Chapitre 2.
Test Fonctionnel (Test Boite Noire)

a.hettab@centre-univ-mila.dz

Introduction

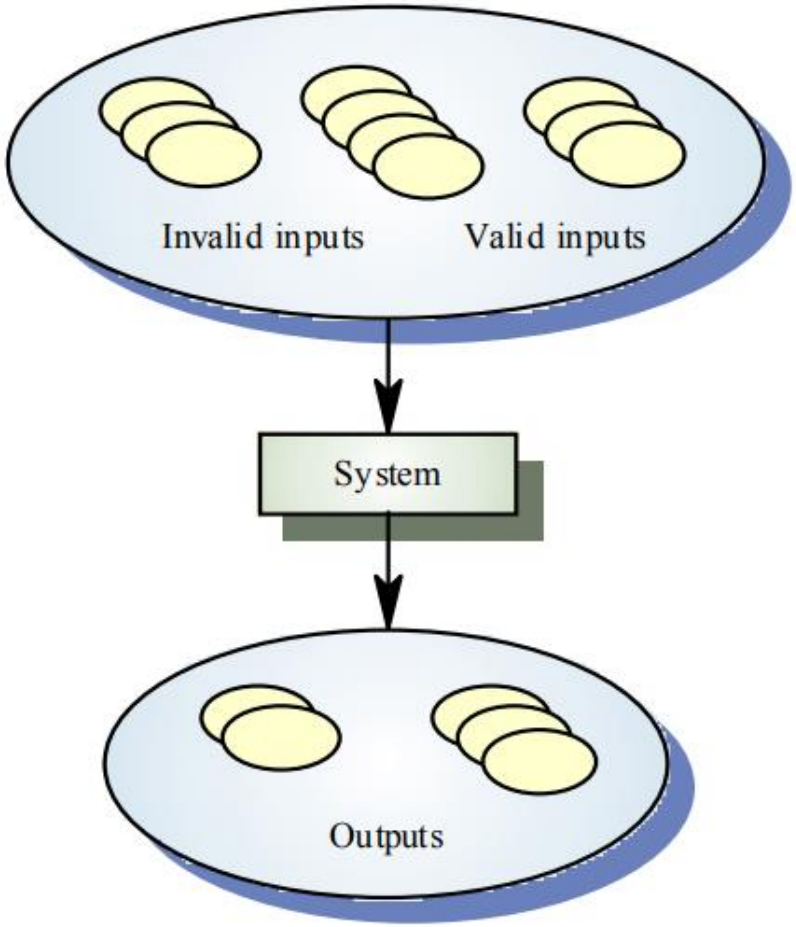
- Le **test fonctionnel** est un processus visant à identifier d'éventuels écarts entre un programme et sa spécification initiale.
- La **spécification initiale** est une description précise du comportement attendu du programme du point de vue de l'utilisateur final.
- Le test fonctionnel est généralement réalisé selon une approche **boîte noire**, ce qui signifie qu'il se concentre sur les entrées et sorties du système sans prendre en compte son implémentation interne.
- Pour mener un test fonctionnel, il est essentiel d'analyser la spécification afin d'en déduire un ensemble de **cas de test** permettant de vérifier le bon fonctionnement du programme.

Méthodes de test fonctionnel

- Il existe plusieurs techniques pour générer des **cas de test** et des **données de test** à partir d'une spécification. Chaque technique de **test fonctionnel** propose une méthode spécifique, basée sur un **critère de sélection ou de production** des données de test à partir des spécifications. Parmi les techniques de test fonctionnel les plus courantes, on retrouve:
- **Partitionnement en classes d'équivalence**
- **Test aux limites**
- **Test combinatoire – Algorithmes Pairwise**
- **Test aléatoire**
- **Graphe causes – effets / tables de décision**
- **Génération automatique de tests à partir d'une spécification**

Partitionnement en classes d'équivalence

- **classe d'équivalence:**
- **Une classe d'équivalence** correspond à un ensemble de données de tests supposées tester le même comportement, c'est-à-dire activer le même défaut.



Partitionnement en classes d'équivalence

- **Exemple**
- Soit un programme qui lit trois nombres réels a, b et c, représentant les **longueurs** des côtés d'un **triangle**. Le programme doit d'abord vérifier si ces trois valeurs peuvent réellement former un triangle. Si ce n'est pas le cas, il affiche un message approprié. Dans le cas contraire, il détermine et indique le type de triangle correspondant :

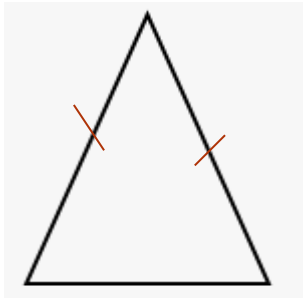
  - équilatéral

  - isocèle: peut être **acutangle (La plus grande angle < 90°), rectangle (La plus grande angle = 90°), ou obtusangle (La plus grande angle > 90°).**

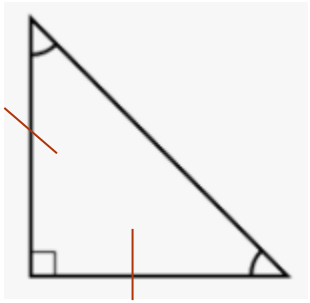
  - scalène (quelconque) : peut être **acutangle, rectangle, ou obtusangle.**

Partitionnement en classes d'équivalence

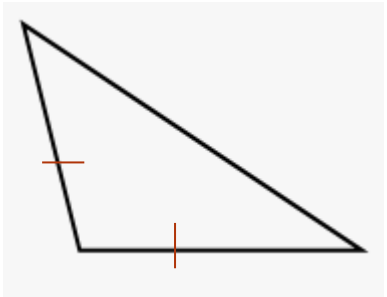
- **Exemple (suite)**



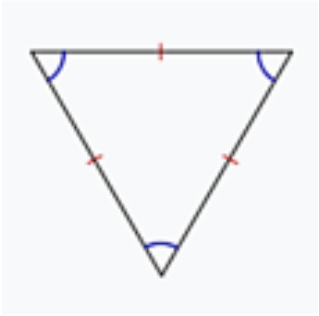
Isocèle acutangle



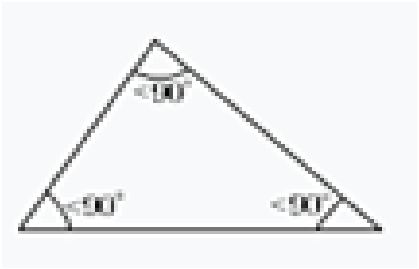
Isocèle rectangle



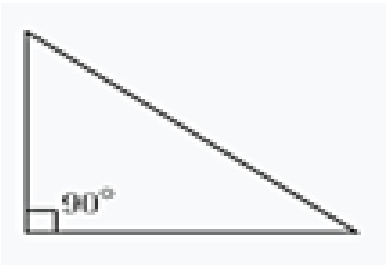
Isocèle obtusangle



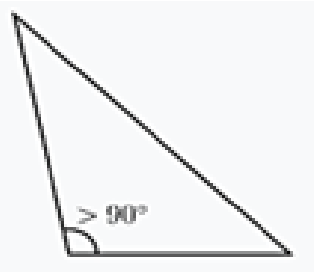
Equilatéral



Scalène acutangle



Scalène rectangle



Scalène obtusangle

Partitionnement en classes d'équivalence

- **Exemple (suite)**
- **Algorithme:**
- Si ces trois nombres ne correspondent pas à un triangle **((a=<0) ou (b=<0) ou (c=<0) ou (a+b<c)ou(b+c<a)ou(a+c<b))**, imprimer le message « pas de triangle ».
- Dans le cas d'un triangle, le programme examine s'il s'agit d'un triangle

  - équilatéral: **si (a=b) et (b=c)** et **par conséquent (a=c)** **sinon**

Partitionnement en classes d'équivalence

- **Exemple (suite)**

- isocèle: si (a=b) ou (a=c) ou (b=c) :

- isocèle acutangle (La plus grande angle < 90°) si ((a*a<b*b+c*c) et (a>=b et a>=c)) ou ((b*b<a*a+c*c) et (b>=a et b>=c)) ou ((c*c<a*a+b*b) et (c>=a et c>=b))

- isocèle rectangle (La plus grande angle = 90°): si (a*a=b*b+c*c) ou (b*b=a*a+c*c) ou (c*c=a*a+b*b)

- isocèle obtusangle (La plus grande angle > 90°): si ((a*a>b*b+c*c) et (a>=b et a>=c)) ou ((b*b>a*a+c*c) et (b>=a et b>=c)) ou ((c*c>a*a+b*b) et (c>=a et c>=b)) sinon

Partitionnement en classes d'équivalence

- **Exemple (suite)**

- scalène (quelconque):

- scalène acutangle (La plus grande angle < 90°) si ((a*a<b*b+c*c) et (a>=b et a>=c)) ou ((b*b<a*a+c*c) et (b>=a et b>=c)) ou ((c*c<a*a+b*b) et (c>=a et c>=b))

- scalène rectangle (La plus grande angle = 90°): si (a*a=b*b+c*c) ou (b*b=a*a+c*c) ou (c*c=a*a+b*b)

- scalène obtusangle (La plus grande angle > 90°): si ((a*a>b*b+c*c) et (a>=b et a>=c)) ou ((b*b>a*a+c*c) et (b>=a et b>=c)) ou ((c*c>a*a+b*b) et (c>=a et c>=b))

Partitionnement en classes d'équivalence

- **Exemple (suite)**
- Dans cet exemple il y a 8 classes d 'équivalence (7 valides et 1 invalide) et pour chaque classe on définit les données de test appropriées.

	Acutangle	Rectangle	Obtusangle
Scalène	6, 5, 4	3, 4, 5	5, 6, 10
Isocèle	6, 1, 6	√2, 2, √ 2	7, 4, 4
Équilatéral	4, 4, 4	Impossible	Impossible
+ non triangle -1, 2, 8			

Partitionnement en classes d'équivalence

- **Stratégie de Test:**
- Pour chaque donnée d'entrée, identification des classes d'équivalence **valides** et **invalides** en fonction des domaines de valeurs.
- Sélection d'un **représentant** pour chaque classe d'équivalence.
- Combinaison des données d'entrée par **produit cartésien** afin d'établir les données de test (DT).
- Application du test par **classes d'équivalence fort**.
- Remarque : Il existe également le test par **classes d'équivalence faible**, qui consiste à choisir une seule valeur représentative par classe d'équivalence, sans appliquer le produit cartésien.

Partitionnement en classes d'équivalence

- **Stratégie de Test (suite) :**
- **Identification des classes d'équivalence:**
- Si la **valeur** appartient à un **intervalle**, construire :
  - **une classe** pour les **valeurs inférieures** à l'intervalle,
  - **une classe** pour les **valeurs supérieures** à l'intervalle,
  - **n classes valides**.
- Si la **donnée** est un **ensemble de valeurs**, construire :
  - **une classe** avec l **'ensemble vide**,
  - **une classe** avec **trop de valeurs**,
  - **n classes valides**.

Partitionnement en classes d'équivalence

- **Stratégie de Test (suite)**
- Si **la donnée** est une **obligation** ou **une contrainte** (forme, sens, syntaxe), construire :
  - **une classe** avec la **contrainte respectée**,
  - **une classe** avec la **contrainte non-respectée.**

Partitionnement en classes d'équivalence

- **Exemple:** Quelles **données de test** pour une méthode qui s'appelle ***NextDate*** qui calcule le **lendemain** d'une date (jour, mois, année) passée en paramètre?
- **Données** : **mois**, **jour**, **année** représentant une **date:**
  - **1 ≤ mois ≤ 12**
  - **1 ≤ jour ≤ 31**
  - **1000 ≤ année ≤ 3000**
- **Résultat : date du jour suivant.**
- **Remarque:** la date donnée doit considérer les années **bissextiles** : Année bissextile **si divisible par 4** et n'est pas divisible par 100. Ou **siècle si multiple de 400.**

Partitionnement en classes d'équivalence

- **Exemple (suite):**
- Dans cet exemple les données de test forment des **intervalles** avec les **contraintes** de **l'année bissextile** et les mois qui contient **30 ou 31 jours.**
- Les classes d'équivalence pour la variable **mois** sont les:
- **M1 = { MOIS: MOIS à 30 jours}**
- **M2 = { MOIS: MOIS à 31 jours}**
- **M3 = { MOIS: MOIS est Février}**

Partitionnement en classes d'équivalence

- **Exemple (suite):**
- Les classes d'équivalence la variable **jour** sont les :
- **D1 = {JOUR: 1<= JOUR <= 28}**
- **D2 = {JOUR: JOUR = 29}**
- **D3 = {JOUR: JOUR = 30}**
- **D4 = {JOUR: JOUR = 31}**
- **D5 = {JOUR: JOUR < 1}** **// invalide**
- **D6 = {JOUR: JOUR > 31}** **// invalide**

Partitionnement en classes d'équivalence

- **Exemple (suite):**
- Les classes d'équivalence la variable **année** sont les :
- **Y1 = {ANNEE: ANNEE = 1900}**
- **Y2 = {ANNEE: ANNEE = 2000}**
- **Y3 = {ANNEE: 1000 <= ANNEE <= 3000 et (ANNEE != 1900) et (ANNEE mod 4 = 0)}**
- **Y4 = {ANNEE: (1000 <= ANNEE <= 3000 et ANNEE mod 4 != 0)}**
- **Y5 = {ANNEE: (ANNEE <1000 } // invalide**
- **Y6 = {ANNEE: (ANNEE >3000 } // invalide**

