Lazi : présentation

Emmanuel Chantréau

1er septembre 2016

Table des matières

0.1 Introduction

Ce texte présente Lazi sous différents angles : qu'est-ce que Lazi, les motivations, les conséquences, ce qui est déjà fait etc.

Le document meta-doc informe des différents documents existant au sujet de Lazi.

0.2 Lazi en résumé

Lazi est une nouvelle fondation mathématique de laquelle est dérivé un langage informatique. Les deux sont largement en dehors de ce que l'on connaît déjà. Par exemple les fondations mathématiques ne reposent pas sur le calcul des prédicats ni les ensembles. Quand au langage informatique dérivé il a lui aussi des propriétés spéciales qui permettent par exemple d'arriver très vite à l'expression de types paramétrés évolués.

Chapitre 1

Considérations épistémologiques

1.1 Introduction

Les personnes qui ne s'intéressent pas à ce sujet pourront passer ce chapitre, les seuls dommages seront probablement une compréhension moins profonde de Lazi. Pourquoi faire de l'épistémologie pour présenter Lazi ? L'activité scientifique se produit dans un certain cadre culturel. Comme l'explique le livre "La Structure des révolutions scientifiques" ce cadre est déterminant dans la pratique scientifique et certaines avancées scientifiques passent inévitablement par la remise en cause d'éléments de ce cadre.

La motivation à découvrir Lazi provient d'une remise en cause d'un des paradigmes couramment adopté.

1.2 La frontière du rasoir d'Ockham

À quoi s'applique le principe du rasoir d'Ockham ? Nous sommes habitués à ce que les représentations de certains phénomènes (comme la matière) se décomposent en éléments simples et uniques. Ainsi nous n'avons qu'une représentation de la matière (la vision particulaire ou ondulatoire sont deux sous-représentations complémentaires pour former une représentation de loin la plus juste connue). Ces principes de décomposition en éléments simples (principe du rasoir d'Ockham) et d'unicité de la représentation est bien établie mais les limites de son application ne l'est pas.

Existe-t-il une propriété "être régi par le principe du rasoir d'Ockham et celui de l'unicité de la représentation ?" est une question ouverte. Nous allons détailler deux réponses possibles.

Le "relata" : Le texte de Gil Jannes "Some comments on "The Mathematical Universe"" défend l'existence d'une propriété "relata" qui détermine si une structure mathématique existe ou non dans l'univers physique. On pourrait croire que cette vision défend l'idée que la notion d'existence est relative à un système et qu'ainsi dans notre univers la notion d'existence est différente de la notion mathématique ou d'une autre notion plus large. Mais la notion de relata est plutôt celle d'une unique propriété d'existence (celle de la réalité physique), principe premier et inaccessible à la science. Dans cette vision les éléments en dehors (comme par exemple le nombre 2, ou encore les fondations mathématiques), n'étant pas "réels", n'obéissent pas forcément aux principes du rasoir d'Ockham ou de l'unicité de la représentation. Cette vision postule donc l'existence d'une propriété (le relata) non mathématique déterminant ce qui existe, propriété elle même non existante au sens de la réalité physique.

L'existence comme notion relative : Cette vision défend l'idée que les principes utilisés en science (rasoir d'Ockham, unicité de la représentation) sont applicables à la notion la plus large possible de l'existence. Les mathématiques nous fournissant la structure la plus ouverte pour représenter toute chose, c'est la notion d'existence mathématique qui nous est la plus large possible, et cette notion d'existence n'a comme seule frontière que la cohérence logique. Ainsi, dans cette vision on peut prétendre "tout est mathématique" dans le sens ou "tout" (ce qui existe) est déterminé par l'existence mathématique. Notre univers étant alors une partie de ce tout, est lui aussi un objet mathématique. Cette dernière assertion est défendue par exemple par Max Tegmark.

