Test et Vérification d'algorithmes

Informatique Théorique 2
Licence 3 informatique

Sébastien Verel
verel@univ-littoral.fr
http://www-lisic.univ-littoral.fr/~verel

Université du Littoral Côte d'Opale
Laboratoire LISIC
Equipe OSMOSE

Objectifs de la séance 7

- savoir proposer un jeux de tests simples pour un algorithme
- connaître la différence entre jeux de tests et preuve de correction
- prouver la correction d'algorithmes utilisant l'affectation, les conditions simples, les itérations

Objectifs de la séance 7

- savoir proposer un jeux de tests simples pour un algorithme
- connaître la différence entre jeux de tests et preuve de correction
- prouver la correction d'algorithmes utilisant l'affectation, les conditions simples, les itérations

Questions principales du jour :

Comment savoir qu'un algorithme donne le résultat espéré ?

Plan

Quelques algorithmes

Recherche du maximum dans un tableau de réels

Algorithme maximum(t : tableau de réel, n : entier)réel
début
variable max : réel
variable i : entier
  $max \leftarrow t[0]$
  pour i **de** 1 **à** n-1 **faire**
    si $t[i-1] < t[i]$ **alors**
      $max \leftarrow t[i]$
    fin si
  fin pour
  retourner *max*
fin

Quelques algorithmes

Recherche du maximum dans un tableau de réels

Algorithme maximum(t : tableau de réel, n : entier)réel
début
variable max : réel
variable i : entier
  $max \leftarrow t[0]$
  pour i **de** 1 **à** n-1 **faire**
    si $t[i-1] < t[i]$ **alors**
      $max \leftarrow t[i]$
    fin si
  fin pour
  retourner *max*
fin

- L'algorithme donne-t-il bien l'élément maximum du tableau ?

Quelques algorithmes

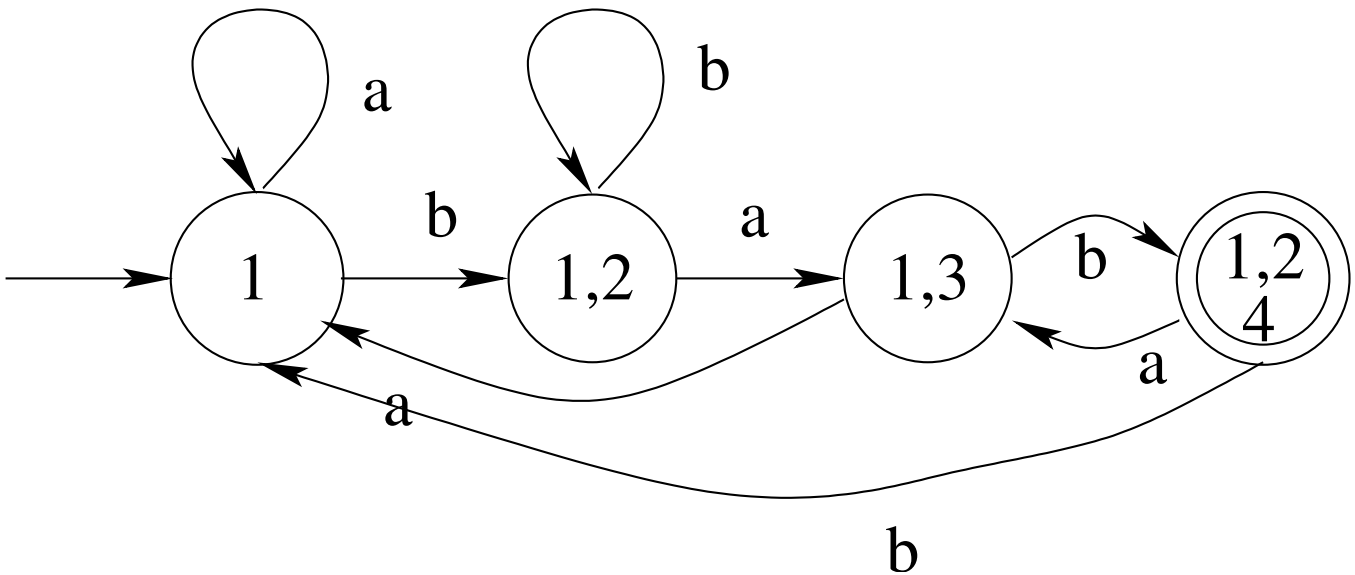
Recherche du maximum dans un tableau de réels

Algorithme maximum(t : tableau de réel, n : entier)réel
début
variable max : réel
variable i : entier
  $max \leftarrow t[0]$
  pour i **de** 1 **à** n-1 **faire**
    si $t[i-1] < t[i]$ **alors**
      $max \leftarrow t[i]$
    fin si
  fin pour
  retourner *max*
fin

- L'algorithme donne-t-il bien l'élément maximum du tableau ?
- En général, les étudiants sont très confiants de leur algorithme....

