GÉNÉRALITÉS SUR LES TESTS
TESTS INTÉGRÉS

2012-10-01

© Emmanuel Chieze
Département d'informatique, UQAM
INF3135

Plan

2

- Généralités sur les tests
- Scripts Shell
- Expressions régulières
- Tests intégrés

Preuves formelles

3

- Démonstration mathématique qu'un programme fonctionne, basée sur la notion de pré- et postconditions
- Garantit l'absence de bugs ou de mauvais fonctionnement
- Difficile à mettre en pratique
  - pour des programmes de taille réaliste
  - lorsqu'il y a des fonctions non pures
- Certains langages séparent les fonctions pures et les autres pour faciliter les preuves formelles
  - exemple : Haskell

Preuves formelles

4

- Exemple de tailles typiques de logiciels (en milliers de lignes de code)
  - Compilateur 10 (C)
  - Système de commutation X25 100 (C)
  - Contrôle de sous-marin nucléaire 1000 (ADA)
  - Contrôle de station spatiale 10.000 (ADA)
- Palliatifs lorsque les preuves formelles sont trop complexes ou impossibles
  - Validation du code par des tests
  - Vérification du code par des êtres humains

Validation

5

- Les tests
  - Garantissent seulement le bon fonctionnement du programme dans les cas testés
  - Garantissent la présence d'erreur lorsqu'un test échoue
  - Ne peuvent garantir que le programme est exempt de mauvais fonctionnement

Validation

6

- Exhaustivité des tests
  - exemple : soit la fonction f définie sur des entiers

```
int f (unsigned n) {
  while (n > 1)
     if (n % 2  == 0)
        n = n / 2 ;
     else
        n = 3 * n + 1 ;
  return (n) ;
}
```

Validation

7

- Prouver que f retourne toujours 1
  - si entiers codés sur 4 octets
    - tests exhaustifs = 2^{32} = 4.3 10^9 valeurs à tester.
    - si un test prend 1ms, presque 50 jours sont nécessaires !
  - si entiers codés sur 8 octets
    - tests exhaustifs = 2^{64} = 18 10^{18} valeurs à tester !
    - 585 millions d'années sont nécessaires !
- Impossibilité de tests exhaustifs en général

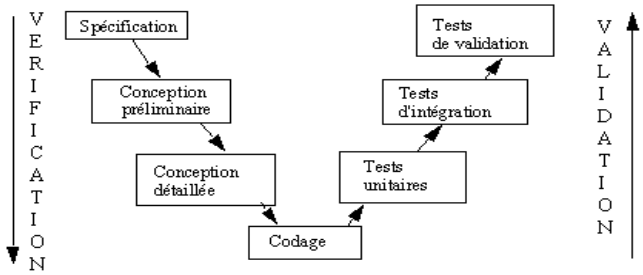
Validation

8

- Il faut adopter des stratégies pour concevoir des tests
  - représentatifs des différentes possibilités d'utilisation d'un programme
    - cas usuels
    - cas limites
    - cas à problème
  - couvrant l'ensemble du code écrit
  - de taille raisonnable
    - ne pas introduire de tests redondants
    - chaque test doit avoir sa justification

Validation

- Plusieurs niveaux de tests : unitaires, intégrés, système, d'acceptation



Différents types de tests

- Tests à la charge de l'équipe de développement
  - Tests unitaires
    - vérifie le bon fonctionnement de chaque module et fonction exportable
    - sous la responsabilité du programmeur
  - Tests intégrés et tests de système
    - vérifie le bon fonctionnement d'un sous-système et/ou d'une application dans son ensemble
    - vérifie notamment l'arrimage des modules composant le sous-système/l'application
    - sous la responsabilité de l'analyste
  - Tests de performance

Différents types de tests

11

- Tests à la charge du client
  - Tests d'acceptation
    - similaires aux tests intégrés/de système
    - nécessaires pour des raisons contractuelles
    - 2 étapes
      - alpha : tests d'acceptation préliminaires, en environnement de développement.
      - beta : tests d'acceptation live, dans un environnement de production réel.

