

Test catégorie-partition

Test catégorie-partition: Étapes

- ❑ Le test par catégorie-partition intègre les classes d'équivalence, les valeurs limites et rajoute
 - Des dépendances explicites entre classes d'équivalence (ici des "choix")
 - Des caractéristiques externes (*environnement* sous lequel la fonction opère etc.)
- ❑ 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.
- ❑ 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).

Test catégorie-partition: 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 générées, 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.

A2 et B2 ne vont
pas être combinés

Catégorie B :

Choix B1,
Choix B2 [si X et Z].

Annotation spéciale : [Erreur], [Simple]

Pour des choix qui peuvent être testés avec un seul cas de test, c.-à-d. qui n'ont pas à être combinés avec d'autres choix.

Petit exemple

Un système électronique contrôlant l'accès à un coffre-fort est relié à trois panneaux mécaniques. Le premier panneau (étiqueté S) affiche une des 10 premières lettres de l'alphabet, soit de A à J; les 2^{ème} (étiqueté A) et 3^{ème} (étiqueté N) panneaux affichent chacun un entier entre 0 et 9. Sur la base de ces trois panneaux, un code d'accès numérique est généré. Ce code est le produit des deux entiers et de la position (dans l'ordre alphabétique) de la lettre du panneau S.

Par exemple, pour $S = B$, $A=5$, $N=4$, le code est $2*5*4=40$.

Les panneaux sont soumis à un ensemble de contraintes mécaniques : la lettre doit être une *consonne suivie par une consonne (dans l'ordre alphabétique)* et on doit pouvoir retrouver dans l'ensemble des deux chiffres *un nombre premier* et un *chiffre pair*. Toute autre combinaison est mécaniquement impossible.

La fonction *generer_code(S, A, N)* génère un code d'accès si les requis ci-dessus sont respectés, sinon elle génère une ERREUR.

Donnez des jeux de tests pour *generer_code* avec la méthode de catégorie-partition pour le critère **AC (c.-à-d. toutes les combinaisons)**.

- ❑ Rappel : Un **nombre premier** est un entier naturel qui admet *exactement* deux diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même)

Solution (simplifiée)

- ❑ S
 - S1 {a,d,e,h,i} : [erreur]
 - S2 {b,c,f,g,j} : [si pair et premier]
- ❑ A
 - A1: {0,4,6,8} [p pair]
 - A2: {2} [p pair] [p premier]
 - A3: {3,5,7} [p premier]
 - A4: {1,9}
- ❑ N
 - N1: {0,4,6,8} [p pair]
 - N2: {2} [p pair] [p premier]
 - N3: {3,5,7} [p premier]
 - N4: {1,9}

AC:

S2 A1 {N2, N3}
S2 A2 {N1, N2, N3, N4}
S2 A3 {N1, N2}
S2 A4 N2

Exemple complet

❑ Spécification :

- Le programme demande à l'utilisateur de taper un nombre entier positif entre 1 et 20 ainsi qu'une chaîne de caractères de la même longueur.
- Le programme demande ensuite de taper un caractère et retourne la (première) position du caractère dans la chaîne ou un message indiquant l'absence du caractère dans la chaîne.
- L'utilisateur 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 l'intervalle » (1-20) ou « hors de l'intervalle »
- ❑ Les catégories pour la longueur de a : minimale, maximale, 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 intervalle : 0, 21.
- ❑ Nombre entier x , dans l'intervalle : 1, 2-19, 20.
- ❑ Longueur de la 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. La solution ci-dessus combine analyse des limites, robustesse et classes d'équivalence

Spécifications du test formel

x:

- | | | |
|-----|------|---|
| x1) | 0 | [erreur] |
| x2) | 1 | [<u>propriété chaineok*</u> , 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 première position dans la chaîne | [si chaineok] |
| c2) | à la dernière position dans la chaîne | [si chaineok et not longueur1] |
| c3) | au milieu de la chaîne | [si chaineok et not longueur1] |
| c4) | pas dans la chaîne | [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 = 'ABCDEFGH', c = 'A'
x 3a2c2	x = 7, a = 'ABCDEFGH', c = 'G'
x 3a2c3	x = 7, a = 'ABCDEFGH', c = 'D'
x 3a2c4	x = 7, a = 'ABCDEFGH', c = 'X'
x 4a3c1	x = 20, a = 'ABCDEFGHJKLMNOPQRST', c = 'A'
x 4a3c2	x = 20, a = 'ABCDEFGHJKLMNOPQRST', c = 'T'
x 4a3c3	x = 20, a = 'ABCDEFGHJKLMNOPQRST', c = 'J'
x 4a3c4	x = 20, a = 'ABCDEFGHJKLMNOPQRST', c = 'X'
x 5	x = 21

12 cas de tests contre 60 (5*3*4) pour SECT

Critères utilisant les choix

- ❑ Toutes les combinaisons (**AC**) : Une valeur pour chaque choix de chaque paramètre doit être utilisée avec une valeur de chaque choix (possible) de chaque autre catégorie. [**Critère par défaut**]
- ❑ Chaque choix (**EC**) : Une valeur de chaque choix pour chaque catégorie doit être utilisée dans un cas de test.
- ❑ Choix de base (**BC**) : Un choix de base est sélectionné pour chaque catégorie. Un premier test de base est formé en utilisant le choix de base pour chaque catégorie. Les autres tests sont ensuite construits en remplaçant un (et un seul à la fois) choix de base par un autre choix de sa catégorie. [**Critère de Compromis**]

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.
- ❑ Une fois que la première étape est complétée, la technique est simple et peut être automatisée.
- ❑ La technique pour la réduction des cas de test le rend utile pour le test pratique.

