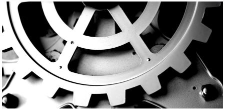
Algorithmique et structures de données
Types abstraits de données



Gaël Mahé

slides : Elise Bonzon et Gaël Mahé

Université Paris Descartes

Licence 2

Types abstraits de données

Types abstraits de données

Pourquoi des types abstraits?

- Conception initiale d'un algorithme souvent indépendante d'une implémentation particulière
  - $\Rightarrow$ Représentation des données non fixée
  - ? *Faut-il représenter les données par un tableau, un fichier, un pointeur?*
- Les données sont considérées de manière abstraite
  = **types abstraits de données**

Notations

Comment décrire les données, les opérations que l'on peut leur appliquer ainsi que leurs propriétés?

- Pour décrire un type abstrait, il faut :
  - La **signature** du type = définition syntaxique
    - Nom du domaine
    - Autres types abstraits nécessaires
    - Déclaration de certaines opérations
  - Les **propriétés** du type = définition sémantique
    i.e. que font les opérations ?
- Puis recherche des structures de données les plus adaptées.
  Elles dépendent du langage de programmation

Un exemple : les Booléens

- Signature :
  - Nom du domaine : $\mathbb{B}$
  - Autres types abstraits nécessaires : les valeurs du domaine $\{\mathrm{vrai}, \mathrm{faux}\}$
  - Opérations de base (ici, les opérateurs logiques) :
    - $\mathrm{Non} : \mathbb{B} \to \mathbb{B}$
    - $\mathrm{Et} : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$
    - $\mathrm{Ou} : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$
- Propriétés de ces fonctions de base données par les tables de vérité

Un exemple : les entiers naturels

Signature :

- Nom du domaine : $\mathbb{N}$
- Autres types abstraits nécessaires :
  - les valeurs du domaine en quantité infiniment dénombrable (il existe un processus d'énumération);
  - les Booléens.
- Opérations de base :
  - La fonction constante Zero (ou 0)
  - $\mathrm{Succ}\colon \mathbb{N} \to \mathbb{N}$
    $\mathrm{Succ}(\mathrm{Succ}(\mathrm{Succ}(0)))$?
  - $\mathrm{Pred}\colon \mathbb{N} \setminus \{0\} \to \mathbb{N}$
  - $\mathrm{Egal\_Zero}\colon \mathbb{N} \to \mathbb{B}$
  - $\mathrm{Add}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

Vocabulaire

Pour l'opération Pred, $n \neq 0$ est une **pré-condition**.

Un exemple : les entiers naturels

Signature :

- Nom du domaine : $\mathbb{N}$
- Autres types abstraits nécessaires :
  - les valeurs du domaine en quantité infiniment dénombrable (il existe un processus d'énumération);
  - les Booléens.
- Opérations de base :
  - La fonction constante Zero (ou 0)
  - $\mathrm{Succ}: \mathbb{N} \to \mathbb{N}$
    $\mathrm{Succ}(\mathrm{Succ}(\mathrm{Succ}(0)))?$ 3
  - $\mathrm{Pred}: \mathbb{N} \setminus \{0\} \to \mathbb{N}$
  - $\mathrm{Egal\_Zero}: \mathbb{N} \to \mathbb{B}$
  - $\mathrm{Add}: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

Vocabulaire

Pour l'opération Pred, $n \neq 0$ est une **pré-condition**.

Un exemple : les entiers naturels

Signature :

- Nom du domaine : $\mathbb{N}$
- Autres types abstraits nécessaires :
  - les valeurs du domaine en quantité infiniment dénombrable (il existe un processus d'énumération);
  - les Booléens.
- Opérations de base :
  - La fonction constante Zero (ou 0)
  - $\mathrm{Succ}\colon \mathbb{N} \to \mathbb{N}$
    $\mathrm{Succ}(\mathrm{Succ}(\mathrm{Succ}(0)))?$ 3
  - $\mathrm{Pred}\colon \mathbb{N} \setminus \{0\} \to \mathbb{N}$
  - $\mathrm{Egal\_Zero}\colon \mathbb{N} \to \mathbb{B}$
  - $\mathrm{Add}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

Vocabulaire

Pour l'opération Pred, $n \neq 0$ est une **pré-condition**.