On pourrait penser que ces différentes visions sont sans conséquences pratiques. Mais elles déterminent ce qui nous parait important d'étudier. Dans la deuxième vision les fondations mathématiques ont une importance capitales, car trouver les "bonnes fondations" est encore plus fructueux que de trouver une représentation correcte de la matière. Par contre la première vision place les fondations mathématiques au rang d'outils de facture humaine, pratique mais non sujet au principe du rasoir d'Ockham ou de l'unicité de la représentation.

Pour des raisons trop longues et profondes à expliquer, je ne chercherai pas à argumenter ici en faveur de la seconde position, mais c'est celle motivant la recherche de fondations mathématiques bien plus simples que celle déjà découvertes.

Il se trouve que Lazi répond tout à fait au principe du rasoir d'Ockham car cette fondation est d'une esthétique et simplicité dépassant de loin les fondations déjà trouvées.

1.3 De l'outil au centre

La théorie des ensembles place (suivant la vision "relata" définie plus haut) les mathématiques comme outil d'étude de certains objets (les ensembles) permettant de représenter toute sorte de choses.

Au contraire le centre d'intérêt de Lazi est Lazi lui-même ! Ce principe, qui peut sembler d'un stupide nombrilisme, est en fait impliqué par la vision relative de la notion d'existence, car les mathématiques sont alors la structure originelle de tout autre.

Cela se traduit en pratique par des mathématiques minimales permettant de se représenter elles-même et de prouver la validité d'extensions mathématiques. Une extension mathématique est une représentation d'une fondation mathématique ayant des symboles et règles de déductions supplémentaires (nous détaillerons plus loin). On a en Lazi "mathématiques = méta-mathématiques", c'est à dire que la preuve d'une assertion peut passer par une preuve dans une extension de Lazi. Cette propriété est unique pour des fondations mathématiques.
Par exemple on peut définir une extension de la théorie des ensembles et prouver sa validité. La théorie des ensembles ne permet pas de prouver une assertion par une représentation de preuve à l'intérieur de la théorie des ensembles car les symboles mathématiques (comme "$\forall$") ne sont pas des objets mathématiques en théorie des ensembles.

Chapitre 2

Quelques caractéristiques spécifiques à Lazi

2.1 Introduction

Les caractéristiques listées ci-dessous peuvent paraître improbables voir impossibles pour qui est habitué aux fondations classiques. J'invite le lecteur rebuté, à vérifier par lui-même en lisant le texte de définition de Lazi.

2.2 Minimalité

Contrairement aux fondations mathématiques usuelles Lazi demande peu de bases et très rapidement les fondations se font dans le langage Lazi lui-même. Par exemple les règles de déduction comportent un maximum de 3 variables fixées (x,y et z). Les fondations comprennent une définition complète en Lazi de ce qu'est une preuve Lazi, ce qui permet de fournir des fondations définies très précisément. Les 6 règles de base sont toutes très simples. Les règles supplémentaires sont traduisibles (la démonstration n'est pas encore faite), c'est à dire qu'une preuve prouvant une formule F et utilisant les règles supplémentaires peut être traduite en une preuve de F utilisant uniquement les 6 règles de base.

Lazi ne comporte aucun axiome.

Les 6 règles de base n'incluent pas la notion de variable (qui est une notion bien trop complexe pour une base si petite). À la place nous avons un mots clé "distribute" et la règle `distribute x y z = (x z) (y z)` qui permet de placer les arguments d'une fonction dans le corps de celle-ci.

2.3 Logique du premier ordre = logique du deuxième ordre

La logique d'ordre supérieur permet de représenter en mathématiques les entités mathématiques (comme les propriétés).

En Lazi la syntaxe et la grammaire du langage est réduite au minimum (les formules sont les mots ou formules assemblés par deux, à la lambda calcul mais juste la partie application), il n'y a pas différents types de mots ni mêmes de variables. Par exemple "if" est un mot clé du langage, mais "if if" est une formule correcte de Lazi. Il en découle que l'on peut tout représenter. Le lecteur pourrait penser que Lazi est d'une permissivité absurde, mais il faut plutôt le voir comme une "micro-logique".

