UNIVERSITÉ PARIS SUD

L3 informatique et MIAGE

Année 2016-2017

Génie logiciel avancé

Introduction au test et méthodes informelles

Delphine Longuet

delphine.longuet@lri.fr

http://www.lri.fr/~longuet/Enseignements/16-17/L3-GLA

Définitions du test

Norme IEEE (Standard Glossary of Software Engineering Terminology)

« Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus. »

- Validation dynamique (exécution du système)
- Comparaison entre système et spécification

Définitions du test

« *Tester peut révéler la présence d'erreurs mais jamais leur absence* »

↳ Vérification partielle : le test ne peut pas montrer la conformité du système (nécessité d'une infinité de tests)

« *Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts* »

↳ Objectif : détection des bugs

Bug ?

Anomalie (fonctionnement) : différence entre comportement attendu et comportement observé

Défaut (interne) : élément ou absence d'élément dans le logiciel entraînant une anomalie

Erreur (programmation, conception) : comportement du programmeur ou du concepteur conduisant à un défaut

erreur ⟶ défaut ⟶ anomalie

Un bon test est un test qui permet de découvrir un défaut en déclenchant une anomalie

Échauffement

Spécification : Le programme prend en entrée trois entiers, interprétés comme étant les longueurs des côtés d'un triangle. Le programme retourne la propriété du triangle correspondant : scalène, isocèle ou équilatéral.

Écrire un ensemble de tests pour ce programme.

Vocabulaire du test

Objectif de test : comportement du système à tester

Données de test : données à fournir en entrée au système de manière à déclencher un objectif de test

Résultats d'un test : conséquences ou sorties de l'exécution d'un test (affichage à l'écran, modification des variables, envoi de messages...)

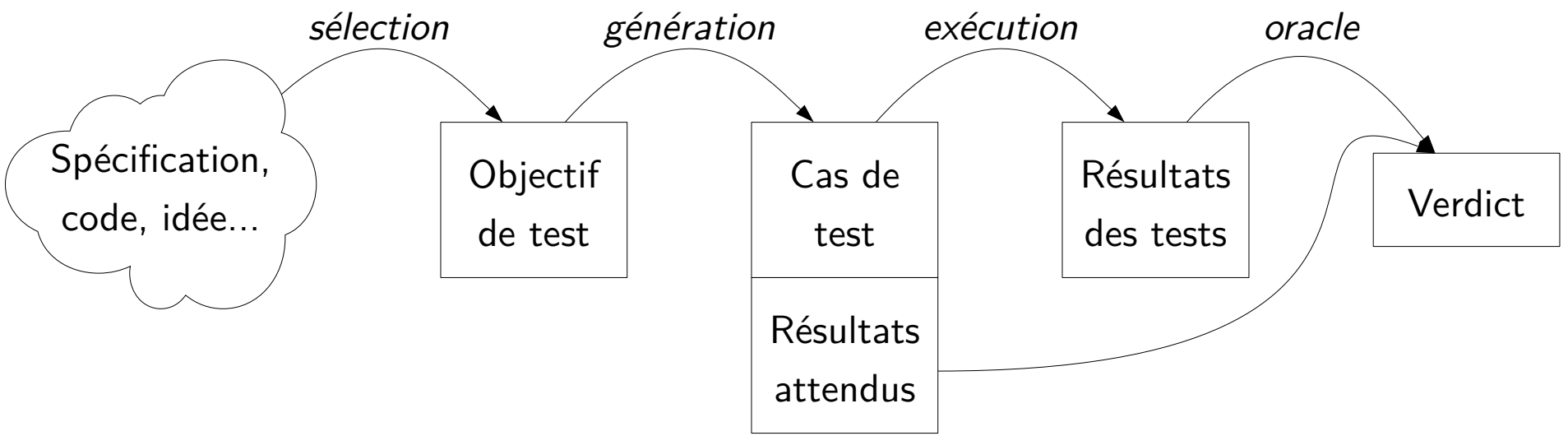
↳ Cas de test : données d'entrée et résultats attendus associés à un objectif de test

