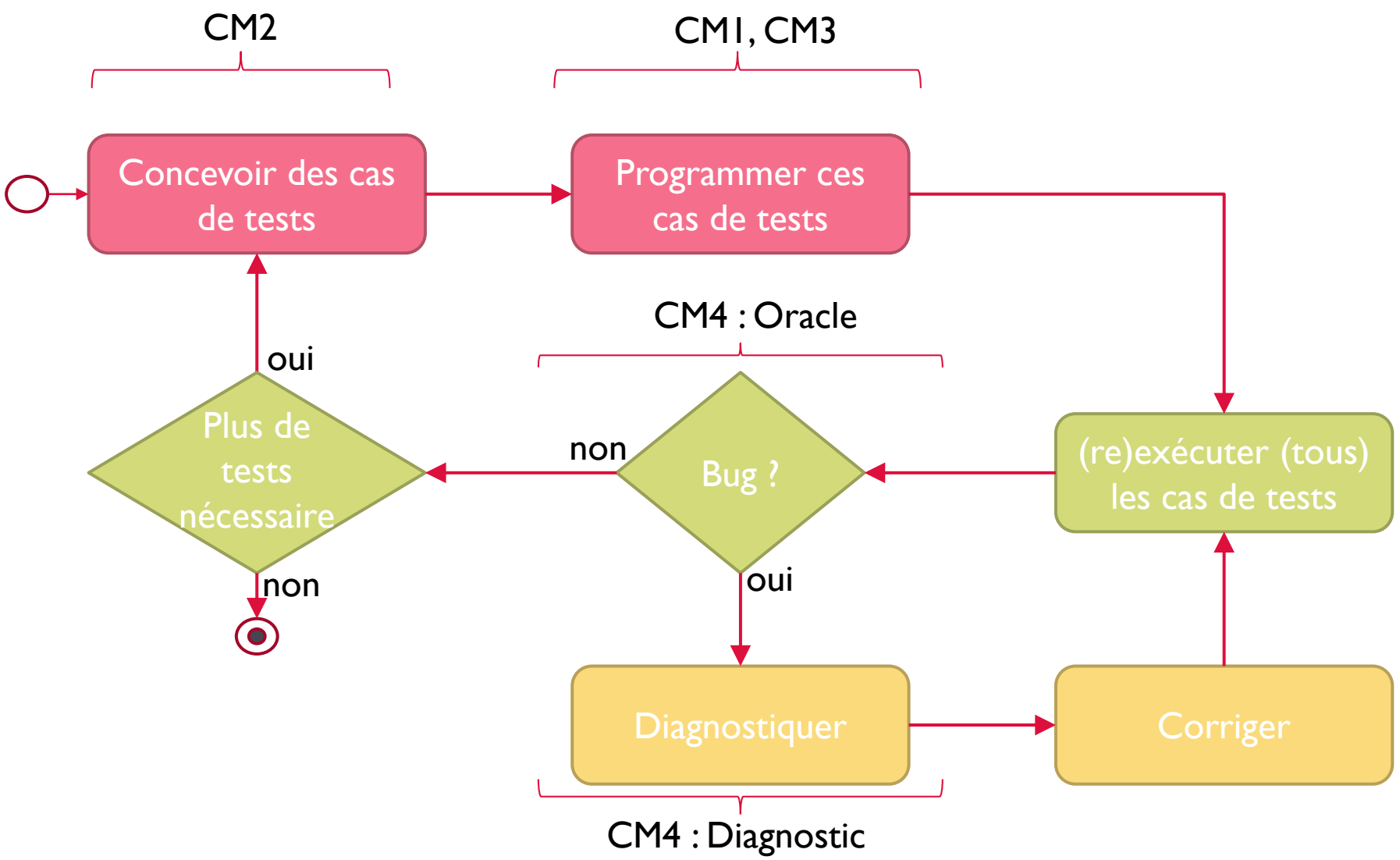
IUT Nantes
Pôle Sciences et technologie
Nantes Université

Oracle - Diagnostic

Jean-Marie Mottu

Le cycle de test dynamique



IUT Nantes
Pôle Sciences et technologie

Nantes Université

Oracle

Oracle du test

- L'oracle est supposé détecter un mauvais comportement
- Comportement
  - Fonctionnel
    - Mauvais résultat renvoyé par la méthode testée
    - Mauvais état d'un objet en sortie
  - Extra-Fonctionnel
    - Temps de réponse
    - Quantité de mémoire utilisée