Tables de décision

Tables de décision: Motivations


- ❑ Aide à exprimer les spécifications de test directement dans une forme utilisable.
- ❑ Facile à comprendre et supporte la dérivation systématique des tests.
- ❑ Supporte la génération automatique ou manuelle des cas de test.
- ❑ Une réponse particulière ou un sous-ensemble de réponses doit être choisie en évaluant plusieurs conditions reliées.
- ❑ Idéal pour décrire les situations dans lesquelles différentes 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 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

Condition	c1	Vrai			Faux		
	c2	Vrai		Faux	Vrai		Faux
	c3	V	F	—	V	F	—
Action	a1	X	X		X		
	a2	X				X	
	a3		X		X	X	
	a4			X			X


Règle

Exemple de table

c1: a, b, c triangle?	F					V				
c2: a = b?	—			V				F		
c3: a = c?	—		V		F		V		F	
c4: b = c?	—	V	F	V	F	V	F	V	F	
a1: Pas un triangle?	X									
a2: Scalène					X		X	X		X
a3: Isocèles		X							X	
a4: Équilatéral			X	X						
a5: Impossible			X			X				

Table de vérité

conditions											
<u>c1: $a < b + c$?</u>	F	V	V	V	V	V	V	V	V	V	V
<u>c2: $b < a + c$?</u>	-	F	V	V	V	V	V	V	V	V	V
<u>c3: $c < a + b$?</u>	-	-	F	V	V	V	V	V	V	V	V
c4: $a = b$?	-	-	-	V	V	V	V	F	F	F	F
c5: $a = c$?	-	-	-	V	V	F	F	V	V	F	F
c6: $b = c$?	-	-	-	V	F	V	F	V	F	V	F
a1: Pas un Vriangle	X	X	X								X
a2: Scalène							X		X	X	
a3: Isocèles				X							
a4: ÉquilaVéral					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 réponse sera sélectionnée en fonction des cas distincts de variables d'entrée.
- ❑ Ces cas peuvent être modélisés par des expressions 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.

Cas spéciaux

- ❑ La condition “*can't happen*” reflète des suppositions selon lesquelles plusieurs entrées sont mutuellement exclusives, ou ne peuvent pas être produites dans l'environnement.
- ❑ **A tester. Sinon 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, ce sont des bogues de spécification.

Graphes de cause-effet

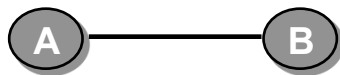
Graphes de cause-effet: Définition

- ❑ Une technique graphique qui aide à dériver les tables de décision.
- ❑ Vise à supporter l'interaction avec les experts de domaines et l'ingénierie inverse des spécifications, à des fins de tests.
- ❑ Identifie les causes (conditions d'entrées, signaux) et les effets (sorties, changements dans l'état du système).
- ❑ Les causes doivent être formulées de manière à 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 "significatives" d'entrées-sorties de manière à construire de plus petites tables de décision.

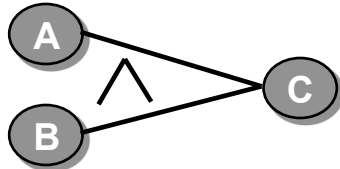
Structure des graphes de cause-effet

- ❑ 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 (\wedge) et *ou* logique (\vee) placés entre les lignes.
- ❑ Une cause dont la négation est nécessaire est désignée par un *non* logique (\sim).
- ❑ 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.

