Lazi / Les types informatiques / La base

Emmanuel Chantréau

24 octobre 2016

Table des matières

1 Introduction

Ce texte fait suite à la présentation des types Lazi. Il explique les bases des types définis dans les fondations de Lazi.

2 Les types en pratique

Un type est un dictionnaire contenant les entrées "domain", "object" ainsi que d'autres moins importantes ("equal", "isType" etc). L'appel des fonctions dans les dictionnaires peut se faire comme dans les objets en C++, c'est à dire que le premier argument "this" permet l'accès au dictionnaire entier.

3 La chaîne d'instances (*incha*)

Le besoin d'identification : En informatique comme dans notre esprit, nous utilisons des choses faciles à manipuler et dont la structure nous est apparente. Par exemple si nous réfléchissons à un projet de vacances alors le mot vacances est dans notre esprit ainsi que les éléments de la structure des vacances (période de temps, lieu, loisir etc). Si nous réfléchissions à ce projet sans identifiant ni structure alors notre esprit ferait plutôt des rêveries sur des vacances passées ou imaginaires.

Comment identifier : Pour identifier une chose nous partirons de son type le plus général pour aller vers le particulier. Par exemple pour identifier le chat du voisin nous utiliserons "animal/mammifère/félin/chat/Victor". "Victor" est une instance du type chat, le type chat est l'objet de l'instance "chat" du type félin, le type félin est l'objet de l'instance "félin" du type mammifère etc , "animal" est une instance du type "topT" qui regroupe les types de tête. Nous dirons que "animal/mammifère/félin/chat/Victor" est une chaîne d'instances de topT (car la

première instance est une instance de topT). “chat/Victor” est donc une chaîne d’instance (ou “*incha*” pour “**in**stance **cha**in”) du type félin (attention : du type et non de type).

Par exemple si nous voulons manipuler le nombre 8 en tant qu’entier naturel nous utiliserons “integer/8” et en tant que réel nous utiliserons “real/8”.

Les instances en informatique : Nous donnons des exemple en langage courant avec des instances qui sont des mots, en informatique nous pourrons avoir toute sorte de structures comme des listes binaires, des arborescences, des mots etc. Comme une instance doit être reconnue par la fonction domain, cela impose des contraintes, par exemple il n’est pas possible de reconnaître une fonction, et donc les instances ne peuvent être ou contenir des fonctions.

4 Tester l’égalité et les inchas

Il est pratiquement toujours utile de tester l’égalité entre incha, c’est pourquoi cette fonction est toujours définie dans les types définies pour Lazi.

5 topT

Le type topT n’a pas d’autre vocation que d’être le type “top”. Pour ses instances nous utilisons des noms, l’objet de l’instance est le type associé au nom. Par exemple si nous avons un type integerT qui n’a pas de type parent (à part topT) alors nous associons dans topT le mot “integer” au type integerT. topT contient un dictionnaire des associations, les instances de topT sont donc les mots en entrée du dictionnaire.

On peut voir topT comme le contexte de plus haut niveau, ou encore le cœur de Lazi, car il définit les types déduction et assertion. Ainsi changer de fondation mathématique (par exemple pour utiliser une extension) revient à changer topT.

6 inchaT t

Le type `inchaT t` a pour argument un type et pour instance les inchas de t. L’objet d’une instance de `inchaT t` est l’objet final, c’est à dire que l’on calcul l’objet de la première instance (qui est de type t), puis si l’incha (qui est une liste) contient un autre élément on considère l’objet calculé comme un type et on répète jusqu’à la fin de la liste.

Tous les types possèdent l’entrée “isType” qui indique si l’objet de l’instance est un type. Nous dirons qu’une incha est *terminale* si l’objet de l’incha n’est pas un type.

7 Types paramétrés et inchas

pairOfT : Si nous voulons définir un type “paire” où les deux éléments de la paire ont un type prédéfini, alors nous devons définir la fonction type pairOfT prenant en argument les types des deux éléments. Ainsi si t1 et t2 sont deux types alors `pairOfT t1 t2` sera le type d’une paire où le premier élément est de type t1 et le deuxième de type t2.

pairOfT ou plus ? Nous pouvons utiliser pairOfT ainsi sans autre développement et sans problème. Mais si nous voulons effectuer des traitements de haut niveau, nous aurons alors besoin de “réfléchir” sur le type “pairOfT t1 t2”, c’est à dire que nous aurons besoin de le représenter sous forme d’incha de topT. Pour cela nous aurons besoin d’un type pairT qui a pour instance les inchas correspondant aux types t1 et t2 et pour objet un “pairOfT t1 t2”. Nous voyons donc que pour effectuer des traitements de haut niveau nous aurons besoin de représenter les types par des inchas de topT.

Avec un tel système de représentation des types nous pourrons construire des types de haut niveau, par exemple un type de base qu’est la structure du langage C est construite en Lazi comme un conteneur. On peut par exemple définir une fonction qui transforme le type d’une structure en remplaçant les types entiers en types réels.

Remarque : Le type “pairOfT” n’est pas défini ainsi dans les fondations, nous utilisons une forme proche mais qui permet de factoriser les parties communes des conteneurs.

8 Les types conteneurs

Informellement se sont des types servant à stocker des éléments sous une certaine forme (par exemple les paires, les listes, les dictionnaires). Les types conteneurs devront fournir deux fonctions :

toList Retourne la liste des éléments.

fromList Change les éléments à partir de la liste en argument.

Les types des éléments d'un conteneur sera paramétré par une liste des types. Nous pourrons utiliser une liste finie, ce qui déterminera aussi la taille du conteneur, ou une liste infinie d'un type répété pour ne pas imposer de taille et fournir un type unique d'instances.

D'autre part l'objet d'un type conteneur pourra être soit la structure contenant les objets, soit le type d'un conteneur similaire dont les types des instances sont données par les objets (qui doivent donc être dans ce cas des types) du premier conteneur.

Par exemple nous pourrons construire un type de liste où les instances représentent des types et où l'objet est le type d'une liste de taille identique où les instances de la liste sont les objets de la première liste. L'incha d'un triplet de type (word, formula, integer) aura la forme `$L[ 'liste' , $P[ 1 , $P[ 1 , $L[$L['word'],$L[ 'formula'] , $L['integer']] ] ]`, le premier "1" est le flag indiquant que l'objet doit être un type (de liste), le deuxième "1" indique que la liste est finie.

Si nous voulons construire le type des listes finies nous utiliserons l'incha `$L[ 'liste' , $P[ 1 , $P[ 0 , $L[ 'typeT' ] ] ]`, Le 0 indique que la liste est une liste infinie répétant le type représenté par l'incha $L['typeT']. typeT est le type des inchas qui ont pour objet des types.