Un exemple : les entiers naturels

Signature :

- Nom du domaine : $\mathbb{N}$
- Autres types abstraits nécessaires :
  - les valeurs du domaine en quantité infiniment dénombrable (il existe un processus d'énumération);
  - les Booléens.
- Opérations de base :
  - La fonction constante Zero (ou 0)
  - $\mathrm{Succ}\colon \mathbb{N} \to \mathbb{N}$
    $\mathrm{Succ}(\mathrm{Succ}(\mathrm{Succ}(0)))$? 3
  - $\mathrm{Pred}\colon \mathbb{N} \setminus \{0\} \to \mathbb{N}$
  - $\mathrm{Egal\_Zero}\colon \mathbb{N} \to \mathbb{B}$
  - $\mathrm{Add}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

Vocabulaire

Pour l'opération Pred, $n \neq 0$ est une **pré-condition**.

Un exemple : les entiers naturels

- **Propriétés**, données par récurrence. $\forall x, y \in \mathbb{N}$
  - $\mathrm{Pred}(\mathrm{Succ}(x)) = x$
  - $\mathrm{Egal\_Zero}(0) = \mathrm{Vrai}$
  - $\mathrm{Egal\_Zero}(\mathrm{Succ}(x)) = \mathrm{Faux}$
  - $\mathrm{Add}(x, 0) = x$
  - $\mathrm{Add}(x, \mathrm{Succ}(y)) = \mathrm{Succ}(\mathrm{Add}(x, y))$
  - $\mathrm{Add}(x, y) = \mathrm{SI\ Egal\_Zero}(y)$
    $\mathrm{ALORS}\ x$
    $\mathrm{SINON\ Succ}(\mathrm{Add}(x, \mathrm{Pred}(y)))$

= **définition axiomatique des propriétés**

→ Pas de contradiction entre les axiomes ? (**consistance**)
→ Sont-ils suffisants pour décrire les opérations ? (**complétude**)

Un exemple : les entiers naturels

- **Propriétés**, données par récurrence. $\forall x, y \in \mathbb{N}$
  - $\mathrm{Pred}(\mathrm{Succ}(x)) = x$
  - $\mathrm{Egal\_Zero}(0) = \mathrm{Vrai}$
  - $\mathrm{Egal\_Zero}(\mathrm{Succ}(x)) = \mathrm{Faux}$
  - $\mathrm{Add}(x, 0) = x$
  - $\mathrm{Add}(x, \mathrm{Succ}(y)) = \mathrm{Succ}(\mathrm{Add}(x, y))$
  - $\mathrm{Add}(x, y) = \mathrm{SI}\ \mathrm{Egal\_Zero}(y)$
$\mathrm{ALORS}\ x$
$\mathrm{SINON}\ \mathrm{Succ}(\mathrm{Add}(x, \mathrm{Pred}(y)))$

$=$ **définition axiomatique des propriétés**

$\rightarrow$ Pas de contradiction entre les axiomes ? (**consistance**)
$\rightarrow$ Sont-ils suffisants pour décrire les opérations ? (**complétude**)

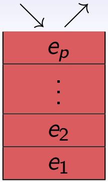
Structures séquentielles

- Types qui existent dans presque tous les langages de programmation : *listes, piles, files...*
- Organisation de données sous forme de listes linéaires
  $\rightarrow$ adapté à un traitement séquentiel...
- En général, structure dynamique :
  on peut ajouter ou supprimer des éléments

Types abstraits de données

Type Pile

- Pile : Type P
- LIFO (Last In First Out)
  - constante pilevide $\in P$
  - empiler : $E \times P \to P$
  - depiler : $P \setminus \{\text{pilevide}\} \to P$
  - sommet : $P \setminus \{\text{pilevide}\} \to E$
  - est_vide : $P \to \mathbb{B}$