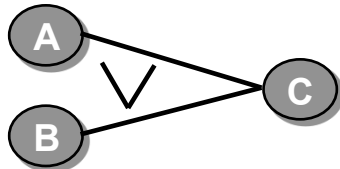
Remarque



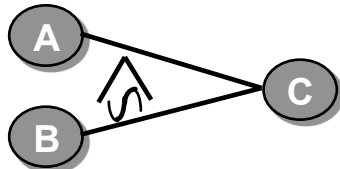
Si A alors B



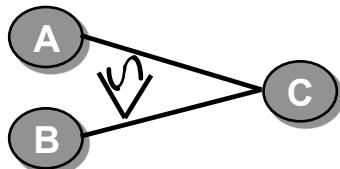
“*AND*” : Si (A et B) alors C



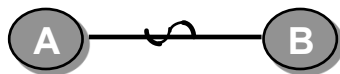
“*OR*” : Si (A ou B) alors C



“*NAND*” : Si pas (A et B) alors C

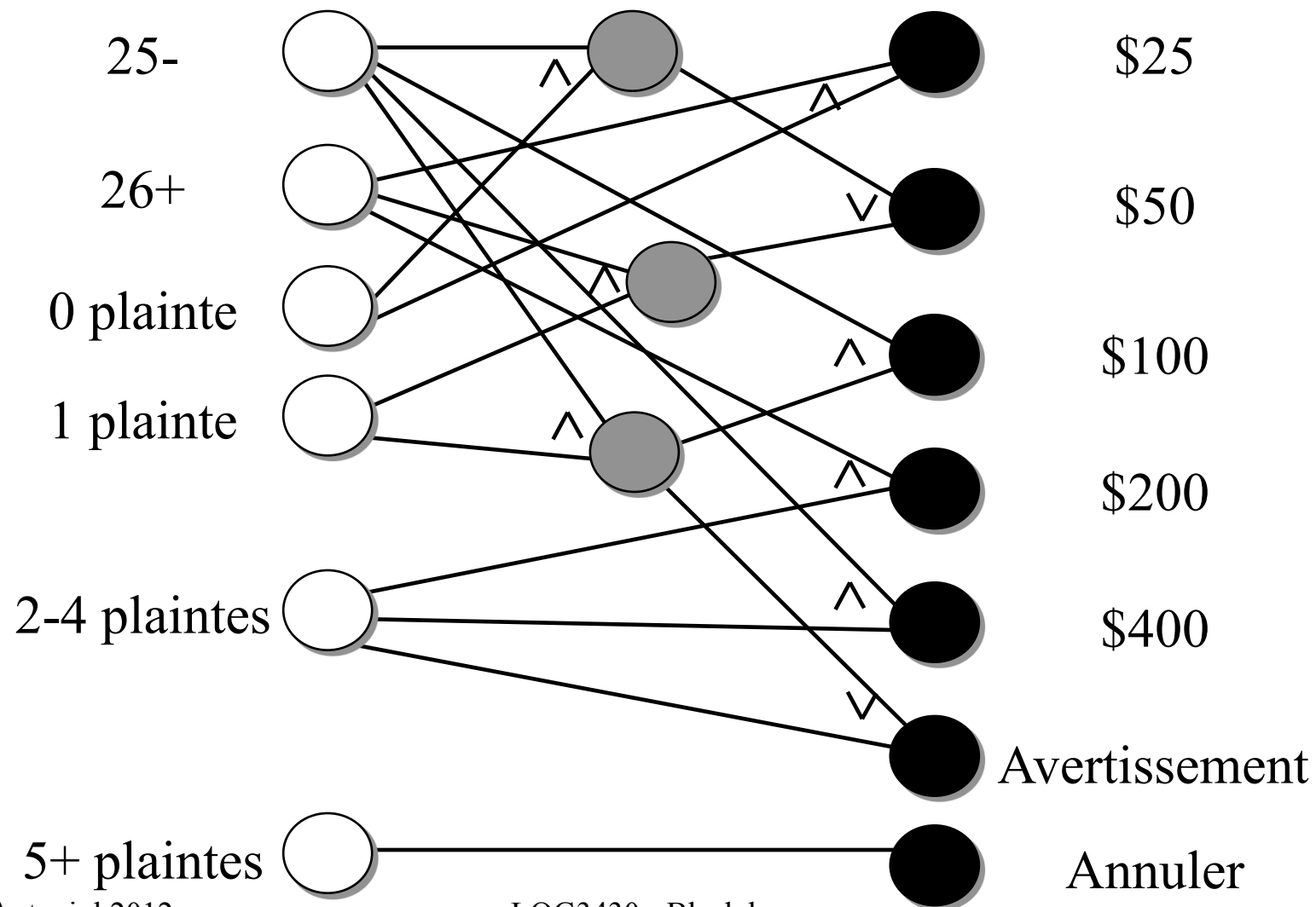


“*NOR*” : Si (ni A et ni B) alors C



“*NOT*” : Si (pas A) alors B

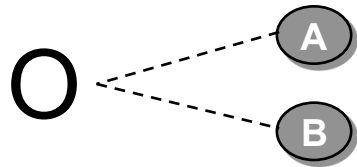
Exemple de renouvellement d'assurance



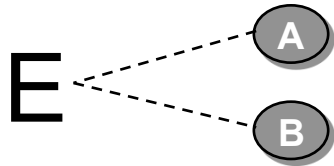
Autre exemple : Table de renouvellement d'assurance

	Section condition		Section action		
Variant	Plaintes	Âge	Prime augmentée \$	Envoi 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

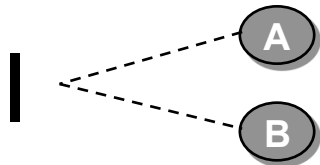
Contraintes additionnelles



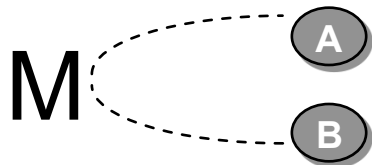
“*EXACTEMENT UN*” de A et B doit être vrai.



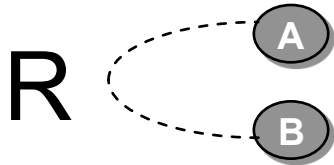
“*AU PLUS UN*” de A et B doit être vrai.



“*AU MOINS UN*” de A et B peut être vrai.



