de De Morgan)
- Z = AD + A~E + AF + AG + AH + ~BD + ~B~E + ~BF + ~BG + ~BH + ~CD + ~C~E + ~CF + ~CG + ~CH (La loi distributrice est utilisée pour obtenir la somme de produits).

Lois d'algèbre booléenne

- Loi d'associativité :
  - (A+B)+C = A+(B+C), (AB)C = A(BC)
- Loi de la distributivité :
  - A+(BC) = (A+B)(A+C), A(B+C) = AB+AC
- Lois de De Morgan :
  - ~(A+B)=~A~B, ~(AB)=~A+~B
- Loi d'absorption :
  - A + AB = A
  - A(A+B) = A
  - A+(~AB) = A+B, A(~A+B) = AB
  - AB+AC+B~C = AC+B~C

Modèle de fautes pour le test basé sur la logique

- Faute de négation d'expression (ENF) : Une fonction logique est implémentée comme sa négation.
- Clause de négation de faute (CNF) : Une clause dans un terme particulier est remplacée par sa négation.
- Faute d'omission de terme (TOF) : Un terme particulier dans une fonction logique est omis.
- Faute de référence d'opérateur (ORF):
  Un opérateur binaire "*ou*" dans une fonction logique est implémenté comme "*et* " ou vice-versa.
- Faute d'omission de clause (COF) :
  Une clause dans un terme particulier d'une fonction logique est omise.
- Faute d'insertion de clause (CIF) :
  Une clause n'apparaissant pas dans un terme particulier d'une fonction logique est insérée dans ce terme.
- Faute de référence de clause (CRF) :
  Une clause dans un terme particulier d'une fonction logique est remplacée par une autre clause n'apparaissant pas dans le terme.

Critères de base de test

- L'objectif est de tester une implémentation et être certain qu'elle est cohérente avec sa spécification, comme modelé par la fonction logique (ou graphe).
- Il existe un nombre de critères de couverture de test qui ne suppose pas une forme normale disjonctive de prédicats:
  - la couverture de prédicats,
  - la couverture de clauses,
  - la couverture combinatoire,
  - la couverture de clauses (in)actives.
- Notation : P est un ensemble de prédicats, C est un ensemble de clauses et P, Cp est l'ensemble de clauses dans un prédicat p.

Couverture de prédicats

- Couverture de prédicats :*Pour p ∈ P, nous avons deux conditions requises de test : p évalué à vrai, et p évalué à faux.*
- Pour A(B~C+D), les deux tests qui satisfont la couverture des prédicats sont : s

  (1) (A=vrai, B=vrai, C=faux, D=faux),

  (2) (A=faux, B=faux, C=vrai, D=vrai).
- Problème : les clauses individuelles ne sont pas testés.

Couverture de clauses

- Couverture de clauses : *Pour chaque c ∈ C, nous avons deux conditions requises de test : c évalué à vrai et c évalué à faux..*

- Pour (A+B)C, deux tests qui satisfont la couverture de clauses :

  (1) (A=vrai, B=vrai, C=faux),

  (2) (A=faux, B=faux, C=vrai).

- Remarque : la couverture de clause ne subsume pas la couverture de prédicats ou vice-versa.

Exemple

- Z = A+B
- t1 = (A = vrai; B = vrai) => Z
- t2 = (A = vrai; B = faux) => Z
- t3 = (A = faux; B = vrai) => Z
- t4 = (A = faux; B = faux) => ~Z
- Si nous choisissons la paire de cas de test T1 = {t1; t2}, il ne satisfait ni la couverture de clauses (parce que A n'est jamais faux) ni la couverture de prédicats (parce que Z n'est jamais faux).
- Ensemble de test T2 = {t2; t3} satisfait la couverture de clauses, mais pas la couverture de prédicats (parce que Z n'est jamais faux).
- Ensemble de test T3 = {t2; t4} satisfait la couverture de prédicats, mais pas celle de couverture de clauses (parce que B n'est jamais vrai).
- Ensemble de test T4 = {t1; t4} est la seule paire satisfaisant les deux (couverture de clauses et couverture de prédicats).

Couverture combinatoire

- Couverture combinatoire : *Pour chaque p ∈ P, nous avons des conditions requises de test pour les clauses dans Cp pour évaluer chaque combinaison possible des valeurs de vérité.*

- Subsume la couverture de prédicats

- Il y a $2^{|Cp|}$ possible assignations de valeurs de vérité.
-

- Problème : Non pratique pour les prédicats avec plus de quelques clauses.