Vocabulaire

- Opération dont le résultat est du type défini = **opération interne**
- Opération dont au moins un argument est du type défini et le résultat est du type pré-défini = **observateur**

Propriétés caractéristiques des piles

$\forall e \in E, \forall p \in P$

- $\text{depiler}(\text{empiler}(e, p)) = ?$

Propriétés caractéristiques des piles

$\forall e \in E, \forall p \in P$

- $\mathrm{depiler}(\mathrm{empiler}(e, p)) = p$
- $\mathrm{sommet}(\mathrm{empiler}(e, p)) = ?$

Propriétés caractéristiques des piles

$\forall e \in E, \forall p \in P$

- $\text{depiler}(\text{empiler}(e, p)) = p$
- $\text{sommet}(\text{empiler}(e, p)) = e$
- $\text{est\_vide}(\text{pilevide}) = ?$

Propriétés caractéristiques des piles

$\forall e \in E, \forall p \in P$

- $\text{depiler}(\text{empiler}(e, p)) = p$
- $\text{sommet}(\text{empiler}(e, p)) = e$
- $\text{est\_vide}(\text{pilevide}) = \text{Vrai}$
- $\text{est\_vide}(\text{empiler}(e, p)) =?$

Propriétés caractéristiques des piles

$\forall e \in E, \forall p \in P$

- $\mathrm{depiler}(\mathrm{empiler}(e, p)) = p$
- $\mathrm{sommet}(\mathrm{empiler}(e, p)) = e$
- $\mathrm{est\_vide}(\mathrm{pilevide}) = \mathrm{Vrai}$
- $\mathrm{est\_vide}(\mathrm{empiler}(e, p)) = \mathrm{Faux}$

Propriétés caractéristiques des piles

$\forall e \in E, \forall p \in P$

- $\mathrm{depiler}(\mathrm{empiler}(e, p)) = p$
- $\mathrm{sommet}(\mathrm{empiler}(e, p)) = e$
- $\mathrm{est\_vide}(\mathrm{pilevide}) = \mathrm{Vrai}$
- $\mathrm{est\_vide}(\mathrm{empiler}(e, p)) = \mathrm{Faux}$

Ces propriétés sont nécessaires et suffisantes.

Algorithme Duplique

Algorithme 1 : Algorithme Duplique

```
begin
    /* INPUT: Une pile P */
    /* OUTPUT: La pile P modifiée */
    if not(est_vide(P)) then
        e := sommet(P)
        empile(e, P)
    return P
end
```

Algorithme Premier_Dernier

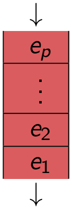
Algorithme 2 : Algorithme Premier_Dernier

```
begin
    /* INPUT: Une pile P */
    /* OUTPUT: La pile P modifiée */
    if not(est_vide(P)) then
1       e := sommet(P) ; depile(P) ; Q := pilevide
        while not(est_vide(P)) do
2           x := sommet(P); empile(x, Q) ; depile(P)
3       empile(e, P)
        while not(est_vide(Q)) do
4           x := sommet(Q); empile(x, P) ; depile(Q)
    return P
end
```

Types abstraits de données

Type File

- File : Type F
- FIFO (First In First Out)
  - constante $\text{filevide} \in F$
  - $\text{enfiler}: E \times F \to F$
  - $\text{defiler}: F \setminus \{\text{filevide}\} \to F$
  - $\text{premier}: F \setminus \{\text{filevide}\} \to E$
  - $\text{est\_vide}: F \to \mathbb{B}$



Propriétés caractéristiques des files

$\forall e \in E, \forall f \in F$

- $\mathrm{est\_vide}(f) \Rightarrow \mathrm{premier}(\mathrm{enfiler}(e, f)) = ?$

Propriétés caractéristiques des files

$\forall e \in E, \forall p \in P$

- $\mathrm{est\_vide}(f) \Rightarrow \mathrm{premier}(\mathrm{enfiler}(e, f)) = e$
- $\mathrm{Non}(\mathrm{est\_vide}(f)) \Rightarrow \mathrm{premier}(\mathrm{enfiler}(e, f)) =?$