“*A MASQUE B*”, c.-à-d. $A \Rightarrow \text{PAS } B$



“*A REQUIERT B*”, c.-à-d. $A \Rightarrow B$

Autre exemple

- ❑ Donnée : La syntaxe de la fonction est “LEVEL(A,B)” où A est la hauteur en mètres de l’eau derrière le barrage et B est la quantité de pluie (en centimètres) dans la dernière période de 24 heures.
- ❑ Processus : La fonction calcule si le niveau de l’eau est (1) normal, (2) élevé ou (3) bas.
- ❑ Sorties : un des messages suivants :
 - LEVEL = SAFE (for normal et low)
 - LEVEL = HIGH
 - INVALID SYNTAX

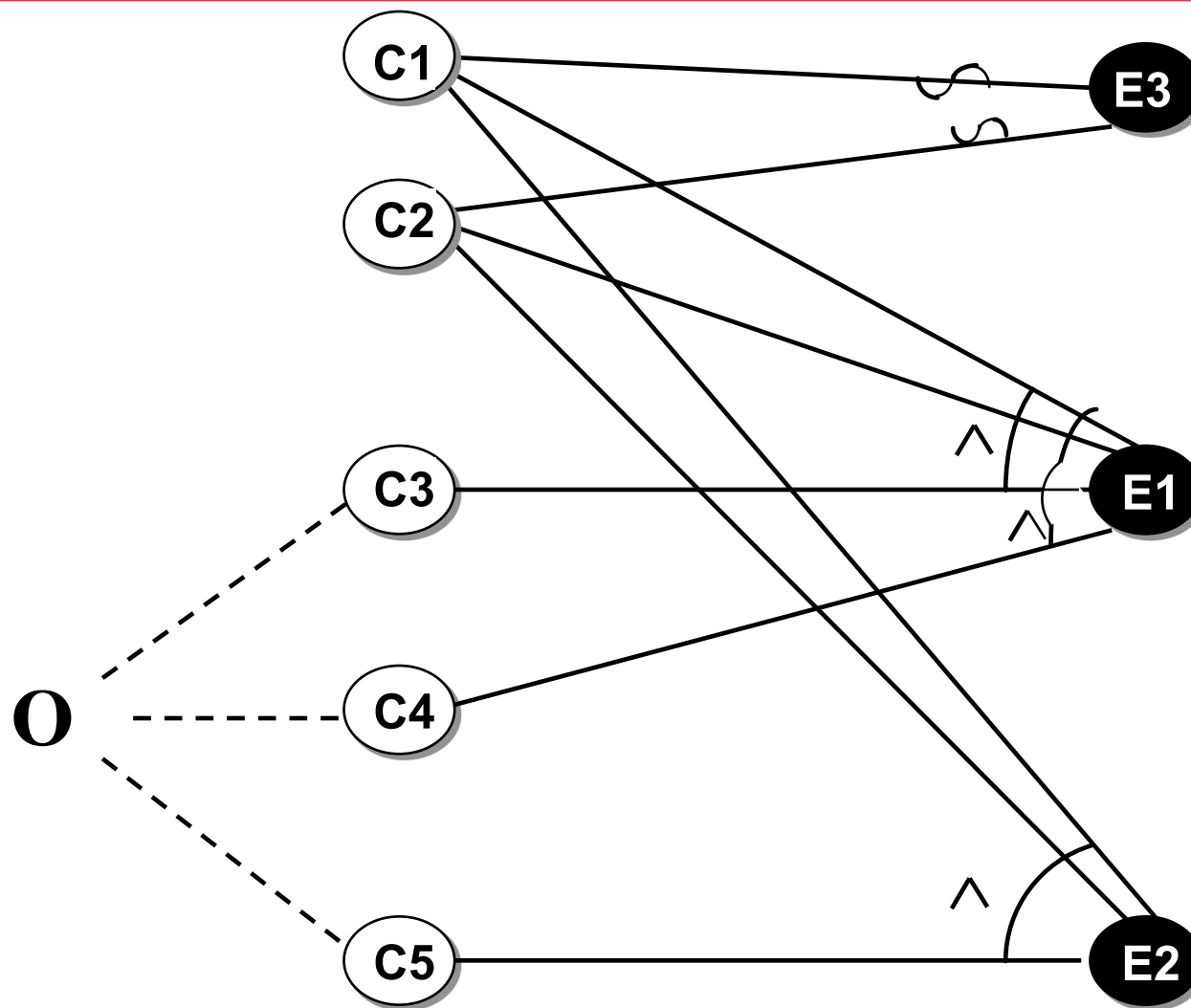
Identifier les causes

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 paramètres A et B sont des nombres réels tels que le niveau de l'eau est calculé pour être "HIGH".

Identifier les effets

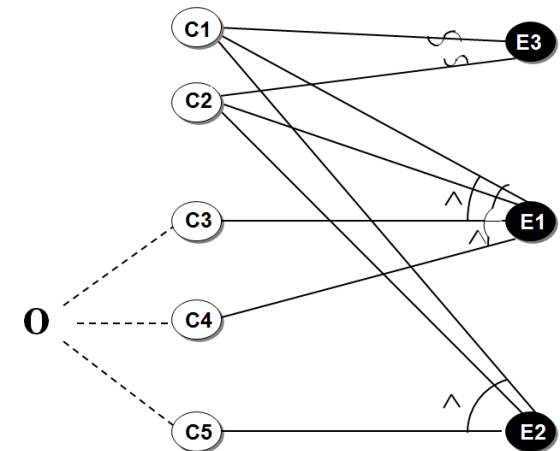
1. "LEVEL = SAFE" (correspondant à LOW ou NORMAL) est affiché à l'écran.
2. "LEVEL = HIGH" est étalé sur l'écran.
3. "INVALID SYNTAX" est imprimé.