Oracle du test

- Définition
  - L'oracle vérifie que l'exécution d'un test respecte la spécification en analysant les résultats du test.
- L'oracle produit le verdict du test : passe ou échoue
  - Peut donner davantage d'indications, cf partie du CM suivante
- L'oracle n'est pas systématiquement la donnée de sortie attendue

Difficulté de l'oracle

- L'oracle doit analyser des données parfois complexes pour produire le verdict
  - Type simple dans le cas de méthode renvoyant un résultat
  - Type complexe
    - Objets (avec toutes leurs propriétés)
    - Structures complexes
      - Base de donnée
      - Programme compilé
      - IHM
      - etc.
    - Propriétés extra-fonctionnelles
      - Temps, environnement (consommation mémoire, processeur), etc.

Difficulté de l'oracle

- L'oracle vérifie le respect de spécifications variées

- Les spécifications sont exprimées
  - Formellement (idéal)
  - Semi-formellement
  - Textuellement

- Les spécifications peuvent être
  - Difficilement interprétables
  - Impossible à traiter automatiquement (majorité des cas)

Pas d'oracle = pas de vérification

- Simple exécution de données de test à partir d'un état donné du logiciel sous test
  - Ne vérifie pas le comportement du logiciel
  - Une seule utilité : le test de robustesse, l'absence de crash du système

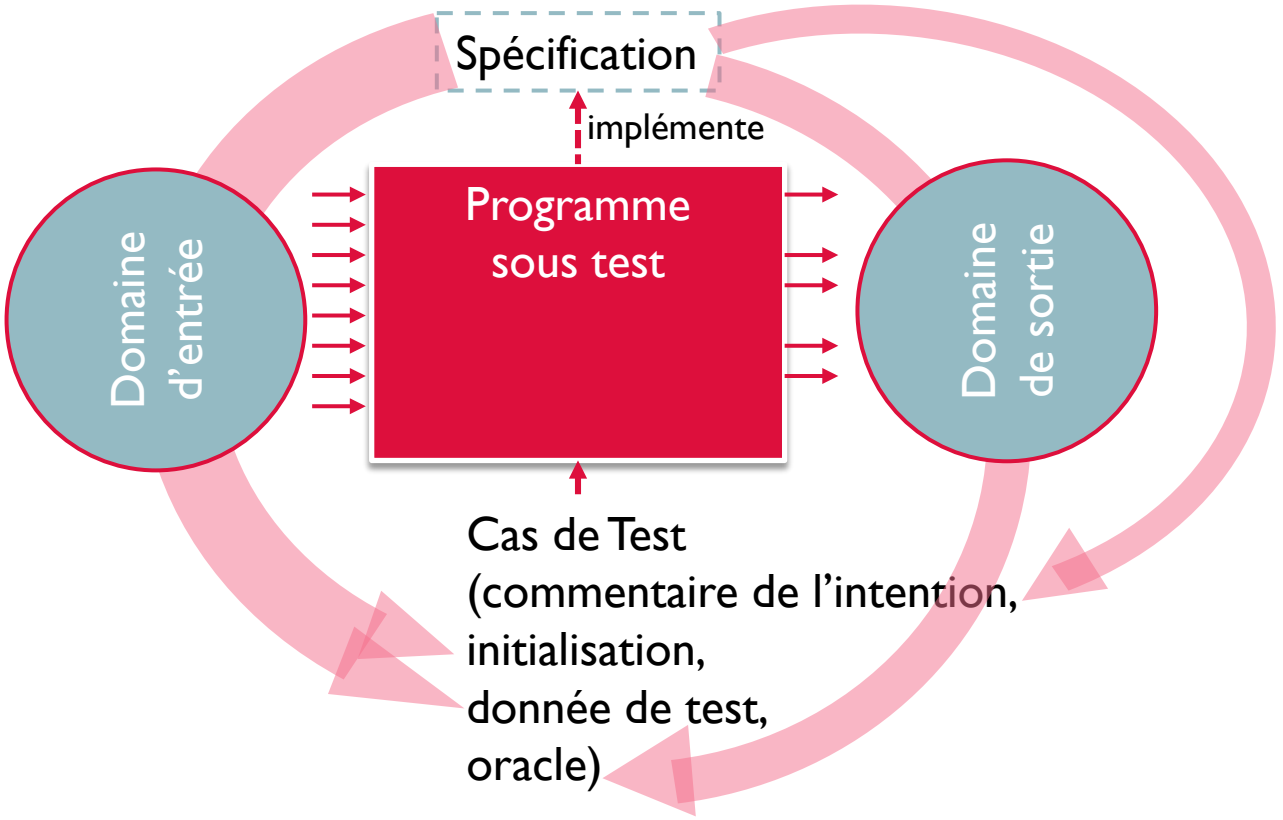
- Il ne suffit pas de générer et d'exécuter des données de test pour faire des suppositions sur l'exactitude d'un logiciel