Partitionnement en classes d'équivalence

- **Exemple (suite):**
- Les cas de test obtenu par le produit cartésien des classes d'équivalence sont:

Id Cas de test	Mois	Jours	Année	Sortie Attendue
1	4	28	2004	29-04-2004
2	4	29	2004	30-04-2004
3	4	30	2004	1-05-2004
4	4	31	2004	Erreur
5	4	0	2004	Erreur
6	4	33	2004	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
7	4	28	2009	29-04-2009
8	4	29	2009	30-04-2009
9	4	30	2009	1-05-2009
10	4	31	2009	Erreur
11	4	0	2009	Erreur
12	4	33	2009	Erreur
13	4	28	1900	29-04-1900
14	4	29	1900	30-04-1900
15	4	30	1900	01-05-1900
16	4	31	1900	Erreur
17	4	0	1900	Erreur
18	4	33	1900	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
19	4	28	2000	29-04-2000
20	4	29	2000	30-04-2000
21	4	30	2000	01-05-2000
22	4	31	2000	Erreur
23	4	0	2000	Erreur
24	4	33	2000	Erreur
25	4	28	950	Erreur
26	4	29	950	Erreur
27	4	30	950	Erreur
28	4	31	950	Erreur
29	4	0	950	Erreur
30	4	33	950	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
31	4	28	3100	Erreur
32	4	29	3100	Erreur
33	4	30	3100	Erreur
34	4	31	3100	Erreur
35	4	0	3100	Erreur
36	4	33	3100	Erreur
37	1	28	2004	29-01-2004
38	1	29	2004	30-01-2004
39	1	30	2004	31-01-2004
40	1	31	2004	01-02-2004
41	1	0	2004	Erreur
42	1	33	2004	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
43	1	28	2009	29-01-2009
44	1	29	2009	30-01-2009
45	1	30	2009	31-01-2009
46	1	31	2009	01-02-2009
47	1	0	2009	Erreur
48	1	33	2009	Erreur
49	1	28	1900	29-01-1900
50	1	29	1900	30-01-1900
51	1	30	1900	31-01-1900
52	1	31	1900	01-01-1900
53	1	0	1900	Erreur
54	1	33	1900	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
55	1	28	2000	29-01-2000
56	1	29	2000	30-01-2000
57	1	30	2000	31-01-2000
58	1	31	2000	01-02-2000
59	1	0	2000	Erreur
60	1	33	2000	Erreur
61	1	28	950	Erreur
62	1	29	950	Erreur
63	1	30	950	Erreur
64	1	31	950	Erreur
65	1	0	950	Erreur
66	1	33	950	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
67	1	28	3100	Erreur
68	1	29	3100	Erreur
69	1	30	3100	Erreur
70	1	31	3100	Erreur
71	1	0	3100	Erreur
72	1	33	3100	Erreur
73	2	28	2004	29-02-2004
74	2	29	2004	01-03-2004
75	2	30	2004	Erreur
76	2	31	2004	Erreur
77	2	0	2004	Erreur
78	2	33	2004	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
79	2	28	2009	01-03-2004
80	2	29	2009	Erreur
81	2	30	2009	Erreur
82	2	31	2009	Erreur
83	2	0	2009	Erreur
84	2	33	2009	Erreur
85	2	28	1900	01-03-2009
86	2	29	1900	Erreur
87	2	30	1900	Erreur
88	2	31	1900	Erreur
89	2	0	1900	Erreur
90	2	33	1900	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
91	2	28	2000	29-02-1900
92	2	29	2000	01-03-2000
93	2	30	2000	Erreur
94	2	31	2000	Erreur
95	2	0	2000	Erreur
96	2	33	2000	Erreur
97	2	28	950	Erreur
98	2	29	950	Erreur
99	2	30	950	Erreur
100	2	31	950	Erreur
101	2	0	950	Erreur
102	2	33	950	Erreur

Partitionnement en classes d'équivalence

Id Cas de test	Mois	Jours	Année	Sortie Attendue
103	2	28	3100	Erreur
104	2	29	3100	Erreur
105	2	30	3100	Erreur
106	2	31	3100	Erreur
107	2	0	3100	Erreur
108	2	33	3100	Erreur

Partitionnement en classes d'équivalence

- **Exemple (suite):**
- **Remarque:** Si l'on applique le **test par classes d'équivalence faible** (c'est-à-dire en choisissant une seule valeur représentative par classe d'équivalence, sans produit cartésien), on obtient les **cas de test** suivants :

Id Cas de test	Mois	Jours	Année	Sortie Attendue
1	2	28	2004	29-04-2004
2	4	29	2009	30-04-2009
3	1	30	1900	31-01-1900
4	2	31	2000	Erreur
5	4	0	950	Erreur
6	1	33	3100	Erreur

Partitionnement en classes d'équivalence

Discussion : Quand choisir le test par classes d'équivalence forte ou faible ?

- **Test par classes d'équivalence fort: Si** des **conditions d'erreurs** sont une **haute priorité**, nous devrions étendre le test par **classes d'équivalence forte** en incluant **les classes invalides**.
- **Test par classes d'équivalence faible** : Approprié lorsque les **données d'entrée** sont définies en termes de **domaines et d'ensembles de valeurs discrètes**.
- **Remarque:** La méthode des **classes d'équivalence fortes** fait la supposition que les **variables sont indépendantes – les dépendances génèreront des "erreurs"** de cas de test.

Partitionnement en classes d'équivalence

- **Discussion (suite): Les avantages de cette méthode de partitionnement en classe d'équivalence sont:**
- Approche **systématique** et elle donne **une bonne couverture**.
- **Praticable** dans les **tests** des **logiciels réels**.
- Permet de **détecter des incohérences au niveau des spécifications**.
- **Inconvénients:**
- **Nécessite** un **effort d'abstraction** et **d'analyse** des **spécifications**.
- La **spécification ne définit pas** toujours les **résultats attendus** pour les **cas de tests invalides**.
- La prise en compte de **toutes les classes d'équivalences** peut amener à générer un **très grand nombre de cas de test.**

Test aux limites

- Le test aux limites est une méthode de test fonctionnel consiste à vérifier le comportement du logiciel aux frontières de ses domaines de fonctionnement.
- Cette technique de test est complémentaire au test de partitionnement en classe d'équivalence vu précédemment.
- Le but est de détecter les erreurs typiques cachées dans les limites et frontières des classes d'équivalence.

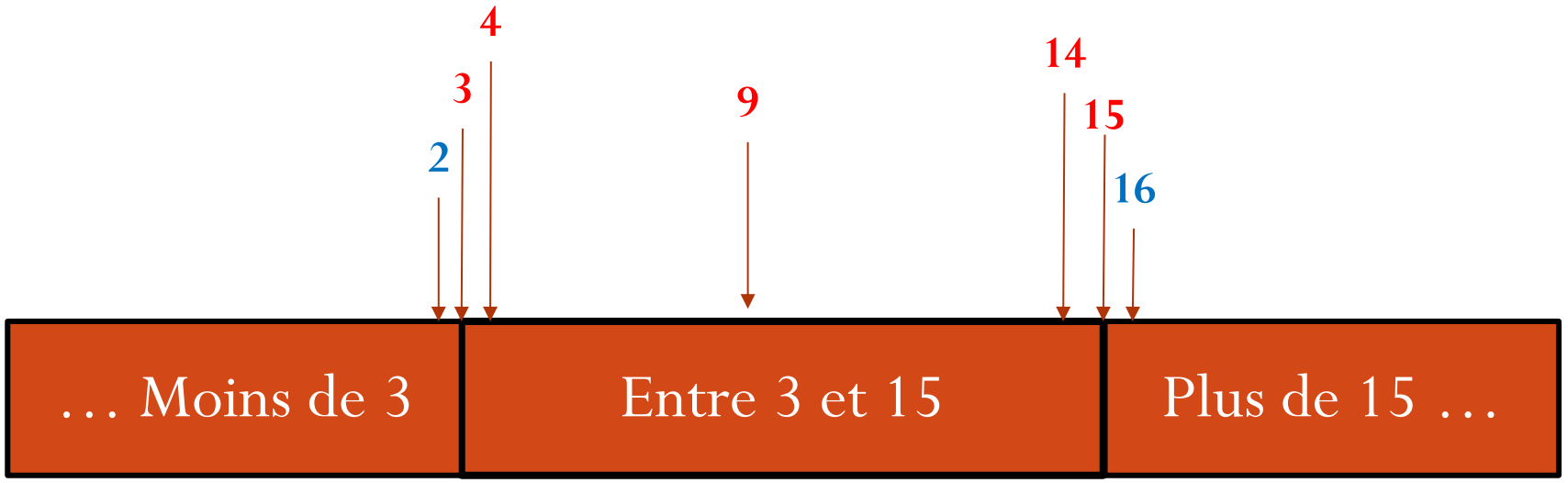
Test aux limites

- **Principe** :
- le principe de la technique de test aux limites **est de partitionner** les domaines des variables d'entrées en **classes d'équivalence** et de **s'intéresser aux bornes des intervalles** comme suit:
- **Si les variables appartiennent à un intervalle**, on prend **une valeur au milieu, les 2 valeurs** correspondant **aux 2 limites ou frontières** de l'intervalle, et **les 4 valeurs** correspondant aux valeurs **des limites de l'intervalle ± le plus petit delta possible**.

Test aux limites

- **Principe** (suite):
- **Exemple**:

Si n ∈ [3 .. 15] alors **v1 = 3, v2 = 15, v3 = 2, v4 = 4, v5=9, v6 = 14, v7 = 16**



Test aux limites

- **Principe (suite)** :
- **Si les variables appartiennent à un ensemble ordonné de valeurs**, on prend un **élément en milieu**, le **premier** élément, le **second**, l **'avant dernier** et le **dernier**.

Exemple:

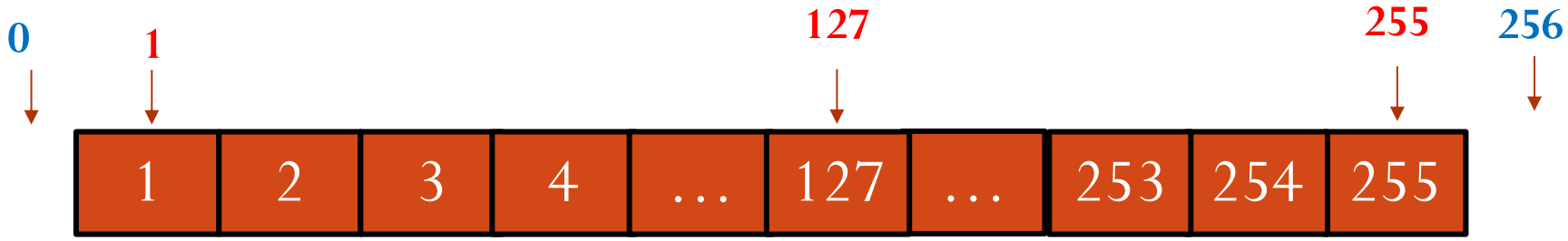
Si n ∈ {-7, 2, 3, 10, 38, 95,123, 157, 200} alors **v1 = -7**, **v2 = 2**, **v3=95**, **v4 = 157**, **v5 = 200.**

Test aux limites

- **Principe (suite)** :
- **Si les variables forment une condition qui spécifie un nombre de valeurs**, alors les cas de test sont définis à partir du **nombre minimum** et **maximum** de valeurs, une **valeur au milieu**, plus les nombres de valeurs **hors limites invalides** supérieures et inferieures.

Exemple:

Si On a un fichier d'entrée qui contient entre **1-255 records**, alors on obtient des données de test pour **0**, **1**, **127**, **255** **et** **256**.



Test aux limites

- **Principe (suite)** :
- les données d'entrée **ne sont pas toujours** des **valeurs numériques**. Elles peuvent être des **caractères** ou **des chaines de caractères**, des valeurs **booléens**, des **images**, de **son**, …etc.
- Dans ces catégories de données, l'espace des valeurs est partitionné en classes d'équivalence, puis les conditions aux limites sont extraites:
  - **True / False** pour les booléens
  - **Fichier plein / Fichier vide** pour les fichiers
  - **Nuances de couleur** dans les images
  - …**etc.**

Test aux limites

- **Principe (suite)** :
- **Exemple:**
- Si on a en entrée une fonction qui vérifie si une chaîne de caractère correspond à "**oui**" ou "**non**" , alors on obtient des données de test suivantes **:**

  - "oui",

  - "non" ,

  - et une autre chaine quelconque **"adfg"** par exemple.

Test aux limites

- **Exemple**:
- On reprend l'exemple de la méthode ***NextDate*** qui calcule le **lendemain** d'une date (jour, mois, année) passée en paramètre.
- **Données** : **mois**, **jour**, **année** représentant une **date:**

  - **1 ≤ mois ≤ 12**

  - **1 ≤ jour ≤ 31**

  - **1000 ≤ année ≤ 3000**

- **Résultat : date du jour suivant.**

Test aux limites

- **Exemple (suite)**:
- Pour **l'année** on prend:

- l'année 999
- l'année 1000
- L'année1001
- L'année 2999
- L'année 3000
- L'année 3001
- Une année au milieu 2003
- Une année bissextile et pas siècle 2004
- Une année bissextile et siècle 2000
- Une année non bissextile et siècle 1900
- Une année non bissextile et non siècle 2015

Test aux limites

- **Exemple (suite)**:
- Pour le **mois** on prend **:**

  - le mois 0

  - le mois 1

  - le mois 2

  - le mois 6

  - le mois 11

  - le mois 12

  - le mois 13

Test aux limites

- **Exemple (suite)**:
- Pour le **jour** on prend **:**

- avec un mois avec 31 jours

- le jour 0 - le jour 1 - le jour 2

- un jour au milieu 16

- Le jour 30 - le jour 31 - le jour 32

- avec un mois avec 30 jours

- On prend les jours suivants: {0,1,2,15,29,30,31}

- avec le mois de février

- On prend les jours suivants: {0,1,2,14,27,28,29,30}

Test aux limites

- **Discussion**:
- Le test aux limites est une méthode efficace pour sélectionner les données d'entrée au sein des classes d'équivalence, car la plupart des erreurs se situent aux limites et aux frontières des domaines des variables.
- Les tests aux limites produisent à la fois des cas de test nominaux (dans l'intervalle) et de robustesse (hors intervalle).
- **Inconvénient**
- Le test aux limites n'est utilisable que une relation d'ordre est présente sur la donnée d'entrée.