---

Glossary of Testing Terms, ISTQB (International Software Testing Qualifications Board)

Processus de test

1. Choisir les comportements à tester (objectifs de test)
2. Choisir des données de test permettant de déclencher ces comportements + décrire le résultat attendu pour ces données
3. Exécuter les cas de test sur le système + collecter les résultats
4. Comparer les résultats obtenus aux résultats attendus pour établir un verdict



Problème de l'oracle

Oracle : décision de la réussite de l'exécution d'un test, en comparant le résultat obtenu au résultat attendu

Problème : décision pouvant être complexe

- types de données sans prédicat d'égalité
- système non déterministe : sortie possible mais pas celle attendue
- heuristique : approximation du résultat optimal attendu

Solution : description du résultat attendu, selon les cas :

- la valeur attendue
- énumération des valeurs possibles
- ensemble de conditions

Problème de l'oracle

Trouver le minimum d'une liste d'entiers

Entrée : [4; 2; 3; 6] Sortie attendue : 2

Oracle : Égalité sur les entiers ✔

Calculer l'itinéraire le plus rapide entre deux villes

Entrée : Paris – Lyon Sortie attendue : ... A6...

Oracle : Égalité des chemins ? ✘

Trajet de 4h17 (quel que soit l'itinéraire choisi) ✔

Problème du sac à dos (résolu avec une heuristique)

Oracle : Résultat raisonnablement éloigné du résultat optimal ? ✘

Résultat = résultat optimal + 5% ✔

Problème de l'oracle

Oracle : En général, résultat attendu = ensemble de conditions si plusieurs résultats possibles et énumération impossible

Risques : Échec d'un programme conforme si définition trop stricte du résultat attendu

↳ Faux positif (*false-fail*)

Voir l'exemple du calcul d'itinéraire dans lequel on impose un chemin

Faux positifs et faux négatifs

Validité des tests : Les tests n'échouent que sur des programmes incorrects

Faux positif (false-fail) : fait échouer un programme correct

Complétude des tests : Les tests ne réussissent que sur des programmes corrects

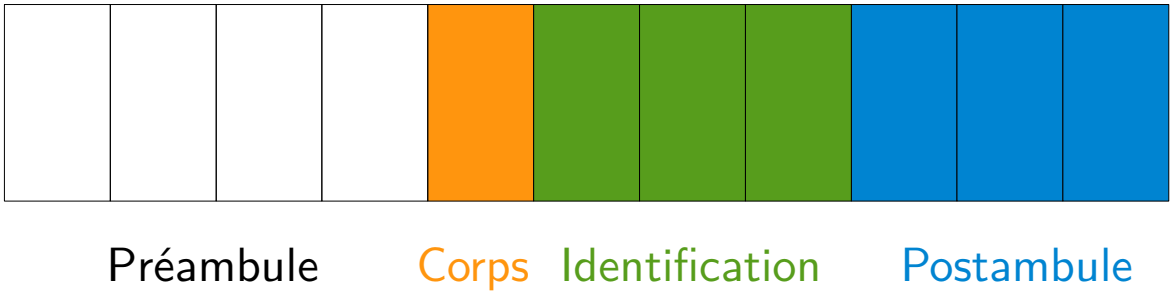
Faux négatifs (false-pass) : fait réussir un programme incorrect

Validité indispensable, complétude impossible en pratique

↳ Toujours s'assurer de la validité des tests

Scénario de test (unitaire)