Verdict des oracles

- Verdict complet ou partiel
  - L'exécution d'un cas de test peut entraîner de nombreuses modifications
  - Un verdict partiel assure que ce test a vérifié une partie de la spécification (partie de la spécification, dont le domaine de sortie)
  - Un verdict complet assure que ce cas de test respecte la globalité de la spécification
    - Attention les tests ne sont pas complets pour autant

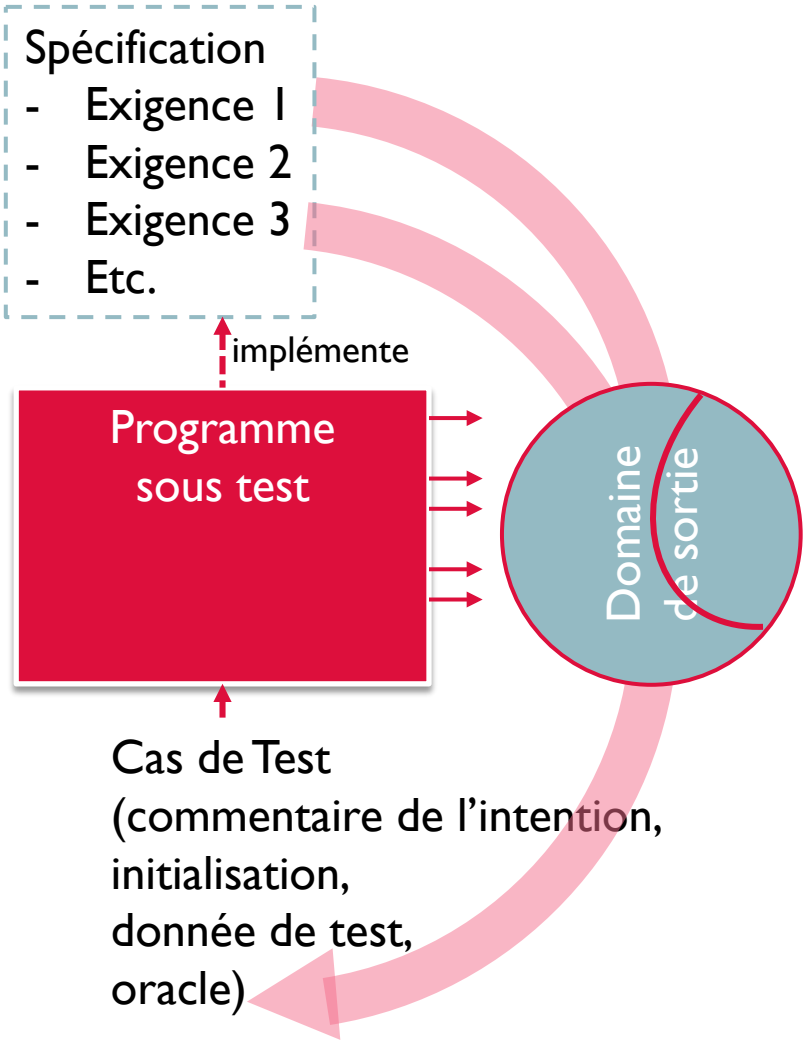
Verdict des oracles

- Un cas de test et son oracle considère :



Verdict des oracles

- Un cas de test et son oracle considère:
  - une partie (multiple) de la spécification
    - les exigences qui sont concernées par le cas de test
      - Un verdict partiel en considère certaines
      - Un verdict complet les considère toutes
  - l'effet du programme sur une partie du domaine de sortie
    - des propriétés sont concernées
      - Un verdict partiel dira si certaines d'entre elles ont été bien modifiées
      - Un verdict complet les considérera toutes
        - Voire même les effets de bords potentiels