Automatisation des tests

12

- Permet les tests de non-régression
- Il faut maintenir à jour le fichier de tests lors de modifications du module
  - ajout de cas testant chaque bug corrigé
  - ajout de cas pour les nouvelles fonctionnalités
  - modification de cas pour la modification de fonctionnalités
- Techniques :
  - shell script,
  - programme C et but *test* dans *makefile,*
  - outils dédiés

Vérification de code

13

- Étape complémentaire aux tests
  - Exemple : un programme doit traiter l'entrée standard sans fixer de limites a priori sur la taille des lignes
    - Impossibilité de valider cette contrainte par des tests
    - Nécessité de recourir à la vérification de code

Vérification de code

14

- La vérification est entièrement réalisée par des humains
  1. Vérifier le respect des normes de programmation
  2. Vérifier la lisibilité du code
  3. Vérifier la pertinence des commentaires
  4. Vérifier l'absence d'hard-codage de valeurs
  5. Vérifier la cohérence du code par rapport aux documents d'analyse organique
  6. Vérifier le respect des interfaces

Vérification de code

15

7. Recherche des défauts les plus courants dans le code
   - utilisation de variables non initialisées
   - définition de variables non utilisées (donc inutiles)
   - sauts dans des boucles
   - affectations incompatibles
   - boucles infinies
   - débordements de tableaux
   - allocations et libérations impropres de zones mémoires
   - mauvaises correspondances entre arguments et paramètres formels
   - tests d'égalité entre valeurs flottantes (erreurs d'arrondi)

- Certaines des erreurs ci-dessus peuvent être repérées par certains compilateurs

Plan

16

- Généralités sur les tests
- Scripts Shell
- Expressions régulières
- Tests intégrés

Shell

17

- (Langage) Shell = langage simple pour combiner et automatiser l'exécution de commandes UNIX
- Script Shell = programme écrit en shell
  - Script interprété
  - Deux grandes familles :
    - C-shell : csh, tcsh, …
    - Bourne shell : sh, ksh, rksh …
- Commandes UNIX
  - exécutables
  - indépendantes du shell

Processus en UNIX

18

- Processus parents/enfants
  - Le lancement d'une commande (exécutable compilé, shell script interprété) crée un processus enfant (généralement)
- Commande ps –f
- Variables d'environnement
  - Héritées par les processus enfants
- Définition des variables d'environnement (csh)
  - Fichiers .login, .cshrc

Shell

19

- Un langage shell gère
  - l'environnement (variables)
  - les processus (commandes exécutées)
  - les redirections de canaux et les pipelines
  - les substitutions …
- Un langage shell offre
  - une gestion de variables locales
  - des structures de contrôle
  - des commandes spécifiques au shell …

Écriture de scripts shell (csh)

20

- Spécifier le shell utilisé au début du script
  - #!/bin/csh
- Valeur des arguments du script :
  - $0 : nom du script
  - $1, $2 à $n : valeurs des arguments 1, 2 …
  - $* : liste de tous les arguments
  - $#argv : nombre d'arguments

Commandes

21

- Commandes spécifiques à csh
  - setenv X toto
  - set X=toto
  - shift
    - supprime le premier argument de la liste des arguments
- Commandes spécifiques à ksh
  - X=toto ; export X
  - X=toto
- Commandes indépendantes du shell
  - rm, man, ls, grep, diff, wc …

Écriture de scripts shell (csh)

22

- Variables
  - désignées par un nom
    - variables d'environnement (exportées) créées par setenv
    - variables locales (non-exportées) créées par set
  - contenu désigné par $nom
  - nombre de mots du contenu désigné par $#nom
- Commentaires suivent #
- Affichage : echo
  - echo Bonjour $1

Écriture de scripts shell (csh)

23

- Structures de contrôle
  - if (condition) then
    (else)
    endif
  - while (condition)
    end
  - foreach var (liste)
    end

Écriture de scripts shell (csh)

24

- Conditions
  - syntaxe similaire à celle du C
    - mais contenu d'une variable var désigné par $var
    - != et == s'appliquent à des chaînes de caractère
  - plus opérateurs additionnels sur les fichiers :
    - -e filename : vrai si le fichier existe
    - -z filename : vrai si le fichier est vide
    - -f filename : vrai si le fichier est un fichier
    - -d filename : vrai si le fichier est un répertoire
    - …

Écriture de scripts shell (csh)