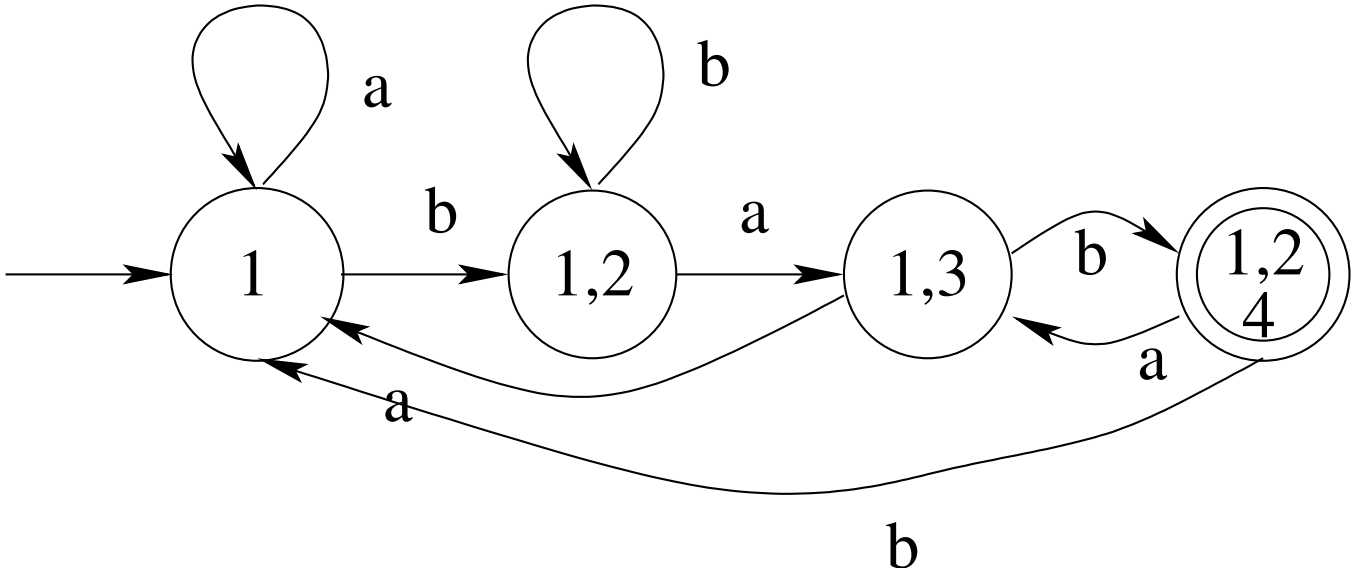
Avec un automate

$$L = \{ubab \mid u \in \Sigma^*\}$$



Avec un automate

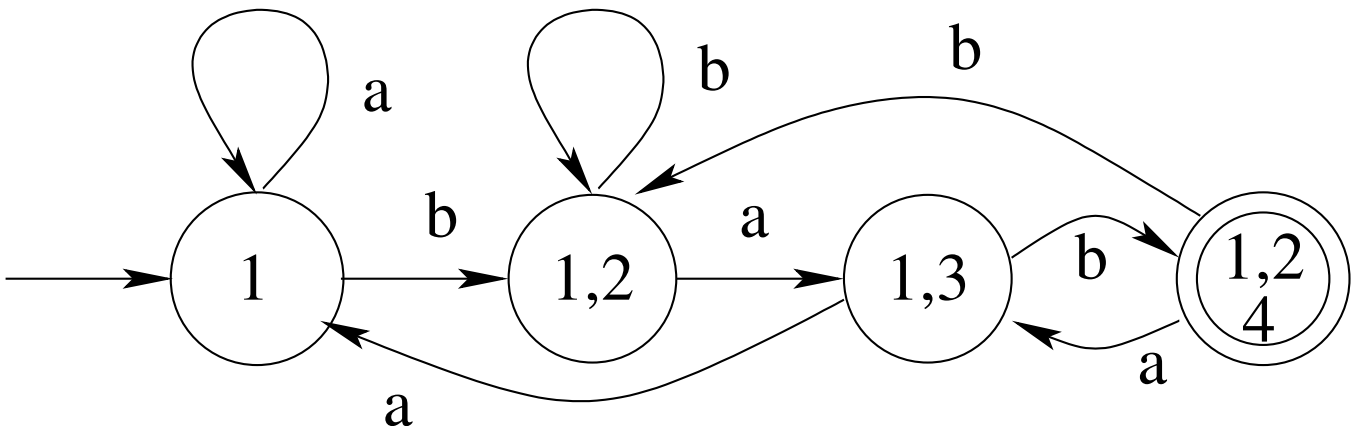
$$L = \{ubab \mid u \in \Sigma^*\}$$



Est-ce l'automate reconnait bien le langage L ?

Avec un automate

$$L = \{ubab \mid u \in \Sigma^*\}$$



Est-ce l'automate reconnait bien le langage L ?

Algorithmes critiques

Quelques exemples où il est nécessaire d'avoir des algorithmes corrects :

- contrôle de pilote automatique
- contrôle des commandes d'un avion
- centrale nucléaire
- robots médicaux
- bases de données critiques : sncf, banque, etc.
- circuits électroniques
- sécurité des systèmes d'information
- sécurité des puces électroniques

Plusieurs méthodes

Comment faire en sorte que l'on soit sùr de la correction de l'algorithme ?

Plusieurs méthodes

Comment faire en sorte que l'on soit sùr de la correction de l'algorithme ?

- Générer un ensemble de tests :
  mais la plupart du temps,
  pas de garantie absolue que l'algorithme est correct :
  on augmente la confiance

Plusieurs méthodes

Comment faire en sorte que l'on soit sùr de la correction de l'algorithme ?

- Générer un ensemble de tests :
  mais la plupart du temps,
  pas de garantie absolue que l'algorithme est correct :
  on augmente la confiance
- Vérifier que l'algorithme corresponde bien aux attentes :
  couvre plusieurs méthodes, parfois compliqué mais très sùr

Plusieurs méthodes

Comment faire en sorte que l'on soit sùr de la correction de l'algorithme ?

- Générer un ensemble de tests :