Test combinatoire

- **Le test combinatoire** est une technique de génération de cas de tests utilisée pour tester un objet ayant plusieurs paramètres, et chaque paramètre ayant plusieurs valeurs. Les cas de test sont générés en identifiant un sous-ensemble approprié de combinaisons de tests pour atteindre un niveau de couverture prédéfini.

Test combinatoire

- **Problématique de test combinatoire**
- La combinaison de toutes les valeurs des domaines d'entrée entraîne une explosion combinatoire.
- **Exemple** : Options d'une boite de dialogue MS Word (voir la diapositive suivante).

Nombre de tous les combinaisons possibles est : $2^{15}*3*3 = 294912$

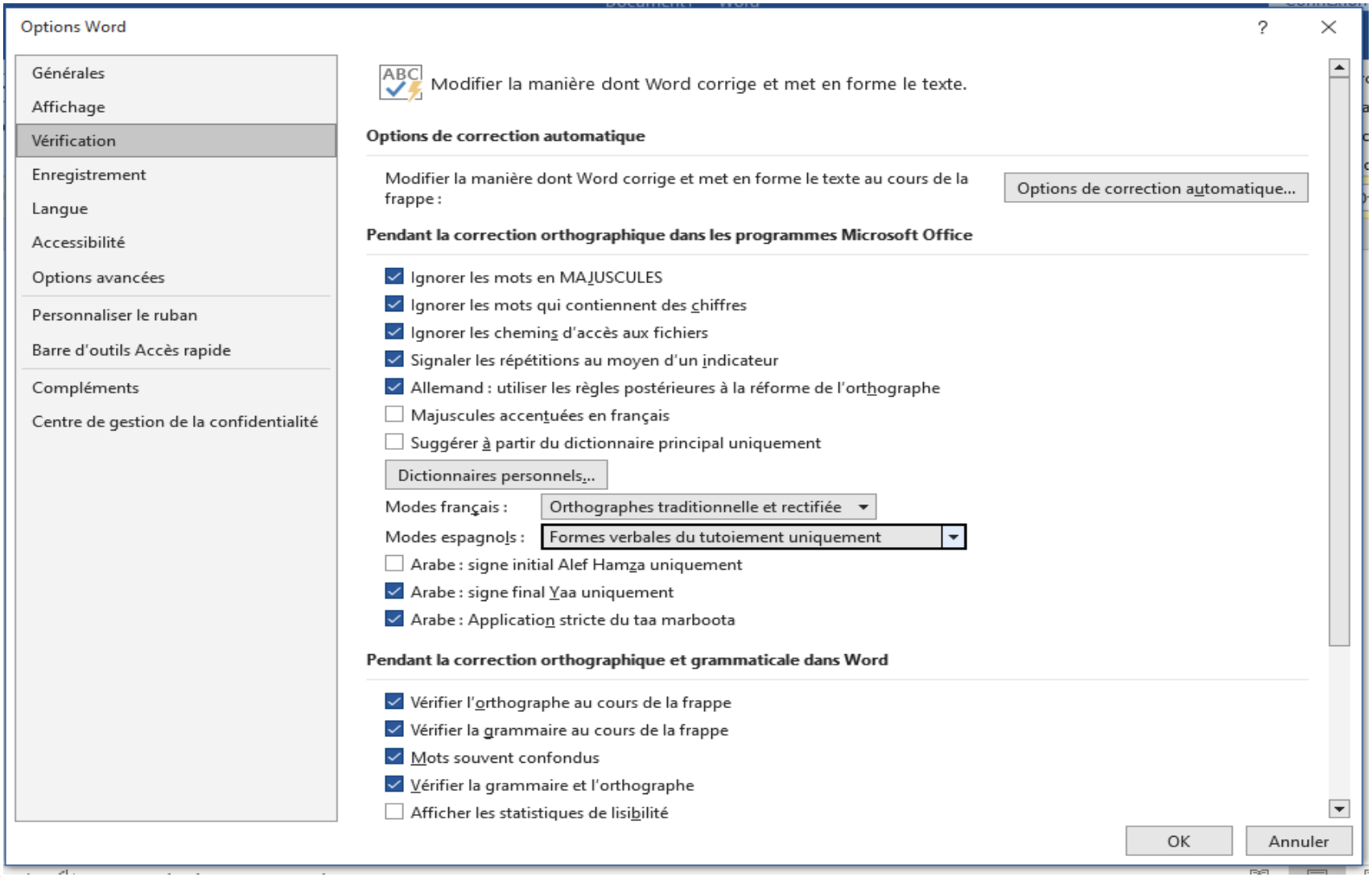
2 Le nombre de choix dans chaque case à cocher (cochée ou décochée)

15 le nombre de case à cochers

3 est le nombre d'item dans le premier menu déroulant

3 est le nombre d'item dans le second menu déroulant

Test combinatoire



Test combinatoire

Approche de Test par table orthogonal

- **Test par table orthogonal:** Une approche de test basée sur les tableaux orthogonaux (des tableaux à deux dimensions construit à partir de propriétés mathématiques particulières) utilisée pour tester toutes les combinaisons de paires de valeurs des paramètres.

Test combinatoire

Approche de Test par table orthogonal

- **Construction d'une table orthogonale:**
- Prenons l'exemple d'une situation avec **trois paramètres**, chacun ayant **deux valeurs**. Les étapes de construction de la table orthogonale sont les suivantes :
- **1.** Dans la première colonne de la table, les deux valeurs du premier paramètre sont répétées comme suit: valeur1, valeur2, valeur1, valeur2,…

2. Dans la seconde colonne de la table, les deux valeurs du deuxième paramètre sont répétées comme suit: valeur1, valeur1, valeur2, valeur2,…

3. Dans la troisième colonne de la table, les deux valeurs du troisième paramètre sont répétées comme suit: valeur1, valeur1, valeur1, valeur1, valeur2, valeur 2, valeur2, valeur2….

Test combinatoire

Approche de Test par table orthogonal

Paramètre 1	Paramètre 2	Paramètre 3
P1.V1	P2.V1	P3.V1
P1.V2	P2.V1	P3.V1
P1.V1	P2.V2	P3.V1
P1.V2	P2.V2	P3.V1
P1.V1	P2.V1	P3.V2
P1.V2	P2.V1	P3.V2
P1.V1	P2.V2	P3.V2
P1.V2	P2.V2	P3.V2

Test combinatoire

Approche de Test par table orthogonal

- Mathématiquement, cette table orthogonale s'écrit comme suit: $L_8(2^3)$.
- **De manière plus générale**, une table orthogonale s'exprime mathématiquement sous la forme: $L_x(\prod^n)$ avec:

- x le nombre de lignes totales de la table obtenus par le produit $\prod n^y$ où **n** est le nombre de valeurs de chaque paramètre, et **y** est le nombre de paramètres.

- **Exemple:**

Pour construire une table à partir de **2 paramètres** qui contient **2 valeurs** chacun, **1 paramètre** qui a **3 valeurs**, et **2 paramètres** qui ont **4 valeurs**, nous avons besoin de 192 lignes et 5 colonnes. L_{192} $(2^2\ 3^1\ 4^2)$.

Test combinatoire
réduire la combinatoire

- Pour réduire **la complexité combinatoire** tout en maintenant une **bonne qualité de couverture**, deux approches sont possibles :
- **Approche all singles (onewise):** Génère des jeux de tests où chaque valeur de chaque paramètre est utilisée au moins une fois.
- **Approche all pairs (pairwise) :** Génère des jeux de tests couvrant toutes les combinaisons possibles de paires de valeurs entre les paramètres d'entrée.

Test combinatoire
Approche all singles (onewise)

- Dans l'approche **All Singles (One-wise)**, pour chaque paramètre, on sélectionne **un** cas de test qui couvre l'une de ses valeurs.
- Le nombre de cas de test correspond au cardinal du paramètre ayant le plus grand ensemble de valeurs.
- Les valeurs des autres paramètres (ayant un ensemble de valeurs plus petit) sont réparties parmi les cas de test construits..

Test combinatoire
Approche all singles (onewise)

- **Exemple**:
- Pour tester un logiciel générant des scripts de validation du comportement des serveurs web, nous considérons les quatre paramètres suivants en entrée :
  - **Le type de système d'exploitation (OS) du client** :
    Ubuntu, **OpenSuse**, **Windows**, **Mac OS X**, **Chrome OS**
  - **Le type de navigateur web** : **Firefox**, **Opera**, **Chrome**
  - **Le type de données téléchargées** : **jpeg**, **avi**, **ogg**, **mp3**
  - **Le fait d'utiliser une connexion: sécurisée** ou **non**
- **Test exhaustif** : $5 \times 3 \times 4 \times 2 = 120$ cas de tests différents

Test combinatoire

Approche all singles (onewise)

- **Exemple (suite)**:
- En appliquant la méthode de test All Singles (One-wise), seuls cinq cas de test sont nécessaires, correspondant au cardinal du paramètre ayant le plus grand nombre de valeurs (systèmes d'exploitation: 5 valeurs):

	système d'exploitation	navigateur web	type de données téléchargées	fait d'utiliser une connexion
Cas de test 1	Ubuntu	Firefox	jpeg	Oui
Cas de test 2	OpenSuse	Opera	avi	Non
Cas de test 3	Windows	Chrome	ogg	Oui
Cas de test 4	Mac OS X	Firefox	mp3	Non
Cas de test 5	Chrome OS	Chrome	jpeg	Oui

Test combinatoire
Approche all singles (onewise)

- **Avantages:**
- L'approche **all singles (onewise)** permet de **réduire considérablement** le **nombre** de cas de tests
- Elle **garantie** une certaine **qualité** de test : chaque valeur d'un paramètre est testée au minimum une fois
- Elle permet la **détection** des **grosses erreurs** et les **oublis de réalisation**.
- **Inconvénients**
- C'est une méthode moins efficace de détecter les erreurs liées à une combinaison de paramètres.

Test combinatoire
Approche All Pairs (Pairwise)

- **Approche All Pairs (Pairwise):**
- Approche All Pairs (Pairwise) est une technique de test fonctionnel, consiste à tester toutes les combinaisons de paires de valeurs possibles des paramètres d'entrée.
- .

Test combinatoire
Approche All Pairs (Pairwise)

- **Exemple:**
- Un magasin de jouets vend des **petits trains** qui peuvent être **personnaliser** en utilisant une **application disponible** à l'entrée du magasin.
- **Le client** qui veut acheter des trains pourra :

  - choisir le **matériau** du train: **fer**, **bois** ou **plastique**

  - choisir la **couleur** du train: **Noir**, **Gris**, **Vert**

  - choisir d'ajouter **un logo** sur le dessus du train: **logo de la SNCF**, **logo TGV** ou **aucun logo**.

Test combinatoire
Approche All Pairs (Pairwise)

- **Exemple (suite):**
- En analysant cette spécification, **3 paramètres** sont identifiés et **chaque paramètre** contient **3 valeurs**:

  - **Matériau**: **fer**, **bois**, **plastique**

  - **Couleur**: **Noir**, **Gris**, **Vert**

  - **Logo**: **SNCF**, **TGV**, **Aucun**

- La combinatoire globale pour tester toutes les combinaisons possibles est de 3*3*3 = 27 combinaisons et notre table orthogonal se noterait $L_{27}(3^3)$.

Test combinatoire
Approche All Pairs (Pairwise)

- **Exemple (suite):**
- Pour réduire cette combinatoire nous utiliserons l'approche Pairwise comme suit:
- **La première étape** consiste à identifier **les combinaisons deux à deux**:

- matériau + couleur, **- matériau + Logo** , et **- couleur + logo**.

Cette première étape nous donne 9 tableaux de combinaison. Alors le nombre de lignes de notre tableau final est 9 et par conséquence le nombre cas de test qui permet de couvrir toutes les paires de valeurs est 9.

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

Matériau	Couleur
Fer	Noir
Bois	Noir
Plastique	Noir

Logo	Matériau
SNCF	Fer
TGV	Fer
Aucun	Fer

Logo	Couleur
SNCF	Noir
TGV	Noir
Aucun	Noir

Matériau	Couleur
Fer	Gris
Bois	Gris
Plastique	Gris

Logo	Matériau
SNCF	Bois
TGV	Bois
Aucun	Bois

Logo	Couleur
SNCF	Gris
TGV	Gris
Aucun	Gris

Matériau	Couleur
Fer	Vert
Bois	Vert
Plastique	Vert

Logo	Matériau
SNCF	Plastique
TGV	Plastique
Aucun	Plastique

Logo	Couleur
SNCF	Vert
TGV	Vert
Aucun	Vert

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**
- **La deuxième étape** consiste à établir les premières paires, en commençant par l'association du **matériau et de la couleur**.

