Amorçage des types inductifs

Erik Helmers

Contents

L'objectif de ce projet de recherche est d'implémenter une unique langage dans lequel il est possible d'inspecter et de manipuler les types eux-mêmes.

1 Design

1.1 Mise en jambes et types dépendants

Les types dépendants constituent le socle de ce projet, il est donc de bon ton de commencer par un rapide rappel sur la notion.

Le lecteur famillier avec un language de programmation typé moderne aura été exposé à la notion de type générique (ou polymorphique). Ce sont les types qui sont fonctions d'autre types. Par exemple, les listes en OCaml sont décrites par `'a list` où `'a` représente nécessairement un type (`int`, `char`, ...). Ce mécanisme rend plus ergonomique la manipulation de structure de données.

Les types dépendants eux, sont fonction de termes. Un exemple habituel est celui des listes dont la longueur est représentée dans le type. On pourrait imaginer la notation `(nat * 'a) list` où `(5, int) list` est le type des listes d'entier de longueur 5.

Ce qui rend cruciale l'introduction de cette expressivité, est qu'elle permet d'encoder, en plus de sa structure, la logique d'un type. Par nature, les types dépendants décrivent donc un comportement dynamique.

Pour illustrer ce propos, prenons en exemple les fonctions de choix de l'axiome éponyme.

- Mathématiquement, soit X un ensemble, alors une fonction f est de choix si elle a pour structure

  $$f : \mathcal{P}(X) \to X$$

  et qu'elle vérifie

  $$\forall S \in \mathcal{P}(X), f(S) \in S$$

- Dans un language simplement typé, en supposant que X et $\mathsf{P(X)}$ désignent des types, on peut décrire la structure d'une fonction de choix:

  ```
  val f : P(X) -> X
  ```

- Avec des types dépendants, il est aussi possible d'en décrire la logique:

  ```
  val f : (S : P(X)) -> S
  ```

  Un lecteur attentif pourrait remarquer que cette déclaration n'est pas strictement compatible, à priori, avec les règles décrites plus tard. En théorie, il est possible de résoudre ce problème en stratifiant l'espace des types mais cela n'entre pas dans le périmètre de ce projet.

De l'introduction des types dépendants, il découle un phénomène supplémentaire. Types et valeurs sont nécessairement applatis sur le même plan syntaxique. Le gain d'expressivité est donc aussi accompagné d'une simplification du language. L'objectif final de ce projet est de reproduire un résultat analogue, cette fois sur la description de types dépendants inductifs.

1.2 Description des structures inductives

Maintenant que nous nous sommes familiarisés avec la notion de types dépendants, nous pouvons essayer d'expliquer la démarche de ce projet. Reprenons notre précédente discussion sur les listes, voici comment en déclarer le type :

```
type    nat = Zero | Suc of nat
type 'a list = Nil  | Cons of 'a * 'a list
type 'a tree = Leaf | Node of 'a tree * 'a * 'a tree
```

Il est assez facile de remarquer les similarités entre la déclaration d'une liste avec celle des naturels ou des arbres. Peut-on exploiter ces ressemblances à notre profit, par exemple en facilitant la réutilisation de code ? Pour pouvoir répondre à cette question, il est nécessaire de pouvoir examiner et manipuler les déclarations de types.

Le reste de ce rapport développe les étapes nécessaires pour pouvoir avoir une représentation des types de données au sein même de la théorie des types.

2 Construction de la théorie

Cette partie est la construction progressive d'un système dans lequel il est possible de répresenter des types (sous forme de description). Pour cela, nous étendons trois fois notre langage en suivant le même déroulement : définition de la syntaxe puis des règles d'évaluations et enfin nous explicitons les règles de typages.

2.1 Lambda calcul minimal

La première étape du projet consiste en l'établissement d'un système avec types dépendants.

2.1.1 Syntaxe

e, σ	$::=$	$e : \sigma$	annotated term
	$\mid$	x	variable
	$\mid$	$\lambda x \mapsto e$	lambda
	$\mid$	$e\ e'$	application
	$\mid$	$(x : \sigma) \to \sigma'$	pi type
	$\mid$	(e, e')	tuple
	$\mid$	fst e	fst
	$\mid$	snd e	snd
	$\mid$	$(x : \sigma) \times \sigma'$	sigma type
	$\mid$	$\star$	type of types

où e, σ représentent des valeurs, des types respectivement.