  mais la plupart du temps,
  pas de garantie absolue que l'algorithme est correct :
  on augmente la confiance
- Vérifier que l'algorithme corresponde bien aux attentes :
  couvre plusieurs méthodes, parfois compliqué mais très sùr
  (à condition de ne pas faire d'erreur dans la preuve...)

Plusieurs méthodes

Tester et Vérifier sont des méthodes complémentaires :

- La vérification a pour but de prouver des propriétés sur des systèmes en manipulant des représentations formelles
- Le test manipule directement le système réel

Importance de la correction

Coût de plus en plus important dans les applications :
- 60% pour certaines applications

Importance de la correction

Coût de plus en plus important dans les applications :
60% pour certaines applications

Ces méthodes (test et vérification) peuvent être produites :

- A la main : lent
- Automatiquement : rapide mais pas toujours possible

Nécessite une spécification formelle (dans un langage abstrait) du programme et de ces propriétés

Définition

Test (définition)

Le test consiste à essayer de trouver des erreurs ou à augmenter la confiance dans la correction d'une implantation par rapport à sa spécification

spécification : propriété (abstraite) qui caractérise une fonctionnalité de l'algorithme

implantation : programme exprimant un algorithme dans un langage donné

Définition

Test (définition)

Le test consiste à essayer de trouver des erreurs ou à augmenter la confiance dans la correction d'une implantation par rapport à sa spécification