TC	Matériau	Couleur	Logo
1	Fer	Noir	
2	Bois	Noir	
3	Plastique	Noir	
4	Fer	Gris	
5	Bois	Gris	
6	Plastique	Gris	
7	Fer	Vert	
8	Bois	Vert	
9	Plastique	Vert	

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**
- **Troisième étape** consiste à compléter le tableau avec les paires qui restent.

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	
5	Bois	Gris	
6	Plastique	Gris	
7	Fer	Vert	
8	Bois	Vert	
9	Plastique	Vert	

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

- Le choix précédent permet d'éliminer 3 autres paires induites.

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	
5	Bois	Gris	
6	Plastique	Gris	
7	Fer	Vert	
8	Bois	Vert	
9	Plastique	Vert	

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

- Puis nous poursuivons ainsi de suite jusqu'à avoir utiliser toutes les paires.

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	TGV
5	Bois	Gris	
6	Plastique	Gris	
7	Fer	Vert	Aucun
8	Bois	Vert	
9	Plastique	Vert	

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	TGV
5	Bois	Gris	
6	Plastique	Gris	
7	Fer	Vert	Aucun
8	Bois	Vert	
9	Plastique	Vert	

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	TGV
5	Bois	Gris	Aucun
6	Plastique	Gris	
7	Fer	Vert	Aucun
8	Bois	Vert	SNCF
9	Plastique	Vert	

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	TGV
5	Bois	Gris	Aucun
6	Plastique	Gris	
7	Fer	Vert	Aucun
8	Bois	Vert	SNCF
9	Plastique	Vert	

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	TGV
5	Bois	Gris	Aucun
6	Plastique	Gris	SNCF
7	Fer	Vert	Aucun
8	Bois	Vert	SNCF
9	Plastique	Vert	TGV

Test combinatoire

Approche All Pairs (Pairwise)

- **Exemple (suite):**

Matériau	Couleur	Logo	Matériau	Logo	Couleur
Fer	Noir	SNCF	Fer	SNCF	Noir
Bois	Noir	TGV	Fer	TGV	Noir
Plastique	Noir	Aucun	Fer	Aucun	Noir
Fer	Gris	SNCF	Bois	SNCF	Gris
Bois	Gris	TGV	Bois	TGV	Gris
Plastique	Gris	Aucun	Bois	Aucun	Gris
Fer	Vert	SNCF	Plastique	SNCF	Vert
Bois	Vert	TGV	Plastique	TGV	Vert
Plastique	Vert	Aucun	Plastique	Aucun	Vert

TC	Matériau	Couleur	Logo
1	Fer	Noir	SNCF
2	Bois	Noir	TGV
3	Plastique	Noir	Aucun
4	Fer	Gris	TGV
5	Bois	Gris	Aucun
6	Plastique	Gris	SNCF
7	Fer	Vert	Aucun
8	Bois	Vert	SNCF
9	Plastique	Vert	TGV

Test combinatoire
Approche All Pairs (Pairwise)

- **Discussion:**
- **L'approche Pairwise** est une méthode efficace pour **réduire la complexité combinatoire** en assurant la couverture de **toutes les paires de valeurs des paramètres**.
- Le tableau suivant illustre comment la méthode Pairwise permet de diminuer la combinatoire.

Valeurs et paramètres d'entrée	100% couverture (combinatoire totale)	Réduction possible avec Pairwise
$3^2 2^2 1^1$	36	9
$10^4 7^4 8^3 6^2 3^2 5^2 4^2 9^1$	14.338.695.168.000.000	144
10^{15}	1.000.000.000.000.000	199
$10^5 5^6 3^5 2^5$	2.430.000.000.000	142
$3^3 4^2 5^2 6^2 7^1 8^2 10^1 9^1$	15.676.416.000	96

Test combinatoire
Approche All Pairs (Pairwise)

- **Discussion (suite):**
- **Inconvénients:**
- L'utilisation de la méthode **pairwise** cause des fois des **paires impossibles** (deux paires incompatibles).
- **L'augmentation de la combinatoire** conduit à de tables pairwise de **grande taille**. Cela rend la **construction** de ses tables **irréaliste sans automatisation à l'aide d'un outil**. **Le site (http://pairwise.org/)** contient **45 outils** permettant **l'automatisation** de la construction des tables **pairwise**.

Graphe causes-effets/tables de décision

tables de décision

- **L'approche de test par table de décision** est une technique simple et fiable utilisée pour identifier les scénarios de test à partir de la **logique métier** complexe d'une application.
- **La logique métier** d'une application définit les schémas d'interaction entre l'application et l'utilisateur. Lorsqu'un utilisateur se connecte, remplit un formulaire d'inscription, etc., toutes ces actions sont traitées dans un ordre précis, formant ainsi des scénarios d'exécution et de test.

Graphe causes-effets/tables de décision

tables de décision

- **Un diagramme de flux de processus logique** est une représentation graphique sous forme de logigramme, modélisant le séquencement des activités et des tâches d'un processus.
- Pour une **exigence complexe**, un diagramme de flux inclut de nombreuses branches, nœuds et boîtes de décision.
- **L'approche de test par table de décision** est particulièrement utile dans le domaine de la logique métier. Elle permet de couvrir efficacement toutes les branches et d'explorer en profondeur l'ensemble d'un arbre logique complexe.
- Dans la suite, nous expliquerons comment préparer et présenter des cas de test pour une logique métier complexe en utilisant des tables de décision.

Graphe causes-effets/tables de décision

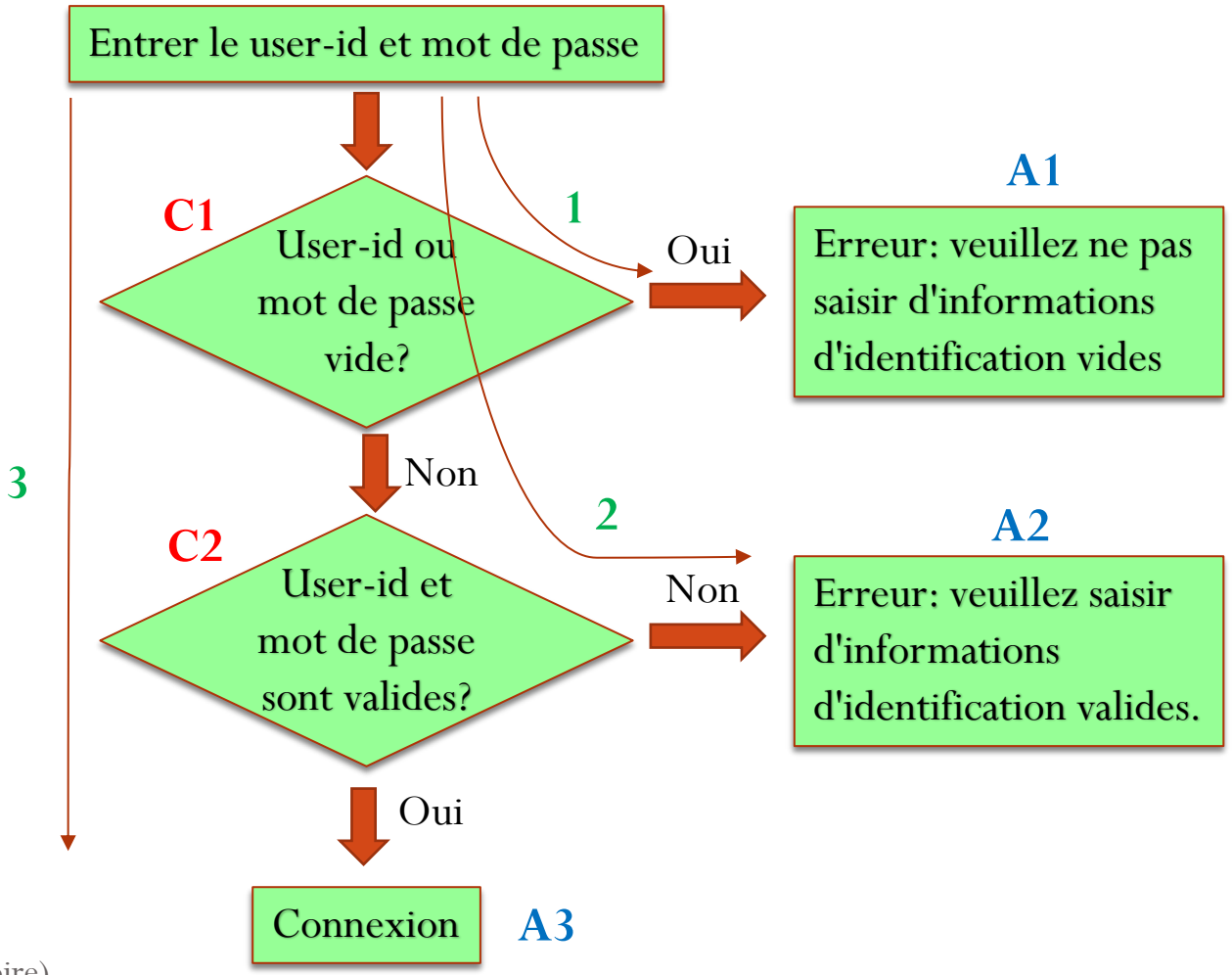
tables de décision

- **Construction de la table de décision:** Une des manières les plus simples de construire une table de décision est la suivante :
  - Identifier toutes les conditions de décisions (C1,C2,…,Cn)
  - Identifier toutes les actions résultantes (A1,A2,….,Am)
  - Identifier les relations entre les actions et les conditions (il y a des actions qui sont produites en réponse à une seule conditions et il y a des actions qui sont produites en réponse à une combinaison de décision)
  - Vérifier la cohérence et la complétude de la table.

Graphe causes-effets/tables de décision

tables de décision

- **Exemple:**



Graphe causes-effets/tables de décision

tables de décision

- **Exemple (suite) : Identification des conditions et des actions**
- **1 -ère étape:** Identifier toutes les conditions de décisions:
  - **C1**: User-id ou mot de passe est vide
  - **C2**: User-id et mot de passe sont valides
- **2 -ème étape:** Identifier toutes les actions:
  - **A1**: Erreur: veuillez ne pas saisir d'informations d'identification vides
  - **A2**: Erreur: veuillez saisir d'informations d'identification valides
  - **A3**: Connexion

Graphe causes-effets/tables de décision

tables de décision

- **Exemple (suite) :**
- **3 -ère étape:** Identifier les relations entre les actions et les conditions :
  - **C1 => A1**
  - **Non(C1) et non(C2) => A2**
  - **Non(C1) et C2 => A3**
- **4 -ème étape:** Construction de la table de décision:
  - Le nombre de ligne de la table sera **5** : 2 conditions + 3 actions
  - Le nombre de colonnes sera **4** = 2 × 2 (2 valeurs pour C1 et 2 valeurs pour C2)

Graphe causes-effets/tables de décision

tables de décision

- **Exemple (suite) :**

- Pour chaque condition ou action on met 1 pour True et 0 pour False.

Condition/ Action				
C1	0	1	0	1
C2	0	0	1	1
A1		1		1
A2	1			
A3			1	

Graphe causes-effets/tables de décision

tables de décision

- **Exemple (suite) :**

- Les cases vide seront complétées par des 0.

Action				
C1	0	1	0	1
C2	0	0	1	1
A1	0	1	0	1
A2	1	0	0	0
A3	0	0	1	0

Graphe causes-effets/tables de décision

tables de décision

- **Exemple (suite) :**
- **5 -ème étape:** Identification des cas de test:

Cas de test numéro	Branche	feuille	Validations		Résultat attendu	combinaisons
			User-id ou mot de passe vide	User-id et mot de passe sont valides		
1	1	A1	Oui	Non applicable	Erreur: veuillez ne pas saisir d'informations d'identification vides	- les deux vides - l'un d'eux vide
2	2	A2	Non	Non	Erreur: veuillez saisir d'informations d'identification valides	- les deux invalides - l'un d'eux invalide
3	3	A3	Non	Oui	Connexion	Non applicable

Graphe causes-effets/tables de décision

tables de décision

- **Discussion:**
- Dans **la table de décision**, il faut vérifier que :
  - Toutes les **validations** mentionnées dans les **cases de décision** doivent être présentes dans la table de décision.
  - Tous les **résultats attendus** mentionnés dans le diagramme de flux en tant que **feuilles** doivent être présentés dans la table de décision.
  - Toutes **les combinaisons** d'entrées nécessaires pour prendre une décision doivent être représentées dans le tableau de décision (Rubrique : **colonne de combinaisons**).

Graphe causes-effets/tables de décision

tables de décision

- **Discussion (suite):**

- **Avantages :** Cette technique

- Permet de modéliser facilement chaque flux métier complexe pouvant être représenté par un diagramme.

- Donne une confiance rapide sur les cas de test.

- Autoriser n'importe qui à créer des cas de test.

- Offre une très bonne couverture de test dès le premier coup.

Inconvénients:

- difficile de combiner cette technique avec d'autres techniques de test fonctionnel telles que le partitionnement en classes d'équivalence et les tests aux limites.

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Le graphe cause-effet** est une technique permettant de concevoir des cas de test où les causes représentent les conditions d'entrée du système et les effets correspondent aux résultats attendus de ces conditions.
- Il permet de générer un nombre minimal de cas de test à partir des exigences fonctionnelles, tout en assurant une couverture de test maximale.
- Cette approche contribue à réduire le temps et le coût d'exécution des tests.
- **Le graphe cause-effet** est utilisé pour modéliser les conditions d'entrée et de sortie d'une spécification fonctionnelle sous forme de relations logiques, en employant des opérateurs booléens tels que ET, OU et NON.