Mise en œuvre : Oracle discret

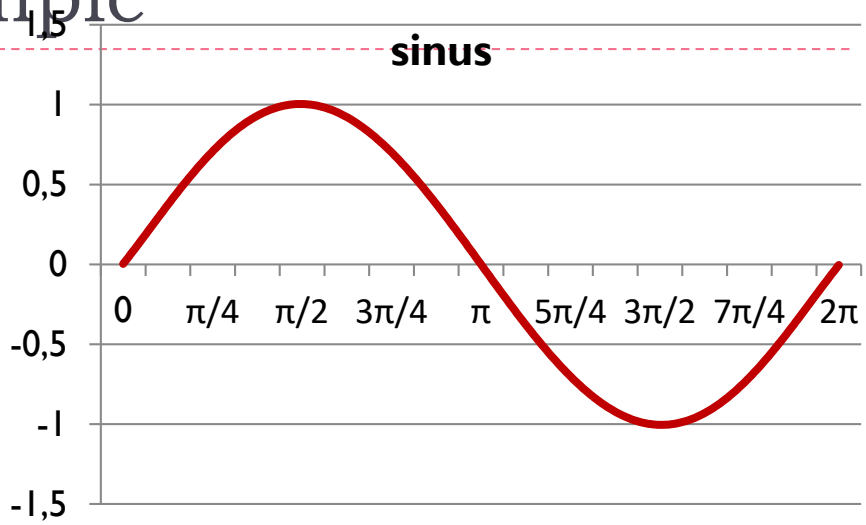
- Directement relié à la détermination d'un ensemble fini de données de test

- Un oracle par résultat de sortie qui vérifie
  - Les données renvoyées en sortie
  - L'état du système le cas échéant

- Solution la plus importante
  - Parce que l'écriture (majoritairement) manuelle des oracles nécessite une limitation de la quantité de test à un nombre fini
  - Néanmoins les tests discrets peuvent laisser passer des erreurs

Oracle discret
Exemple – Contre-exemple

- Exemple du test de la fonction sinus : y = sin(x)
- Entre 0 et 2π
  - Données de test identifiables fonctionnellement
    - Les valeurs provoquant un changement de sens : π/2, 3π/2
    - Les valeurs dont le sinus est nul : 0, π, 2π
    - Des valeurs dans les intervalles : π/4, 3π/4, 5π/4, 7π/4

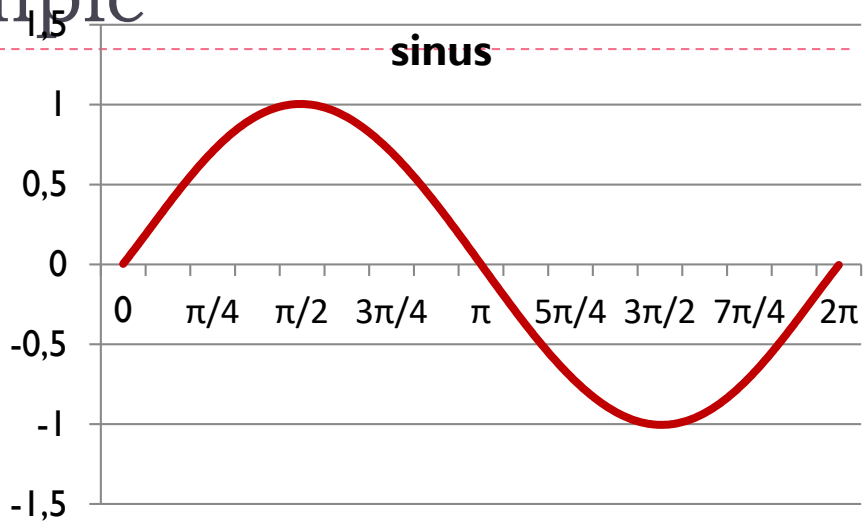


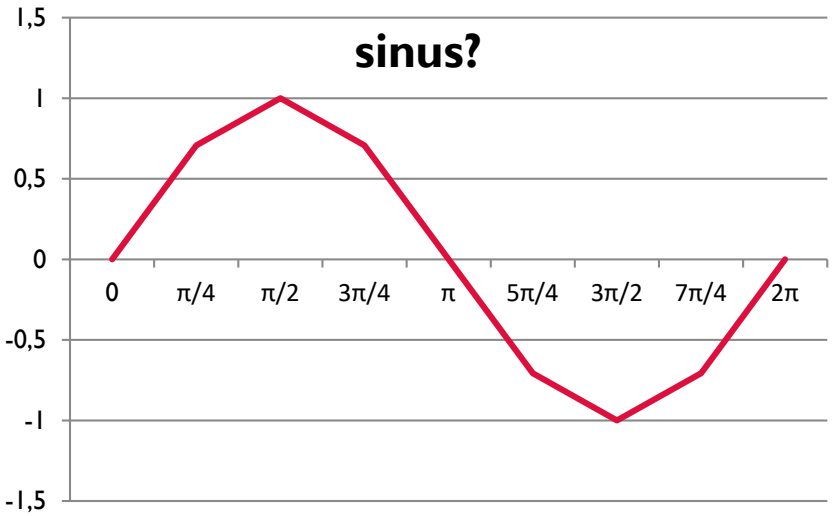
- CT:

DT	Oracle
0	0
π/4	√2/2
π/2	1
3π/4	√2/2
π	0
5π/4	-√2/2
3π/2	-1
7π/4	-√2/2
2π	0

Oracle discret
Exemple – Contre-exemple

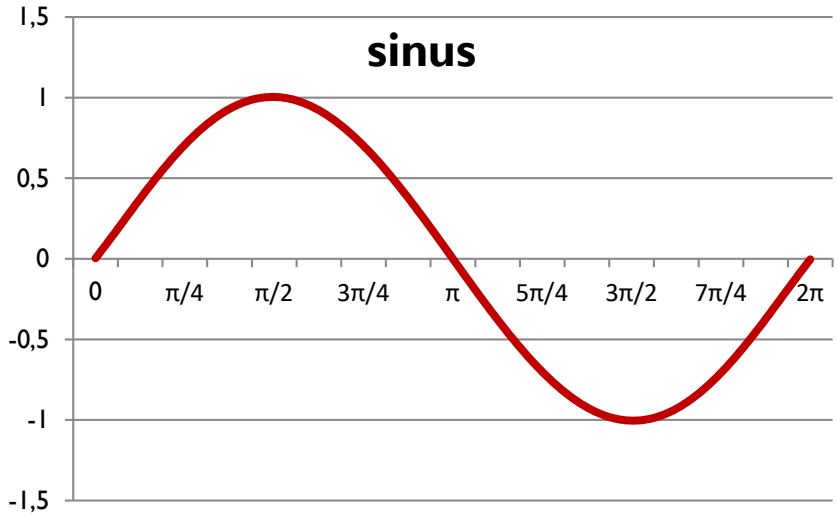
- Exemple du test de la fonction sinus : y = sin(x)
- Entre 0 et 2π
  - Données de test identifiables fonctionnellement
    - Les valeurs provoquant un changement de sens : π/2, 3π/2
    - Les valeurs dont le sinus est nul : 0, π, 2π
    - Des valeurs dans les intervalles : π/4, 3π/4, 5π/4, 7π/4