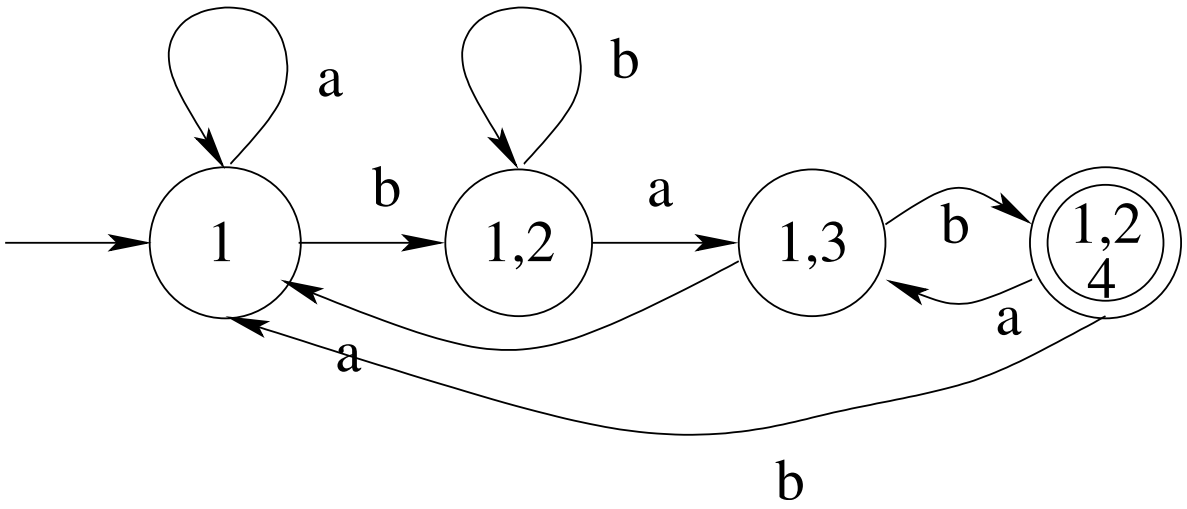
spécification : propriété (abstraite) qui caractérise une fonctionnalité de l'algorithme

implantation : programme exprimant un algorithme dans un langage donné

En fait, le test n'a pas pour but d'établir la correction d'un algorithme...

Exemple avec l'automate

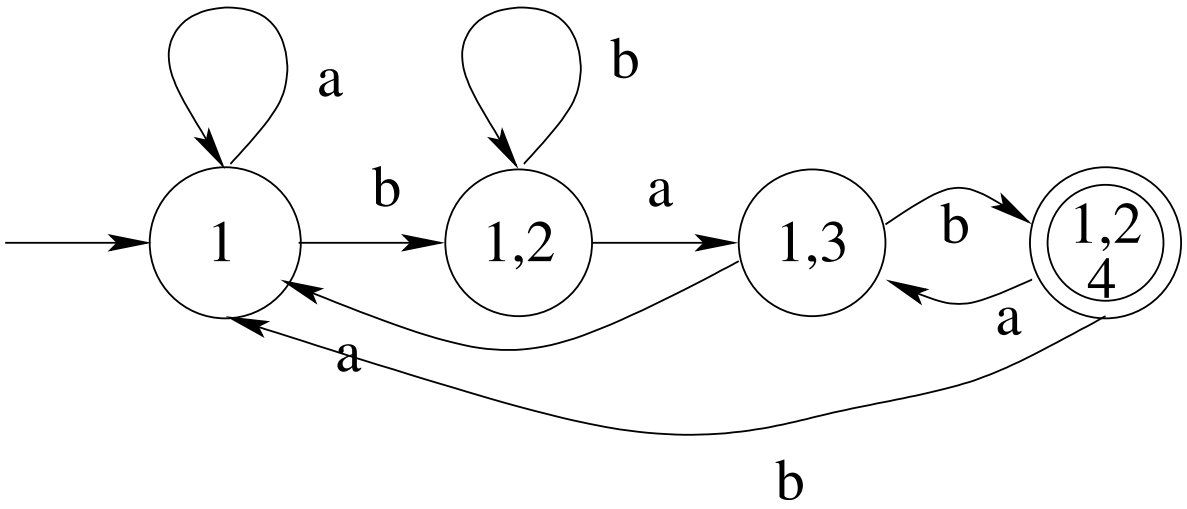
$$L = \{ubab \mid u \in \Sigma^*\}$$



Pouvez-vous trouver un test qui mette en défaut l'automate ?

Exemple avec l'automate

$$L = \{ubab \mid u \in \Sigma^*\}$$



Pouvez-vous trouver un test qui mette en défaut l'automate ?

