Martin-Löf on n'a pas $\alpha \leq \beta \vee \beta < \alpha$

mais $\forall \gamma \left[\alpha, \beta < \gamma \rightarrow [\alpha \leq \beta \vee \beta < \alpha \right.$

About functional hierarchy and ordinal notations

J.-Y. Girard
Dec 88

Introduction

The aim of this note is to explain the representation of ordinals in a type system. As an illustration, we present the translation of the proof of the theorem of comparison of hierarchies (Girard 75) in this formalism. This proof has been type-checked in the version 4.6 of the system of constructions in CAML, following closely S. Wainer "Slow growing versus fast growing", Leeds Tech. Report 1987 (our treatment is however less general since we restrict ourselves to the first three levels of ordinals).

In order to simplify the notations, we have supposed to have ML notation for concrete data type and pattern-matching, with overloading for the name of constructors (dual to the overloading for labels in records). This facility is not implemented yet (in the implemented version, they are axiomatised, which makes the proof harder). To have such a notation seems to be a good objective for a system with inductive types.

Our treatment doesn't pretend to any mathematical novelty. The goal is only to present a test example for systems that handle inductively defined types, and show how such a formulation suggests a discussion on which is the equality (intensional or extensional) used. In a first part, we describe the type theoretic representation of ordinals. An important point is that we represent ordinal notations rather than ordinals. In the second part, we explain the theorem of hierarchy, and we show that we make an intensional use of the notations for ordinals.

1 Motivations

This section is only here to motivate the theorem we shall prove in a formal system, and is so at the "meta"-level w.r.t. the rest of the paper.

We consider two functional hierarchies defined over integers, indexed over ordinal notations. The slow hierarchy γ is defined inductively by $\gamma_0(x) = 0$, $\gamma_{\alpha+1}(x) = \gamma_\alpha(x) + 1$ and for $\alpha = \bigvee(\alpha_n)$, $\gamma_\alpha(x) = \gamma_{\alpha_x}(x)$. It grows indeed very slowly. For instance $\gamma_\omega(x) = x$, $\gamma_{\omega^\omega+\omega^3+5} = x^x + x^3 + 5$. The fast hierarchy λ is defined by $\lambda_0(x) = x$, $\lambda_{\alpha+1}(x) = \lambda_\alpha(x+1)$ and for $\alpha = \bigvee(\alpha_n)$, $\lambda_\alpha(x) = \lambda_{\alpha_x}(x)$ For instance $\lambda_\omega(x) = 2x$, $\lambda_{\omega^2}(x) = x2^x$, and λ_{ω^ω} is of the same order as Ackerman's function.

Notice, and this is a very important remark, that these hierarchies are indexed not really over ordinals, but over ordinal notations.

The theorem of hierarchy (Girard 75) shows that these two hierarchies overlap, provided the ordinal notation for γ is high enough. For instance, it is possible to show that λ_{ϵ_0} [1], which grows more quickly than any function provably total in Peano arithmetic, is also of the form γ_α for one ordinal notation α.

[1] here the ordinal ϵ_0 is taken with its "natural" fundamental sequence

2 Representation of ordinals in type theory

A reference is Martin-Löf, "Notes on Constructive Mathematics" Almqvist & Wiksell, Stockholm. The type of integers N0, is described as an ML concrete data type.

```
type N0 = 0 | S of N0;;
```

The induction principle on N0 is

```
(P:N0->Prop)(P 0)->((u:N0)(P x)->(P (S x)))->(x:N0)(P x).
```

It is the type of unary representation of integers. For the type of "countable" ordinals N1, we take (with overloading, in order to simplify the notation):

```
type N1 = 0 | S of N1 | Lim of N0 -> N1 ;;
```

The induction principle on N1 is

```
(P:N1->Prop)(P 0)->((u:N1)(P u)->(P (S u)))->
        ((f:N0->N1)((u:N0)(P (f u)))->(P (Lim f)))->(x:N1)(P x).
```