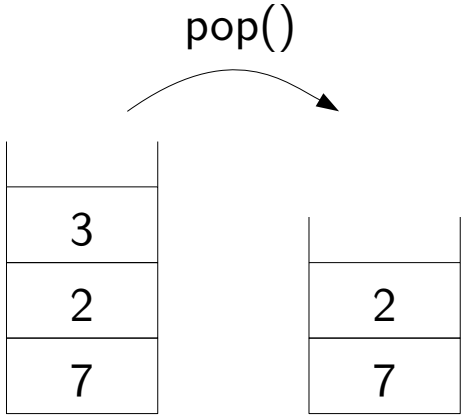
- Préambule : Suite d'actions amenant le programme dans l'état nécessaire pour exécuter le cas de test
- Corps : Exécution des fonctions du cas de test
- Identification : Opérations d'observation rendant l'oracle possible
- Postambule : Suite d'actions permettant de revenir à un état initial



Exécution d'un test

Pop (supprimer le sommet d'une pile)

Cas de test :



Exécution du test :

Préambule	push(7)
	push(2)
	push(3)
Corps	pop()
Identification	top() = 2
	pop()
	top() = 7
	pop()
	empty() = true

Attention

- Les tests doivent être exécutés dans des conditions et sur des données connues et contrôlées.
- Le résultat d'un test doit être observable.
- Il doit exister un moyen sûr de comparer le résultat observé au résultat attendu afin de déterminer le succès ou l'échec du test.
- Les cas de test ne doivent pas être redondants : plusieurs cas ne doivent pas cibler la même faute
- Un cas de test correspond à un comportement cible unique : pas de choix pendant l'exécution du test

Méthodes informelles de sélection de tests

Problématique du test

Impossible d'exécuter un programme sur toutes ses entrées possibles

↳ Nécessité de sélectionner les tests

Objectif : détecter un maximum de défauts avec un minimum de tests

Ensemble de tests idéal

- praticable : qu'on peut exécuter en temps fini et raisonnable
- représentatif : susceptible de trouver un grand nombre de fautes

Analyse partitionnelle

Principe : Définir des classes d'équivalence sur les domaines des entrées (ou des sorties) d'un système pour en déduire différents objectifs de test

Hypothèse : Pour chaque classe d'équivalence, toutes les entrées génèrent le même comportement

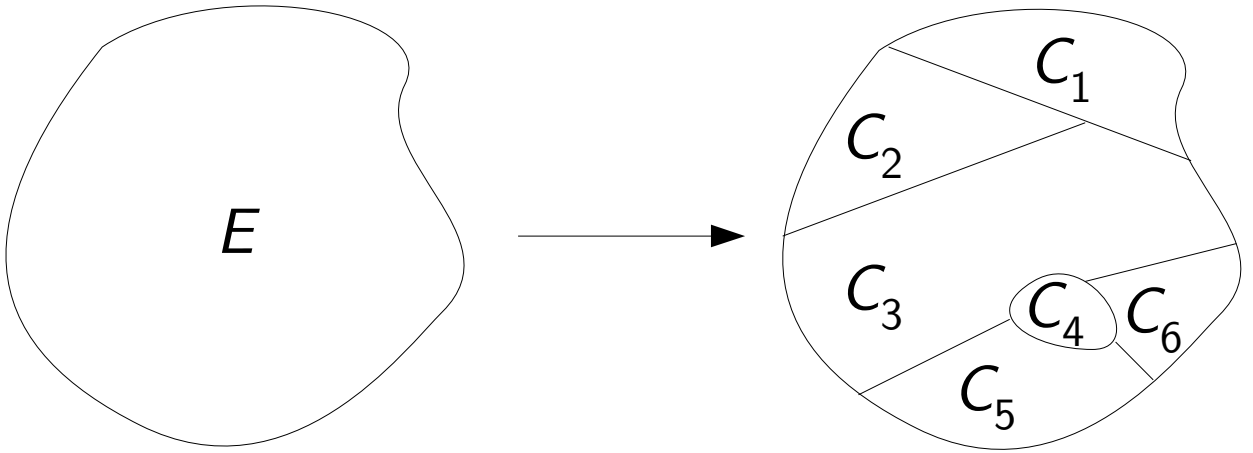
↳ Détection du même défaut

But : Tester chaque comportement de façon indépendante, en fonction du défaut cherché

Analyse partitionnelle

Construction à partir de l'ensemble E des entrées du programme d'une partition de E en un ensemble de classes C_i telles que :