Dans notre langue courante nous pouvons dire "La phrase 'Je mange du pain.' est vraie." et l'interlocuteur comprend que le locuteur mange du pain. En théorie des ensemble nous pouvons représenter une formule de la théorie des ensembles mais nous ne pouvons traduire cette représentation en une vérité car nous ne pouvons pas utiliser les propriété, opérateurs logiques ou quantificateur comme objet. Cette limite n'existe pas en Lazi, ce qui nous conduit à la deuxième caractéristique.

2.4 mathématique = métamathématique

Lazi est une fondation minimale permettant de prouver la validité d'extensions. On peut alors prouver qu'une preuve dans une extension est valide et en déduire que la traduction de la conclusion de la preuve est vrai. Cela permet de fournir des fondations très petites, par exemples les quantificateurs ne sont pas définis dans les fondations mais par des extensions.

Il n'y a pas besoin de distinguer mathématique et métamathématique car une vérité démontrée en Lazi sur les mathématiques peut être directement utilisée.

Par opposition, regardons la théorie des ensembles. Imaginons que nous ayons un logiciel qui vérifie les preuves dans ces fondations et que l'on définie par exemple des fondations utilisant des axiomes et symboles supplémentaires (comme l'analyse non standard). On pourra démontrer que l'extension ne produit pas d'inconsistance et que toute vérité prouvée dans le langage de base est vraie dans la théorie des ensembles. Néanmoins nous ne pourrons pas utiliser cette extension dans le logiciel de vérification des preuves. Cette limitation n'existe pas en Lazi.

2.5 Lazi comme base d'un langage informatique

Sa base est composée de 6 règles de déductions dont 5 sont calculatoires (par exemple `if 1 x y = x`). Cela permet de définir des fonctions (sans domaine) et de les calculer.

À partir de cette base nous pouvons très rapidement définir un système de types puissant (les types sont paramétrables et peuvent être eux-même manipulés par les fonctions Lazi). Cela est nécessaire à la définition de Lazi et s'y trouve donc intégré.

Très rapidement la définition des fondations se fait grâce à des fonctions Lazi, il en découle que la définition de Lazi est directement utilisable (grâce à un interpréteur) pour, par exemple, vérifier la validité d'une preuve.

On arrive à un langage proche d' Haskell mais avec un système de types bien plus puissant.

Du fait de la nature mathématique de Lazi on peut prouver des assertions sur les types et les fonctions pour par exemple vérifier qu'une fonction correspond à son type déclaré. Pour les cas simples on peut utiliser un générateur de preuve automatique (c'est ce qui est fait implicitement par les compilateurs quand ils vérifient les types).

Les compilateurs usuels (ghc, g++ etc) sont des gros blocs logiciels garantissant des relations entre les sources et l'exécutable correspondant. Utiliser directement un langage mathématique permet de réduire énormément la partie logiciel garantissant des propriétés. On peut alors utiliser Lazi comme un outils de construction modulaire de compilateurs. Il en résulte l'absence des compromis usuels puisque le développeur peut descendre au niveau nécessaire pour produire l'exécutable tout en ayant les garanties habituelles. Il en résulte un compilateur bien plus modulaire que ceux existant actuellement.

On peut alors envisager diverses applications, par exemple il devient possible d'intégrer des modules de preuves automatiques pour faciliter la vérification de la qualité des logiciels produits (par exemple prouver qu'il n'y a pas de boucle infinie ou encore calculer le temps d'exécution.). De plus il est possible d'accompagner un module de la preuve de son innocuité, ce qui permet de l'intégrer automatiquement.

Chapitre 3

Exemples de ce qu'apporte Lazi

3.1 Introduction

Cette partie fournit des exemples de ce que peuvent être les avancées obtenues grâce à Lazi. Cela s'adresse surtout aux personnes voyant les mathématiques et l'informatique comme de simples outils (voir le chapître "Considérations épistémologiques").