2.1.2 Contexte

Γ	$::=$	ϵ	empty context
	$\mid$	$\Gamma, x : \tau$	adding a variable

$$\frac{}{\text{valid}(\epsilon)} \qquad \frac{\text{valid}(\Gamma) \qquad \Gamma \vdash \tau \Leftarrow \star}{\text{valid}(\Gamma, x : \tau)}$$

2.1.3 Evaluation

ν, τ	$::=$	n	neutral term
	$\mid$	$\lambda x \mapsto \nu$	lambda
	$\mid$	$(x : \tau) \to \tau'$	dependent function space
	$\mid$	(ν, ν')	tuple
	$\mid$	$(x : \tau) \times \tau'$	dependent pair space
	$\mid$	$\star$	type of types

n	$::=$	x	variable
	$\mid$	$n\ \nu$	neutral app
	$\mid$	fst n	neutral first projection
	$\mid$	snd n	neutral second projection

$$\frac{}{\star \Downarrow \star}\ (\textsc{Star}) \qquad \frac{}{x \Downarrow x}\ (\textsc{Var}) \qquad \frac{e \Downarrow \nu}{e : \sigma \Downarrow \nu}\ (\textsc{Ann})$$

$$\frac{e \Downarrow \nu}{\lambda x \mapsto e \Downarrow \lambda x \mapsto \nu}\ (\textsc{Lam}) \qquad \frac{e \Downarrow \nu \qquad e' \Downarrow \nu'}{(e, e')(\Downarrow, \nu)\nu'}\ (\textsc{Tuple})$$

$$\frac{e \Downarrow \lambda x \mapsto \nu \qquad \nu[x \mapsto e'] \Downarrow \nu'}{e\ e' \Downarrow \nu'}\ (\textsc{App}) \qquad \frac{e \Downarrow n \qquad e' \Downarrow \nu'}{e\ e' \Downarrow n\ \nu'}\ (\textsc{NApp})$$

$$\frac{e \Downarrow (\nu, \nu')}{\text{fst } e \Downarrow \nu}\ (\textsc{Fst}) \qquad \frac{e \Downarrow (\nu, \nu')}{\text{snd } e \Downarrow \nu'}\ (\textsc{Snd}) \qquad \frac{e \Downarrow n}{\text{fst } e \Downarrow \text{fst } n}\ (\textsc{NFst})$$

$$\frac{e \Downarrow n}{\text{snd } e \Downarrow \text{snd } n}\ (\textsc{NSnd}) \qquad \frac{\sigma \Downarrow \tau \qquad \sigma' \Downarrow \tau'}{(x : \sigma) \to \sigma' \Downarrow (x : \tau) \to \tau'}\ (\textsc{Pi})$$

$$\frac{\sigma \Downarrow \tau \qquad \sigma' \Downarrow \tau'}{(x : \sigma) \times \sigma' \Downarrow (x : \tau) \times \tau'}\ (\textsc{Sigma})$$

2.1.4 Typage

Dans la suite, nous notons $e \Rightarrow \tau$ quand le terme e se synthétise en τ et $e \Leftarrow \tau$ quand il est possible de vérifier que e est d'un type τ.

$$\frac{\Gamma \vdash e \Rightarrow \tau}{\Gamma \vdash e \Leftarrow \tau}\ (\textsc{Chk}) \qquad \frac{\Gamma \vdash \sigma \Leftarrow \star \qquad \sigma \Downarrow \tau \qquad \Gamma \vdash e \Leftarrow \tau}{\Gamma \vdash (e : \sigma) \Rightarrow \tau}\ (\textsc{Ann})$$

$$\frac{}{\Gamma \vdash \star \Leftarrow \star}\ (\textsc{Star}) \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x \Rightarrow \tau}\ (\textsc{Var})$$

$$\frac{\Gamma, x : \tau \vdash e \Leftarrow \tau'}{\Gamma \vdash \lambda x \mapsto e \Leftarrow (x : \tau) \to \tau'}\ (\textsc{Lam})$$

$$\frac{\Gamma \vdash e \Leftarrow \tau \qquad \Gamma \vdash e' \Leftarrow \tau'}{\Gamma \vdash (e, e') \Leftarrow (x : \tau) \times \tau'}\ (\textsc{Tuple})$$

