Lazi / Les types informatiques / Présentation

Emmanuel Chantréau

16 octobre 2016

Table des matières

1 Introduction

Ce texte présente les types du point de vue du langage informatique Lazi. Contrairement aux langages classiques le cœur de Lazi n'est pas son système de types, mais plutôt l'utilisation de fondations mathématiques spéciales. Néanmoins le système de types a un rôle important dans la définition des fondations mathématiques, il apporte de plus des avancées que je trouve conséquentes aux systèmes classiques.

2 La méthode

Avec de l'expérience en conception de systèmes nous savons que souvent les premières moutures sont défectueuses ou peuvent être largement améliorées. Pour beaucoup nous avons découvert admiratif le monde de la programmation objet, nous avons appris à l'utiliser et à l'apprécier.

Nous allons voir ici qu'il s'agit d'une première mouture que l'on peut largement améliorer. Mais pour cela nous devons nous munir d'un peu d'énergie pour lutter contre nos habitude

et remettre les choses à plat. Nous sommes tous sujet à l'effet d'attitude (appelé aussi effet d'Einstellung).

Pour remettre les choses à plat il va nous falloir donc un temps oublier cette (déjà merveilleuse) version 1 du monde de la programmation objet et n'utiliser que notre bon sens et notre logique, dans le cas contraire on ressentirait un malaise fasse au décalages dans les paradigmes.

Nous allons étudier la question suivante : "Comment organiser un système de traitement de l'information ?". Nous connaissons déjà de tels systèmes : l'esprit humain ou encore les ordinateurs.

Les systèmes de traitement de l'information sophistiqués tendent à refléter l'univers extérieur à l'intérieur du système. Ainsi si nous regardons un chat, l'image inversée du chat se forme sur nos cônes rétiniens, ainsi que dans la partie visuelle de notre cerveau, pour finalement se refléter sous forme symbolique dans notre conscience. C'est pourquoi nous allons chercher à répondre à la question "Comment représenter les choses de notre univers ?"

3 Les choses, Les types, les instances, les objet

3.1 Quoi représenter ?

Définition : Une chose désigne un objet, une idée ou une abstraction quelconque (c'est le sens du dictionnaire).

3.2 Comment représenter ?

3.2.1 Les choses et les types

Pour se représenter quelque **chose**, il faut déjà avoir un moyen de l'*identifier*. Par exemple si on n'a pas de moyen de parler de la bicyclette du voisin sans parler en même temps de la sienne on aboutit alors à un système inopérant.

D'autre part nous remarquons que les choses sont divisées en types, par exemple les choses de type "chat", les choses de type "avion" etc.

Pour chacun des types, nous avons besoin au minimum de 2 informations :
— Reconnaître si une chose est de ce type. On dira alors que c'est une **instance** du type.
— Trouver toutes les informations disponibles sur cette chose.
Par exemple si "Victor" est pour vous le chat du voisin, à partir du nom "Victor" vous reconnaissez que cette chose est de type "chat". De plus, à partir de l'identifiant "Victor" vous pouvez retrouver dans votre esprit les divers informations au sujet de ce chat.

3.2.2 Les types de types

Remarquons qu'un système de traitement de l'information doit pouvoir représenter toutes sortes de choses, même les concepts comme le nombre 2 ou le concept "type chat". Par exemple le type "entier naturel" a pour instance 2 ou 18. Les types, comme toutes choses, obéissent eux-mêmes aux règles de représentation que nous venons de voir. Il convient donc de les représenter en trouvant des types de types. Par exemple le type "chat" et le type "cheval" peuvent être reliés au type "mamifère".

3.2.3 Les types plats et structurés

Il faut ici se dissocier légèrement de l'habitude du langage courant : nous structurons notre représentation en type et sous-types (par exemple animal/mammifère/espèce chat/Victor), mais tous ces types sont "*plats*", c'est à dire que leurs instances identifient des choses qui ne sont pas des types. Par exemples on dira "Les chats sont de type mammifère" et non "l'espèce chat est de type mammifère". Cette habitude dans notre langue courante est pratique car cela nous permet de parler de types et sous-types sans pour autant définir une hiérarchie stricte.
Mais cela ne conviendra pas à un système de représentation simple, car par exemple "Les chats sont des mammifères." est une expression mathématique non simple ("pour tout x, si x est de type chat alors x est de type mammifère"). Il nous sera donc préférable que le type mammifère ait pour instance des espèces et non des individus. Ainsi dans notre systèmes de types nous dirons "le type chat est de type mammifère" (et non "les chats sont de type mammifère").

Quand un type a pour instance d'autre types nous dirons que c'est un type *structuré*, sinon nous dirons que c'est un type *plat*. Par exemple le type chat sera plat car ses instances sont des choses (comme Victor) qui ne sont pas des types, par contre le type mammifère sera structuré car ses instances seront des types (les espèces).

