Tests de boîte noire

Tests de boîte noire
Sujets de ce thème

- Définition
- Le partitionnement en classes
- L’analyse des valeurs limites
- Les tests pseudo-aléatoires

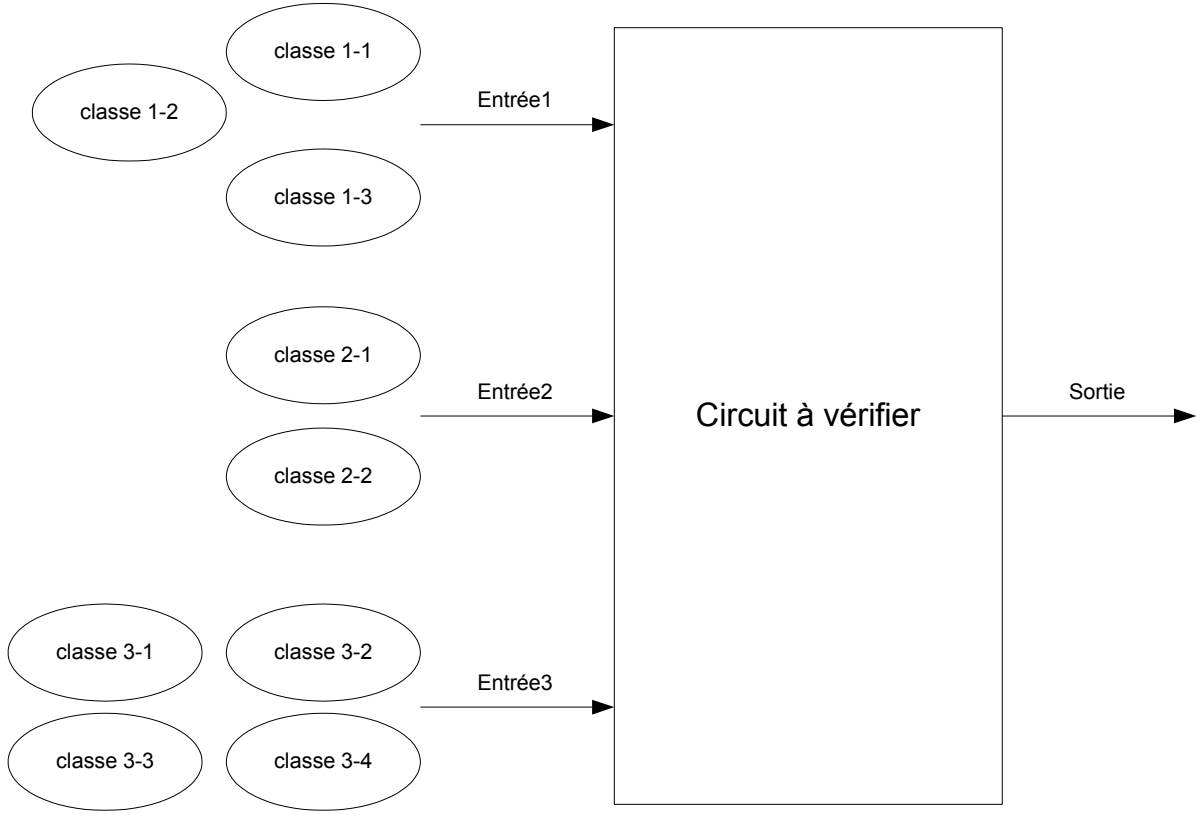
Tests de boîte noire (ou tests fonctionnels)

- Le terme « test de boîte noire » fait référence à un test qui ne suppose aucune connaissance de l'implémentation du système.
- En anglais: *black box*, *opaque box*, *closed box*, *functional test*.
- Les tests de boîte noire s'appuient uniquement sur les spécifications du système.
- Plus le système est complexe, et plus il faut utiliser ce genre de test.
- Il est difficile de déterminer à quel point le système a été vérifié par ce genre de test.
- Les tests de boîte noire ne permettent pas en général de découvrir des comportements non spécifiés du système.