3.2 En mathématique

3.2.1 Fournir un cadre souple

La théorie des catégories a montré que les fondations classiquement utilisées sont trop étroites. Lazi est une base très petite mais où les extensions font parti intégrantes de l'activité mathématique.

La structure de Lazi fait que l'on rencontrera difficilement un cadre limitant les représentations et que l'on pourra créer les extensions permettant de pratiquer la forme de mathématique correspondant le mieux à son activité.

3.2.2 Favoriser l'utilisation de l'informatique

L'informatique peut être utilisée pour vérifier les preuves, répertorier les théorèmes, aider à la recherche de preuve. Voir par exemple le projet de Vladimir Voedovski.

Lazi offre une base solide pour représenter les notations nécessaires aux différents champs mathématiques (avec leurs traductions). Avec Lazi, ajouter des notations et les preuves de leur validité fait partie du processus normal des mathématiques. Il permet donc d'offrir un cadre simple et puissant pour l'utilisation de l'outil informatique en mathématique.

3.3 En informatique

3.3.1 Des types sans limite

Les langages informatiques actuels sont divisés en deux parties : le langage traitant des données et celui traitant des types. Un des langages des plus modernes qu'est Haskell a même un troisième niveau avec les kinds.

En intégrant les types au langage de base nous pouvons utiliser toute la puissance du langage de base pour les types, c'est ce que fait Lazi. Pour plus de détail sur le système de types utilisé dans la définition de Lazi voir le texte "Lazi : Les types informatiques".

3.3.2 Ajouter un cœur mathématique

Il existe actuellement des utilisations cachées des mathématiques en informatique, voici quelques exemples :

Les types : On peut tout à fait voir les types comme des ensembles et les structures comme des structures mathématiques.

La vérification des types : "char f(int x)" signifie en C que si l'argument de f est un entier et si f ne boucle pas alors le résultat par f sera un caractère. Il s'agit d'une affirmation mathématique.

Les fonctions sans effet de bord : Elles peuvent se représenter par des fonctions mathématiques, c'est ce que fait avec succès Haskell.

Les raisonnements logiques des développeurs : Le développeur doit raisonner pour s'assurer que les fonctions vérifient les propriétés souhaitées. Il est possible de fournir des outils de raisonnement pouvant aider le développeur à vérifier les qualités souhaités de son logiciel.

En intégrant complètement les mathématiques à l'informatique on peut alors avoir un design plus propre des langages informatiques et offrir de nouvelles possibilités (par exemple prouver qu'une fonction ne boucle pas ou encore offrir des outils d'aide à la preuve de propriétés du logiciel).

Chapitre 4

Architecture des fondations Lazi

4.1 Définition du langage

La syntaxe Lazi est très simple et rapide à définir. Du fait du besoin de représenter simplement Lazi elle comporte des simplicités pas pratiques pour les humains, c'est pourquoi nous définissons rapidement des notations pour compenser.

4.2 Définition du système de production de vérités

Cette définition répond à la question "Comment déduire une vérité ?" .

On utilise le système le plus simple possible : une liste de vérités déjà prouvées et des règles. Une règle est définies par un ensemble de variables, des conditions et une conclusion.

4.3 Définition des 6 règles de base

4.4 Définition de la dernière règle

Grâce à des notations, cette partie (la dernière) est fait en code Lazi lui-même.

La septième règle est plus grosse et il faut une trentaine de pages pour la définir. Elle signifie que si x est une représentation d'une preuve prouvant une représentation y d'une formule alors l'interprétation de y est vrai.
On entend par interprétation de y le calcul de ce que représente y. Par exemple pour la langue française, *"Je mange du pain."*, du fait des guillemets, est une représentation d'une phrase qui a pour interprétation *Je mange du pain.* . On pourrait représenter les phrases à l'envers, on aurait alors : l'interprétation de *".niap ud egnam eJ"* est *Je mange du pain.* .