- Évaluation d'une commande ` cmd `
  - renvoie le résultat de cmd
  - pour affectation à une variable le plus souvent
    - set jour = `date +%Y-%m-%e`
    - echo $jour
  - ou à une liste
    - foreach fic (`ls`)
    - …
    - end

Substitutions

- Noms de fichiers :
  - * remplace n'importe quelle chaîne de caractères (y compris la chaîne vide)
  - ? remplace exactement un caractère
  - […] : remplace exactement un caractère parmi ceux spécifiés entre crochets
    - spécification par énumération : [1b3]
    - spécification par intervalle : [a-z]
    - spécification par exclusion : [^1b3], [^a-z]

Substitutions

27

- Chaînes de caractères
  - entre ' ' : aucune substitution
  - entre " " : remplacement de $var par la valeur de var, et substitution de noms de fichiers
  - entre ` ` : résultat de (substitution de $var par la valeur correspondante, substitution de noms de fichiers, exécution de la commande résultante)

Exemple : la commande rmvide

28

- Créer un nouveau fichier texte appelé rmvide et taper les lignes suivantes :

```
#!/bin/csh
foreach f (`ls`)
        if ((-z $f) && (-f $f)) then
                echo Suppression du fichier vide $f
                rm -f $f
        endif
end
```

- Rendre le script exécutable

```
chmod u+x rmvide
```

Redirections

29

- Entrée standard
  - commande <fic1
- Sortie standard
  - commande >fic1
  - commande >>fic1
  - commande >/dev/null
- Sortie standard et canal d'erreur
  - commande >&fic1
  - commande >>&fic1
- Pipelines

Variables spéciales

30

- cwd : répertoire courant
- home : répertoire d'entrée
- path : liste des répertoires dans lesquels les exécutables sont recherchés
- prompt : chaîne affichée automatiquement au début de chaque ligne, en attente d'une entrée de l'utilisateur
- status : statut de la dernière commande exécutée

Utilité des scripts shell

31

- Automatisation de chaînes de traitement
- Intégration d'applications
- Approche courante :
  - utilisation de modules élémentaires (commandes UNIX, exécutables ad-hoc écrits en C)
  - chaque module agit comme un filtre
  - les modules sont reliés entre eux par des pipes
  - ou le fichier en sortie de l'un est utilisé en entrée par l'autre (pour conserver les résultats intermédiaires)

Utilité des scripts shell pour des tests

32

- Automatisation des tests intégrés dans des cas simples
  - lancement des tests
  - ***et*** comparaison entre les résultats obtenus et ceux attendus
  - on n'affiche que les résultats des cas à problèmes
    - au complet,
    - ou seulement les sections présentant des différences
- Utilité des scripts shell pour des tests unitaires ?

Plan

33

- Généralités sur les tests
- Scripts Shell
- Expressions régulières
- Tests intégrés

Outils pour manipuler les fichiers texte

34

- Utilitaires s'intègrant bien aux scripts shell
  - grep : repère les lignes contenant un patron donné
  - sed : éditeur de lignes (simple)
  - awk : éditeur de lignes (plus complet)
- perl : langage de programmation complet incluant la manipulation de chaînes de caractères
- Tous ces outils se basent sur les Expressions Régulières (ER)

Expressions régulières

35

- Expressions régulières
  - mini-langage décrivant des patrons syntaxiques simples
  - associées à la théorie des automates à états finis
  - implémentées de façon efficace

Expressions régulières

36

- Expressions régulières d'1 caractère
  - tout caractère autre que . * [ ] \ ^ $ / désigne lui-même
  - pour désigner les caractères ci-dessus, utiliser \
  - . désigne n'importe quel caractère
  - [*liste*] désigne n'importe quel caractère dans *liste*
  - [^*liste*] désigne tout autre caractère que ceux dans *liste*
  - [*c*-*c'*] désigne tout caractère compris entre *c* et *c'*
  - [^*c*-*c'*] désigne tout autre caractère que ceux compris entre *c* et *c'*

Expressions régulières

37

- Expressions régulières de plusieurs caractères (ER désigne une expression valide)
  - *ER* * : 0 occurrences ou + de *ER*
  - *ER* \{*m*\} : *m* occurrences de *ER*
  - *ER* \{*m*,\} *:* au moins *m* occurrences de *ER*
  - *ER* \{*m*,*n*\} *:* entre *m* et *n* occurrences de *ER*
  - L'algorithme est cupide (*greedy*) : il identifie le maximum d'occurrences possible

