B Langage OCaml

La présente annexe liste limitativement les éléments du langage OCaml (version 4 ou supérieure) dont la connaissance, selon les modalités de chaque sous-section, est exigible des étudiants. Aucun concept sous-jacent n'est exigible au titre de la présente annexe.

B.1 Traits et éléments techniques à connaître

Les éléments et notations suivants du langage OCaml doivent pouvoir être compris et utilisés par les étudiants sans faire l'objet d'un rappel, y compris lorsqu'ils n'ont pas accès à un ordinateur.

Traits généraux

— Typage statique, inférence des types par le compilateur. Idée naïve du polymorphisme.
— Passage par valeur.
— Portée lexicale : lorsqu'une définition utilise une variable globale, c'est la valeur de cette variable au moment de la définition qui est prise en compte.
— Curryfication des fonctions. Fonction d'ordre supérieur.
— Gestion automatique de la mémoire.
— Les retours à la ligne et l'indentation ne sont pas signifiants mais sont nécessaires pour la lisibilité du code.

Définitions et types de base

— `let`, `let rec` (pour des fonctions), `let rec` … `and` …, `fun` x y `->` e.
— `let` v `=` e `in` e', `let rec` f x `=` e `in` e'.
— Expression conditionnelle `if` e `then` e_V `else` e_F.
— Types de base : `int` et les opérateurs `+`, `-`, `*`, `/`, l'opérateur `mod` quand toutes les grandeurs sont positives ; exception `Division_by_zero` ; `float` et les opérateurs `+.`, `-.`, `*.`, `/.` ; `bool`, les constantes `true` et `false` et les opérateurs `not`, `&&`, `||` (y compris évaluation paresseuse). Entiers et flottants sont sujets aux dépassements de capacité.
— Comparaisons sur les types de base : `=`, `<>`, `<`, `>`, `<=`, `>=`.
— Types `char` et `string` ; `'`x`'` quand x est un caractère imprimable, `"`x`"` quand x est constituée de caractères imprimables, `String.length`, s`.[`i`]`, opérateur `^`. Existence d'une relation d'ordre total sur `char`. Immuabilité des chaînes.

Types structurés

— n-uplets ; non-nécessité d'un `match` pour récupérer les valeurs d'un n-uplet.
— Listes : type `'a list`, constructeurs `[]` et `::`, notation `[`x`;` y`;` z`]` ; opérateur `@` (y compris sa complexité) ; `List.length`. Motifs de filtrage associés.
— Tableaux : type `'a array`, notations `[|`…`|]`, t`.(`i`)`, t`.(`i`)` `<-` v ; fonctions `length`, `make`, et `copy` (y compris le caractère superficiel de cette copie) du module `Array`.
— Type `'a option`.
— Déclaration de type, y compris polymorphe.
— Types énumérés (ou sommes, ou unions), récursifs ou non ; les constructeurs commencent par une majuscule, contrairement aux identifiants. Motifs de filtrage associés.
— Filtrage : `match` e `with` p_0 `->` v_0 `|` p_1 `->` v_1 … ; les motifs ne doivent pas comporter de variable utilisée antérieurement ni deux fois la même variable ; motifs plus ou moins généraux, notation `_`, importance de l'ordre des motifs quand ils ont des instances communes.

Programmation impérative

— Absence d'instruction ; la programmation impérative est mise en œuvre par des expressions impures ; `unit`, `()`.
— Références : type `'a ref`, notations `ref`, `!`, `:=`. Les références doivent être utilisées à bon escient.
— Séquence `;`. La séquence intervient entre deux expressions.
— Boucle `while` c `do` b `done` ; boucle `for` v `=` d `to` f `do` b `done`.

Divers

- Usage de `begin` … `end`.
- `print_int`, `print_float`, `print_string`, `read_int`, `read_float`, `read_line`.
- Exceptions : levée et filtrage d'exceptions existantes avec `raise`, `try` … `with` … ; dans les cas irrattrapables, on peut utiliser `failwith`.
- Utilisation d'un module : notation $M.f$. Les noms des modules commencent par une majuscule.
- Syntaxe des commentaires, à l'exclusion de la nécessité d'équilibrer les délimiteurs dans un commentaire.

B.2 Éléments techniques devant être reconnus et utilisables après rappel

Les éléments suivants du langage OCaml doivent pouvoir être utilisés par les étudiants pour écrire des programmes dès lors qu'ils ont fait l'objet d'un rappel et que la documentation correspondante est fournie.

Traits divers

- Types de base : opérateur `mod` avec opérandes de signes quelconques, opérateur `**`.
- Types enregistrements mutables ou non, notation {c_0 : t_0; c_1 : t_1; …}, {c_0 : t_0; `mutable` c_1 : t_1; …}; leurs valeurs, notations {c_0 = v_0; c_1 = v_1; …}, $e.c$, $e.c$ <- v.
- Fonctions de conversion entre types de base.
- Listes : fonctions `mem`, `exists`, `for_all`, `filter`, `map`, `iter` du module `List`.
- Tableaux : fonctions `make_matrix`, `init`, `mem`, `exists`, `for_all`, `map` et `iter` du module `Array`.
- Types mutuellement récursifs.
- Filtrage : plusieurs motifs peuvent être rassemblés s'ils comportent exactement les mêmes variables. Notation `function` p_0 -> v_0 | p_1 -> v_1 ….
- Boucle `for` v = f `downto` d `do` b `done`.
- Piles et files mutables : fonctions `create`, `is_empty`, `push` et `pop` des modules `Queue` et `Stack` ainsi que l'exception `Empty`.
- Dictionnaires mutables réalisés par tables de hachage sans liaison multiple ni randomisation par le module `Hashtbl` : fonctions `create`, `add`, `remove`, `mem`, `find` (y compris levée de `Not_found`), `find_opt`, `iter`.
- `Sys.argv`.
- Utilisation de `ocamlc` ou `ocamlopt` pour compiler un fichier dépendant uniquement de la bibliothèque standard.

Gestions des ressources de la machine

- Gestion de fichiers : fonctions `open_in`, `open_out`, `close_in`, `close_out`, `input_line`, `output_string`.
- Fils d'exécution : recours au module `Thread`, fonctions `Thread.create`, `Thread.join`.
- Mutex : recours au module `Mutex`, fonctions `Mutex.create`, `Mutex.lock`, `Mutex.unlock`.