Vecteurs de test choisis par partitionnement en classes

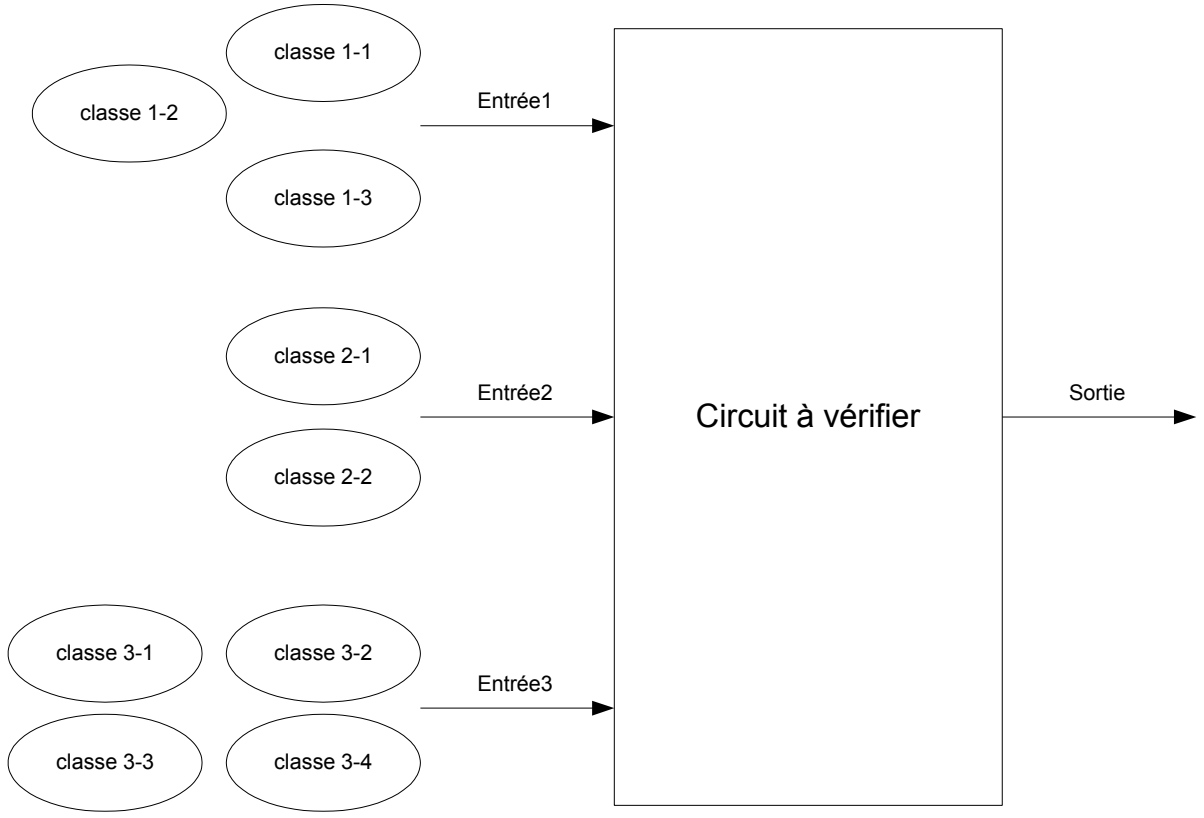
- L'ensemble des valeurs d'entrée du système est partitionné en classes séparées.
- Le principe du partitionnement en classes s'appuie sur la supposition suivante:

  « Deux données d'une même classe sont similaires et le comportement du système est semblable pour elles »



Vecteurs de test choisis par partitionnement en classes

- Pour composer un ensemble de vecteurs de tests, on doit choisir des données qui appartiennent à chacune des classes.
  - Un test faible consiste en un ensemble de vecteurs de tests où au moins un élément de chacune des classes serait présent au moins une fois.
  - Un test fort consiste en un ensemble de vecteurs de test où toutes les interactions entre les classes seraient présentes au moins une fois. Le nombre de vecteurs de test est alors égal au produit du nombre de classes pour chacune des entrées du système.



Test faible: 4 vecteurs
Test fort: 24 vecteurs

Vecteurs de test choisis par partitionnement en classes
Exemple

- Un module d'un chronomètre doit calculer la date du lendemain à la fin de la journée.
- On identifie les classes de données d'entrée suivantes.
  - Pour les années :{année divisible par 4 mais pas par 100}, {année divisible par 400}, {année pas divisible par 4}, {année divisible par 100 mais pas par 400}.
  - Pour les mois : {mois de 30 jours}, {mois de 31 jours}, {mois de février}.
  - Pour les jours : {jour est entre 1 et 28}, {jour est 29}, {jour est 30}, {jour est 31}.