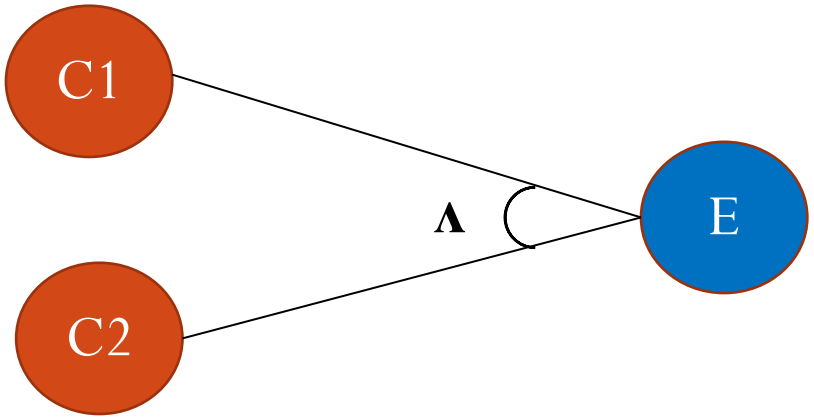
Graphe causes-effets/tables de décision

Graphe causes-effets

- **Notations utilisées:**
- **ET:** Pour que l'effet ***E*** soit vrai,

les deux causes ***C1*** et **C2**

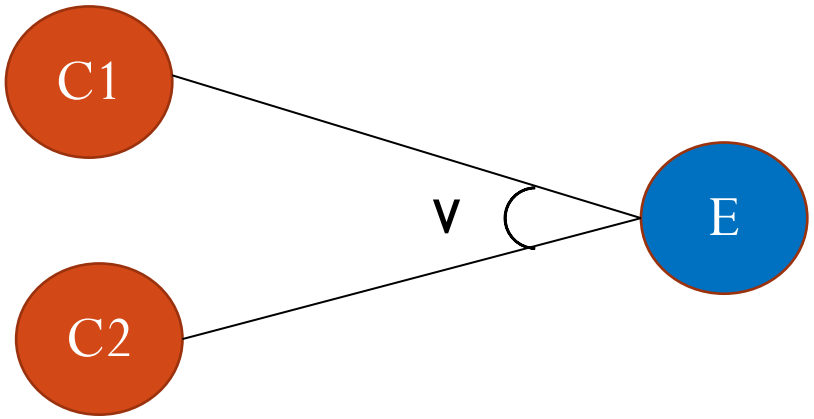
doivent être vraies.



- **OU:** Pour que l'effet ***E*** soit vrai,

l'une des causes ***C1*** ou ***C2***

doit être vraie.



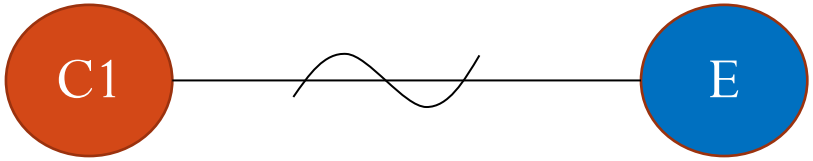
Graphe causes-effets/tables de décision

Graphe causes-effets

- **Notations utilisées (suite):**

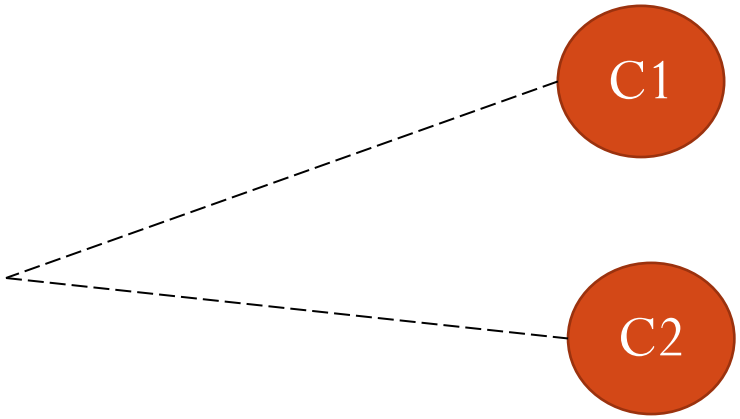
- **NON:** Pour que l'effet **E** soit vrai,

la cause ***C1*** doit être fausse.



- **Exclusion mutuelle :** lorsqu'une

seule des causes est vraie.



Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple:**
- Soit un logiciel d'impression qui lit deux caractères et imprime des messages, en fonction de valeurs de ces caractères.
- Le système fonctionne comme suit:

- Le premier caractère doit être un «A» ou un «B».
- Le deuxième caractère doit être un chiffre.
- **Si** les caractères sont corrects (le premier caractère est («A» ou «B») et le second caractère est un chiffre) **alors** le système imprime le fichier approprie.
- **Si** le premier caractère est incorrect (pas «A» ou «B») **alors** le système imprime le message X.
- **Si** le deuxième caractère est incorrect (pas un chiffre) **alors** le système imprime le message Y.

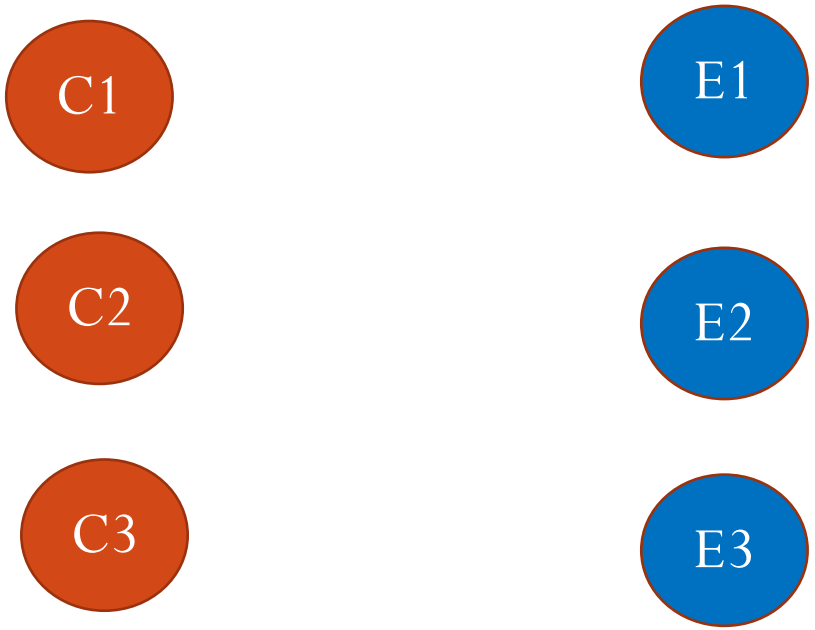
Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Identification des causes et effets:** dans ce système:
- Les **causes** sont:

- *C1* - **Le premier caractère est A**
- *C2* - **Le premier caractère est B**
- *C3* - **le deuxième caractère est un chiffre**

- **Les effets sont:**
- *E1* – **Imprimer le message approprie**
- *E2* - **Imprimer le message «X»**
- *E3* - **Imprimer le message «Y»**



Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Identification des relations entre les causes et les effets:**
- En commence par l'effet ***E1 (Imprimer le message approprie)***:
- Le fichier approprie est imprimé lorsque:

- Le premier caractère est ***«A»*** et le deuxième caractère est un **chiffre** => **C1 et C3 doivent être vrais**

- Le premier caractère est ***«B»*** et le deuxième caractère est un **chiffre** => **C2 et C3 doivent être vrais**

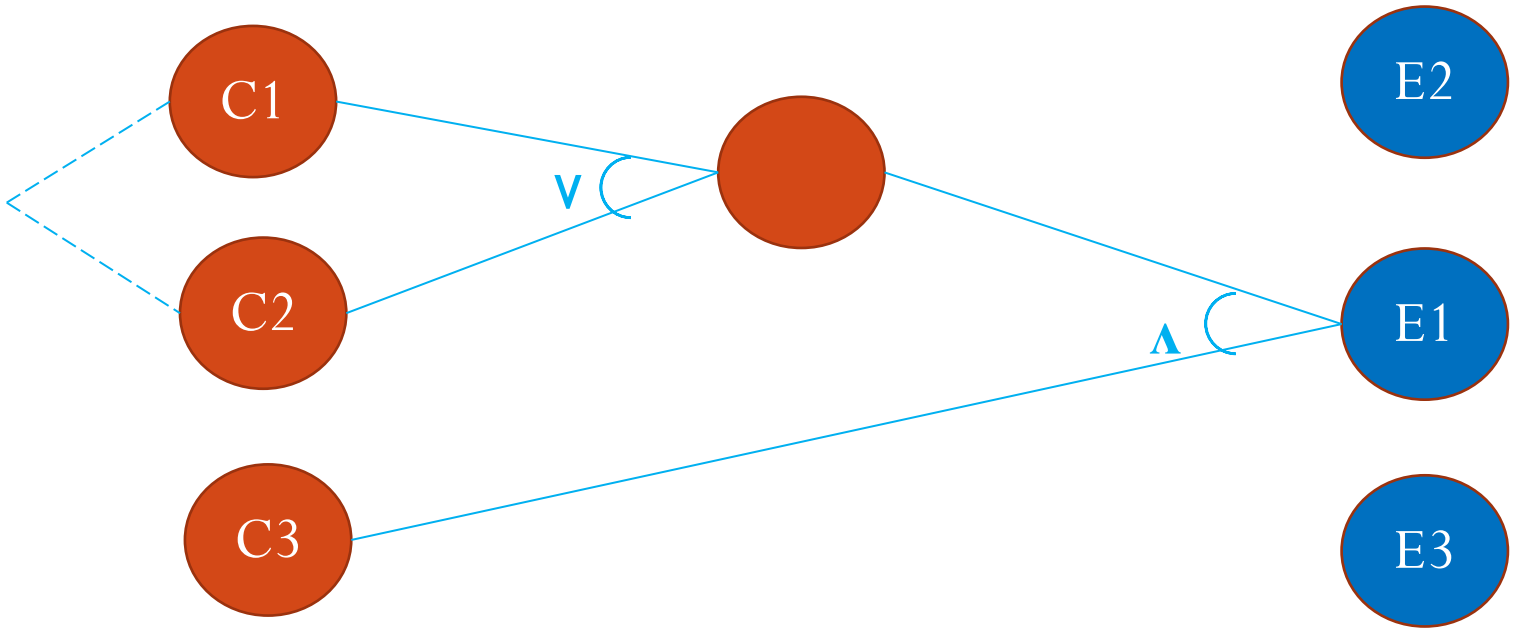
- Le premier caractère peut être **«A»** ou «B» et ne peut pas être les deux en même temps => **C1 et C2 sont en exclusion mutuelle**

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**

- **Identification des relations entre les causes et les effets (suite):**



Graphe causes-effets/tables de décision

Graphe causes-effets

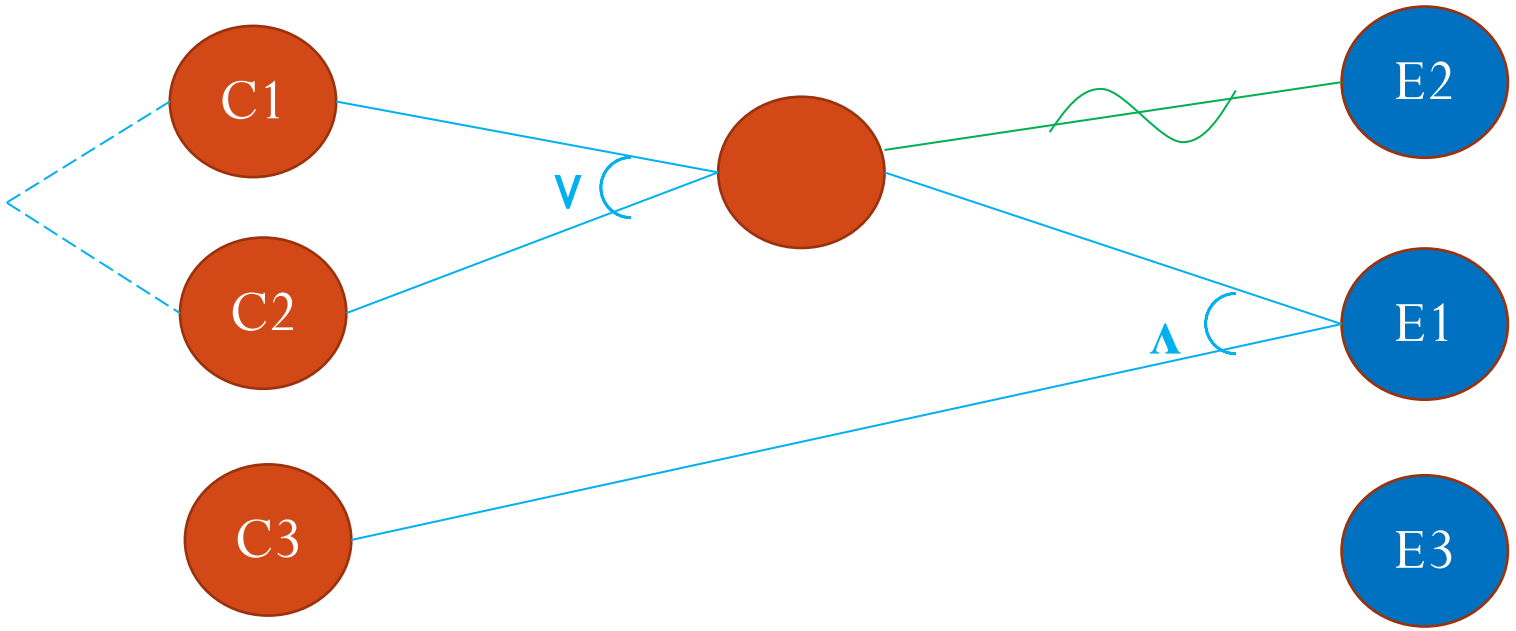
- **Exemple (suite):**
- **Identification des relations entre les causes et les effets (suite):**
- Pour l'effet ***E2 (Imprimer le message « X »)***:
- Le message « X » est imprimé lorsque:

- Le premier caractère ni ***«A»***, ni **« B »** => **C1 est fausse et C2 est fausse** => **(C1 ou C2) soit faut.**

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Identification des relations entre les causes et les effets (suite):**



Graphe causes-effets/tables de décision

Graphe causes-effets

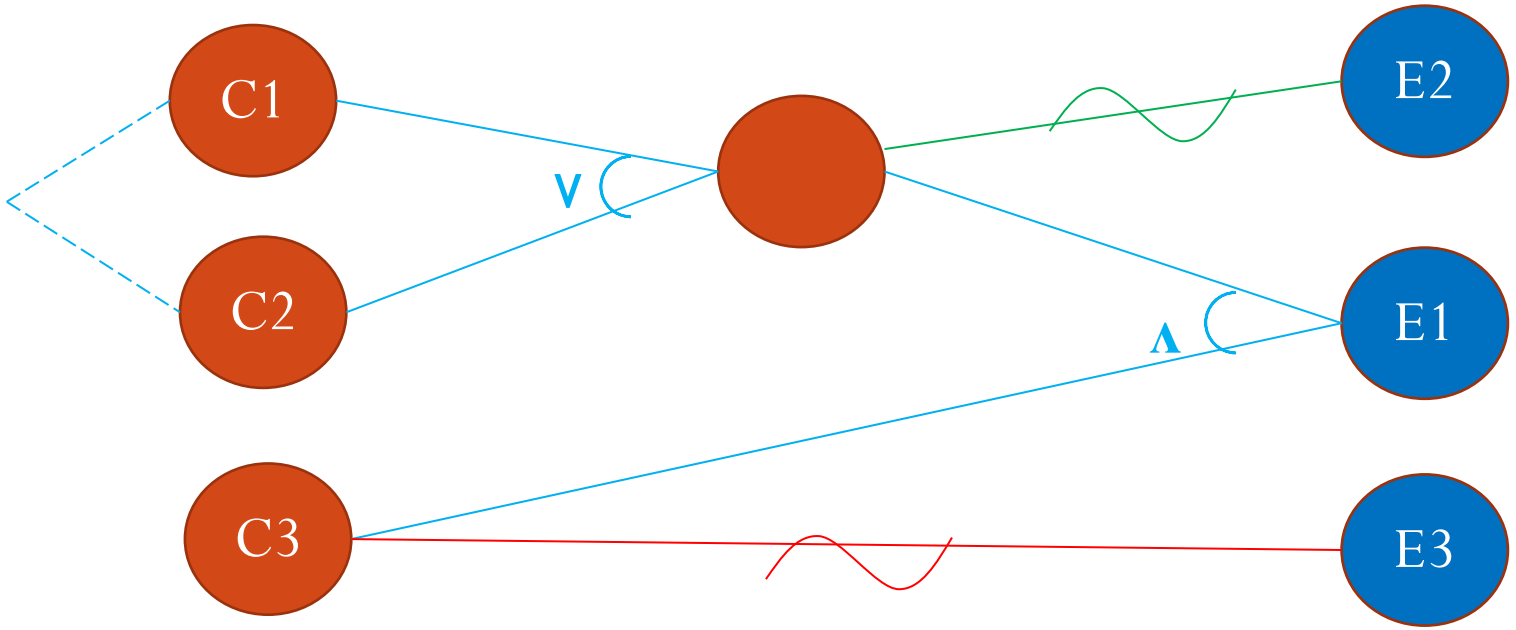
- **Exemple (suite):**
- **Identification des relations entre les causes et les effets (suite):**
- Pour l'effet ***E3 (Imprimer le message « Y »)***:
- Le message « Y » est imprimé lorsque:

  - Le second caractère est incorrect => **C3 est fausse**.

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Identification des relations entre les causes et les effets (suite):**



Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Construction de la table de décision:**
- La première étape consiste à mettre les causes et les effets dans une seule colonne

Action
C1
C2
C3
E1
E2
E3

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Construction de la table de décision (suite):**
- Dans la table de décision on va de bas en haut.
- On commence par l'effet E1 qui soit vrai,

  si (C1 v C2) ^C3 soit vrai.
- On met l'effet E1 comme True dans la colonne suivante (True =1 et False=0).

Action	
C1	
C2	
C3	
E1	1
E2	
E3	

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Construction de la table de décision (suite):**
- Pour que E1 soit vrai =1, Il faut que:

  - C1 ET C3 sera vrai

  - C2 ET C3 sera vrai

- On complète la table de décision comme suit:

Action		
C1	1	
C2		1
C3	1	1
E1	1	1
E2		
E3		

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Construction de la table de décision (suite):**
- Pour que E2 soit Vrai, il faut que:

  - C1 ou C2 doit être Faux

- On complète la table de décision comme suit:

Action				
C1	1		0	0
C2		1	0	0
C3	1	1	0	1
E1	1	1		
E2			1	1
E3				

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Construction de la table de décision (suite):**
- Pour que E3 soit vrai, il faut que:

  - C3 doit être faux.

- On complète la table de décision comme suit:

Action						
C1	1		0	0	1	0
C2		1	0	0	0	1
C3	1	1	0	1	0	0
E1	1	1				
E2			1	1		
E3			1		1	1

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**
- **Construction de la table de décision (suite):**
- On va Compléter la table de décision en ajoutant 0 dans les cases vides.

Action						
C1	1	0	0	0	1	0
C2	0	1	0	0	0	1
C3	1	1	0	1	0	0
E1	1	1	0	0	0	0
E2	0	0	1	1	0	0
E3	0	0	1	0	1	1

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**

- **Identification des cas de test à partir de la table de décision**

Id cas de test	nom	description	étapes	Résultat attendu
1	Imprimer fichier 1	Imprimer le fichier si le premier caractère est « A » et le seconde est un chiffre	1. Ouvrir le programme 2. Enter le premier caractère « A » 3. Entrer le second caractère qui est un chiffre	Imprimer le fichier
2	Imprimer fichier 2	Imprimer le fichier si le premier caractère est « B » et le seconde est un chiffre	1. Ouvrir le programme 2. Enter le premier caractère « B » 3. Entrer le second caractère qui est un chiffre	Imprimer le fichier

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite):**

- **Identification des cas de test à partir de la table de décision (suite)**

Id cas de test	nom	description	étapes	Résultat attendu
3	Imprimer message X1	Imprimer « X » si le premier caractère ni « A » ni « B » et le second caractère n'est pas un chiffre	1. Ouvrir le programme 2. Enter le premier caractère qui défère de « A » et de « B » 3. Entrer le second caractère qui n'est pas un chiffre	Imprimer « X »
4	Imprimer message X2	Imprimer « X » si le premier caractère ni « A » ni « B » et le second caractère est un chiffre	1. Ouvrir le programme 2. Enter le premier caractère qui défère de « A » et de « B » 3. Entrer le second caractère qui est un chiffre	Imprimer « X »

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Exemple (suite): Identification des cas de test à partir de la table de décision (suite)**

Id cas de test	nom	description	étapes	Résultat attendu
5	Imprimer message Y1	Imprimer «Y » si le premier caractère est « A » et le second caractère n'est pas un chiffre	1. Ouvrir le programme 2. Enter le premier caractère qui est « A » 3. Entrer le second caractère qui n'est pas un chiffre	Imprimer «Y »
6	Imprimer message Y2	Imprimer «Y » si le premier caractère est « B » et le second caractère n'est pas un chiffre	1. Ouvrir le programme 2. Enter le premier caractère qui est « B » 3. Entrer le second caractère qui n'est pas un chiffre	Imprimer «Y »
7	Imprimer message Y3	Imprimer "Y" si le premier caractère n'est ni "A" ni "B" et que le deuxième caractère n'est pas un chiffre	1. Ouvrir le programme 2. n'est ni "A" ni "B" » 3. Entrer le second caractère qui n'est pas un chiffre	Imprimer «Y »

Graphe causes-effets/tables de décision

Graphe causes-effets

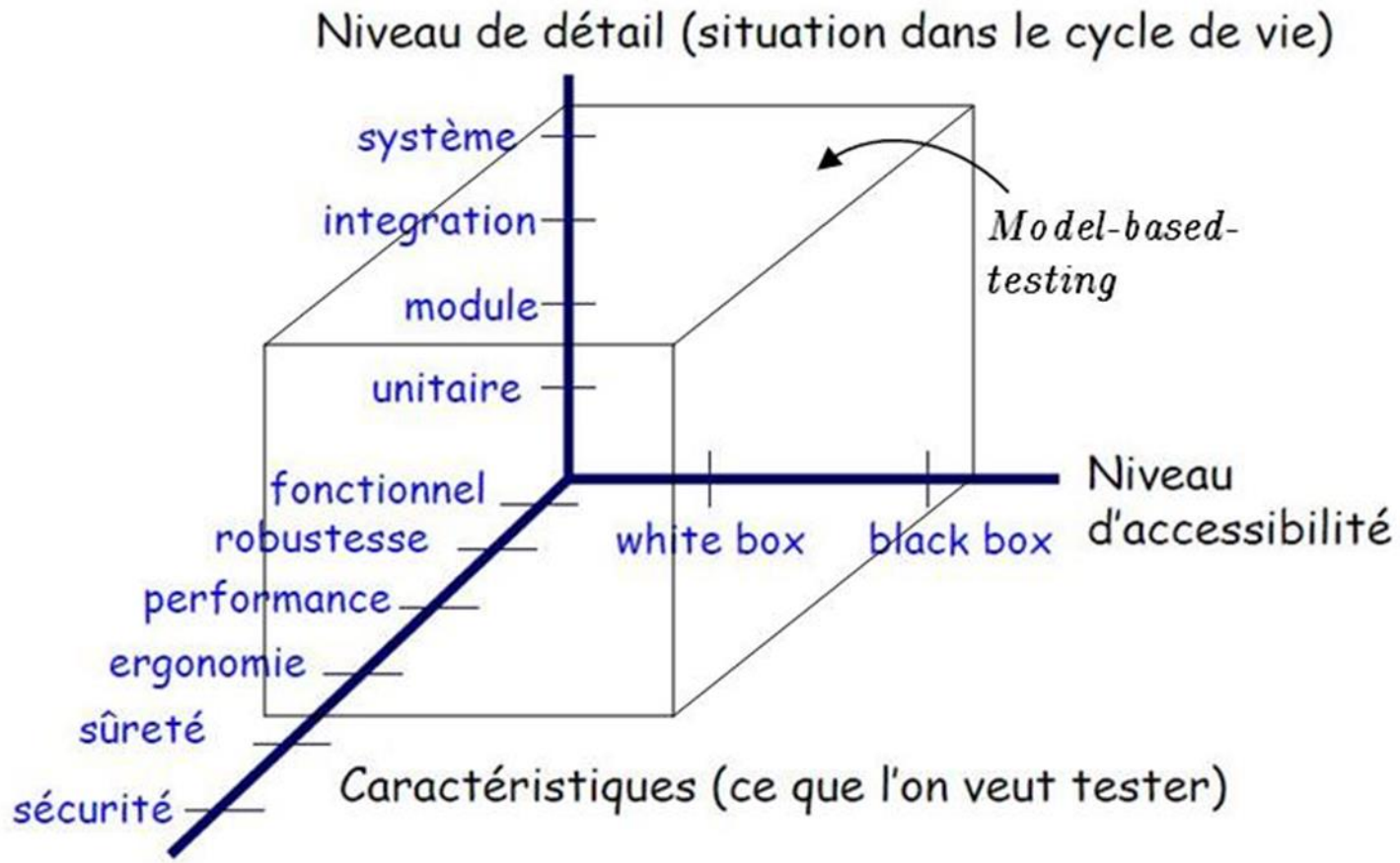
- **Exemple (suite):**
- **Remarque:** Un cas de test réel doit contenir de nombreux autres attributs comme:
- les conditions préalables,
- les données de test,
- la gravité ,
- la priorité,
- la construction,
- la version,
- l'environnement, etc.

Graphe causes-effets/tables de décision

Graphe causes-effets

- **Discussion**
- Le graphe de **cause - effet** est une technique qui permet de réduire significativement l'effort d'exécution des tests par réduire le nombre total de cas de test. Cette technique préserve la qualité de l'application par la génération des cas de test en respectant une couverture maximale.
- **Inconvénients**
- L'inconvénient major de cette technique est que la modélisation de toutes les exigences fonctionnelles avec le graphe de cause - effet prend beaucoup de temps.