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 segments distincts mènent vers l'effet E3, chacun correspondant à un cas de test, quatre 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	-
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

A: effet absent

P: effet présent

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, on peut essayer de le réduire en trouvant, pour chaque combinaison admissible d'effets, les combinaisons de causes qui la déclenchent. L'idée ici est de grouper effets et/ou causes pour plus de lisibilité du graphe.
- ❑ À cause de contraintes additionnelles sur le graphe, on peut être plus restrictif qu'avec les tables de décision classiques.

Fonctions logiques

Fonctions logiques: Définitions

- ❑ Un prédicat est une expression qui prend une valeur booléenne.
- ❑ Les prédicats peuvent contenir des variables booléennes, des variables non booléennes utilisées avec des opérateurs comparateurs $\{>, <, =, \dots\}$, et des appels de fonction (retournant une valeur booléenne).
- ❑ La structure interne d'un prédicat est créée par les *opérateurs logiques* $\{\text{non}, \text{et}, \text{ou}, \dots\}$.
- ❑ 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 simples à lire, nous utiliserons la contiguïté pour l'opérateur et, + pour l'opérateur ou et un \sim pour l'opérateur de négation non.
- ❑ Exemple :
mettre en marche ou hors d'état l'ignition d'une chaudière se basant sur quatre variables d'entrée :
 - NormalPressure (A) : la pression dans une limite d'utilisation sécuritaire ?
 - CallForHeat (B) : la température ambiante sous le point de consigne ?
 - DamperShut (C) : le conduit du tuyau d'échappement fermé ?
 - ManualMode (D) : sélection manuelle du mode d'utilisation ?
- ❑ Fonction logique : $Z = A(B\sim C + D) \sim$ > Table de vérité.

Table de vérité de la chaudière – Variant 0-7

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 - Variant 8-15

Numéro du vecteur d'entrées	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 reliés par l'opérateur *ou*.
- ❑ ***L'implicant** :*
Chaque terme de l'expression somme de produit – condition suffisante et pas nécessaire pour la sortie *True* de cette expression.
- ❑ *Les **implicants premiers** :*
Aucun sous-ensemble (sous-terme propre) n'est aussi un implicant.
- ❑ *La minimisation de la logique :*
dérivation d'expressions booléennes compactes (non redondantes) mais équivalentes en utilisant l'algèbre booléenne.

Exemple de la chaudière

❑ La fonction logique : $Z=A(B\sim C+D)$.

❑ La forme somme de produits (DNF) :

$$Z=A(B\sim C+D) = AB\sim C+AD.$$

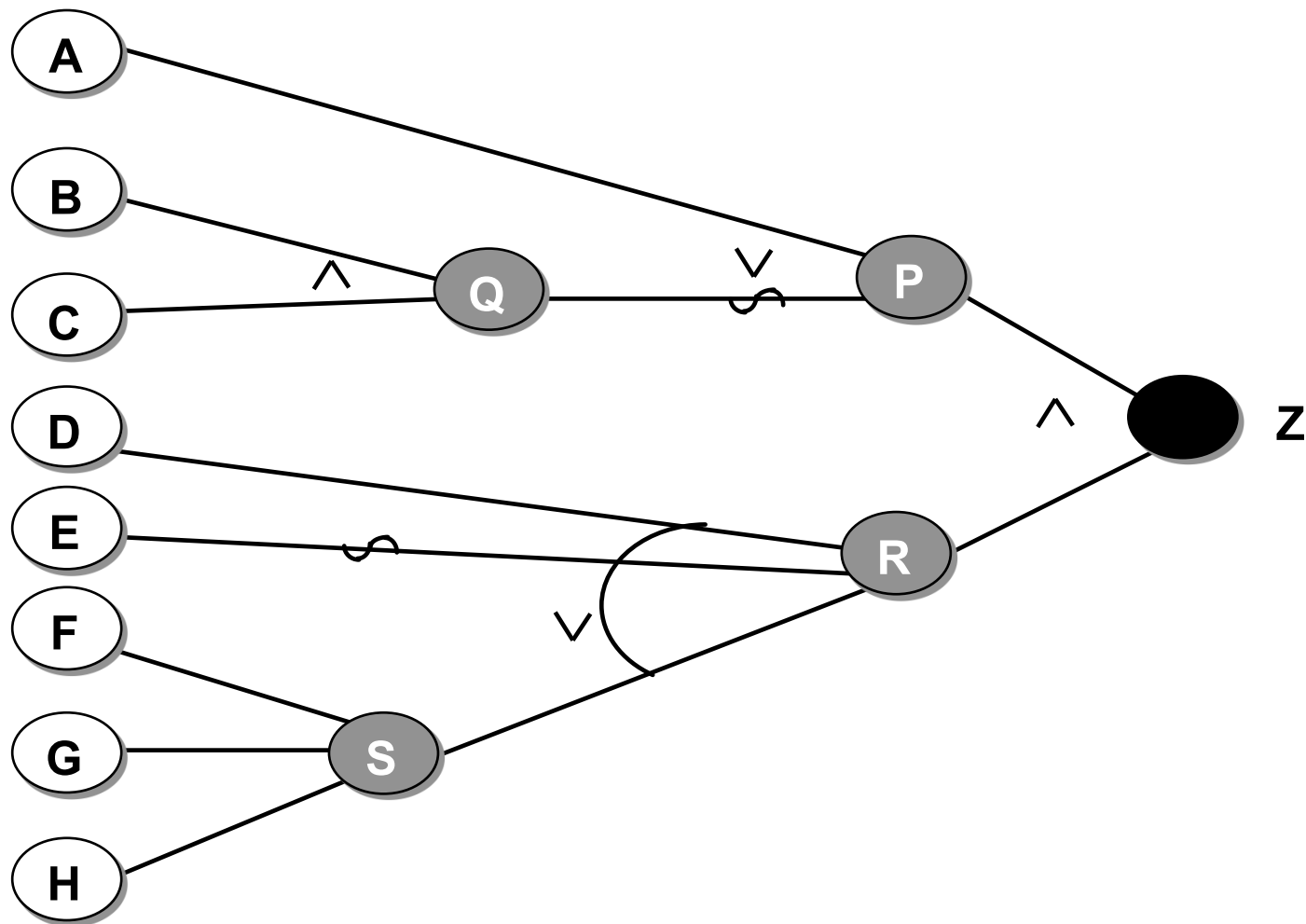
❑ Les implicants : $AB\sim C$, AD .

