Extensions

Suivant les langages de programmation, modules plus avancés :

- modules imbriqués
- modules paramétrés par des modules (foncteurs)
  - généricité
- modules de première classe : peuvent être manipulés comme des valeurs du langage

Dictionnaire

Types abstraits

Au moment de la conception du programme, on est amené à définir des types abstraits en spécifiant quelles propriétés ils doivent vérifier

- Le développeur du type devra respecter ces propriétés, mais sera libre de l'implémentation concrète
- L'utilisateur pourra supposer que les propriétés sont vérifiées pour son code

Spécifications algébriques

Les spécifications algébriques sont une des manières de procéder ainsi. Elles contiennent

- les prototypes des fonctions manipulant le type abstrait
- des équations mettant en relation ces fonctions, expliquant ainsi les propriétés qu'elles doivent vérifier

De la spécification algébrique à l'implémentation

prototypes $\longrightarrow$ interface d'un module

comment vérifier que l'implémentation vérifie les propriétés ?

- test
  - à la main
  - génération de tests

  exhaustivité ?
- certification (preuve)
  - à la main
  - formelle (interactive ou automatique)

  $\pm$ facile suivant le langage

Dictionnaire

Exemple concret :
les dictionnaires (aussi appelés tableaux associatifs ou tables d'association)

On veut associer des clefs $k \in \mathcal{K}ey$ à des valeurs $v \in \mathcal{V}al$

Il est possible

- de créer un dictionnaire vide
- d'insérer une nouvelle association
- de rechercher à quelle valeur est associée une clef
- de supprimer une association

En général, $\mathcal{K}ey$ est supposé totalement ordonné.
Dans la suite, on supposera qu'à une clef n'est associée qu'une seule valeur au maximum (le dictionnaire est une fonction au sens mathématique)

Exemples d'utilisation

Omniprésent en informatique :

- Table des symboles dans un compilateur
  $\mathcal{K}ey=$ symboles, $\mathcal{V}al=$ informations (type, visibilité, ...)
- Système de fichier
  $\mathcal{K}ey=$ chemins, $\mathcal{V}al=$ emplacements disque
- Mémoïsation
  $\mathcal{K}ey=$ arguments, $\mathcal{V}al=$ résultats
- Moteur de recherche
  $\mathcal{K}ey=$ mots-clefs, $\mathcal{V}al=$ pages associées
- Représentation d'un ensemble
  $\mathcal{K}ey=$ élément, $\mathcal{V}al=\{est\_dedans\}$
- ...

Spécification algébrique d'un dictionnaire

```
type ('k,'v) dict

creer : int -> dict
inserer : dict -> 'k -> 'v -> dict
rechercher : dict -> 'k -> ('v | ERR)
supprimer : dict -> 'k -> (dict | ERR)

rechercher(creer(i), k) = ERR
rechercher(inserer(d,k,v),k) = v
rechercher(inserer(d,k,v),k') = rechercher(d,k')  k≠k'
rechercher(supprimer(d,k),k) = ERR
rechercher(supprimer(d,k),k') = rechercher(d,k')  k≠k'
```

Interface : C

```
typedef int key;
typedef char* value;

typedef struct dict_base *dict;

dict creer(int);

value rechercher(dict, key);

dict inserer(dict, key, value);

dict supprimer(dict, key);
```

Interface : OCaml

```
type ('key,'value) dict

val creer : int -> ('key,'value) dict

val rechercher :
  ('key,'value) dict -> 'key -> 'value

val inserer :
  ('key,'value) dict -> 'key -> 'value
    -> ('key,'value) dict

val supprimer :
  ('key,'value) dict -> 'key -> ('key,'value) dict
```

Implémentations

Dans ce cours, trois implémentations

- listes d'association
- arbres bien équilibrés
- table de hachage

Comparaison de la complexité en temps de chacune des fonctions

- en moyenne
- dans le pire des cas