1 Définition mathématique

1.1 Définition

Un type de données abstrait (TDA) est une formule logique conjonctive à $n+1$ paramètres ($n \in \mathbb{N}$) $\Psi(\varepsilon, \varphi_1, \ldots, \varphi_n)$.

Plus concrètement, on définit un TDA en se donnant :

- un symbole d'ensemble ε ;
- un ensemble fini de symboles fonctionnels $Fonc = \{\varphi_1, \ldots, \varphi_n\}$;
- d'un ensemble fini d'axiomes Ax portant sur les symboles de $Fonc$ et les éléments de ε :

$$Ax = \{\Phi_1(\varepsilon, \varphi_1, \ldots, \varphi_n), \ldots, \Phi_k(\varepsilon, \varphi_1, \ldots, \varphi_n)\}$$

la formule $\Psi(\varepsilon, \varphi_1, \ldots, \varphi_n)$ est alors la formule $\Phi_1 \wedge \ldots \wedge \Phi_k$.

Une instance d'un tel TDA est un couple (E, F) formé d'un ensemble E et d'un ensemble de fonctions $F = \{f_1, \ldots, f_n\}$ qui satisfait la formule $\Psi : \Psi(E, f_1, \ldots, f_n)$ est vraie, ou autrement dit qui satisfont tous les axiomes de Ax : pour tout $j \in \{1, \ldots, k\}$, $\Phi_j(E, f_1, \ldots, f_n)$ est vraie.

Dans un TDA,

- le symbole d'ensemble ε représente l'ensemble qu'on veut définir ;
- les symboles de fonctions représentent les actions qu'on a le droit de faire sur les éléments de ε, elles sont généralement partagées en deux espèces non disjointes : celles qui ont un élément de ε dans leurs paramètres et qui calculent un objet qui donnent une information partielle sur l'élément, qu'on appelle des observateurs, et celles qui calculent un « nouvel » élément de ε, qu'on appelle des constructeurs ;
- les axiomes représentent les règles qui régissent les rapports entre les éléments de ε et les différentes actions possibles, avec leurs ensembles de définition éventuels.

1.2 Exemples

La plupart des définitions de structures algébriques sont en fait des TDA.

a) Groupes

Un groupe est un couple $(G, *)$ tel que G soit un ensemble non vide, $*$ une loi de composition interne sur G qui possède un élément neutre e, est associative et tel que tout élément de G possède un inverse pour la loi $*$.

Réécrivons cette définition pour montrer que le type « groupe » est en fait un TDA :

- dans cette définition, le symbole G est un symbole d'ensemble ;
- dans cette définition, il y a trois symboles fonctionnels (trois constructeurs) qui apparaissent :
  - le symbole $\ell = *$, qui représente une fonction de $G \times G$ dans G, partout définie,
  - le symbole e, qui représente une fonction de $\{\varnothing\}$ dans G (autrement dit une constante) : par abus de notation, on notera $e = e()$,
  - le symbole $\text{inv} = {}^{-1}$, qui représente une fonction de G dans G, partout définie ;
- enfin, pour que G soit un groupe, on doit vérifier les axiomes suivants :
  - $\forall (x, y) \in G^2 \quad \ell(x, y) \in G$
  - $e \in G$
  - $\forall x \in G \quad \text{inv}(x) \in G$
  - $\forall (x, y, z) \in G^3 \quad \ell(x, \ell(y, z)) = \ell(\ell(x, y), z)$
  - $\forall x \in G \quad \ell(x, e) = x \wedge \ell(e, x) = x$
  - $\forall x \in G \quad \ell(x, \text{inv}(x)) = e \wedge \ell(\text{inv}(x), x) = e$

Le TDA « groupe » est donc la formule $\Psi(G, \ell, e, \text{inv}) =$

$$\big(\forall (x, y) \in G^2 \ \ell(x, y) \in G\big) \wedge \big(e \in G\big) \wedge \big(\forall x \in G \quad \text{inv}(x) \in G\big) \wedge \big(\forall (x, y, z) \in G^3 \ \ell(x, \ell(y, z)) = \ell(\ell(x, y), z)\big)$$
$$\wedge \big(\forall x \in G \ \ell(x, e) = x \wedge \ell(e, x) = x\big) \wedge \big(\forall x \in G \ \ell(x, \text{inv}(x)) = e \wedge \ell(\text{inv}(x), x) = e\big)$$

Un groupe est donc un ensemble G auquel on peut associer 3 fonctions telles que $\Psi(G, \ell, e, \text{inv})$ soit vraie.

Par exemple, $\Psi(\mathbb{Z}, +, 0, x \mapsto -x)$ est vraie donc $\mathbb{Z}$ est un groupe pour l'addition,
$\Psi(\mathbb{C}^*, \times, 1, x \mapsto 1/x)$ est vraie donc $\mathbb{C}^*$ est un groupe pour la multiplication,
$\Psi(GL_2(\mathbb{R}), \times, I_2, M \mapsto M^{-1})$ est vraie donc $GL_2(\mathbb{R})$ est un groupe pour la multiplication des matrices.

b) Listes

La définition des listes d'éléments de E est en fait un TDA. Soit E un ensemble, on cherche en fait à définir l'ensemble $L = \mathscr{L}(E)$ des listes d'éléments de E.