❑ Les implicants premiers : $AB\sim C = VVFx = \{VVFV, VVFF\}$,
 $AD=VxxV=\{VFFV, VFVV, VVFV, VVVV\} \Rightarrow$ les deux termes
sont des implicants premiers.

Graphe de cause-effet \rightarrow fonction logique

- ❑ Une fois le graphe cause-effet révisé et considéré correct, nous voulons trouver 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.
- ❑ Dériver une fonction booléenne à partir d'un graphe d'une manière systématique.

Graphe \rightarrow Fonction: Exemple



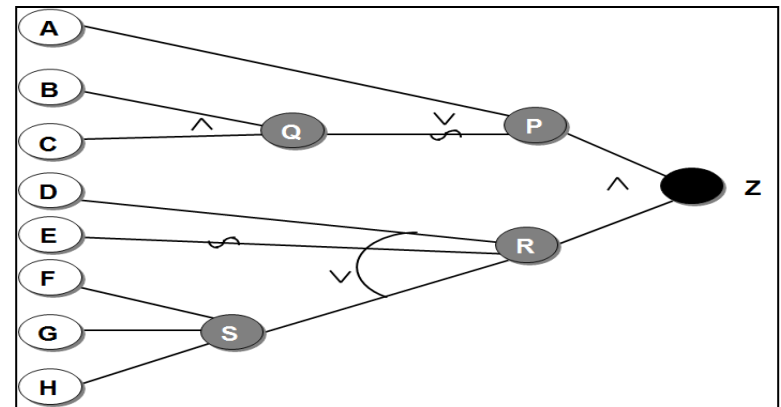
Graphe \rightarrow Fonction: Démarche

- ❑ 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.

Graphe \rightarrow Fonction: Exemple (suite)

- ❑ $Z = PR$ (effet)
- ❑ $P = A + \sim Q$ (intermédiaire)
- ❑ $Q = BC$ (intermédiaire)
- ❑ $R = D + \sim E + S$ (intermédiaire)
- ❑ $S = F + G + H$ (intermédiaire)
- ❑ $Z = (A + \sim(BC))(D + \sim E + (F+G+H))$ (substitution)
- ❑ $Z = (A + \sim B + \sim C)(D + \sim E + F + G + H)$ (Loi de De Morgan)
- ❑ $Z = AD + A\sim E + AF + AG + AH + \sim BD + \sim B\sim E + \sim BF + \sim BG + \sim BH + \sim CD + \sim C\sim E + \sim CF + \sim CG + \sim CH$ (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 :

➤ $\sim(A+B)=\sim A\sim B, \sim(AB)=\sim A+\sim B$

❑ Loi d'absorption :

➤ $A + AB = A$

➤ $A(A+B) = A$

➤ $A+(\sim AB) = A+B, A(\sim A+B) = AB$

➤ $AB+AC+B\sim C = AC+B\sim C$

Modèle de fautes pour les tests basés 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 d'être certain qu'elle est cohérente avec sa spécification, comme modélisé par la fonction logique (ou le graphe).
- ❑ Certains critères de couverture de test ne requièrent pas une forme normale disjonctive (DNF) 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, C_p est l'ensemble de clauses dans un prédicat p .

Couverture de prédicats

- ❑ Couverture de prédicats : *Pour $p \in P$, nous avons deux conditions requises de test : p évalué à vrai, et p évalué à faux.*
- ❑ Pour $A(B \sim C + D)$, les deux tests qui satisfont la couverture des prédicats sont : (1) $(A=\text{true}, B=\text{false}, C=\text{true}, D=\text{true})$, (2) $(A=\text{false}, B=\text{false}, C=\text{true}, D=\text{true})$,
- ❑ Branch coverage!
- ❑ Problème : les clauses individuelles ne sont pas testées.