Finally, we need also the "ordinals of class 3"

```
type N2 = 0 | S of N2 | Lim of N0 -> N2 | Sup of N1 -> N2 ;;
```

The induction principle on N2 is

```
(P:N2->Prop)(P 0)->((u:N2)(P u)->(P (S u)))->
        ((f:N0->N2)((u:N0)(P (f u)))->(P (Lim f)))->
        ((f:N1->N2)((z:N1)(P (f z)))->(P (Sup f)))->(x:N2)(P x).
```

Note that N1 for instance, is not the type of ordinals, but really the type of ordinal *notations*. For instance, if we define

```
let rec (L0:N0->N1) = function 0->0 | S(u) -> S(L0(u));;
```

We have (at least) two distinct notations for the ordinal ω. For instance both (Lim L0), and (Lim (S o L0)) represent ω. We thus get a (meta-)equivalence relations between ordinal notations: "to represent the same ordinal". Surprisingly, we will use *intensional* operations that are not extensional, that is functions on ordinal notations that depend on the denotation and not only on the ordinal value of the argument.

We need also the coercion from N1 to N2:

```
let rec (i1:N1 ->N2) = function
 0->0
|S(u) -> S(i1(u))
|Lim f -> Lim (i1 o f) ;;
```

We can also define a version of the slow hierarchy, seen as a collapsing from N1 to N0 (intuitively ω becomes n, ω^2 becomes n^2, $\omega^\omega + \omega + 1$ becomes $n^n + n + 1, \cdots$)

```
let rec (G1:N1->N0->N0) x n = match x with
 0 -> 0
|S(y) -> S(G1 y n)
|Lim f -> G1 (f n) n ;;
```

and we will need also

```
let rec (L1:N1 -> N0 ->N2) n = function
 0->0
|S(u) -> S(L1(u))
|Lim f -> Sup (fun z -> L1 (f (G1 z n))) ;;
```

We define a lifting of G1 at level 2, seen as a collapsing from N2 to N1

```
let rec (G2:N2->N0->N1) x n = match x with
 0 -> 0
```

```
|S(y) -> S(G2 y n)
|Lim f -> G2 (f n) n
|Sup f -> Lim (fun p -> G2 (f (L0 p)) n) ;;
```

We consider also a version of the fast hierarchy, where ordinals are translated into functions from integers to integers

```
let rec (B1:N1->N0->N0) x n = match x with
 0 -> n
|S(y) -> B1 y (S(n))
|Lim f -> B1 (f n) n ;;
```

and the same at level 2, where ordinals of N2 are translated as functions from N1 to N1,

```
let rec (B2:N2->N1->N1) x z = match x with
 0 -> z
|S(y) -> B2 y (S(z))
|Lim f -> Lim (fun n -> B2 (f n) z)
|Sup f -> B2 (f z) z ;;
```

Notice that the functions B1 and B2 on ordinal notations are not extensional. For instance, we have

```
B1 (Lim L0) (S(0)) = B1 (S(0)) (S(0)) = S(S(0))
```

but

```
B1 (Lim S o L0) S(0) = B1 (S(S(0))) (S(0)) = S(S(S(0)))
```

though, extensionally, (Lim L0) and (Lim (S o L0)) represent the same ordinal ω.

We will need define a weak extensional equality on the type N1. We define a binary relation Eq on N1 by induction. We use the notation (n:N0)(P n) for the universal quantification of a predicate P over N0.

```
let rec (Eq:N1->N1->Prop) n m = match (n,m) with
 (0,0) -> true
|(S(p),S(q)) -> Eq p q
|(Lim f,Lim g) -> (n:N0)(Eq (f n) (g n))
| _ -> false ;;
```

Notice that this equality is still not the equality of the associated ordinals. Indeed, we have that (Eq (Lim L0) (Lim (S o L0))) = false, though (Lim L0) and (Lim (S o L0)) represent the same ordinal.

Another equality on any type is Leibniz equality, that we write "x = y" and which means that (P:A→Prop)(P y)⇒(P x), where A is the common type of x and y.