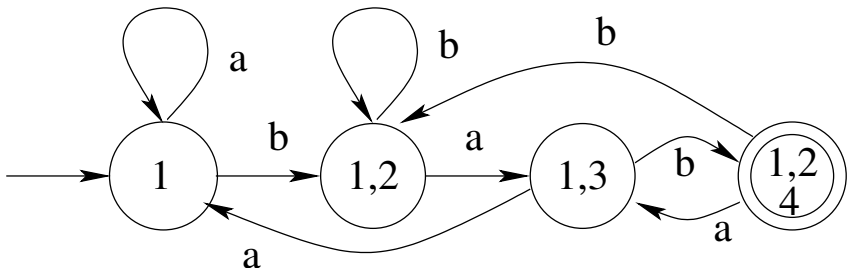
babbab

Méthode plus automatique

Principe d'une méthode plus automatique :

- Tester que toutes les transitions entre deux états conduisent à ce qui est attendu.
- Trouver les chemins qui permettent de tester l'ensemble des transitions.

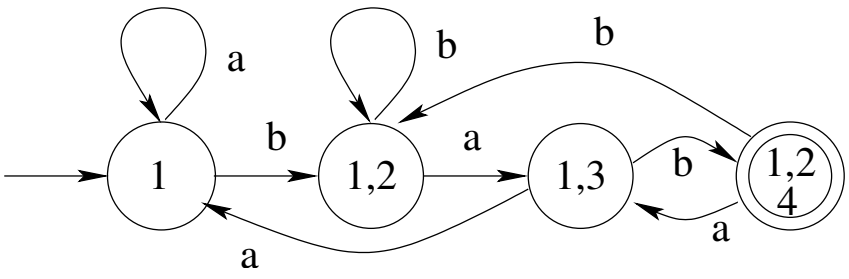
Exemple avec l'automate



attendu dans :

- état 4 : $(a+b)^*bab$
- état 3 : $(a+b)^*ba$
- état 2 : $(a+b)^*b$
- état 1 : $(a+b)^*$

Exemple avec l'automate



attendu dans :

- état 4 : $(a+b)^*bab$
- état 3 : $(a+b)^*ba$
- état 2 : $(a+b)^*b$
- état 1 : $(a+b)^*$
- $3 \rightarrow 4$: ok
- $2 \rightarrow 3$: ok
- $1 \rightarrow 2$: ok
- $4 \rightarrow 3$: ok
- $4 \rightarrow 2$: ok
- ...

Avec un langage impératif

- Dans ce cas l'état peut être l'état de la mémoire :
  Après chaque opération, un état de la mémoire est attendu

Avec un langage impératif

- Dans ce cas l'état peut être l'état de la mémoire :
  Après chaque opération, un état de la mémoire est attendu
- Il faut alors tester toutes les transitions possibles et comparer à l'état attendu

Avec un langage impératif

- Dans ce cas l'état peut être l'état de la mémoire :
  Après chaque opération, un état de la mémoire est attendu
- Il faut alors tester toutes les transitions possibles et comparer à l'état attendu
- Ou alors exiber un contre-exemple,
  trouver une exécution où le résulat attendu est différent du résultat constater.

il est possible d'automatiser cette procèdure mais nous n'allons pas le faire dans ce cours

Exemple

Recherche du maximum dans un tableau de réels

Algorithme maximum(t : tableau de réel, n : entier) : réel
début
variable max : réel
variable i : entier
  $max \leftarrow t[0]$
  pour i **de** 1 **à** n-1 **faire**
    si $t[i-1] < t[i]$ **alors**
      $max \leftarrow t[i]$
    fin si
  fin pour
  retourner *max*
fin

on s'attend que max contienne toujour le maximum du tableau lu entre 0 et i

Il est facile de trouver un contre-exemple :

Exemple

Recherche du maximum dans un tableau de réels

Algorithme maximum(t : tableau de réel, n : entier) : réel
début
variable max : réel
variable i : entier
  $max \leftarrow t[0]$
  pour i **de** 1 **à** n-1 **faire**
    si $t[i-1] < t[i]$ **alors**
      $max \leftarrow t[i]$
    fin si
  fin pour
  retourner *max*
fin

on s'attend que max contienne toujour le maximum du tableau lu entre 0 et i

Il est facile de trouver un contre-exemple :