- Pour un test faible, il suffit de prendre un élément de chaque classe au moins une fois.
- Pour un test fort, on devrait choisir au moins un vecteur de test pour chacune des combinaisons de classes.
  - cas du mois de 31 jours avec les quatre classes de jour et les trois classes d'années.
  - cas du mois de février, le jour 31, l'année centenaire (pas valide).
  - etc.
- La combinaison de vecteurs de test couvrant toutes les combinaisons de classes peut être très grande.

Vecteurs de test choisis par partitionnement en classes
Exemple

Année divisible par 100 mais pas par 400 (ex. 1700, 1800, 2100, etc.)

Année divisible par 4 mais pas par 100 (ex. 2004, 2008, 2012)

Année divisible par 400 (ex. 1600, 2000, 2400)

Année pas divisible par 4 (ex. 2013, 2015)

Jour 1 à 28

Jour = 29

Jour = 30

Jour = 31

Février

Mois de 30 jours

Mois de 31 jours

1. Choix d'une donnée par classe

Vecteurs de test choisis par partitionnement en classes
Exemple

Année divisible par 100 mais pas par 400 (ex. 1700, 1800, 2100, etc.)

Année divisible par 4 mais pas par 100 (ex. 2004, 2008, 2012)

Année divisible par 400 (ex. 1600, 2000, 2400)

Année pas divisible par 4 (ex. 2013, 2015)

Jour 1 à 28

Jour = 29

Jour = 30

Jour = 31

Février

Mois de 30 jours

Mois de 31 jours

2. Ensemble de vecteurs de test faible

Vecteurs de test choisis par partitionnement en classes
Exemple

Année divisible par 100 mais pas par 400 (ex. 1700, 1800, 2100, etc.)

Année divisible par 4 mais pas par 100 (ex. 2004, 2008, 2012)

Année divisible par 400 (ex. 1600, 2000, 2400)

Année pas divisible par 4 (ex. 2013, 2015)

Jour 1 à 28

Jour = 29

Jour = 30

Jour = 31

Février

Mois de 30 jours

Mois de 31 jours

3. Ensemble de vecteurs de test fort

Vecteurs de test choisis par analyse des valeurs limites

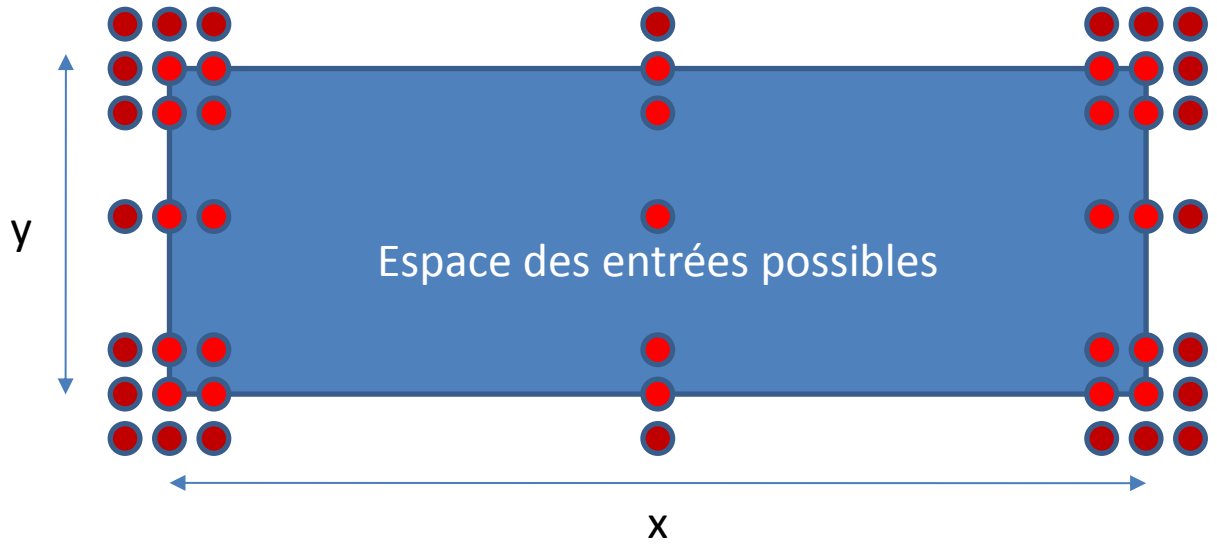
- Cette approche découle de l'observation que les erreurs se produisent souvent aux valeurs limites des données.
- Par exemple, il peut y avoir un débordement lors d'une addition.
- On devrait inclure dans les vecteurs de test, pour chaque donnée, jusqu'à sept valeurs différentes.
- Il n'est pas nécessairement possible d'appliquer des valeurs interdites au système.