- E est l'union des C_i
- les classes C_i sont deux à deux disjointes

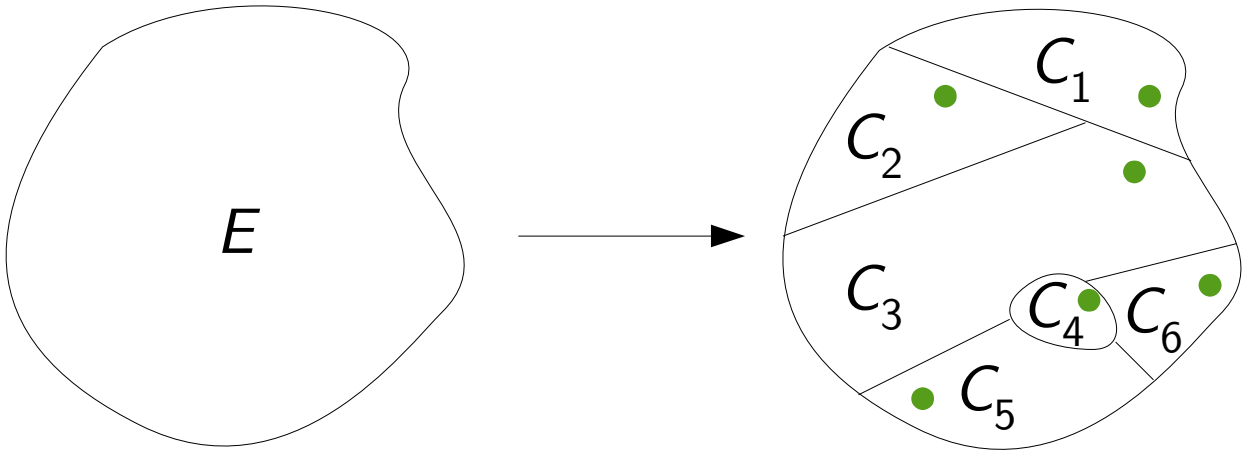


Analyse partitionnelle

Hypothèse d'uniformité : Une seule valeur dans chaque C_i suffit à tester le comportement représenté par C_i

$$\exists\ d \in C_i,\ \text{correct}(\text{Prog},d) \Rightarrow \forall\ d \in C_i,\ \text{correct}(\text{Prog},d)$$

S'il existe une valeur d dans C_i telle que le programme est correct pour d, alors le programme est correct pour toutes les valeurs de C_i



Analyse partitionnelle

Triangles

Le programme prend en entrée trois réels, interprétés comme étant les longueurs des côtés d'un triangle. Si ces longueurs forment un triangle, le programme renvoie la propriété du triangle correspondant (scalène, isocèle ou équilatéral) ainsi que la propriété de son plus grand angle (aigu, droit ou obtus).

Donner les classes d'équivalence associées à ce programme ainsi qu'un cas de test possibles pour chacune de ces classes.

Test aux limites

Principe : Construire des tests pour les valeurs limites du programme

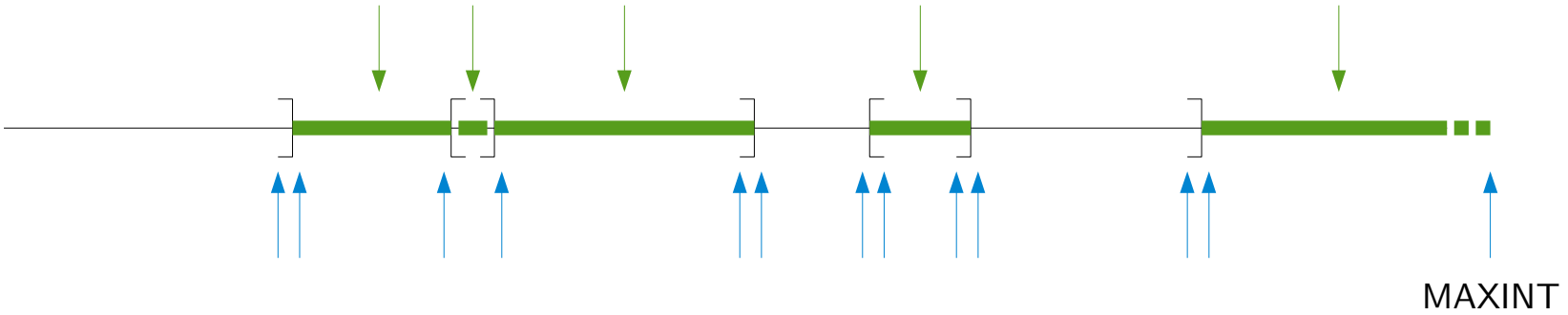
- Condition de boucle
- Valeurs très grandes
- Valeurs non valides