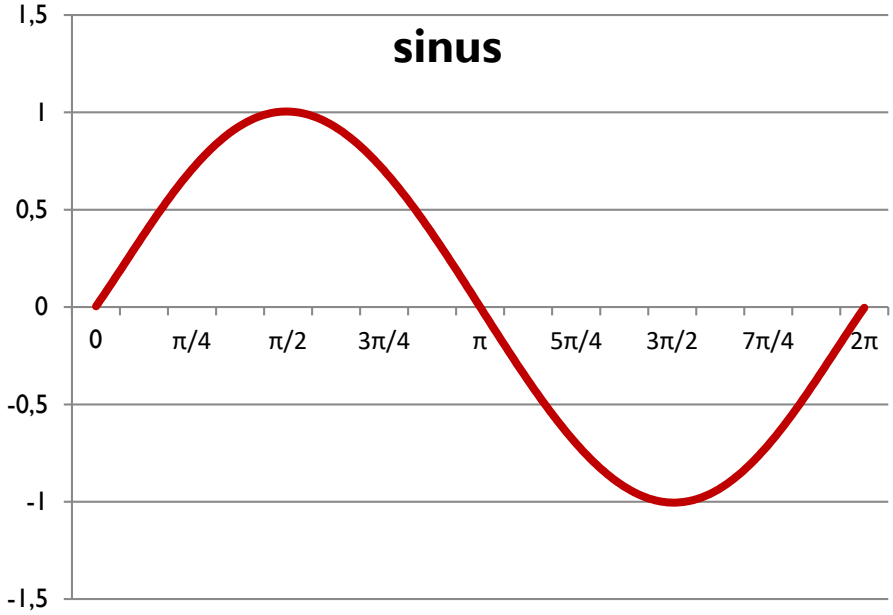
Mise en œuvre : Oracle heuristique

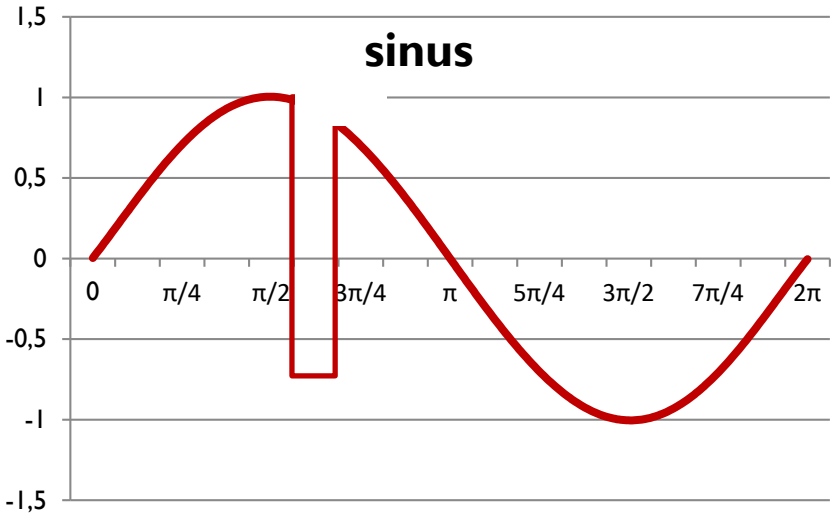
- Un oracle heuristique permet de vérifier certaines propriétés d'un groupe de résultats
- Se base particulièrement sur des liens entre données d'entrée et données de sortie
- Exploite le partitionnement
- Cf : Property Based Testing



Exemple d'heuristiques

- y = sin(x) :
  - -1=< y <= 1 quelque soit x
  - 0 > y si x mod 2π < π
  - 0 < y si x mod 2π > π



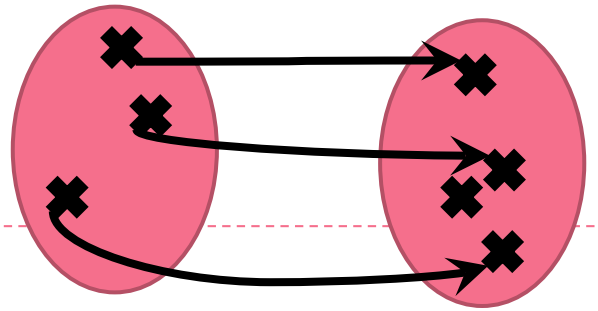


Oracle vrai (true oracle)

- Un oracle vrai vérifie n'importe quelle exécution d'une donnée de test en renvoyant un verdict complet
  - Très souple car il peut être utilisé dans n'importe quel test
  - Très difficile à obtenir
    - Aussi complexe que le logiciel qu'on vérifie
      - Le découpage en tests perd son intérêt
      - Risque d'erreur
- Logiciel de référence comme oracle
  - Permet les vérifications fonctionnelles
    - Généralement pas adapté aux vérifications extra-fonctionnelles
  - Valable si un logiciel de référence est disponible
    - Cas des changements de plate-forme
    - Cas des améliorations extra-fonctionnelles
  - On n'écrit pas un logiciel de référence spécifiquement pour en faire un oracle
    - Il faudrait aussi le tester