Valeur	Exemple pour le jour, mois de janvier
Une valeur juste inférieure à la valeur minimale	0
La valeur minimale	1
Une valeur juste supérieure à la valeur minimale	2
Une valeur nominale	15
Une valeur juste inférieure à la valeur maximale	30
La valeur maximale	31
Une valeur juste supérieure à la valeur maximale	32

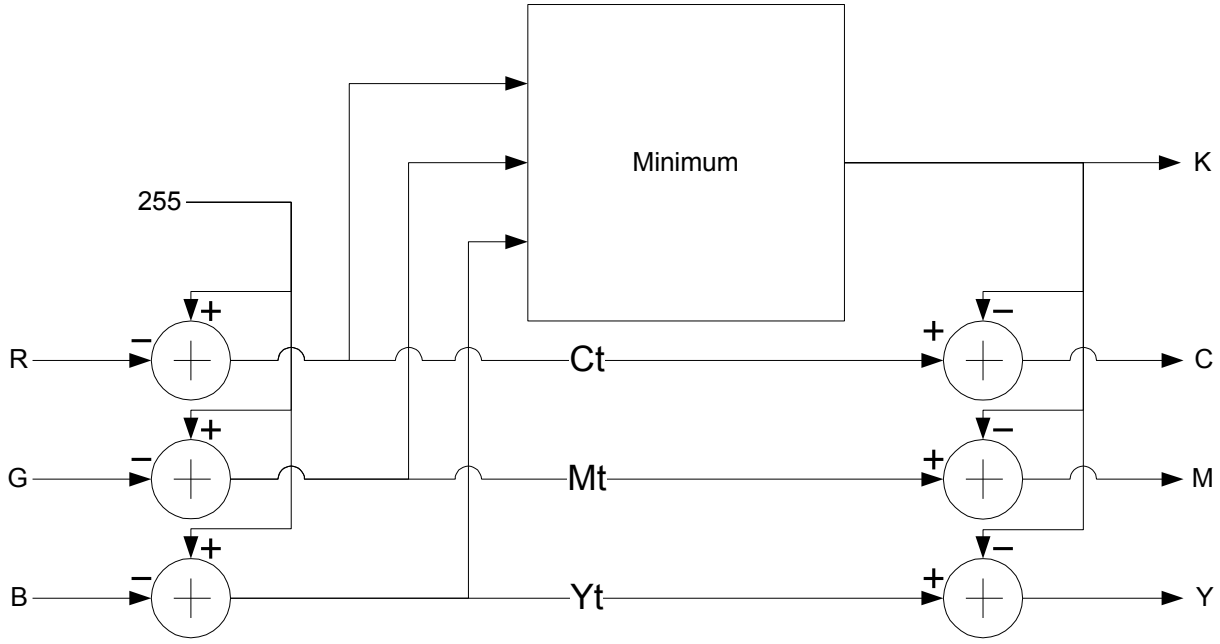
Vecteurs de test choisis par analyse des valeurs limites

- Vérifier toutes les combinaisons possibles des valeurs limites de chacune des variables:
  7^n cas différents pour *n* variables.
- Si le nombre de variables est trop grand, on peut éliminer les valeurs interdites (sous le minimum et en haut du maximum).
  On obtient alors 5^n cas différents.
- On peut encore réduire le nombre de combinaisons en fixant toutes les variables à leur valeur nominale, sauf une ou deux à la fois qui passent à travers leurs valeurs limites.