$$\frac{\Gamma \vdash e \Rightarrow (x : \tau) \to \tau' \qquad \Gamma \vdash e' \Leftarrow \tau \qquad \tau'[x \mapsto e'] \Downarrow \tau''}{\Gamma \vdash e\ e' \Rightarrow \tau''}\ (\textsc{App})$$

$$\frac{\Gamma \vdash e \Rightarrow (x : \tau) \times \tau'}{\Gamma \vdash \text{fst } e \Rightarrow \tau}\ (\textsc{Fst})$$

$$\frac{\Gamma \vdash e \Rightarrow (x : \tau) \times \tau' \qquad \tau'[x \mapsto \text{fst } e] \Downarrow \tau''}{\Gamma \vdash \text{snd } e \Rightarrow \tau''}\ (\textsc{Snd})$$

$$\frac{\Gamma \vdash \sigma \Leftarrow \star \qquad \sigma \Downarrow \tau \qquad \Gamma, x : \tau \vdash \sigma' \Leftarrow \star}{\Gamma \vdash (x : \sigma) \to \sigma' \Leftarrow \star}\ (\textsc{Pi})$$

$$\frac{\Gamma \vdash \sigma \Leftarrow \star \qquad \sigma \Downarrow \tau \qquad \Gamma, x : \tau \vdash \sigma' \Leftarrow \star}{\Gamma \vdash (x : \sigma) \times \sigma' \Leftarrow \star}\ (\textsc{Sigma})$$

2.2 Interlude des booléens

Avant d'avancer plus loin, il est bon de pouvoir s'assurer que l'implémentation correspond à nos attentes.

2.2.1 Syntaxe

$$
\begin{array}{rcll}
e, \sigma, \kappa & ::= & ... & \\
& | & \text{true} & \\
& | & \text{false} & \\
& | & \text{cond } e\ [x.\sigma]\ e'\ e'' & \text{condition} \\
& | & \text{bool} & \text{type of a bool}
\end{array}
$$

2.2.2 Evaluation

$$
\begin{array}{rcll}
\nu, \tau & ::= & ... & \\
& | & \text{true} & \\
& | & \text{false} & \\
& | & \text{bool} &
\end{array}
$$

$$
\begin{array}{rcll}
n & ::= & ... & \\
& | & \text{cond } \nu\ [x.\tau]\ \nu'\ \nu'' &
\end{array}
$$

$$\frac{}{\text{true} \Downarrow \text{true}}\ (\textsc{True}) \qquad \frac{}{\text{false} \Downarrow \text{false}}\ (\textsc{False})$$

$$\frac{e \Downarrow \text{true} \qquad e' \Downarrow \nu}{\text{cond } e\ [x.B]\ e'\ e'' \Downarrow \nu}\ (\textsc{CondT}) \qquad \frac{e \Downarrow \text{true} \qquad e'' \Downarrow \nu}{\text{cond } e\ [x.B]\ e'\ e'' \Downarrow \nu}\ (\textsc{CondF})$$

$$\frac{e \Downarrow n \qquad e' \Downarrow \nu \qquad e'' \Downarrow \nu'}{\text{cond } e\ [x.B]\ e'\ e'' \Downarrow \text{cond } n\ [x.\tau]\ \nu\ \nu'}\ (\text{NCond})$$

$$\frac{}{\text{bool} \Downarrow \text{bool}}\ (\textsc{BoolTy})$$

2.2.3 Typing

$$\frac{}{\text{true} \Leftarrow \text{bool}}\ (\textsc{True}) \qquad \frac{}{\text{false} \Leftarrow \text{bool}}\ (\textsc{False})$$

$$\frac{\Gamma \vdash e \Leftarrow \text{bool} \qquad \Gamma, x : \text{bool} \vdash B \Leftarrow \star \qquad B[x \mapsto e] \Downarrow \tau}{\Gamma \vdash \text{cond } e\ [x.B]\ e'\ e'' \Rightarrow \tau}\ (\textsc{Cond})$$

$$\frac{}{\text{bool} \Leftarrow \star}\ (\textsc{BoolTy})$$

2.3 Enumerations

2.3.1 Syntax

$$
\begin{array}{rcll}
e, \sigma, \kappa & ::= & \ldots & \\
& | & \text{nil} & \\
& | & \text{unit} & \\
& | & {}'\text{t} & \text{label} \\
& | & \text{label} & \text{label type} \\
& | & \text{nil} & \\
& | & [t\ l] & \\
& | & \text{labels} & \text{labels type}
\end{array}
$$