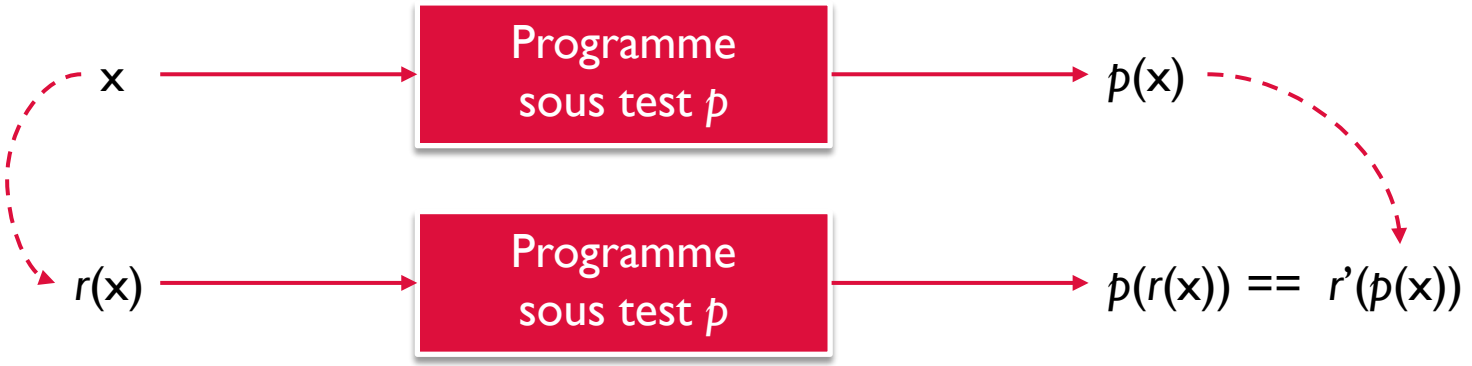
Logiciel inverse comme oracle

- Utiliser un logiciel réalisant l'inverse du logiciel testé
  - Permet les vérifications fonctionnelles
  - Ne permet pas les vérifications extra-fonctionnelles
- On n'écrit pas un logiciel inverse spécifiquement pour en faire un oracle
  - Il faudrait aussi le tester
- Valable si un logiciel inverse est disponible
  - Utopique à l'échelle d'un logiciel
  - Possible à l'échelle d'une fonction
    - En particulier les fonctions mathématiques sous la condition qu'elles soient des applications injectives



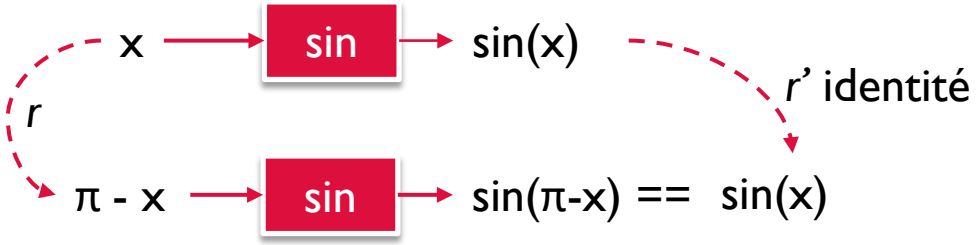
Metamorphic testing

- Utiliser des propriétés liant entrées et sorties
  - Si 2 entrées ont une relation,
    alors leurs sorties peuvent avoir une relation connues



  - Ainsi si l'égalité n'est pas respectée, le test échoue
  - Pour appliquer cette méthode, il faut identifier les relation r et r'

  - Par exemple avec *p*
    la fonction sinus,
    alors pour *r*(x) = π-x
    on a *r'*(y)=y : l'identité



Synthèse Oracle

- Différentes mises en œuvre
- Quelque soit l'oracle on ne sera pas exhaustif
  - Même difficulté que pour les données de test

- Compromis
  - Utilisation d'oracle discret et heuristique

IUT Nantes
Pôle Sciences et technologie

Nantes Université

Diagnostic

Jean-Marie Mottu
Nantes Université

Diagnostic : analyse de l'échec des cas de test

- Le diagnostic exploite
  - L'oracle
    - Les vérifications (multiples) effectuées par l'oracle donnent des indications sur la différence entre
      - le comportement attendu et
      - le comportement obtenu
  - La trace
    - Il s'agit de remonter aux sources de l'échec des cas de test
    - Top-down
      - Depuis l'entrée du programme vers la faute causant la divergence
    - Bottom-up
      - Depuis l'erreur (ce que l'oracle a observé)
      - Vers la faute (qu'il faudra corriger)
    - Souvent mixte

Analyse de trace

- Produire une trace est
  - Statique : technique de « model checking »
    - Le code est modélisé puis ce modèle est analysé
  - Dynamique : les tests sont exécutés pour produire la trace

- L'analyse de la trace dynamique est
  - Statique : la trace est stockée pour être analysée à postériori
    - Stockée dans un modèle, dans des logs (plus ou moins structurés)
  - Dynamique : pendant l'exécution on peut observer le comportement du cas de test

Diagnostic
Trace + Oracle

- Trace et oracle vont conjointement permettre d'identifier
  - Les divergences de valeurs
    - Quand une variable n'a pas la valeur attendue
    - Quand elle est utilisée
      - En écriture : quand elle est définie (passage de paramètre, affectation)
      - En lecture : quand elle est utilisée
        - Dans des conditionnelles
        - Dans des affectations
  - Les divergences dans le flot d'exécution
    - Quand l'exécution vient à passer dans des lignes de code où, elle ne devrait pas