Test à partir de modèles (Model-Based Testing)

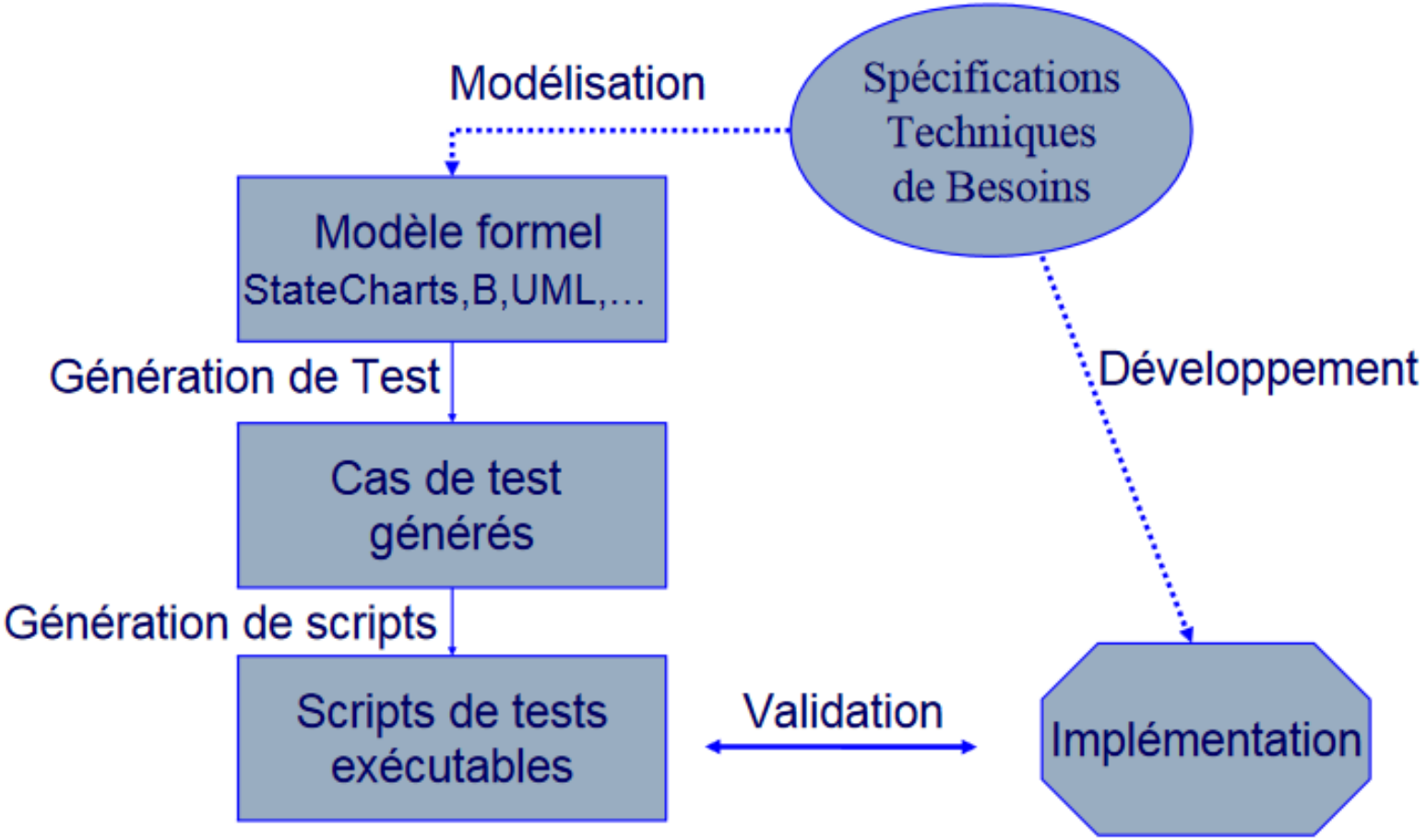


Test à partir de modèles (Model-Based Testing)

- **Le test basé sur un modèle (Model-Based-testing (MBT))** est une technique de test de logiciel où le **comportement d'exécution** du logiciel **testé est vérifié** par rapport aux **prédictions** faites par un **modèle**.
- **Un modèle** est une description du **comportement** d'un système. **Le comportement** peut être décrit en termes de **séquences d'entrée**, **d'actions**, de **conditions**, de **sortie** et de **flux de données** de **l'entrée à la sortie**. Il doit être pratiquement compréhensible et peut être réutilisable ; partageable doit avoir une description précise du système testé.

Test à partir de modèles (Model-Based Testing)

- **Processus de test à partir de modèles.**



Test à partir de modèles (Model-Based Testing)

- **Remarque:**

Les scénarios de test générés à partir de **modèles abstraits** ne peuvent pas être exécutés **directement** sur le code exécutable en raison de **la différence d'abstraction** entre les **modèles** et le **code source**.

Cette distinction nécessite souvent l'intervention **manuelle** d'un ingénieur de test qui apporte une adaptation de la conception pour passer d'une série de tests abstraits à des tests exécutables, cette étape est généralement appelée **concrétisation**.

Test à partir de modèles (Model-Based Testing)

- De **nombreux modèles** sont disponibles et décrivent différents aspects du comportement du système.

Voici des exemples de modèle :

- **Les automates à états finis**
- **Flux de données**
- **Flux de contrôle**
- **Les diagrammes UML notamment statecharts et les diagrammes d'activités.**

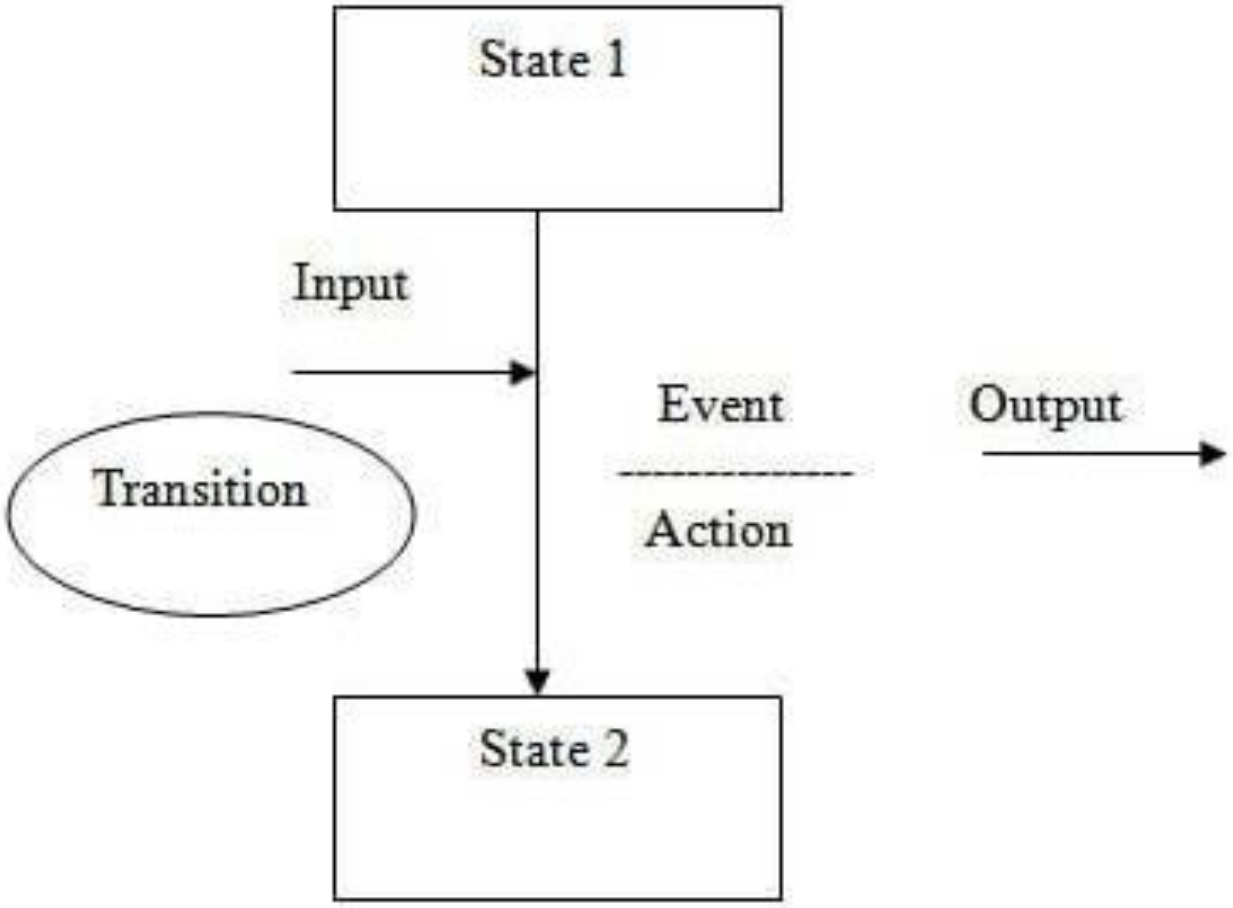
..etc

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- On prend le **diagramme d'états-transitions UML** comme **exemple**.
- **Le diagramme d'états-transitions UML** est utilisé pour modéliser le **changement d'états** d'un système ou d'un objet.
- **Le passage** d'un **état** à un **autre** est appelé **transition**.
- **La transition** est **déclenchée** en repense à un **événement reçu**, et faire basculer l'objet dans **un nouvel état** en exécutant certaines **actions**.
- Le **franchissement** d'une **transition** peut être **conditionné** par une ou plusieurs **conditions de franchissement**.

Test à partir de modèles (Model-Based Testing)

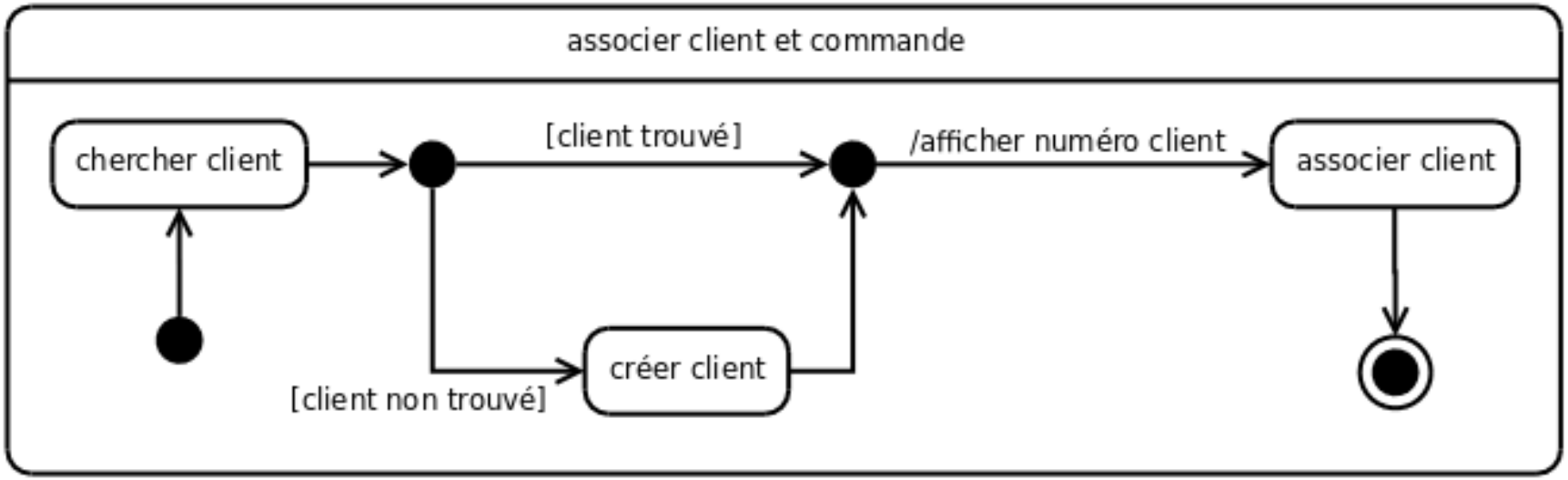
Diagramme d'états-transitions UML



Test à partir de modèles (Model-Based Testing)

Diagramme d'états-transitions UML

- **Graphiquement** le diagramme **d'états-transitions** est un graphe dont les **nœuds** sont les **états** et les **arcs** sont les **transitions**.
- **Les arcs** sont **étiquetés** par les **évènements**, les **actions** et les **conditions de franchissement** de la transition associée.



Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Génération de cas de test :**
- Pour générer les cas de test à partir des diagrammes d'états-transitions, ce dernier doit être simple a comprendre et assez détaillé pour prendre en compte tous les comportements intéressants pour le test.

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Génération de cas de test (suite):** Il existe **plusieurs critères de couvertures** pour générer les **cas de test** à partir des diagrammes **d'états-transitions**:
- **Tous les états (All-States):** chaque état doit être atteindre au moins une fois,
- **Toutes les transitions (All-Transitions):** exercer au moins une fois chaque transition
- **Toutes les paires de transitions (All-Transition-Pairs)**: Précise que pour chaque état, chaque couple de transitions sortantes doit être tiré au moins une fois.
- **Toutes les configurations (All-Configurations):** Chaque configuration du diagramme d'états-transitions est visitée au moins une fois. Ce critère de couverture est le même que **la couverture de tous les états** pour les systèmes sans parallélisme.