Effets de masquage

- Quand nous introduisons des tests au niveau des clauses, nous voulons avoir un effet sur les prédicats.
- Les expressions logiques (clauses) peuvent se masquer les unes les autres.
- Dans l'attribut AB, si B = *faux*, B peut masquer A, parce que peu importe la valeur de A, AB sera toujours *faux.*
- Nous avons besoin de considérer les circonstances sous lesquelles une clause affecte la valeur d'un prédicat, pour détecter des défaillances possibles d'implémentation.

Détermination

- Détermination : *Étant donné une clause c_i dans un prédicat p, appelée clause majeure, nous disons que c_i détermine p si le restant des clauses mineures c_j [?] p, j <> i ont des valeurs de telle manière qu'en changeant la valeur de vérité de c_i, on change la valeur de vérité de p.*

- Nous aimerions tester chaque clause sous les circonstances où il détermine le prédicat.

- L'ensemble de test T4 dans un transparent précédent satisfait les deux couvertures de prédicats et de clauses mais ne teste ni A ni B efficacement.

Couverture de clauses actives

- **Couverture de clauses actives (ACC) :**

  Pour chaque p [?] P et chaque clause majeure c_i [?] Cp, choisir des clauses mineures c_j, j <> i de telle façon que c_i détermine p. Nous avons 2 conditions requises de test pour chaque c_i : c_i évaluée à vraie et c_i évaluée à fausse.
- Par exemple, pour Z = A+B, nous terminons avec un total de quatre conditions requises de test, deux pour la clause A, deux pour la clause B.
- Pour la clause A, A détermine Z si, et seulement si, B est fausse. Donc, nous avons les deux conditions requises de test {(*A = vraie; B = fausse*) *;* (*A = fausse; B = fausse*)}.
- Pour la clause B, B détermine Z si, et seulement si, A est fausse. Donc, nous avons les deux conditions requises de test {(*A = fausse; B = vraie*)*;* (*A = fausse; B = fausse*)}, la dernière est commune avec A.
- ACC est presque identique à MCDC dans la couverture de code.
- Les questions les plus importantes sont que (1) ACC devrait subsumer PC, (2) les clauses mineures c_j ont besoin d'avoir les mêmes valeurs quand la clause majeure c_i est vraie ainsi quand c_i est fausse.

ACC corrélées (CACC)

- Pour chaque p ? P et chaque clause majeure c_i ? Cp, choisir les clauses mineures c_j, j <> i de telle façon que c_i détermine p. Il y a deux conditions requises de test pour chaque c_i : c_i évaluée à vraie et c_i évaluée à fausse. Les valeurs choisies pour les clauses mineures c_j doivent causer p à être vrai pour une valeur de la clause majeure c_i et faux pour l'autre, aussi, il est requis que p(c_i = vraie) <> p(c_i = fausse).

- CACC est subsumé par la couverture combinatoire de clauses et subsume la couverture de clauses/prédicats.

ACC restreintes (RACC)

- Pour chaque p ? P et chaque clause majeure c_i ? Cp, choisir des clauses mineures c_j, j <> i afin que c_i détermine p. Il y a deux conditions requises de test pour chaque c_i : c_i évaluée à vraie et c_i évaluée à fausse. Les valeurs choisies pour les clauses mineures c_j doivent être les même quand c_i est vraie que quand c_i est fausse, il est requis que $c_j(c_i = \text{vraie}) = c_j(c_i = \text{fausse})$ pour tous les c_j.
- RACC le rend plus facile de déterminer la cause du problème que CACC, si l'une est détectée : clause majeure.
- Mais est ce qu'il est fréquent en spécification d'avoir des contraintes entre les clauses, rendant RACC impossible d'être réalisée ?
- Ceci correspond à MCDC pour la couverture de code.