3.2.4 Tout représenter par le même système

Nous avons vu que les types sont des choses soumises eux-mêmes au besoin de représentation. Nous avons besoin de simplicité, et donc de pouvoir utiliser le même système de représentation pour les types que pour le reste. Ceci est le cas dans le langage courant, mais pas en informatique classique où si on veut définir une fonction prenant en argument un type il faudra passer par un autre système de représentation (par exemple les templates en C++). Pour un langage à la pointe du progrès comme Haskell on trouve même une troisième forme de représentation avec les "kind".

Remarque : Pour arriver à cela il nous faudra assouplir la contrainte des langages classiques qui obligent à ce que chaque entité du langage de base ait un type (nous pourrons néanmoins imposer cette contrainte pour certains code sources).

3.2.5 L'objet d'un type et d'une instance

"Victor" peut être une instance du type "chat". Pour retrouver les informations sur Victor nous devons utiliser la fonction du type chat "object" qui associe les informations à l'instance. On peut voir le résultat de la fonction "objet" comme ce qui se rapproche le plus de l'objet réel. Si le poids au milligramme près de Victor nous est inutile alors nous n'avons pas cette information accessible par la fonction "object", il en sera de même en informatique où la fonction "object" ne fournira que les informations nécessaires.

3.2.6 En conclusion

Nous pouvons voir que la représentation des choses exposées ici est très similaire à celle de l'esprit humain et donc naturellement facile à pratiquer. Par exemple si nous avons un chat devant nous, nous avons accès à l'objet. Mais pour réfléchir à la nature du chat nous avons besoin de construire un système de types structuré (vivant, animal, mammifère etc) dont il sera l'objet terminal.

4 En pratique avec Lazi

4.1 Informatique et mathématique

Pour bien comprendre ce qui suit je pense qu'il est important de voir que l'intégration des mathématiques au cœur de Lazi a une incidence structurelle forte. Par exemple la vérification des types est modulaire en Lazi et est vu comme de la vérification et recherche de preuves. Il ne faudrait donc pas que le lecteur s'imagine la définition d'un langage amenant à la conception d'un gros compilateur classique, mais plutôt à la définition informatique de la base d'un compilateur modulaire utilisant la représentation mathématique de lui-même.

4.2 La base

À partir de "if" nous définissons des structures élémentaires (paire, liste, mot, dictionnaire etc).

Un dictionnaire est une liste de paires où le premier élément des paires est un mot. On peut donc l'utiliser pour associer des mots à des choses.

4.3 Un type Lazi

Un type Lazi est un dictionnaire comprenant les entrées suivantes :

domain : Fonction retournant 1 pour les choses qui sont des instances du type.

object : Retourne toute les informations sur la chose correspondant à l'instance.

isType : Retourne 1 si l'instance est celle d'un type (si l'objet correspondant à l'instance est un type).

Remarque : Pour qui a l'habitude de ne manipuler que des fonctions ayant un type défini et de n'utiliser que des mathématiques avec tiers exclu, la fonction "domain" peut paraître bizarre. Lazi ne garantit pas que le résultat soit 0 ou 1, mais on pourra en général prouver que pour des choses d'un certain type le résultat est 0 ou 1.

4.4 Exemples de types Lazi

Le texte "définition" comporte de nombreuses définitions de types, d'abord des simples comme des plus complexes.

5 Comparaison avec l'informatique classique

5.1 En général

Les constructions usuelles sont réalisées facilement en Lazi du fait que les types sont complètement intégrés aux langage. Par exemple :

— types paramétrés
— héritage
— surcharge

Il devient aussi facile de définir des types impossibles à définir classiquement et pourtant utiles (au moins dans les fondations Lazi) comme :

— Une liste où le type du premier élément est donné en paramètre et où le type de l'élément suivant dépend de l'élément précédant.
— Un type liste où les éléments sont de type quelconque, par exemple Haskell doit alors définir une pléthore de types "tuples".

Il est aussi facile de manipuler les types comme tout autre chose, par exemple on pourrait définir une fonction "myType" etc.

Les langages classiques imposent qu'une chose ait un type défini, autrement dit à chaque instance est associé un unique type. En Lazi les types définissent un ensemble d'instances, mais une chose peut être l'instance de plusieurs types (par exemple une liste binaire peut être une instance du type "mot", du type "nombre" ou encore du type "liste binaire"). Si on compare l'informatique classique à la vie, il faudrait imaginer que chaque chose de notre vie ait une étiquette marquée de son type et que si on voulait utiliser un boulon en tant que remblai il fallait utiliser une machine (une fonction de conversion) pour transformer le boulon en type "remblai". On peut serte éloigner l'informatique du fonctionnement du réel, mais il me semble de bon sens de représenter au plus près le réel quand on veut concevoir un système de traitement d'informations évolué.

5.2 Comparaison avec les objets C++

On peut voir une classe C++ comme un type Lazi où l'instance doit comporter l'information sur son type. On peut voir l'objet d'une classe comme l'objet en Lazi où la fonction objet serait prédéterminée par le C++ à partir de la hiérarchie des classes. Par exemple il serait impensable en C++ de définir un héritage différent.

6 Conclusion

Comparé au langage informatique classique, Lazi est bien plus exigeant du point du vue de la logique, mais cet exigence placée sur un élément fondamental permet une grande souplesse aux autres niveaux.