An important point is that B1 is extensional for Eq,

Lemma: If (Eq x y), and n:N0, then (B1 x n) = (B1 y n).

This is provable by a direct induction on x,y.

We can show by a direct induction that G1 (L0 n) p = n, and Eq (G2 (L1 z n) p) z, for any n,p:N0, z:N1.

3 The theorem of comparison of hierarchies

In the framework of the previous section, we can state simply the translation of the theorem of comparison of hierarchies, which becomes a non-trivial property of some inductively defined objects.

We need first to define by induction a relation O2.

```
let rec (O2:N0->N2->Prop) n = function
 0 -> true
|S(x) -> O2 n x
|Lim f -> (p:N0)(O2 n (f p))
|Sup f -> (z:N1)(O2 n (f z)) and
                (z:N1)(Eq (G2 (f z) n) (G2 (f (L0 (G1 z n))) n)) ;;
```

For instance, we have (O2 n (Lim L0)), (O2 n (i1 z)) for any z:N1, n:N0 (by induction on z) and (O2 n (Lim i1)). Indeed, we have to show that for all z:N1,

```
Eq (G2 (i1 z) n) (G2 (i1 (L0 (G1 z n))) n))
```

This is by induction on z. This is true if z is O, and true by induction if z is S(z0). If z is of the form (Lim f), then we want to compare

```
(G2 (i1 (Lim f)) n) = (G2 (Lim (i1 o f)) n)
                    = (G2 (i1 (f n)) n)
```

and

```
(G2 (i1 (L0 (G1 (Lim f) n))) n) = (G2 (i1 (L0 (G1 (f n) n))) n)
```

But we know that this is equal to (G2 (i1 (f n)) n) by induction on z.

We can now state

Proposition: (theorem of comparison of hierarchies) For all x:N2, y:N1, and n:N0, if (O2 n x), we have G1 (B2 x y) n = B1 (G2 x n) (G1 y n).

This will imply that G1 (B2 x0 y0) n0 has the same canonical value than B1 (G2 x0 n0) (G1 y0 n0), for closed terms x0, y0, n0 such that (O2 n0 x0).

Proof: By induction on x. The induction is direct if x is of the form O or S(x0). If x is of the form (Lim f), we get by induction

```
G1 (B2 (Lim f) y) n = G1 (Lim (fun m -> (B2 (f m) y)) n
                    = G1 (B2 (f n) y) n
                    = B1 (G2 (f n) n) (G1 y n)
                    = B1 (G2 (Lim f) n) (G1 y n).
```

Finally, if x is of the form (Sup f), the hypothesis (O2 n x) will imply that

```
(Eq (G2 (f (L0 (G1 z n))) n) (G2 (f z) n))
```

for all z:N1. Since is B1 is extensional for Eq, and by induction hypothesis, we get:

```
G1 (B2 (Sup f) y) n = G1 (B2 (f y) y) n
                    = B1 (G2 (f y) n) (G1 y n)
                    = B1 (G2 (f (L0 (G1 y n))) n) (G1 y n)
                    = B1 (Lim (fun m -> (G2 (f (L0 m)) n))) (G1 y n)
                    = B1 (G2 (Sup f) n) (G1 y n).
```

We make use of this result with the value om0 = (Lim L0) for y. In this case, one can check that we have (G1 om0 n) = n, so that we get

```
G1 (B2 x om0) n = B1 (G2 x n) n
```

if (O2 n x).

We will give to this result a more natural form. Let us define the slow hierachy gamma: N1→N0→N0, by gamma = G1, and the fast hierarchy lambda: N1→N0→N0 by lambda = B1. The equation of the proposition tells us now that if Eq (G2 x n) y is provable for all n:N0, then the function (lambda y) is extensionally equal to gamma (B2 x om0), provided (O2 n x).

For instance, for x = om1 = (Sup i1), we have, for any n:N0,

```
G2 om1 n = Lim (fun p -> G2 (L0 p) n)
```

and G2 (L0 p) n = L0 p, for all p:N0 by induction on p, so that Eq (G2 om1 n) (Lim L0), that is Eq (G2 om1 n) om0. Since we know that (O2 n om1), we get that the function (lambda om0) is extensionally equal to (gamma (B2 om1 om0)). Indeed, both functions represent $n \longmapsto n + n$.