Exemplexxx

- Z=A(B+C)
- Il serait possible de satisfaire la couverture de clauses corrélées actives en respectant la clause *A* avec les deux conditions requises de test :

  {(A = vraie; B = vraie; C = fausse);

  (A = fausse; B =fausse; C = vraie)}

- Mais ça ne satisfait la RACC :

  {(A = vraie; B = vraie; C = fausse);

  (A = fausse; B = vraie; C = fausse)}

- Ce cas est facile …

Couverture de clauses inactives

- Le critère de la couverture de clauses actives vise à rendre certain que les clauses majeures affectent leurs prédicats. Un critère complémentaire à la couverture de clauses actives assure que changer une clause majeure qui ne devrait pas affecter le prédicat n'affecte pas, en fait, le prédicat.
- **Couverture de clauses inactives (ICC) :**

  Pour chaque p ? P et chaque clause majeure c_i ? Cp, choisir des clauses mineures c_j , j <> i de façon que c_i ne détermine pas p. Il y a quatre conditions requises de test pour c_i sous ces circonstances : (1) c_i évaluée à vraie avec p vrai, (2) c_i évaluée à fausse avec p vrai, (3) c_i évaluée à vraie avec p faux, et (4) c_i évaluée à fausse avec p faux.

- ICC est subsumée par la couverture de clauses combinatoires et subsume la couverture de clauses/prédicats.

Critères de couverture de forme normale disjonctive

- Ici, les critères supposent que les prédicats ont été ré-exprimés sous une forme normale disjonctive (DNF).
- Ce qui est intéressant avec DNF sont les critères qui l'accompagnent.
- Critères :
  - la couverture des implicants,
  - la couverture des implicants premiers,
  - la stratégie de négation de variables.

Couverture d'implicants (IC)

- IC : *Étant donné les représentations de DNF d'un prédicat p et sa négation ~p, pour chaque implicant, une condition test requise est que l'implicant est évalué à vrai.*
- Ceci teste différentes situations de test dans lesquels une action devrait (ou ne devrait pas) être prise (e.g., une chaudière allumée).
- p : AB+B~C.
- ~p (une représentation) : ~B+~AC.
- Quatre implicants : {AB, B~C, ~B, ~AC}.
- Plusieurs ensembles de tests satisfont ce critère, exemple, pour ABC, respectivement, nous pouvons utiliser {TTF, FFT}.
- IC subsume la couverture de prédicats, mais pas nécessairement les la couverture de clauses actives.

Problèmes avec

- Un problème avec l'IC est que les tests doivent être choisis de façon qu'un simple test satisfait des implicants multiples (voir l'exemple précédent).
- Quoique cela laisse les testeurs minimiser la dimension des jeux de tests, il est une mauvaise chose de tester les contributions uniques que chaque implicant peuvent apporter à un prédicat.
- Ainsi, nous introduisons une méthode pour forcer une genre d^"indépendance" des implicants.

Implicants premiers

- La première étape est d'obtenir une forme DNF où chaque implicant peut être satisfait sans satisfaire aucun autre implicant.
- Heureusement, des approches stantards existent déjà et peuvent être utilisées. Un *sous-terme propre* d'un implicant est l'implicant avec une ou plusieurs clauses omises.
- Un *implicant premier* est un implicant dont aucun sous-terme propre de l'implicant est aussi un implicant.
- Exemple : ABC+AB~C+B~C.
- ABC n'est pas un implicant premier parce que un sous-terme propre (AB) est aussi un implicant.

Couverture de l'implicant premier (PIC)

- Assumons que notre attribut de DNF contient seulement des implicants primaires.
- Un implicant est redondant s'il peut être omis sans changement de la valeur de l'attribut.
- Dans AB+AC+B~C, AB est redondant.
- PIC : *Étant donné des représentations DNF non redondantes d'implicants premiers d'un prédicat p et sa négation ~p, pour chaque implicant, une conditions requises de test est que l'implicant est évalué à vrai, pendant que les autres implicants sont évalués à faux.*

Exemple & discussion PIC

- p : AB+B~C.
- ~p: ~B+~AC.
- Les deux sont des représentations de l'implicant premier non redondant.
- L'ensemble de tests suivants satisfait le PIC: {TTT, FTF, FFF, FTT}.
- Le PIC est un critère de couverture puissant : aucun critère de la couverture de clauses subsume le PIC.
- Quoique jusqu'aux 2^{n-1} implicants premiers, plusieurs prédicats génèrent un nombre modeste de tests pour le PIC.
- C'est une question ouverte à savoir si le PIC subsume tous les critères de la couverture de clauses.

Stratégie de négation de variables

- Aller même plus loin que le PIC.
- *Points uniques vrais* : variantes qui rendent un et seulement un terme produit vrai.
  - E.g., (TTFF) pour le premier terme produit dans l'exemple de la chaudière (AB~C), AD est faux
- *Points presque faux* : variantes pour chaque terme produit où une clause est rendue négative de telle manière que la fonction logique globale est évaluée fausse.
  - E.g., (TTTF) pour AB~C où ~C est négatif.
- De telles variantes constituent les ensembles de tests candidats (TCS).
- Générer le TCS pour chaque terme produit dans une fonction logique.
- Le jeu de test est formée en choisissant le plus petit jeu qui couvre tous les TCSs.