8	2	3	1

pour $i = 1$, $max = 8$
pour $i = 2$,

Exemple

Recherche du maximum dans un tableau de réels

Algorithme maximum(t : tableau de réel, n : entier) : réel
début
variable max : réel
variable i : entier
  $max \leftarrow t[0]$
  pour i **de** 1 **à** n-1 **faire**
    si $t[i-1] < t[i]$ **alors**
      $max \leftarrow t[i]$
    fin si
  fin pour
  retourner *max*
fin

on s'attend que max contienne toujour le maximum du tableau lu entre 0 et i

Il est facile de trouver un contre-exemple :

8	2	3	1

pour $i = 1$, $max = 8$
pour $i = 2$, $max = 3$

Exemple

Algorithme maximum(t : tableau de réel, n : entier) : réel
début
variable max : réel
variable i : entier
  $max \leftarrow t[0]$
  pour i **de** 1 **à** n-1 **faire**
    si $max < t[i]$ **alors**
      $max \leftarrow t[i]$
    fin si
  fin pour
  retourner *max*
fin

On augmente la probabilité que l'algorithme soit correct avec le nombre de tests.

Erreurs fréquemment rencontrées

Conseil éternel dans votre pratique informatique (ou non) :

Au moins faire 1 test !

Erreurs fréquemment rencontrées

Conseil éternel dans votre pratique informatique (ou non) :

Au moins faire 1 test !

Imaginer des séquences qui peuvent poser problème :

- début et fin de tableau
- répétition d'un même élément
- valeurs critiques : taille nulle, etc.

Erreurs fréquemment rencontrées

Conseil éternel dans votre pratique informatique (ou non) :

Au moins faire 1 test !

Imaginer des séquences qui peuvent poser problème :

- début et fin de tableau
- répétition d'un même élément
- valeurs critiques : taille nulle, etc.

Attention : ce n'est pas parce que l'algorithme fonctionne sur 1 exemple que l'on peut généraliser.

Petit calcul

Une erreur se produit avec une probabilité p sur l'ensemble des données

Combien faut-il faire de tests pour la probabilité de voir l'erreur se produire soit supérieure ou égale à α ?

Petit calcul

Une erreur se produit avec une probabilité p sur l'ensemble des données

Combien faut-il faire de tests pour la probabilité de voir l'erreur se produire soit supérieure ou égale à α ?

probabilité qu'au moins une erreur se produise sur n tests :

$$1 - (1 - p)^n$$

Petit calcul

Une erreur se produit avec une probabilité p sur l'ensemble des données

Combien faut-il faire de tests pour la probabilité de voir l'erreur se produire soit supérieure ou égale à α ?

probabilité qu'au moins une erreur se produise sur n tests :

$$1 - (1 - p)^n$$

d'où $n \geq \frac{\log(1-\alpha)}{\log(1-p)}$

Petit calcul

Une erreur se produit avec une probabilité p sur l'ensemble des données

Combien faut-il faire de tests pour la probabilité de voir l'erreur se produire soit supérieure ou égale à α ?

probabilité qu'au moins une erreur se produise sur n tests :

$$1 - (1 - p)^n$$

d'où $n \geq \frac{\log(1-\alpha)}{\log(1-p)}$

pour $p = 10^{-3}$ et $\alpha = 0.95$,

Petit calcul

Une erreur se produit avec une probabilité p sur l'ensemble des données

Combien faut-il faire de tests pour la probabilité de voir l'erreur se produire soit supérieure ou égale à α ?

probabilité qu'au moins une erreur se produise sur n tests :