Ce qu'est une preuve pour la dernière règle ressemble à ce qu'est une preuve définie par le premier niveau des fondations ("Définition du système de production de vérités" et "Définition des 6 règles de base") mais est plus sophistiqué : les règles sont :

— Les 6 règles de base
— une règle de type "tiers exclu" faible, n'empêchant pas Lazi d'être constructif
— une règle de récurrence sur les listes finies

De plus le système de déduction élargit les assertions (qui sont normalement des formules) aux règles. On peut donc prouver, appliquer des règles prouvées et même prouver des règles où les conditions et la conclusion peuvent être des règles.

Chapitre 5

Ce qui est déjà fait

5.1 Partie mathématiques

5.1.1 Définition de Lazi

C'est fait.

5.1.2 Élimination des erreurs dans la définition

Par utilisation de la définition

C'est fait en bonne partie, la définition de Lazi étant utilisée dans des tests et des extensions.

Cette forme d'élimination d'erreurs n'a pas la fiabilité d'une preuve mathématique, elle est plutôt équivalente à des tests d'un logiciel.

Par preuve

Ce n'est pas encore fait car des extensions doivent encore être développées pour cela, l'interpréteur doit aussi est amélioré.

Les preuves n'apportent pas une garantie absolue (qui ne peut exister) mais la preuve de propriété sur Lazi oblige à vérifier les propriétés des fonctions utilisées et oblige à une vérification profonde et fiable.

5.1.3 Extensions

Les extensions développées comprennent une fonction de traduction d'une preuve avec l'extension vers une preuve sans, mais pas la preuve de la validité de l'extension (ce sera fait plus tard). Les extensions définies sont :

applyARule

Permet de déduire une assertion basée sur une règle en fournissant uniquement la conclusion à déduire. Ainsi il n'est pas nécessaire de préciser la règle utilisée ni les valeurs des variables de la règles, tout est déduit.

Par exemple si vous avez une règle déduisant y à partir de x et x=y, si titi est vrai ainsi que titi=toto, vous avez juste à préciser que vous voulez déduire "toto".

defName

Permet de définir un ensemble de noms avec leurs valeurs. C'est l'équivalent de (par exemple) "soit x = 2". Les valeurs peuvent faire référence aux noms définis, par exemple on peut définir "soit x = x" ou encore "soit x = y et y = x".

La fonction de traduction remplace dans la preuve les variables par leur valeur associées, les boucles sont éliminées par une forme de récursivité des fonctions de placement (utilisant "duplicate").

(pratiquement fini)

5.2 Partie informatique

Le logiciel “lazi-compute” est un interpréteur de Lazi (fait).

5.2.1 Un petit et un gros exemple de code source Lazi

Ces deux exemples sont extraits de la définition des fondations mathématiques Lazi.

Définition du mot `listsUnion`, `$F` est une notation pour une fonction.

```
/*
Union d'une liste de listes ll. eq est la fonction d'égalité sur les éléments.
*/
$Def listsUnion = $F eq,ll -> listRevFold (listUnion eq) 0l ll
```