Discussion

- Si une implémentation du terme produit n'est pas évaluée vraie quand elle devrait – impliquant que au moins une clause dans ce terme produit n'est pas évaluée vraie quand c'est prévu– les cas de test de la TCS *(points uniques vrais)* correspondant au terme seront capables de le détecter, sans l'effet masquant des autres clauses ou termes.
- Si une implémentation du terme produit n'est pas évaluée fausse quand elle devrait, c'est que la négation d'une (au moins) clause n'a pas l'effet attendu sur la fonction logique (faux), les cas de test du TCS (*point presque faux)* correspondant à la clause négative seront capables de le détecter, sans effet masquant des autres clauses ou termes.
- Dans une étude par Weyuker et al, approximativement *6 %* du jeu de test du test "All-Variant" (2^n) sont nécessaire pour satisfaire les *critères de négation de variables*.

Exemple de la chaudière

- Rendre AB~C vrai mais pas AD : un point unique vrai est (TTFF), ou (1100) dans la forme binaire, ou {12} dans la forme décimale.
- Rendre AD vrai mais pas AB~C : ensemble de points uniques vrais {9,1,15}.
- Les points presque faux pour AB~C :{14}, {8}, {4,5} pour la négation ~C, B, et A, respectivement.
- Les points presque faux pour AD :{1, 3, 5, 7}, {8, 10, 14} pour la négation A et D, respectivement.
- Générer l'ensemble des matrices de variables et sélectionner le jeu de tests en couvrant tous les ensembles de candidats {*} au-dessus.
- Parce qu'une variable peut appartenir à plus d'un ensemble de candidats, le nombre de tests requis peut être moins que le nombre d'ensembles de tests candidats.

Matrice de l'ensemble de variables

Var	1	2	3	4	5	6	7	TCS
0								
1							x	
2								
3							x	
4				x				
5				x			x	S
6								
7							x	
8			x			x		S
9					x			
10						x		
11					x			S
12	x							S
13								
14		x				x		S
15					x			

Stratégie VN versus fautes

- Faute de négation d'expression (ENF) :
  Tout point dans l'espace booléen.
- faute de négation de clause (CNF) :
  Tout point unique vrai ou point presque faux pour le terme défectueux et la clause négative.
- Faute d'omission de terme (TOF) :
  Tout point unique vrai pour le terme défectueux.
- Faute de la référence d'opérateur (ORF) :
  - "*ou*" implémenté comme "*et*" : tout point unique vrai d'un de deux termes.
  - "*et*" implémenté comme "*ou*" : tout point presque faux d'un de deux termes.
- Faute d'omission de clause (COF) :
  Tout point presque faux pour le terme défectueux et la clause omise.
- Faute d'insertion de clause (CIF) :
  Tous les points presque faux et les points uniques vrais pour le terme défectueux.
- Faute de référence de clause (CRF) :
  Tous les points presque faux et les points uniques vrais pour le terme défectueux.

Étude de Weyuker et al

- TCASII, système d'évitement de collision aérienne.
- 20 prédicats/fonctions logiques forment les spécifications (dans une notation diagramme d'état modifiée).
- En moyenne 10 clauses distinctes par expression.
- Cinq opérateurs de mutation, définis pour les expression booléennes, nous les utilisons pour produire des fautes dans les spécifications.
- La sélection aléatoire de cas de tests (même dimension) mène à un score moyen de mutation de 42.7%.
- La stratégie de négation de variable est donc beaucoup mieux avec une moyenne de 97.9%.

Sommaire du test fonctionnel

- Toutes les techniques voient un programme comme une fonction mathématique qui relie des entrées aux sorties.
- En ordre de supériorité : (1) l'analyse de la valeur limite, (2) le test par classes d'équivalence, (3) Catégorie-partition (4) les graphes cause-effet.
- (1) Mécanique, (2) invention des classes d'équivalence, (3) partitions, catégories et dépendances logiques (4) dépendances logiques entre les causes elles-mêmes et, entre les causes et les effets.
- Moins de cas de tests avec (3) or (4).
- Compromis entre l'effort d'identification de test et l'effort d'exécution de test.