1. Exemple de méthodes et outils pour la correction de programmes

*: prérequis

Intro. Omniprésence des systèmes info. Nécessité de bon fonctionnement à différents niveaux : expérience utilisateur (service), systèmes critiques (aviation) ...

I. Méthodes formelles de correction

1. Logique de Hoare

a) Notions théoriques

def 1 Un **triplet de Hoare** $\{\varphi\}\, pg\, \{\psi\}$ est la donnée de deux formules logiques* φ (la précondition) et ψ (la postcondition), et d'un programme* pg

exemple 2 : algorithme de tri :

$$\{x = arg\}\ tri(arg)\ \left\{ \begin{array}{l} \exists \sigma \text{ perm. } \sigma(x) = arg \wedge \\ \forall 0 < i < |x|,\ arg[i-1] \leq arg[i] \end{array} \right\}$$

$$\text{assign}\ \frac{}{\{\varphi[e/x]\}\ x := e\ \{\varphi\}} \qquad \text{seq}\ \frac{\{\varphi\}\ p1\ \{\theta\} \quad \{\theta\}\ p2\ \{\psi\}}{\{\varphi\}\ p1;\ p2\ \{\psi\}}$$

$$\frac{\{I \wedge e\}\ c\ \{I\}}{\{I\}\ \text{while } e \text{ do } c\ \{I \wedge \neg e\}}\ \text{while} \qquad [\ldots]$$

fig. 3 : système de preuve sur les triplets de Hoare pour un langage impératif simple

def. 4 (prouvabilité) : $\vdash \{\varphi\}\, pg\, \{\psi\}$ s'il existe un arbre de preuve valide construit à partir de fig. 3.

def. 5 (validité) : $\models \{\varphi\}\, pg\, \{\psi\}$ si $\forall I$ interprétation et $\forall s$ état initial t.q. s valide $[\![\varphi]\!]^I$, pg finit sur s, alors $s \bullet pg^*$ valide ψ^I.

Théorème 6 (correction et complétude) : ||DEV. 1

$$\forall \quad \vdash \{\varphi\}\, pg\, \{\psi\} \iff \models \{\varphi\}\, pg\, \{\psi\}$$

☉☉! la logique de Hoare simple :
- peut donner des preuves difficiles à écrire ;
- dépend intrinsèquement de la logique des formules sous-jacentes (logique modale, ...)

b) Utilisation en pratique

Son utilisation peut être une de deux façons :
- montrer son adéquation sur un système plus performant/pratique (ex. systèmes de type)
- comme système automatisable (Why3) DM/TP

2. Systèmes de type

a) Définitions

def ? Un type est un ensemble de valeurs qui partagent des propriétés communes.
ex : int, unit, float → string, ...

Les syntaxes des types varient et peuvent être très riches (polymorphisme, type dépendant...)

def ? Un système de typage est un ensemble de règles, décrivant les opérations permises sur les types, qui permet d'associer un type à un programme.

exemple 9 (système de type simple)

$$\frac{0 \leqslant n < 2^8}{n : \text{int}} \qquad \frac{\vdash u : A \to B \quad \vdash v : A}{\vdash uv : B}$$

$$\overline{\text{cons} : \text{int} \to \text{list} \to \text{list}}$$

« Well-typed programs cannot go wrong »
— Milner (1976)

les systèmes de type suffisamment riches empêchent l'apparition de comportements indéfinis :
- pas d'erreur de segmentation
- d'adressages non-alignés
- ...

Activité comparaison de programmes équivalents en Caml/python, exécution mal typées...

b) Programmation typée

- Plusieurs langages déploient un typage fort qui fournisse ces garanties : OCaml, Haskell, Rust, ...

Cela implique un surcoût de typage à la compilation, et souvent aussi d'inférence, elle-même liée au principe d'unification. ‖ DEV 2

- le typage est aussi lié aux spécifications : découpage en modules, types abstraits.

II. Discipline et contrats

Une spécification peut être informelle pour de bonnes raisons (tâche difficile, dépend du langage, ...)
→ c'est le cas pour la plupart des gros projets (stdlibs, kernel, ...)

1. Documentation

Texte associé à une fonction/un module qui :
- fait apparaître des préconditions (prérequis)
- décrire les effets et les garanties (postconditions) associées.

⊙⊙! Faire la différence entre commentaire et doc, etc.

Activité Commenter des codes mal documentés

2. Vérifications expérimentales

Quand la spécification est informelle, l'utilisation d'outils de vérification par l'exécution est essentiel

a). Tests

- fonctions dédiées pour vérifier plusieurs niveaux de corrections : tests unitaires, d'intégration, d'acceptation. Objectif :
  - conformité aux spécifications
  - éviter les régressions
- quadriciels (framework) de test--

b.) Débogage

def 11 un outil de débogage est un outil d'instrumentation d'un code source permettant de réaliser des exécutions contrôlées d'un programme pour observer ses effets.

exemple 12
- gdb : exécution pas à pas, observation de la pile ;
- valgrind : observation des bugs, détection des fuites mémoires