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

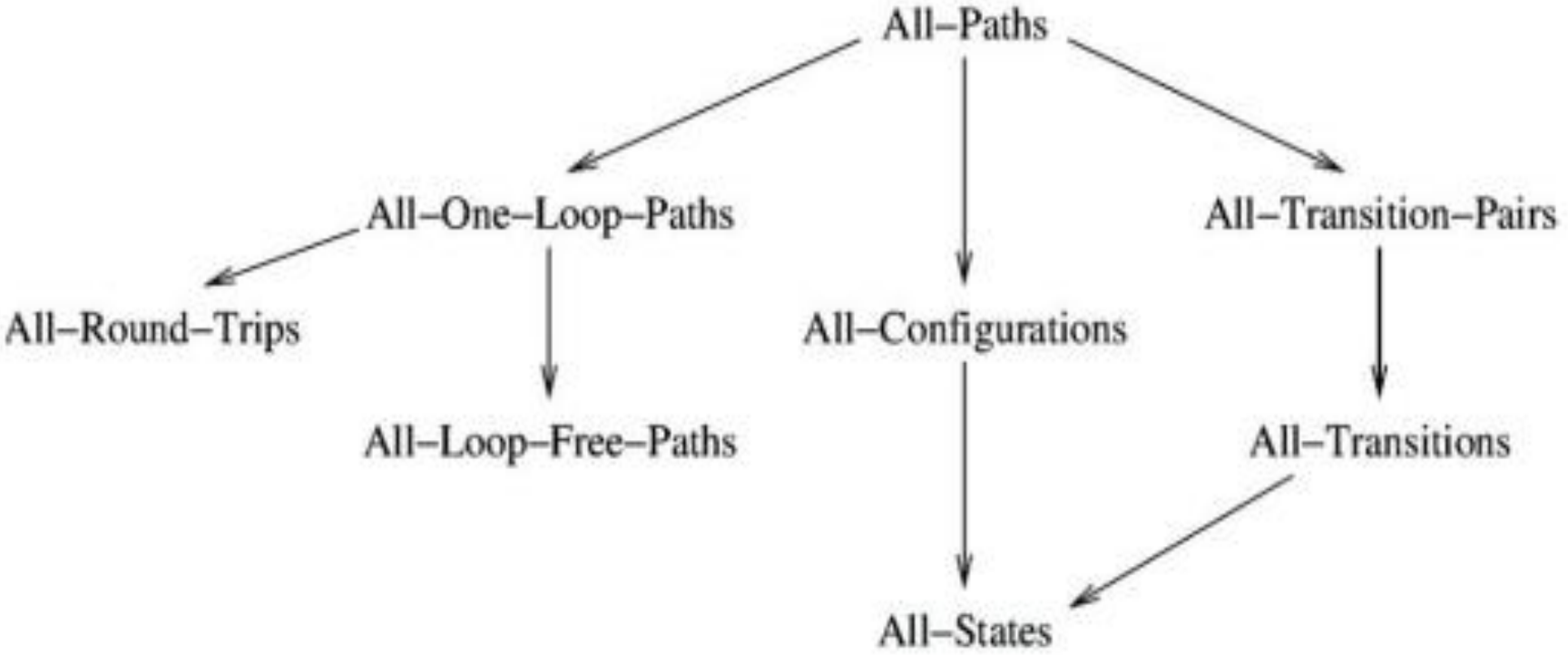
- **Génération de cas de test (suite):**
- **All-One-Loop-Paths:** renvoie tous les chemins contenant un cycle au maximum ; ainsi, chaque chemin généré contient au plus un et un seul état répété. En d'autres termes, cette condition nécessite de visiter tous les chemins sans boucle à travers le modèle, y compris tous les chemins qui bouclent une fois.
- **All-Loop-Free-Paths:** doit traverser chaque chemin de boucle au moins une fois. Un chemin qui ne contient aucun type de répétition est dit sans boucle. Notamment, cette couverture ne couvre pas souvent toutes les transitions. De même, cette couverture ne couvre pas constamment tous les États.

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Génération de cas de test (suite):**
- **Tous les allers-retours (All-Round-Trips)** est similaire au critère de **All-One-Loop-Path** car elle nécessite un test pour chaque boucle du modèle ; de plus, ce test n'a qu'à effectuer une itération autour de la boucle. Néanmoins, cette couverture est plus faible que **All-One-Loop-Path** car tous les chemins précédant ou suivant une boucle ne nécessitent pas de test.
- **Tous les chemins (All Paths):** spécifie que chaque chemin exécutable doit être suivi au moins une fois lors de l'exécution du cas de test. Le critère de tous les chemins correspond au test exhaustif du modèle de diagramme d'états-transitions.

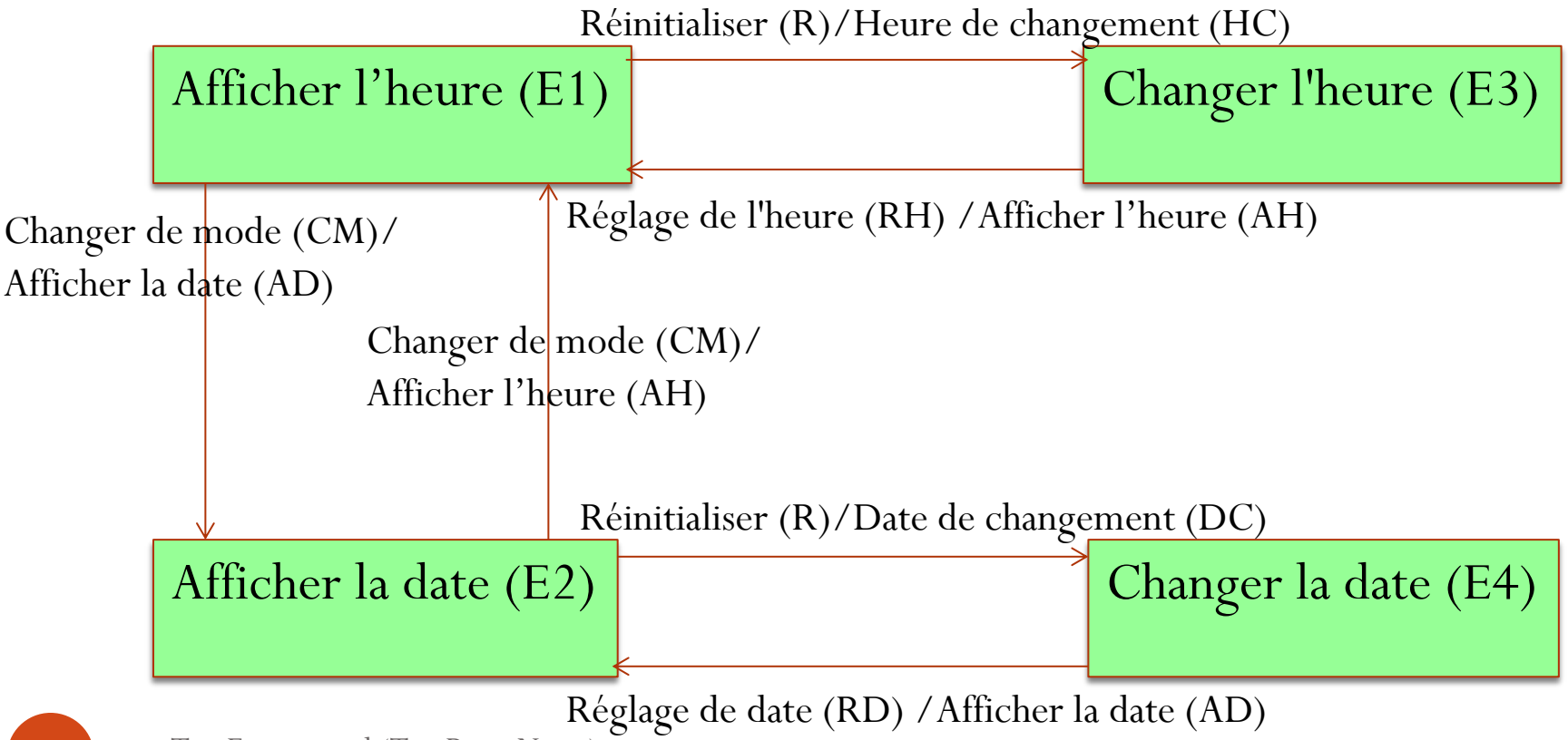
Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Génération de cas de test (suite):**



Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple:** Prenons l'exemple d'un système qui gère les modifications de mode d'affichage de l'heure et/ou de la date.



Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite):**
- Les différents **états** du système sont:
  - **Afficher l'heure (E1)**
  - **Afficher la date (E2)**
  - **Changer l'heure (E3)**
  - **Changer la date (E4)**

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite):** Les différents **transitions** du système sont:
  - **Changer de mode (CM)/ Afficher la date (AD)**: L'activation de cette transition fera changer le l'état du système de **«Afficher l'heure (E1)»** à **«Afficher la date (E2)»**. **Changer de mode (CM)** est un évènement **et Afficher la date (AD)** c'est l'action à entreprendre lorsque l'événement **Changer de mode (CM)** se produit.
  - **Changer de mode (CM)/ Afficher l'heure (AH)** : L'activation de cette transition fera changer le l'état du système de **«Afficher la date (E2)»** à **«Afficher l'heure (E1)».**
  - **Réinitialiser (R)/Heure de changement (HC):** Cette transition permet de régler le mode d'affichage sur **« Changer l'heure (E3) »**.

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite):** Les différents **transitions** du système sont **(suite):**
  - **Réinitialiser (R)/Date de changement (DC):** Cette transition permet de régler le mode d'affichage sur **« Changer la date (E4) »**.
  - **Réglage de l'heure (RH) /Afficher l'heure (AH):** L'activation de cette transition entraînera le retour au état **« Afficher l'heure (E1) »** depuis l'état **« Changer l'heure (E3) ».**
  - **Réglage de date (RD) /Afficher la date (AD) :** L'activation de cette transition entraînera le retour au état **« Afficher la date (E2) »** depuis l'état **« Changer la date (E4) ».**

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite): Identification des entrées/soties du système**
  - Les différentes entrées du système sont les **évènements** suivants:

Changer de mode (CM), Réinitialiser (R), Réglage de l'heure (RH) et Réglage de date (RD) .

  - Les différentes **sorties** du système sont les **actions** suivantes:

Afficher la date (AD), Afficher l'heure (AH) , Heure de changement (HC), Date de changement (DC), Afficher l'heure (AH) et Afficher la date (AD).

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite): La génération de cas de test**
- **Etape1:** Identifiez tous **les états de départ** et leurs **flèches sortantes**.
  - **L'état E1** : contient **deux** flèches sortantes: la première flèche va à **l'état E3** et la deuxième flèche va à **l'état E2**.
  - **L'état E2** : contient **deux** flèches sortantes: la première flèche va à **l'état E1** et la deuxième flèche va à **l'état E4**
  - **L'état E3** : contient **une seule** flèche sortante va à **l'état E1.**
  - **L'état E4** : contient **une seule** flèche sortante va à **l'état E2.**

états de départ	E1	E1	E2	E2	E3	E4

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite): La génération de cas de test (suite)**
- **Etape2:** Mettez sur la table tous les **états de transition finaux** pour chaque **état de départ** comme suit:

états de départ	E1	E1	E2	E2	E3	E4

états final	E2	E3	E4	E1	E1	E2

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite): La génération de cas de test (suite)**
- **Etape3:** Mettez sur la table les **entrées** et les **sorties** de chaque **état de départ** et son **état final correspondant** comme suit:

états de départ	E1	E1	E2	E2	E3	E4
entrée	CM	R	R	CM	RH	RD
sortie	AD	HC	DC	AH	AH	AD
états final	E2	E3	E4	E1	E1	E2

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite): La génération de cas de test (suite)**
- **Etape4:** L'écriture **des cas de test** comme suit:

Id cas de test	Description	Étapes de test	Résultat attendu
1	valider que le système est capable d'effectuer la transition du mode d'affichage de l'heure à l'affichage de la date avec sortie : Afficher la date	1. Ouvrir l'application 2. Ouvrir l'entrée de l'état de départ en tant que Changer de mode	la sortie de l'état de départ doit être « afficher la date » et l'état final est la « Afficher la date »
2	valider que le système est capable d'effectuer la transition de l'état d'afficher l'heure à l'état de changer l'heure avec la sortie: Heure de changement	1. Ouvrir l'application 2. Ouvrir l'entrée de l'état de départ en tant que Réinitialiser	la sortie de l'état de départ doit être « Heure de changement » et l'état final est la « Changer l'heure»

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite): La génération de cas de test (suite)**

- **Etape4:** L'écriture **des cas de test** comme suit:

Id cas de test	Description	Étapes de test	Résultat attendu
3	valider que le système est capable d'effectuer la transition de l'état d'afficher la date à l'état de changer la date avec la sortie: Date de changement	1. Ouvrir l'application 2. Ouvrir l'entrée de l'état de départ en tant que Réinitialiser	la sortie de l'état de départ doit être « Date de changement » et l'état final est « Changer la date»
4	valider que le système est capable d'effectuer la transition du mode d'affichage de la date à l'affichage de l'heure avec sortie : Afficher l'heure	1. Ouvrir l'application 2. Ouvrir l'entrée de l'état de départ en tant que Changer de mode	la sortie de l'état de départ doit être « afficher l'heure » et l'état final est la « Afficher l'heure »

Test à partir de modèles (Model-Based Testing)
Diagramme d'états-transitions UML

- **Exemple (suite): La génération de cas de test (suite)**

- **Etape4:** L'écriture **des cas de test** comme suit:

Id cas de test	Description	Étapes de test	Résultat attendu
5	valider que le système est capable d'effectuer la transition de l'état de changer l'heure à l'état de afficher l'heure avec la sortie: Afficher l'heure	1. Ouvrir l'application 2. Ouvrir l'entrée de l'état de départ en tant que Réglage de l'heure	la sortie de l'état de départ doit être «Afficher l'heure » et l'état final est «Afficher l'heure»
6	valider que le système est capable d'effectuer la transition de l'état de changer la date à l'état de afficher la date avec la sortie: Afficher la date	1. Ouvrir l'application 2. Ouvrir l'entrée de l'état de départ en tant que Réglage de la date	la sortie de l'état de départ doit être «Afficher la date» et l'état final est «Afficher la date»