Couverture de clauses

- ❑ Couverture de clauses : *Pour chaque $c \in C$, 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) $(A=\text{vrai}, B=\text{vrai}, C=\text{faux})$,
 - (2) $(A=\text{faux}, B=\text{faux}, C=\text{vrai})$.

- ❑ Remarque : la couverture de clause n'inclut pas la couverture de prédicats et vice-versa.

Exemple

- ❑ $Z = A+B$
- ❑ $t1 = (A = \text{vrai}; B = \text{vrai}) \Rightarrow Z$
- ❑ $t2 = (A = \text{vrai}; B = \text{faux}) \Rightarrow Z$
- ❑ $t3 = (A = \text{faux}; B = \text{vrai}) \Rightarrow Z$
- ❑ $t4 = (A = \text{faux}; B = \text{faux}) \Rightarrow \sim Z$
- ❑ 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 $p \in P$, nous avons des conditions requises de test pour les clauses dans C_p pour évaluer chaque combinaison possible des valeurs de vérité.*
- ❑ Inclut la couverture de prédicats
- ❑ Il y a $2^{|C_p|}$ assignations possibles de valeurs de vérité.
→ d'où la notion de clause active
- ❑ 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 le prédicat AB , si $B = \textit{faux}$, 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 $c_j \in p, j \neq i$ ont des valeurs de telle manière qu'en changeant la valeur de vérité de c_i , on change la valeur de vérité de p .*
- ❑ **clause majeure: la clause que je vais tester!**
- ❑ Nous aimerions tester chaque clause dans les circonstances où elle détermine le prédicat.
- ❑ L'ensemble de test T4 dans le transparent 82 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 $p \in P$ et chaque clause majeure $c_i \in C_p$, choisir des clauses mineures $c_j, j \neq i$ de telle façon que c_j détermine p . Nous avons 2 conditions requises de test pour chaque c_i : c_i évaluée à vrai et c_i évaluée à faux.

- 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 $\{(A = T; B = F) ; (A = F; B = F)\}$.
- Pour la clause B, B détermine Z si, et seulement si, A est fausse. Donc, nous avons les deux conditions requises de test $\{(A = F; B = T); (A = F; B = F)\}$, la dernière est commune avec A*.
- *Pour un prédicat à n clauses, $n+1$ cas de tests suffisent.*
- La définition ci-dessus permet des ambiguïtés qu'il convient de lever

General ACC (GACC)

- Pour chaque $p \in P$ et chaque clause majeure $c_i \in C_p$, choisir les clauses mineures c_j , $j \neq i$ 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 n'ont pas besoin d'être les mêmes quand c_i vrai et c_i faux.
- GACC ne subsume pas la couverture de prédicats.

Exemple: $p = a \leq b$. a détermine $p \forall b$, b détermine $p \forall a$

$\{VV, FF\}$ satisfait GACC mais p est évalué à Vrai à chaque fois.

ACC corrélées (CACC)

- Pour chaque $p \in P$ et chaque clause majeure $c_i \in C_p$, choisir les clauses mineures $c_j, j \neq i$ 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, c-à-d, il est requis que $p(c_i = \text{vraie}) \neq p(c_i = \text{fausse})$.
- CACC est subsumé par la couverture combinatoire de clauses et subsume la couverture de clauses/prédicats.

Exemple: $p = a \neq b$. Résultat: $\{TT, FT, TF\}$

ACC restreintes (RACC)

- ❑ Pour chaque $p \in P$ et chaque clause majeure $c_i \in C_p$, choisir des clauses mineures c_j , $j \neq i$ 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 mêmes 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 rend plus facile de déterminer la cause du problème que CACC, si l'une est détectée : clause majeure.
- ❑ Mais il arrive d'avoir des contraintes entre les clauses qui rendent RACC impossible à réaliser : Ceci correspond à MCDC (unique cause MCDC) pour la couverture de code.

Exemple

- $Z = A(B + C)$
- Il est possible de satisfaire la couverture de clauses corrélées actives (CACC) par rapport à la clause A avec les deux conditions requises de test :

$\{(A = \text{vraie}; B = \text{vraie}; C = \text{fausse});$

$(A = \text{fausse}; B = \text{fausse}; C = \text{vraie})\}$

- Mais ça ne satisfait pas la RACC :

$\{(A = \text{vraie}; B = \text{vraie}; C = \text{fausse});$

$(A = \text{fausse}; B = \text{vraie}; C = \text{fausse})\}$

Couverture de clauses inactives

- ❑ Le critère de la couverture de clauses actives vise à s'assurer que les clauses majeures affectent leurs prédicats. Un critère complémentaire aux ACC s'assure que **changer une clause majeure qui ne devrait pas affecter le prédicat n'affecte pas le prédicat.**

- ❑ **Couverture de clauses inactives (ICC) :**
Pour chaque $p \in P$ et chaque clause majeure $c_i \in C_p$, choisir des clauses mineures c_j , $j \neq i$ 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.
- ❑ On distingue GICC et RICC (de manière similaire aux ACC)

- ❑ ICC est subsumée par la couverture de clauses combinatoire 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 la DNF sont les critères qui l'accompagnent.
- ❑ Critères :
 - la couverture des implicants,
 - la couverture des implicants premiers,
 - la stratégie de négation de variables (VNS).