L'exemple ci-dessous montre la définition d'un type `applyRuleT` paramétré par le contexte mathématique `maths` et se basant sur le type défini par la fonction `instanceT`. `$P` est une notation pour une paire et `$D` pour un dictionnaire. Quand ces notations se trouve en argument dans une définition de fonction elles servent à assigner des variables correspondant à des parties internes de structures en argument (à la Haskell).

```
///                                    applyRuleT

/*
Le type de déduction pour l'application d'une règle. Une instance de ce type est une paire (règle,
valeurs). "valeurs" est le dictionnaire des valeurs des variables.
Remarquons que si la règle contient des sous-règles ayant des noms de variable qui sont des noms
utilisés dans les valeurs, alors on pourrait avoir une collision de noms, c'est pourquoi ce cas est
hors domaine. Si ce cas arrivait il faudrait alors prouver une règle identique aux noms des variables
près (une extension permetra de le faire automatiquement).
*/
$Def applyRuleT = $F maths ->
  // Le type de l'instance dépend de l'instance car le type de "values" dépend de la règle.
  // Mais en regardant comment les fonctions sont définies on voit qu'il n'y a pas de boucle
  // (car la fonction "domain" de pairT commence par vérifier le premier élément).
  $Let instanceT = $F $P[$D[wordsQualities],values] ->
    pairT ( ruleT maths , dictT . dictMap ( constantF formulaT ) wordsQualities )
  ,
  $D[
    domain      = $F this,i@$P[rule@$D[wordsQualities],values] ->
      ( instanceT i .df `domain' ) i &b
      // Les valeurs ont-elles leurs mots connus et vérifiant les qualités requises.
      listAnd ( dictValues . dict2Map ( formulaHasQualities maths ) wordsQualities values ) &b
      // Les valeurs ne doivent pas avoir des mots rentrant en collision avec les variables des
      // sous-assertions. Remarque : pour cet ordre d'arguments le calcul est rapide en général
      // car la liste des sous-variables est souvent vide.
      isEmptyList
        (
          listIntersect
          (
            wordEqual ,
            ( ruleT maths .df `subVariables' ) rule ,
            concatListMap usedWords ( dictValues values )
          )
        )
    ,
    equal        = $F this,i,j -> (instanceT i .df `equal') i j
    ,
    isValid     = $F this,$P[rule@$D[conditions],values],truths ->
      // La règle est-elle une vérité ?
      ∃l t/truths; ( maths .df `assertionT' .df `equal' ) t $L[ `rule' , rule ] &b
      // Les conditions sont-elles dans les vérités de maths
      isListOfTruths
        maths
        truths
        ( listMap ( maths .df `assertionT' .df `replaceWords' . values ) conditions )

    ,
    isTerminal  = $F this,i -> 1
    ,
    // L'objet d'une déduction est l'assertion déduite.
    object = $F this,$P[$D[wordsQualities,conclusion],values] ->
      ( maths .df `assertionT' .df `replaceWords' ) values conclusion
  ]
```

5.2.2 Optimisation

Calculer des formules Lazi sans utiliser de shortcut ou notation est équivalent à simuler un processeur atome par atome, c'est à dire que c'est matériellement impossible sauf sur de très petits calculs. C'est pourquoi l'interpréteur Lazi comporte de nombreuses optimisations dont l'activation est paramétrable par un paramètre "shortcuts".

5.2.3 Traçage

Une option permet de récupérer toutes les étapes d'un calcul. Le résultat est sous forme HTML (l'étape intermédiaire xml est disponible) où l'on peut déplier les différentes parties. Cela n'est pas adapté quand la trace dépasse la centaine de Mo. En pratique je n'utilise que très rarement le traçage pour déboguer, c'est bien plus pratique par essais.

5.2.4 Profiling

Une option permet d'avoir des informations de profiling. Le profiling et le traçage des systèmes à évaluation paresseuses a difficilement du sens, et donc cela n'a pas la qualité du profiling des langages à exécution immédiate. Mais cela peut donner des idées ou permet de voir si les shortcuts sont complètement utilisés.

5.2.5 Utilisation

L'interpréteur en lui-même ne permet que de lire et produire du xml. Un logiciel complémentaire (lazi-translate) s'occupe des traductions dans les deux sens. On peut alors voir un rapport des exécutions par un navigateur web.

5.2.6 Développements futurs

Il me parait bien plus profitable de développer un compilateur modulaire ayant un cœur mathématique Lazi plutôt que directement un langage complet (comme Haskell) interfacé à de nombreuses librairies. Cela n'empêchera pas d'utiliser Lazi comme langage informatique, mais évitera des limitations inutiles.