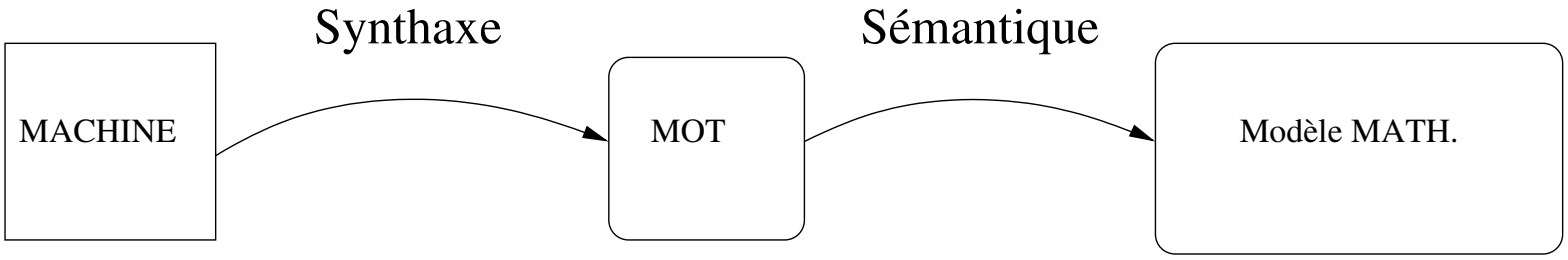
$$1 - (1 - p)^n$$

d'où $n \geq \frac{\log(1-\alpha)}{\log(1-p)}$

pour $p = 10^{-3}$ et $\alpha = 0.95$, $n \geq 2994$

Sous l'hyptothèse d'uniformité d'utilisation des données

Machine / synthaxe / sémantique



Synthaxe concrète / abstraite

Synthaxe abstraite

identifie les composantes significatives des constructions du langage.

Synthaxe concrète

décrit complètement la représentation écrite (placement des parenthèses, ponctuations, etc.)

```
while (x != A[i]) {
  i = i + 1;
}
```

```
while x <> A[i] do
  i = i + 1
od
```

Sémantique

Sémantique

Décrit les relations entre la synthaxe et le “modèle de calcul”

Etude de la sémantique

a pour but de fournir une signification aux différents objets pouvant être construits à l'aide de la synthaxe

Vérification (correction partielle)

Spécification

Une spécification formelle de l'agorithme est basée sur un langage formel dont la sémantique est bien définie.

On exprime une spécification de l'algorithme qui permet décrire une fonctionnalité souhaitée

Vérification

La vérification consiste à comparer une spécification à la réalisation finale de l'algorithme.

Méthodes de vérification formelle

- Méthode opérationnelle
- Méthode axiomatique (aujourd'hui)
- Méthode dénotationnel
- Méthode algébrique

Logique de Hoare (1969)

- Décrit les évolutions possibles d'un programme informatique.

Logique de Hoare (1969)

- Décrit les évolutions possibles d'un programme informatique.

Les évolutions sont modélisés par des régles et
l'état d'un programme par un triplet :

$$\{P\}C\{Q\}$$

P et Q sont des propositions, assertion sur les variables du programmes :

- P : pré-condition
- Q : post-condition

Logique de Hoare (1969)

- Décrit les évolutions possibles d'un programme informatique.

Les évolutions sont modélisés par des régles et
l'état d'un programme par un triplet :

$$\{P\}C\{Q\}$$

P et Q sont des propositions, assertion sur les variables du programmes :

- P : pré-condition
- Q : post-condition

Signifie que si l'état du système vérifie la proposition P alors après exécution de C, l'état du système vérifira Q

Exemples

$$\{n = 4 \wedge a = 1\} a \leftarrow n \{n = 4 \wedge a = 4\}$$

$$\{n = 4 \wedge i = 0\} \textbf{ si } i = 0 \textbf{ alors } \mathrm{n} \leftarrow 8 \textbf{ sinon } \mathrm{n} \leftarrow 6 \textbf{ finSi}$$
$$\{n = 8 \wedge i = 0\}$$

Exemples

Pour une même instruction, plusieurs pré et post conditions sont possibles :

$$\{x \geq 0\}\ x \leftarrow x + 1\ \{x \geq 0\}$$

$$\{x \geq 0\}\ x \leftarrow x + 1\ \{x > 0\}$$

Propriétés

- raffinement de la post-condition :

$$\text{Si } \{P\}C\{Q\} \text{ et } Q \Rightarrow R$$
$$\text{alors}$$

Propriétés

- raffinement de la post-condition :

$$\text{Si } \{P\}C\{Q\} \text{ et } Q \Rightarrow R$$
$$\text{alors } \{P\}C\{R\}$$

- raffinement de la pré-condition :

$$\text{Si } \{P\}C\{Q\} \text{ et } O \Rightarrow P$$
$$\text{alors}$$

Propriétés

- raffinement de la post-condition :

$$\text{Si } \{P\}C\{Q\} \text{ et } Q \Rightarrow R \text{ alors } \{P\}C\{R\}$$

- raffinement de la pré-condition :

$$\text{Si } \{P\}C\{Q\} \text{ et } O \Rightarrow P \text{ alors } \{O\}C\{Q\}$$