Utilisations :

- Conjointement à l'analyse partitionnelle : données choisies aux bornes des intervalles des classes d'équivalence (suppose une relation d'ordre sur les entrées)
- Pour le test de robustesse : données choisies en dehors des valeurs autorisées

Méthode générale

1. Pour chaque paramètre d'entrée, calculer les classes d'équivalence sur les domaines de valeurs de ce paramètre
2. S'il y a plusieurs paramètres, faire le produit cartésien des classes obtenues
3. Choisir des données de test



- Une valeur pour chaque classe obtenue
- Des valeurs aux limites

Analyse partitionnelle + test aux limites

Analyse partitionnelle :

- ✔ Réduction du nombre de cas de test
- ✗ Choix des classes délicat

Test aux limites :

- ✔ Heuristique solide pour le choix des données au sein des classes
- ✔ Production de tests de conformité et de tests de robustesse
- ✗ Relation d'ordre sur les entrées nécessaire
- ✗ Explosion combinatoire des données de test

Inconvénient majeur : Caractère intuitif ou subjectif de la partition en classe et de la notion de limite

↳ Difficulté pour caractériser la couverture des tests

Cas de nombreux paramètres

30 imprimantes

4 valeurs

+ une option complexe

1 entier

5 valeurs + sous-options

2 valeurs