Approche top-down

- Exploitation du mode debug d'un IDE
  - Mise en place de point d'arrêt
    - Nécessite d'avoir localisé la source du problème
      - Basé sur les informations renvoyées par l'oracle
        - Quelle assertion a échoué
        - Quelle variable est impliquée
        - Où est manipulée la variable
  - Avancement pas à pas
    - Pas :
      - Par instruction
      - Par appel
    - Après chaque pas, on connait l'état complet du système (variables, attributs)
    - Nécessite de savoir à quoi va ressembler la divergence
      - Difficile quand l'oracle n'a pu l'observer que bien plus tard

Approche Bottom-up
Remontée de trace

- Stocker la trace
- Analyser la trace depuis l'assertion ayant échoué, jusqu'au point de divergence
  - Difficulté de savoir quel est le point de divergence quand la variable impliquée est écrite plusieurs fois

Approche Bottom-up
Recoupement de traces d'exécution

- Exploitation de plusieurs traces pour identifier une zone potentielle où serait la faute
  - Réduit la zone de recherche

- Spectrum-Based Fault Localization (SBFL) techniques
  - Principe
    - identifier pour chaque cas de test les instructions couvertes
    - Combiner ces couvertures pour ordonner les instructions potentiellement fausses
      - Plus une instruction est couverte par des tests échouant et moins elle est couverte par des tests passant, alors plus elle a de potentiel d'être fausse
      - Différents algorithmes de combinaison de couverture

Algorithme de Jones et al. (diapo Le Traon, Baudry)

- Ordonner les instructions de la plus suspecte à la moins suspecte.
- Exemple :

```
pow(x, y:integer) : float
  local i, p : integer
  i := 0;                    {1}
  Result := 1;               {2}
  if y<0 then p := -x;       {3}
  else p := y;               {4}
  while i<p do
    Result := Result * x;    {5}
    i := i + 1;              {6}
  done
  if y<0 then
    Result := 1/Result;      {7}
end
```

- Fonction puissance
- Une faute a été introduite en {3}

  (l'instruction correcte serait p:=-y)

Algorithme de Jones et al. (diapo Le Traon, Baudry)

- Ordonner les instructions de la plus suspecte à la moins suspecte.
- Exemple :

4 cas de test

		1	2	3	4
		x=2 y=4	x=-2 y=0	x=2 y=-4	x=-3 y=-3
pow(x, y:integer) : float					
local i, p : integer					
i := 0;	{1}	1	1	1	1
Result := 1;	{2}	1	1	1	1
if y<0 **then** **p := -x**;	{3}	0	0	1	1
else p := y;	{4}	1	1	0	0
while i<p **do**					
Result := Result * x;	{5}	1	0	0	1
i := i + 1;	{6}	1	0	0	1
done					
if y<0 **then**					
Result := 1/Result;	{7}	0	0	1	1
end					
		P	P	F	P

Matrice de diagnostic

Algorithme de Jones et al. (diapo Le Traon, Baudry)

Résultats du diagnostic

		1	2	3	4			
		x=2 y=4	x=-2 y=0	x=2 y=-4	x=-3 y=-3	%Passed	%Failed	**Rang**
pow(x, y:integer) : float								
local i, p : integer								
i := 0;	{1}	1	1	1	1	100%	100%	**3**
Result := 1;	{2}	1	1	1	1	100%	100%	**4**
if y<0 **then p := -x**;	{3}	0	0	1	1	33%	100%	**1**
else p := y;	{4}	1	1	0	0	66%	0%	**5**
while i<p **do**								
Result := Result * x;	{5}	1	0	0	1	66%	0%	**6**
i := i + 1;	{6}	1	0	0	1	66%	0%	**7**
done								
if y<0 **then**								
Result := 1/Result;	{7}	0	0	1	1	33%	100%	**2**
end								
		P	P	F	P			

Diagnostic et cas de test

- Les cas de test de nouveau au centre
  - Fournisse explicitement l'oracle
  - Fournisse la trace par la sollicitation du programme avec la donnée de test
- Dépendance entre les données test, les oracles et le diagnostic

- Plus il y aura de cas de test et plus le diagnostic est facile
  - Contraire aux pratiques du test qui minimisent le nombre de cas de test pour atteindre un critère de test donné
  - Itératif
    - Produire des cas de test pour identifier des erreurs
    - Les compléter avec des cas de test pour identifier les fautes

Le cycle de test dynamique