- Dans la définition des listes, le symbole L est un symbole d'ensemble ;
- il y a plusieurs symboles fonctionnels qui apparaissent :
  - le symbole Nil, qui représente la constante « liste vide » (constructeur),
  - le symbole Cons, qui représente la fonction $(a, \ell) \mapsto a :: \ell$ (constructeur),
  - le symbole T, qui représente la fonction tête (observateur),
  - le symbole Q, qui représente la fonction queue (observateur) ;
- les axiomes qui définissent les opérations possibles et leurs rapports :
  - $\text{Nil} \in L$
  - $\forall \ell \in L \quad \big(T(\ell) = \bot \iff \ell = \text{Nil}\big) \vee T(\ell) \in E$
  - $\forall \ell \in L \quad \big(Q(\ell) = \bot \iff \ell = \text{Nil}\big) \vee Q(\ell) \in L$
  - $\forall (x, \ell) \in E \times L \quad \text{Cons}(x, \ell) \neq \text{Nil}$
  - $\forall (x, \ell) \in E \times L \quad T(\text{Cons}(x, \ell)) = x$
  - $\forall (x, \ell) \in E \times L \quad Q(\text{Cons}(x, \ell)) = \ell$
  - $\forall \ell \in L \quad \ell \neq \text{Nil} \Rightarrow \text{Cons}(T(\ell), Q(\ell)) = \ell$

2 SDA en informatique

2.1 Définition

En pratique informatique, une SDA est définie par la donnée de son nom, ce qui définit un **type** et par des **opérations**, qui sont des constructeurs ou des observateurs (chacun défini par son type).

SDA = type + opérations

Les conditions d'utilisation des fonctions précédentes et les relations fonctionnelles qui les relient sont souvent décrites dans les commentaires ou dans le manuel.

Le type « liste d'éléments de `'a` » en CAML (encore !) est défini de la façon suivante :

- il se nomme `'a list`
- il est associé à des opérations :
  - `[]` est une liste, appelée liste vide,
  - Cons, notée infixement `::`, est une fonction de type `'a -> 'a list -> 'a list`, définie partout,
  - `hd` est une fonction de type `'a list -> 'a`, qui n'est définie que si la liste est non vide,
  - `tl` est une fonction de type `'a list -> 'a list`, qui n'est définie que si la liste est non vide,
- pour toute liste `li` de type `'a list` et élément `x` de type `'a`
  - `hd (x :: li) = x`
  - `tl (x :: li) = li`
  - si `li` est non vide, alors `hd l :: tl l = l`

2.2 Modules, interfaces et implémentations

Cette définition ne permet à aucun moment de savoir comment est physiquement représenté un objet de type `'a list` dans la machine et c'est justement ce qui fait son intérêt ! En effet, on peut imaginer que le code nécessaire à la gestion des listes est écrit dans un fichier séparé, qu'on appelle **un module**. La SDA est en quelque sorte le mode d'emploi du module : il dit que le module définit un nouveau type et des fonctions associées, il donne les conditions d'utilisation de ses fonctions et donne des invariants d'utilisation garantis.

Résumons : le module définit un nouveau type, l'ensemble des fonctions, appelé interface du module, donne les moyens de manipuler les objets de chaque instance et les axiomes sont les règles qui explicitent le fonctionnement des fonctions.

Tant que l'utilisateur ne cherche pas à bidouiller le cœur de la machine et qu'il se contente des fonctionnalités qui lui sont offertes, le concepteur garantit le bon fonctionnement des objets définis par la SDA. Si pour une raison ou une autre, le concepteur est amené à modifier le code interne du module mais qu'il ne change pas l'interface du module, alors tous les programmes qui utilisent le module peuvent continuer à l'utiliser sans modification !

En pratique, une SDA informatique donne des informations supplémentaires, comme une fonction taille sur les objets définis par la SDA et les complexités associées des fonctions de l'interface.

En Caml-light, les fonctions de manipulation des listes sont dans un module nommé `List` (original, non ?) dont l'interface est donnée en annexe. Il n'y a nulle trace du code réellement utilisé pour définir ces fonctions, qui est dans un autre fichier (donné en annexe aussi).

Un TDA est donc en général associé à un module, lui-même défini en deux parties : une partie interface et une partie implémentation, qui sont en général définies dans deux fichiers différents.

2.3 SDA permanentes ou impératives

En informatique, on distingue encore les SDA permanentes (ou immuables) des SDA impératives (ou modifiables, ou mutables) : les premières sont purement fonctionnelles, les opérations construisent de nouvelles structures (exemple typique : les listes, les arbres), alors que les secondes sont modifiables en interne, leurs opérations permettant de les modifier en place ou d'en construire d'autres (exemple typique : les tableaux).