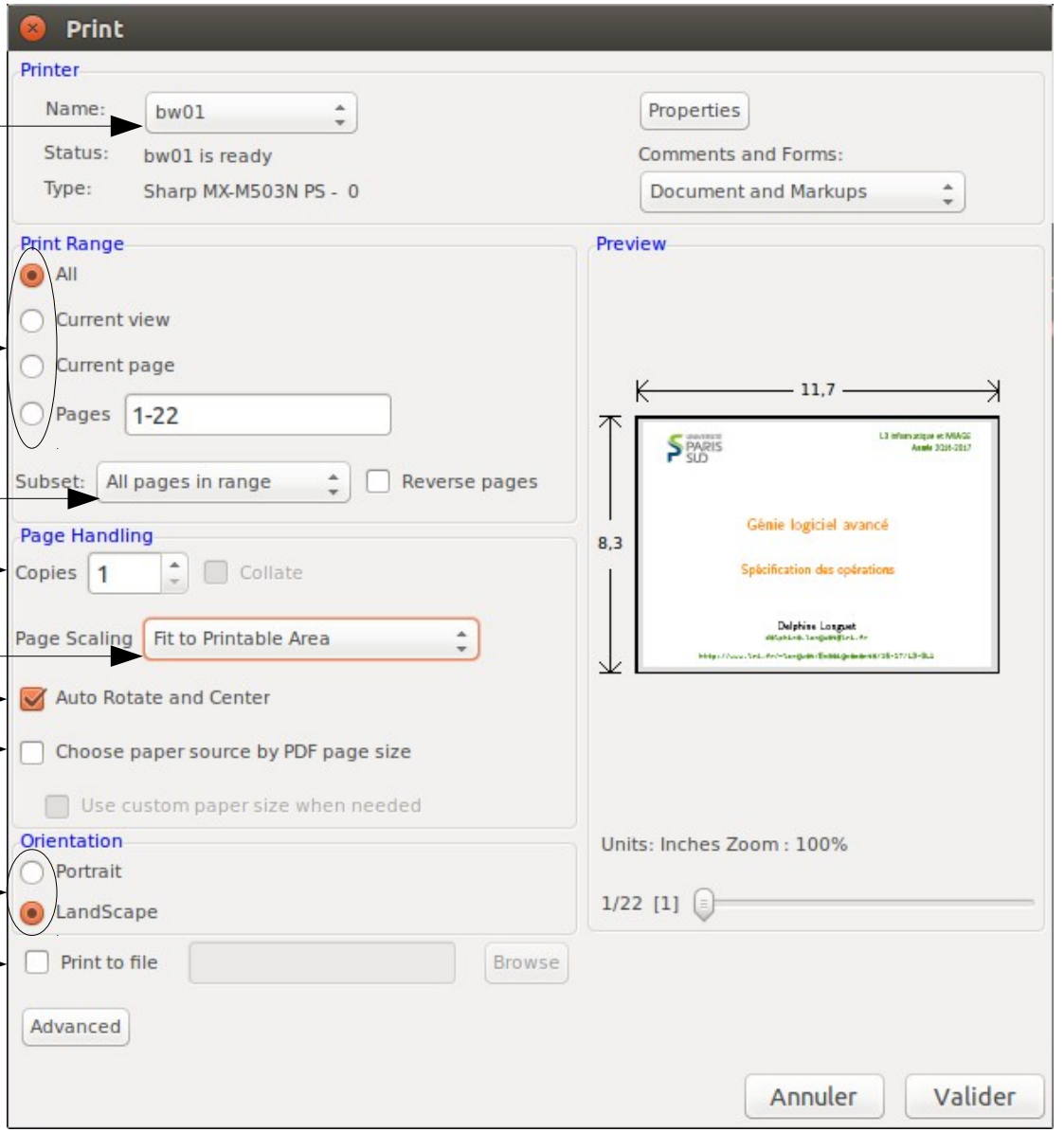
2 valeurs

2 valeurs

2 valeurs

+ choix du fichier

Plus les paramètres avancés...



Test pairwise

Constatation : Explosion combinatoire des valeurs d'entrées dans le cas de nombreux paramètres prenant des ensembles de valeurs finis

Solution : Tester un sous-ensemble des combinaisons de valeurs tel que chaque combinaison de n variables est testée

↳ Pairwise : $n = 2$

Idée sous-jacente : Majorité des fautes détectées par combinaisons de deux valeurs de variables

Complexité : Produit du cardinal des deux plus grands ensembles de valeurs

Test pairwise

Exemple : Trois variables x, y et z :

$x = 1$ ou 2 $\quad$ $y =$ Q ou R ou S $\quad$ $z = 5$ ou 6

Couvrir toutes les combinaisons (12) : 12 tests

Couvrir tous les couples (16) : 6 tests

x	y	z	Couples couverts
1	Q	5	(1,Q) (Q,5) (1,5)
1	R	6	(1,R) (R,6) (1,6)
1	S	5	(1,S) (S,5) (1,5)
2	Q	6	(2,Q) (Q,6) (2,6)
2	R	5	(2,R) (R,5) (2,5)
2	S	6	(2,S) (S,6) (2,6)

Test pairwise

Test pairwise

- ✔ Peut être utilisé avec l'analyse partitionnelle de plusieurs paramètres
- ✔ Génération outillée (voir `http://www.pairwise.org`)
- ✗ Combinaison aléatoire des valeurs
- ✗ Résultat attendu à fournir manuellement

Extension : Combinaisons de trois valeurs, quatre valeurs, etc.