Another version (a little more difficult to type-checked, and which is roughly the one followed by S. Wainer) is to change B1 and B2 following a variation on the fast hierarchy:

```
let rec (Iter:(A:Type)N0 ->(A->A)->A->A) A n f x = match n with
 0 -> x
|S(m) -> f (Iter A m f x) ;;

let rec (F1:N1->N0->N0) x n = match x with
 0 -> S(n)
|S(z) -> Iter N0 (F1 z) n
|Lim f -> F1 (f n) n ;;

let rec (F2:N2->N1->N1) x z = match x with
 0 -> S(z)
|S(y) -> Iter N1 (F y) z
|Lim f -> Lim (fun p -> F2 (L0 p) z)
|Sup f -> F (f z) z
```

We can then show that G1 (F2 x y) n = F1 (G2 x n) (G1 y n), provided (O2 n x), so that we get that F1 (G2 x n) and lambda (F2 x om0) are extensionally equal. For x = om1, we get an ordinal notation (F2 om1 om0), so that gamma (F2 om1 om0) is of the order of Ackerman function.

To get an example of the order of Ackerman functions with the current version, we need to prove more things about O2. We define first:

```
let rec (plus:N1->N1->N1) x = function
 0 -> x
|S(y) -> S(plus x y)
|Lim f -> Lim (fun p -> plus x (f p));;

let rec (plus:N2->N2->N2) x = function
 0 -> x
|S(y) -> S(plus x y)
|Lim f -> Lim (fun p -> plus x (f p))
```

```
|Sup f -> Sup (fun z -> plus x (f z));;
```

We can then prove Eq (G2 (plus x y) n) (plus (G2 x n) (G2 y n)), by induction on y. Furthermore (O2 x n) and (O2 n y) implies (O2 n (plus x y)). We define then:

```
let rec (times:N1->N1->N1) x = function
 0 -> x
|S(y) -> plus x (times x y)
|Lim f -> Lim (fun p -> times x (f p));;

let rec (times:N2->N2->N2) x = function
 0 -> x
|S(y) -> plus x (times x y)
|Lim f -> Lim (fun p -> times x (f p))
|Sup f -> Sup (fun z -> times x (f z));;
```

and prove Eq (G2 (times x y) n) (times (G2 x n) (G2 y n)), by induction on y. Finally, one defines the "exponentiation to ω":

```
let rec (exp:N1 -> N1) = function
 0 -> S(0)
|S(y) -> times (exp y) om0
|Lim f -> Lim (fun p -> exp (f p)) ;;

let rec (exp:N2 -> N2) = function
 0 -> S(0)
|S(y) -> times (exp y) (i1 om0)
|Lim f -> Lim (fun p -> exp (f p))
|Sup f -> Sup (fun z -> exp (f z)) ;;
```

Notice that F1 x is extensionally equal to B1 (exp x) (by induction on x).

It is then possible to show that (O2 n (exp x)) if (O2 n x), and that Eq (G2 (exp x) n) (exp (G2 x n)). As an application, we get that gamma (B2 (exp om1) om0) represents a function which is of the order of Ackerman function.

Conclusion

We have illustrated the theory of inductive definitions in type theory with a non-trivial theorem on ordinal hierarchy that seems to be a reasonable test for a mechanised system. We have limited our analysis to ordinal notations, and it may be an interesting exercice to relate this type-theoretic notion of ordinal notations to ordinals, as name for well-founded linear relations. Also, it might be interesting to look at the corollary that, for instance, $(\exists\alpha)\gamma_\alpha = \lambda_{\omega^\omega}$ and to do a cut-elimination, or pruning, on such a proof (actually, it may even be interesting to look mechanically at direct proof of results like $\gamma_{B_2(\omega^{\omega_1},\omega)} = \lambda_{\omega^\omega}$).

References