2.3.2 Evaluation

$$\frac{l \Downarrow \text{nil}}{\text{record } l \text{ as } x \text{ return } B \Downarrow \text{unit}}\ (\textsc{RecordNil})$$

$$\frac{B[x \mapsto 0] \Downarrow \tau \qquad \begin{array}{c} l \Downarrow [t\ l'] \\ \text{record } l' \text{ as } x \text{ return } B[x \mapsto 1+\ x] \Downarrow \tau' \end{array}}{\text{record } l \text{ as } x \text{ return } B \Downarrow \tau \times \tau'}\ (\textsc{RecordCons})$$

$$\frac{l \Downarrow n}{\text{record } l \text{ as } x \text{ return } B \Downarrow \text{record } n \text{ as } x \text{ return } B}\ (\text{NRecord})$$

$$\frac{e \Downarrow 0 \qquad \text{fst cs} \Downarrow \nu}{\text{case } e \text{ as } x \text{ return } B \text{ with cs} \Downarrow \nu}\ (\textsc{CaseZe})$$

$$\frac{\text{snd cs} \Downarrow \text{cs'} \qquad \begin{array}{c} e \Downarrow 1+\ e' \\ \text{case } e' \text{ as } x \text{ return } B[x \mapsto 1+\ x] \text{ with cs'} \Downarrow \nu \end{array}}{\text{case } e \text{ as } x \text{ return } B \text{ with cs} \Downarrow \nu}\ (\textsc{CaseSuc})$$

$$\frac{e \Downarrow n \qquad \text{cs} \Downarrow \nu}{\text{case } e \text{ as } x \text{ return } B \text{ with cs} \Downarrow \text{case } n \text{ as } x \text{ return } B \text{ with } \nu}\ (\textsc{CaseZe})$$

2.3.3 Typing

$$\frac{}{\text{nil} \Leftarrow \text{unit}}\ (\textsc{Nil}) \qquad \frac{}{\text{unit} \Leftarrow \star}\ (\textsc{Unit})$$

$$\frac{}{\text{'t} \Leftarrow \text{label}}\ (\textsc{Label}) \qquad \frac{}{\text{label} \Leftarrow \star}\ (\textsc{LabelTy})$$

$$\frac{}{\text{nil} \Leftarrow \text{labels}}\ (\textsc{NilL}) \qquad \frac{\Gamma \vdash t \Leftarrow \text{label} \qquad \Gamma \vdash l \Leftarrow \text{labels}}{[t\ l] \Leftarrow \text{labels}}\ (\textsc{ConsL})$$

$$\frac{}{\text{labels} \Leftarrow \star}\ (\textsc{LabelsTy})$$

$$\frac{\Gamma \vdash t \Leftarrow \text{label} \qquad \Gamma \vdash l \Leftarrow \text{labels}}{\Gamma \vdash 0 \Leftarrow \text{enum}\ [t\ l]}\ (\textsc{Zero})$$

$$\frac{\Gamma \vdash t \Leftarrow \text{label} \qquad \Gamma \vdash l \Leftarrow \text{labels} \qquad \Gamma \vdash n \Leftarrow \text{enum}\ l}{\Gamma \vdash 1+\ n \Leftarrow \text{enum}\ [t\ l]}\ (\textsc{Suc})$$

$$\frac{\Gamma \vdash l \Leftarrow \text{labels}}{\Gamma \vdash \text{enum}\ l \Leftarrow \star}\ (\textsc{Enum})$$

$$\frac{\Gamma \vdash l \Leftarrow \text{labels} \qquad \Gamma, x : \text{enum}\ l \vdash B \Leftarrow \star}{\Gamma \vdash \text{record}\ l\ \text{as}\ x\ \text{return}\ B \Rightarrow \star}\ (\textsc{Record})$$

$$\frac{\begin{array}{c}\Gamma \vdash e \Leftarrow \text{enum}\ l \qquad \Gamma, x : \text{enum}\ l \vdash B \Leftarrow \star \\ B[x \mapsto e] \Downarrow \tau \qquad \Gamma \vdash \text{cs} \Leftarrow \text{record}\ l\ \text{as}\ x\ \text{return}\ B\end{array}}{\Gamma \vdash \text{case}\ e\ \text{as}\ x\ \text{return}\ B\ \text{with cs} \Rightarrow \tau}\ (\textsc{Case})$$