Mais : Explosion du nombre de tests

Test pairwise

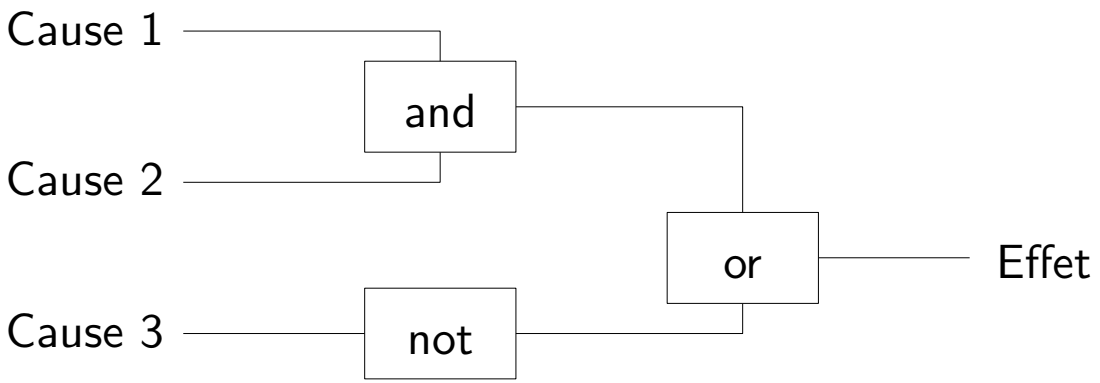
Test de configuration

On veut tester l'impression de fichiers depuis plusieurs applications sur des OS et via des réseaux différents. Donner les tests permettant de couvrir tous les couples de valeurs.

OS	Réseau	Imprimante	Application
Windows 7	IP	HP35	Word
Linux	Wifi	Canon900	Excel
Mac OS X	Bluetooth		PowerPoint
			AcrobatReader

Graphe cause-effet

Représentation graphique des relations logiques entre les entrées et les sorties d'un programme



Principe : couvrir les liens causes-effet

- Un test pour chaque combinaison de causes ayant un effet

Méthode de construction des tests

- Identifier les causes et les effets
- Établir le graphe des relations entre causes et effets
- Convertir le graphe en table de décision
- Réduire la table de décision en éliminant les causes inutiles à un effet
- Construire un test pour chaque façon de produire un effet

Graphe cause-effet

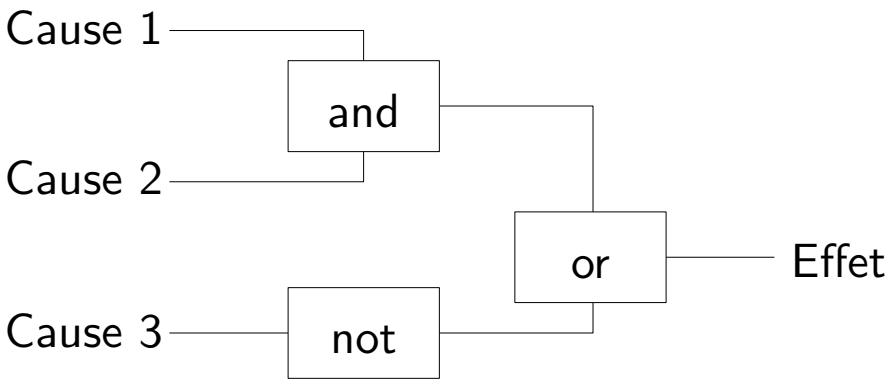


Table de décision

Cause 1	0	0	0	0	1	1	1	**1**
Cause 2	0	0	1	1	0	0	1	**1**
Cause 3	**0**	1	**0**	1	**0**	1	**0**	1
Effet	x		x		x		x	x

Méthode de construction des tests

- Identifier les causes et les effets
- Établir le graphe des relations entre causes et effets
- Convertir le graphe en table de décision
- Réduire la table de décision en éliminant les causes inutiles à un effet
- Construire un test pour chaque façon de produire un effet