Exemple pour deux variables x et y.



Vecteurs de test choisis par analyse des valeurs limites
Exemple

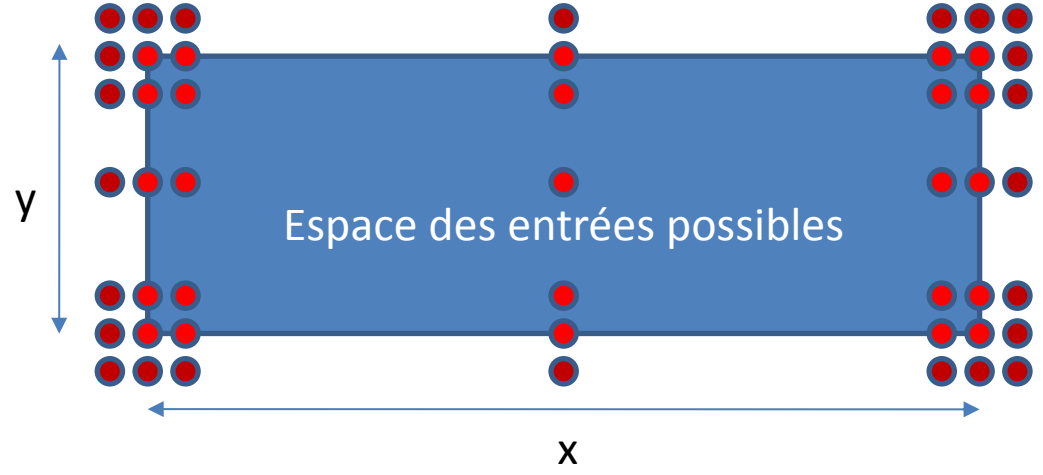



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity convRGB2CMYK is
    port (
    rouge, vert, bleu : in unsigned(7 downto 0);
    cyan, magenta, jaune, noir : out unsigned(7 downto 0)
);
end convRGB2CMYK;
```

Quels vecteurs de test choisir selon l'analyse des valeurs limites?
- en version complète
- en version réduite

Comment ces ensembles se comparent-ils au test exhaustif?



Vecteurs de test pseudo-aléatoires

- Les vecteurs de test pseudo-aléatoires ont l'avantage d'être simples à générer.
- Ils peuvent compléter de façon intéressante un ensemble de vecteurs de tests composé avec une autre méthode.

```
library ieee;
use ieee.math_real.all;

...
process
variable seed1 : positive := 1;
variable seed2 : positive := 2;
variable aleatoire : real;
variable t : integer := -1;
begin
   uniform(seed1, seed2, aleatoire); -- 0.0 < aleatoire < 1.0
   aleatoire := floor(aleatoire * 255.0); -- 0.0 <= aleatoire <= 255.0
   t := integer(aleatoire); -- 0 <= t <= 255
   ... -- appliquer t à l'entité à tester

end process;
```

- Quelques principes doivent être respectés pour les tests pseudo-aléatoires :
  - Il est utile de distinguer entre des valeurs valides et non valides appliquées au système, et s'assurer de générer surtout des valeurs de test valides.
  - Le test doit être reproductible, donc il faut choisir des valeurs de graines connues (et non la valeur de l'horloge) pour lancer la génération des nombres pseudo-aléatoires.
  - Le test devrait avoir un objectif précis en restreignant les valeurs aléatoires générées à une seule classe.
  - La distribution des valeurs aléatoires devrait être connue et contrôlable pour correspondre aux conditions d'opération du système.

Vous devriez maintenant être capable de …

- Décrire et utiliser quelques types de tests de boîte noire. (B2, B3)
- Décrire et utiliser le partitionnement en classe. (B2, B3)
- Décrire et utiliser l'analyse des valeurs limites. (B2, B3)
- Décrire les principes à respecter pour les tests pseudo-aléatoires et les utiliser. (B2, B3)

Code	Niveau (http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom)
B1	Connaissance – mémoriser de l'information.
B2	Compréhension – interpréter l'information.
B3	Application – confronter les connaissances à des cas pratiques simples.
B4	Analyse – décomposer un problème, cas pratiques plus complexes.
B5	Synthèse – expression personnelle, cas pratiques plus complexes.