2.4 Descriptions

2.4.1 Syntax

$$
\begin{array}{rcll}
e, \sigma & ::= & ... & \\
& | & \text{`unit} & \\
& | & \text{`var} & \text{identity functor} \\
& | & \text{`}\Sigma\ \sigma\ e & \\
& | & \text{`}\Pi\ \sigma\ e & \\
& | & [\![e]\!]\ \sigma & \\
& | & \text{desc} & \text{descriptor type} \\
& | & \mu\ e & \\
& | & \text{ctor}\ e &
\end{array}
$$

2.4.2 Evaluation

$$
\begin{array}{rcll}
\nu, \tau & ::= & ... & \\
& | & \text{`unit} & \\
& | & \text{`var} & \text{identity functor} \\
& | & \text{`}\Sigma\ \tau\ D & \\
& | & \text{`}\Pi\ \tau\ D & \\
& | & \text{desc} & \text{descriptor type} \\
& | & \mu & \text{fixpoint} \\
& | & \text{ctor}\ n & \text{constructor}
\end{array}
$$

$$
\begin{array}{rcll}
n & ::= & ... & \\
& | & [\![n]\!]\ \tau &
\end{array}
$$

$$\frac{D \Downarrow \text{`unit}}{[\![D]\!]\ \sigma \Downarrow \text{unit}}\ (\textsc{DecodeNil}) \qquad \frac{D \Downarrow \text{`var} \qquad \sigma \Downarrow \tau}{[\![D]\!]\ \sigma \Downarrow \tau}\ (\textsc{DecodeVar})$$

$$\frac{D \Downarrow \text{`}\Sigma\ \tau\ D' \qquad (e : \tau) \times [\![D'\ e]\!]\ \sigma \Downarrow \tau'}{[\![D]\!]\ \sigma \Downarrow \tau'}\ (\textsc{DecodeSigma})$$

$$\frac{D \Downarrow \text{`}\Pi\ \tau\ D' \qquad (e : \tau) \to [\![D'\ e]\!]\ \sigma \Downarrow \tau'}{[\![D]\!]\ \sigma \Downarrow \tau'}\ (\textsc{DecodePi})$$

2.4.3 Typing

$$\frac{}{\Gamma \vdash \text{`unit} \Leftarrow \text{desc}} \qquad \frac{}{\Gamma \vdash \text{`var} \Leftarrow \text{desc}} \qquad \frac{}{\Gamma \vdash \text{desc} \Leftarrow \star}$$

$$\frac{\Gamma \vdash \sigma \Leftarrow \star \qquad \sigma \Downarrow \tau \qquad \Gamma \vdash D \Leftarrow \tau \to \text{desc}}{\Gamma \vdash \text{`}\Sigma\ \sigma\ D \Leftarrow \text{desc}}\ (\textsc{DSigma})$$

$$\frac{\Gamma \vdash \sigma \Leftarrow \star \qquad \sigma \Downarrow \tau \qquad \Gamma \vdash D \Leftarrow \tau \to \text{desc}}{\Gamma \vdash \text{`}\Pi\ \sigma\ D \Leftarrow \text{desc}}\ (\textsc{DPi})$$

$$\frac{\Gamma \vdash D \Leftarrow \text{desc} \qquad \Gamma \vdash \sigma \Leftarrow \star}{\Gamma \vdash [\![D]\!]\ \sigma \Rightarrow \star}\ (\textsc{Decode}) \qquad \frac{\Gamma \vdash D \Leftarrow \text{desc}}{\Gamma \vdash \mu\ D \Leftarrow \star}\ (\textsc{Mu})$$

$$\frac{\Gamma \vdash \sigma \Leftarrow [\![D]\!]\ (\mu\ D)}{\Gamma \vdash \text{ctor}\ \sigma \Leftarrow \mu\ D}$$

3 Conclusion, futur travail

Nous avons défini les structures et règles nécessaires à la représentation paradoxale des types que nous souhaitions accomplir. Malheureusement, si nous avons réussi à bâtir un système de support solide, nous n'avons pas eu le temps de l'employer. Or, c'est ce dernier point qui aurait constitué une étape importante pour la réussite pour ce projet.