Couverture d'implicants (IC)

- ❑ IC : *Étant données les représentations DNF d'un prédicat p et sa négation $\sim p$, pour chaque implicant, une condition test requise est que l'implicant est évalué à vrai.*
- ❑ Ceci teste différentes situations dans lesquelles une action devrait (ou ne devrait pas) être prise (e.g., une chaudière allumée).
- ❑ p : $AB + B\sim C$.
- ❑ $\sim p$ (une représentation) : $\sim B + \sim AC$.
- ❑ Quatre implicants : $\{AB, B\sim C, \sim B, \sim AC\}$.
- ❑ 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 la couverture de clauses actives.

Problèmes avec IC

- ❑ Un problème avec l'IC est que les tests peuvent être choisis de façon à ce qu'un simple test satisfasse des implicants multiples (voir l'exemple précédent).
- ❑ Quoique cela permette aux testeurs de minimiser la taille des jeux de tests, c'est une mauvaise chose pour tester les contributions uniques que chaque implicant peut apporter à un prédicat.
- ❑ Ainsi, nous introduisons une méthode pour forcer un 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 standards existent 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 : $ABC + AB\sim C + B\sim C$.
- ❑ 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 du prédicat.
- ❑ Dans $AB+AC+B\sim C$, AB est redondant.
- ❑ **PIC** : *Étant donné des représentations DNF non redondantes d'implicants premiers d'un prédicat p et sa négation $\sim p$, pour chaque implicant, une condition requise de test est que l'implicant est évalué à vrai, pendant que les autres implicants sont évalués à faux.*

Exemple & discussion PIC

- ❑ $p : AB + B\sim C$.
- ❑ $\sim p : \sim B + \sim AC$.
- ❑ 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 ne subsume le PIC.
- ❑ Quoiqu'il y ait potentiellement jusqu'à 2^{n-1} implicants premiers, en général, les prédicats génèrent un nombre modeste de tests pour le PIC.
- ❑ Savoir si le PIC subsume tous les critères de la couverture de clauses reste une question ouverte.

Stratégie de négation de variables (VNS)

- ❑ **Un Point Unique Vrai (PUV)** par rapport à un implicant i est une assignation de valeurs telles que i soit à Vrai et tout autre implicant (de p) à Faux
 - E.g., (TTFF) pour le premier terme produit dans l'exemple de la chaudière ($AB\sim C$), AD est faux
- ❑ **Un Point Presque Faux (PPF)** par rapport à une clause c dans un implicant i (de p) est une assignation de valeurs telles que p est à Faux mais si c est « nié » (et les autres clauses restent inchangées), i (et donc p) passe à Vrai.
 - E.g., (TTTF) pour $AB\sim C$ où $\sim C$ est négatif.
- ❑ De telles assignations de valeurs constituent les ensembles de tests candidats (**TCS**). Il faut générer un TCS pour chaque implicant dans la fonction logique p ($\sim p$ n'est pas considéré).
- ❑ Le jeu de test recherché est formé en choisissant le plus petit jeu qui couvre tous les TCS.

Discussion

- ❑ Si l'implémentation d'un terme produit n'est pas évaluée à Vrai quand elle devrait – impliquant qu'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 l'implémentation d'un terme produit n'est pas évaluée à Faux 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 % d'un jeu de test satisfaisant la couverture de clauses combinatoire - "All-Variant" (2^n) - sont nécessaires pour satisfaire les *critères de négation de variables*.

Exemple de la chaudière

- ❑ Rendre $AB\sim C$ 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 $AB\sim C$: ensemble de points uniques vrais {9,11,15}.
- ❑ Les points presque faux pour $AB\sim C$: {14}, {8}, {4,5} pour la négation $\sim C$, 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.

Table de vérité

Implicants

Pour améliorer la visibilité seulement les valeurs vraies sont représentées.

Var	(I1) AB~C	(I2) AD	Z
0			
1			
2			
3			
4			
5			
6			
7			
8			
9		1	1
10			
11		1	1
12	1		1
13	1	1	1
14			
15		1	1

Var	Neg(I1,A) AB~C _A	Neg(I1,B) AB~C _B	Neg(I1,C) AB~C _C	Neg(I2,A) AD _A	Neg(I2,D) AD _D	Z
0						
1				1		
2						
3				1		
4	1					
5	1			1		
6						
7				1		
8		1			1	
9		1				1
10					1	
11						1
12					1	1
13						
14			1		1	
15			1			1

Var	PUV AB~C	PUV AD	PPF AB~C _A	PPF AB~C _B	PPF AB~C _C	PPF AD _A	PPF AD _D	TCS
0								
1						X		
2								
3						X		
4			X					
5			X			X		S
6								
7						X		
8				X			X	S
9		X						S
10							X	
11		X						
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 expressions booléennes (utilisées pour produire des fautes dans les spécifications).
- ❑ La sélection aléatoire des cas de tests mène à un score moyen de mutation de 42.7%.
- ❑ La stratégie de négation de variable est significativement meilleure avec une moyenne de 97.9%.

Sommaire du test fonctionnel

- ❑ Toutes les techniques présentées voient un programme comme une fonction mathématique qui relie des entrées aux sorties.
- ❑ Un compromis est à faire entre l'effort d'identification de test et l'effort d'exécution de test.