Graphe cause-effet

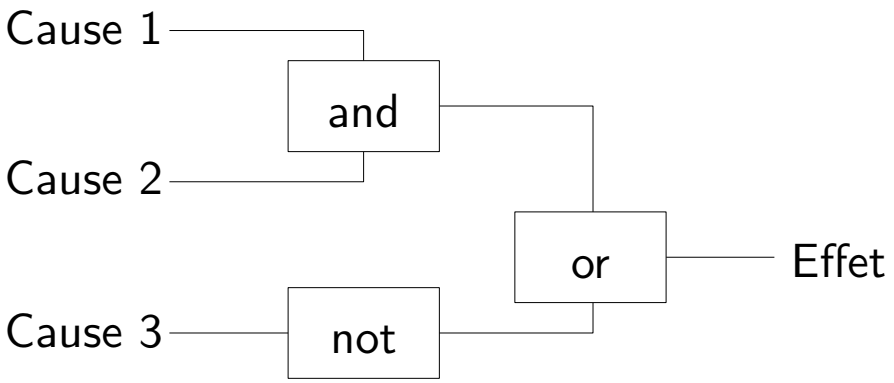


Table de décision simplifiée

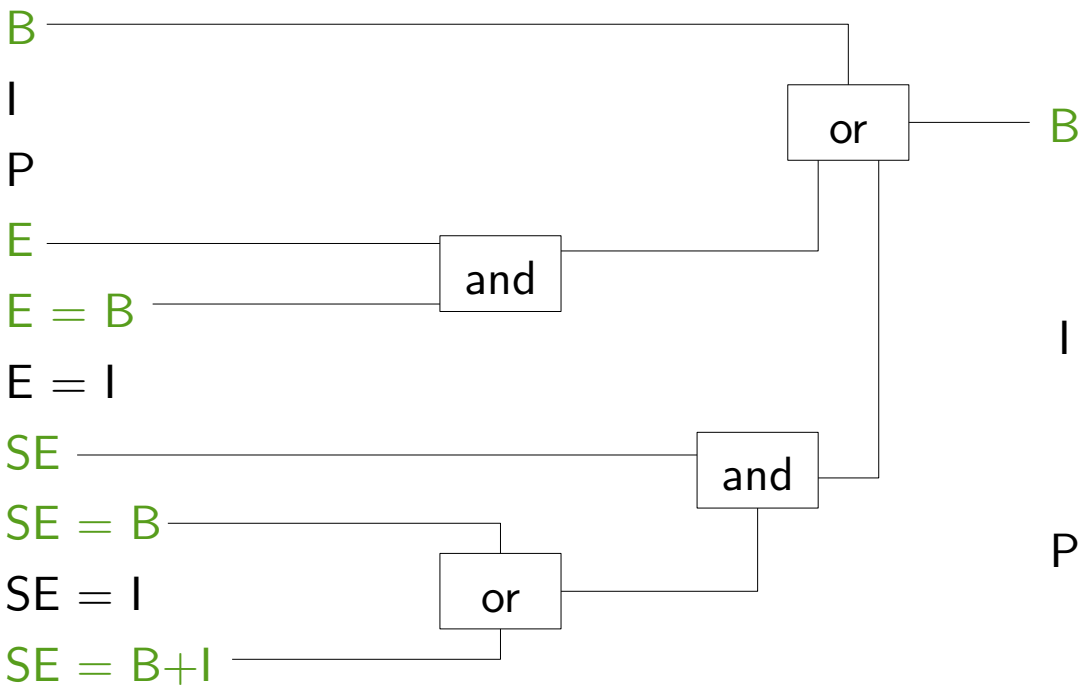
Cause 1	**x**	0	0	1	**1**
Cause 2	**x**	0	1	0	**1**
Cause 3	**0**	1	1	1	**x**
Effet	x				x

Test 1 : Cause 3 fausse

Test 2 : Cause 1 et 2 vraies

Graphe cause-effet

Mise en forme de texte



4 tests pour l'effet B :

- B
- E et E = B
- SE et SE = B
- SE et SE = B+I

B : bold, I : italic, P : plain, E : emphasized, SE : super emphasized