- enchaînement d'opérations :

$$\text{Si } \{P\}C_1\{Q\} \text{ et } \{Q\}C_2\{R\} \text{ alors}$$

Propriétés

- raffinement de la post-condition :

$$\text{Si } \{P\}C\{Q\} \text{ et } Q \Rightarrow R$$
$$\text{alors } \{P\}C\{R\}$$

- raffinement de la pré-condition :

$$\text{Si } \{P\}C\{Q\} \text{ et } O \Rightarrow P$$
$$\text{alors } \{O\}C\{Q\}$$

- enchaînement d'opérations :

$$\text{Si } \{P\}C_1\{Q\} \text{ et } \{Q\}C_2\{R\}$$
$$\text{alors } \{P\}C_1; C_2\{R\}$$

Affectation

Soit $P(z)$ une proposition dépendant d'une variable z
et ne dépendant pas de la variable x

$$\{P(e)\}x \leftarrow e\{P(x)\}$$

Si $P(e)$ est vraie alors en exécutant l'affection $x \leftarrow e$, $P(x)$ est vraie
puisque après l'affectation x est à la valeur e

Exemple affectation

Recherche de la pré-condition :

$$\{?\}x \leftarrow a + 2\{x = 5\}$$

Exemple affectation

Recherche de la pré-condition :

$$\{?\}x \leftarrow a + 2\{x = 5\}$$

Substitution de x par $a + 2$ de la post-condition :

$$\{a + 2 = 5\}x \leftarrow a + 2\{x = 5\}$$

Exemple affectation

Recherche de la pré-condition :

$$\{?\}x \leftarrow a + 2\{x = 5\}$$

Substitution de x par $a + 2$ de la post-condition :

$$\{a + 2 = 5\}x \leftarrow a + 2\{x = 5\}$$

donc, $\{a = 3\}x \leftarrow a + 2\{x = 5\}$ est démontré

Exemple affectation

Recherche de la pré-condition :

$$\{?\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

Exemple affectation

Recherche de la pré-condition :

$$\{?\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

Substitution de a par n de la post-condition :

$$\{n = 4 \wedge n = 4\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

Exemple affectation

Recherche de la pré-condition :

$$\{?\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

Substitution de a par n de la post-condition :

$$\{n = 4 \wedge n = 4\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

$$\text{or, } n = 4 \wedge n = 4 \Leftrightarrow n = 4$$

Exemple affectation

Recherche de la pré-condition :

$$\{?\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

Substitution de a par n de la post-condition :

$$\{n = 4 \wedge n = 4\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

$$\text{or, } n = 4 \wedge n = 4 \Leftrightarrow n = 4$$

$$\text{et, } n = 4 \wedge a = 1 \Rightarrow n = 4$$

Exemple affectation

Recherche de la pré-condition :

$$\{?\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

Substitution de a par n de la post-condition :

$$\{n = 4 \wedge n = 4\}a \leftarrow n\{n = 4 \wedge a = 4\}$$

$$\text{or, } n = 4 \wedge n = 4 \Leftrightarrow n = 4$$

$$\text{et, } n = 4 \wedge a = 1 \Rightarrow n = 4$$

$$\text{donc } \{n = 4 \wedge a = 1\}a \leftarrow n\{n = 4 \wedge a = 4\} \text{ est démontré}$$

Condition simple

Soient b une expression booléenne, C et D deux algorithmes, et P et Q deux propositions.

Si :

1. $\{b \wedge P\} C \{Q\}$
2. $\{\neg b \wedge P\} D \{Q\}$

Alors :

- $\{P\}$ Si b Alors C Sinon D FinSi $\{Q\}$

Iteration tant que

Si :

1. $P \Rightarrow I$
2. $\{b \wedge I\} C \{I\}$
3. $\neg b \wedge I \Rightarrow Q$

Alors :

- $\{P\}$ TantQue b Faire C FinTantQue $\{Q\}$

I est un invariant de boucle

Objectifs de la séance 7

- savoir proposer un jeux de tests simples pour un algorithme
- connaître la différence entre jeux de tests et preuve de correction
- prouver la correction d'algorithmes utilisant l'affectation, les conditions simples, les itérations

Questions principales du jour :

Comment savoir qu'un algorithme donne le résultat espéré ?