Propriétés caractéristiques des files

$\forall e \in E, \forall p \in P$

- $\text{est\_vide}(f) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = e$
- $\text{Non}(\text{est\_vide}(f)) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = \text{premier}(f)$
- $\text{est\_vide}(f) \Rightarrow \text{defiler}(\text{enfiler}(e, f)) =?$

Propriétés caractéristiques des files

$\forall e \in E, \forall p \in P$

- $\text{est\_vide}(f) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = e$
- $\text{Non}(\text{est\_vide}(f)) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = \text{premier}(f)$
- $\text{est\_vide}(f) \Rightarrow \text{defiler}(\text{enfiler}(e, f)) = f$
- $\text{Non}(\text{est\_vide}(f)) \Rightarrow \text{defiler}(\text{enfiler}(e, f)) =?$

Propriétés caractéristiques des files

$\forall e \in E, \forall p \in P$

- $\text{est\_vide}(f) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = e$
- $\text{Non}(\text{est\_vide}(f)) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = \text{premier}(f)$
- $\text{est\_vide}(f) \Rightarrow \text{defiler}(\text{enfiler}(e, f)) = f$
- $\text{Non}(\text{est\_vide}(f)) \Rightarrow \text{defiler}(\text{enfiler}(e, f)) = \text{enfiler}(e, \text{defiler}(f))$

Propriétés caractéristiques des files

$\forall e \in E, \forall p \in P$

- $\mathrm{est\_vide}(f) \Rightarrow \mathrm{premier}(\mathrm{enfiler}(e, f)) = e$
- $\mathrm{Non}(\mathrm{est\_vide}(f)) \Rightarrow \mathrm{premier}(\mathrm{enfiler}(e, f)) = \mathrm{premier}(f)$
- $\mathrm{est\_vide}(f) \Rightarrow \mathrm{defiler}(\mathrm{enfiler}(e, f)) = f$
- $\mathrm{Non}(\mathrm{est\_vide}(f)) \Rightarrow \mathrm{defiler}(\mathrm{enfiler}(e, f)) = \mathrm{enfiler}(e, \mathrm{defiler}(f))$
- $\mathrm{est\_vide}(\mathrm{filevide}) = \mathrm{Vrai}$
- $\mathrm{est\_vide}(\mathrm{enfiler}(e, f)) = \mathrm{Faux}$

Propriétés caractéristiques des files

$\forall e \in E, \forall p \in P$

- $\text{est\_vide}(f) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = e$
- $\text{Non}(\text{est\_vide}(f)) \Rightarrow \text{premier}(\text{enfiler}(e, f)) = \text{premier}(f)$
- $\text{est\_vide}(f) \Rightarrow \text{defiler}(\text{enfiler}(e, f)) = f$
- $\text{Non}(\text{est\_vide}(f)) \Rightarrow \text{defiler}(\text{enfiler}(e, f)) = \text{enfiler}(e, \text{defiler}(f))$
- $\text{est\_vide}(\text{filevide}) = \text{Vrai}$
- $\text{est\_vide}(\text{enfiler}(e, f)) = \text{Faux}$

Ces propriétés sont nécessaires et suffisantes.

Algorithme Enfile_Prem

Algorithme 3 : Algorithme Enfile_Prem

```
begin
    /* INPUT: Une file F */
    /* OUTPUT: La file F modifiée */
    if not(est_vide(F)) then
        e := premier(F)
        defiler(F)
        enfiler(e, F)
    return F
end
```

Algorithme Renversef

Algorithme 4 : Algorithme Renversef

```
begin
    /* INPUT: Une file F */
    /* OUTPUT: La file F modifiée */
    P := pilevide
    while not(est_vide(F)) do
        e := premier(F) ; defiler(F)
        empile(e, P)
    while not(est_vide(P)) do
        e := sommet(P)
        depile(P)
        enfile(e, F)
    return F
end
```