Expressions régulières

38

- Expressions régulières de plusieurs caractères
  - *ERC1 ERC2* : identifie la concaténation des chaînes repérées par *ERC1* et *ERC2*
  - \(*ERC*\) : permet de traiter des *ERC* complexes comme des *ERC* simples
  - ^*ERC* : identifie *ERC* en début de ligne
  - *ERC*$: identifie *ERC* en fin de ligne

Expressions régulières

39

- Exemples : quelles sont les chaînes identifiées par :
  - `[0-9][0-9]*\.[0-9][0-9]*`
  - `[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\}`
  - `[0-9]\{4\}[-\/][0-9]\{2\}[-\/][0-9]\{2\}`
  - `[_a-zA-Z][_a-zA-Z0-9]*`

grep

40

- grep ne reconnaît pas \(..\) ni \{..\}
- mais grep reconnaît :
  - RE+ : au - 1 occurrence de RE
  - RE ? : 0 ou 1 occurrence de RE
- Afficher les lignes vides de toto.c, préfixées de leur numéro de ligne
  - `grep -n '^$' toto.c`
- Afficher les lignes de toto.c contenant *taille* en minuscules ou majuscules
  - `grep -i taille toto.c`

sed

41

- sed (-e commande)+ [fichier]
- sed -f fichier_commandes [fichier]
- option additionnelle -n pour supprimer la sortie par défaut
- commande = [adresse[,adresse]] comm
  - adresse ou adresse, adresse : numéros de lignes absolus ($ = dernière ligne)
  - ou patron /…/ : seule les lignes contenant le patron sont traitées par la commande
- comm :
  - s : substitution de chaînes
  - d : suppression de ligne
  - p : impression …

sed : commande s

42

- s/ER/remplacement/options
  - remplace la chaîne identifiée par ER par une autre chaîne
  - remplacement :
    - \n désigne la nième chaîne entre \(..\)
  - options :
    - g : toutes les occurrences du patron sur la ligne sont remplacées
    - p : imprime la ligne (utile avec sed -n)

sed : commande s

43

- Exemples : fichier toto

```
Emmanuel 321-43-21
Gilles 342-12-13
Rene (450) 211-11-11
Sylvain (450) 202-1212
Tom 819-321-32-18
```

- Supprimer toutes les occurrences de -

```
sed -e 's/-//g' toto
```


sed : commande s

44

- Supprimer les - entre les 4 derniers chiffres

```
sed -e 's/-\([0-9]\{2\}\)-/-\1/' toto
```

- Rajouter (514) lorsque l'indicatif régional n'est pas présent

```
sed -e '/^[A-Za-z]* [0-9]\{3\}-[0-9]\{2\}-
  [0-9]\{2\}$/s/ / (514) /' toto
```

- Combiner les 3 dernières commandes

```
sed -e 's/ \([0-9]\{3\}\)-\([0-9]\{3\}\)-/
  (\1) \2-/'
  -e '/^[A-Za-z]* [0-9]\{3\}-[0-9]\{2\}-
  [0-9]\{2\}$/s/ / (514) /'
```

sed et scripts shell

45

- sed est très utile pour modifier le résultat de commandes UNIX standard, ou pour extraire certaines informations
  - Donner le nombre de lignes de chaque fichier du répertoire sans afficher de ligne de total

```
wc -l *|sed '/total$/d'
```

  - Donner les 10 fichiers les plus gros du répertoire, en terme de nombre de lignes

```
wc -l *| sed '/total$/d' | sort -r | sed -
  n -e '1,10p'
```


Exemple : corrTP

46

```
#!/bin/csh
rm trace_corrections*
rm [a-z]*.res[0-9]
foreach nom (`ls *indente.c`)
        set etudiant=`echo $nom | sed -e 's/\..*$//'`
        set TRACE=trace_corrections.$etudiant.txt
        finger -sfhp $etudiant  >> $TRACE
       cp $nom indente.c
        echo Compilation >> $TRACE
        gcc indente.c
        echo Tests >> $TRACE
        echo ------------------------------------------------- >> $TRACE
        foreach n (0 1 2 3 4 5 6 7 8 9)
                echo test$n >> $TRACE
                echo ===== >> $TRACE
                a.out < test$n > $etudiant.res$n
                diff res$n $etudiant.res$n >> $TRACE
                echo ===== >> $TRACE
        end
        rm indente.c
        rm a.out
end
```

Plan

47

- Généralités sur les tests
- Scripts Shell
- Expressions régulières
- Tests intégrés

Principes

48

- On veut tester la commande cmd
  - peut être un script shell
  - peut être un exécutable
- cmd est un filtre : lit l'entrée standard et écrit sur la sortie standard
  - prévoir n fichiers de tests correspondant à des entrées distinctes
  - prévoir n fichiers de sortie correspondant aux résultats attendus
  - exécuter cmd sur chacun des n fichiers d'entrée
    - comparer la sortie obtenue avec la sortie attendue
    - afficher un message d'erreur en cas de différence

Exemples de scripts

49

- voir [Tremblay, 2005] pour des scripts génériques
  - dans le cas d'un filtre
  - dans le cas d'une commande modifiant un fichier de l'environnement
- à adapter à ses propres besoins

Exemple d'élaboration de plan de tests intégrés

50

- transforme est un programme qui remplace chaque lettre (entre a et z, A et Z) par la lettre obtenue par un décalage circulaire de n caractères dans l'alphabet. Le résultat s'affiche sur la sortie standard
  - z avec un décalage de -1 donne y
  - z avec un décalage de +1 donne a
- usages
  - transforme décalage : transforme l'entrée standard
  - transforme décalage fichier : transforme le fichier

Analyse des cas à tester

51

- 3 dimensions indépendantes
  - arguments
    - 0 arguments
    - un seul argument, le décalage
    - 2 arguments
    - plus que 2 arguments
  - source
    - fichier inexistant
    - fichier existant (spécifié en deuxième argument / ou envoyé sur l'entrée standard)
      - vide (fic0)
      - de petite taille (fic1)
      - de taille normale, composé de lettres minuscules, majuscules et d'autres caractères (incluant les lettres accentuées) (fic2)
      - de grande taille (taille des lignes et/ou nombre de lignes) (fic3)

Analyse des cas à tester

52

- 3 dimensions indépendantes
  - décalage
    - autre chose qu'un entier
    - 0
    - un petit nombre positif : 1 par exemple
    - un petit nombre négatif : -1 par exemple
    - un nombre positif compris entre 1 et 27 : 5 par exemple
    - 27 : doit donner le même résultat que 1

Stratégie de tests associée

53

- On fait varier chaque dimension indépendamment
  - En fixant les autres à une valeur standard
- Si D1 a N1 valeurs possibles …
  - Environ N1 + N2 + N3 tests
  - Et non N1 * N2 * N3 tests

Stratégie de tests associée

- Variation des arguments
  - Le cas standard (2 args) figure plus loin

Réf test	Cas testé	Commande	Résultats attendus
Test 1	0 args	transforme	Message d'erreur …
Test 2	3 args	transforme 2 toto titi	Message d'erreur …
Test 3	1 arg	cat fic2 | transforme 5	res2

Stratégie de tests associée

- Variation des fichiers : on fixe le décalage à une valeur standard (5) et le nb d'args à 2

Réf test	Cas testé	Commande	Résultats attendus
Test 4	fichier inexistant	transforme 5 ficbidon	Message d'erreur …
Test 5	fichier vide	transforme 5 fic0	sortie vide
Test 6	fichier de petite taille	transforme 5 fic1	res1
Test 7	fichier de taille normale	transforme 5 fic2	res2
Test 8	fichier de grande taille	transforme 5 fic3	res3

Stratégie de tests associée

- Variation du décalage : on fixe le fichier à une valeur standard (fic2, fichier de taille normale) et le nb d'args à 2

Réf test	Cas testé	Commande	Résultats attendus
Test 9	décalage 0	transforme 0 fic2	fic2
Test 10	petit décalage positif	transforme 1 fic2	res4
Test 11	petit décalage négatif	transforme -1 fic2	res5
Test 12	grand décalage positif	transforme 27 fic2	res4
Test 13	décalage aberrant	transforme 3.2 fic2	Message d'erreur …

Référence

- [Tremblay, 2005] : Stratégie de tests (unitaires)