Normalisation Proof for Dependent Types

August 9, 2002

Exercices

SASL tautology function

Define a function $F : N \to *$ such that $F\ 0 = B$ and $F\ (S\ n) = B \to F\ n$ Define next conjunction $x \wedge y : B\ [x, y : B]$ and a function $taut : \Pi n : N.F\ n \to B$ such that

$$taut\ 0\ f = f \qquad taut\ (S\ n)\ f = taut\ n\ (f\ 0) \wedge taut\ n\ (f\ 1)$$

Prove directly that $\Pi n : N.F\ n \to B$ is a small type and that we have $taut \Vdash \Pi n : N.F\ n \to B$.

Heyting arithmetic

Assume given types $\perp : *$ and $T : *$ with one element $1 : T$ define a function $eq : N \to N \to *$ such that

$$eq\ 0\ (S\ y) = \perp \qquad eq\ 0\ 0 = T \qquad eq\ (S\ x)\ 0 = \perp \qquad eq\ (S\ x)\ (S\ y) = eq\ x\ y$$

Using this function it is possible to embed Heyting arithmetic (intuitionistic version of Peano arithmetic) in the type-theory we have considered. This embedding is known to be non conservative: there are propositions $A : *$ that are inhabited, but not provable in Heyting arithmetic.

Type-Checker with explicit substitution

Write an type-checker for $* : *$ in CAML, SML or Haskell. The main problem is clash of variables and correct definition of the substitution function. Here is a possible way to structure the program. The intuition, roughly, is to write what should be an arbitrary denotational model of the calculus.

First, the syntax for terms, symbolic values and contexts

$$t \ = \ x \mid [x : t]t \mid t\ t \mid * \qquad v \ = \ v_k \mid v\ v \mid * \mid t\rho \qquad \Gamma \ = \ \Gamma[x : t] \mid$$

(following Automath we identify λ and Π) where ρ is an typed environment, represented as a pair $\rho = \Gamma, vs$ of a context and a list of values.

If $\rho = \Gamma, vs$ we define $\rho, x : A = v$ to be $\Gamma[x : A], vs.v$; we define the lookup function by

$$\epsilon\ x\ (\Gamma[x : A], vs.v) = v \qquad \epsilon\ x\ (\Gamma[y : A], vs.v) = \epsilon\ x\ (\Gamma, vs) \ \ if\ y \neq x$$

and the lookup type function by

$$\tau\ x\ (\Gamma[x : A], vs.v) = A(\Gamma, vs) \qquad \tau\ x\ (\Gamma[y : A], vs.v) = \tau\ x\ (\Gamma, vs)\ if\ y \neq x$$

We define head-reduction on values by

$$x\rho \succ \epsilon\ x\ \rho \qquad (\lambda x : A.t)\rho\ v \succ t(\rho, x : A = v) \qquad (t_1\ t_2)\rho \succ t_1\rho\ (t_2\rho) \qquad *\,\rho \succ *$$

and $u\ v \succ u'\ v$ if $u \succ u'$ (notice that these laws follow indeed the usual denotational semantics of programs).

From this, it is easy to define the conversion algorithm. The type-checking rules have to be modified as follow, where $\rho \vdash e : v$ can be read as "in the model the value $e\rho$ is of type v"

- $\rho \vdash x : v$ iff $\tau\ x\ \rho = v$
- $\rho \vdash [x : A]B : *$ if $\rho \vdash A : *$ and $\rho, x : A = v_k \vdash B : *$
- $\rho \vdash [x : A]t : ([y : C]D)\nu$ if $\rho \vdash A : *$ and $A\rho = C\nu$ and $\rho, x : A = v_k \vdash t : D(\nu, y : C = v_k)$
- $\rho \vdash t_1\ t_2 : B(\nu, x : A = t_2\rho)$ if $\rho \vdash t_1 : ([x : A]B)\nu$ and $\rho \vdash t_2 : A\nu$
- $\rho \vdash * : *$

where v_k denotes a fresh value variable.

This extends easily to the case of definitions in context $\Gamma[x : A = t]$, by taking

$$\epsilon\ x\ (\Gamma[x : A = t], vs) = t(\Gamma, vs) \qquad \tau\ x\ (\Gamma[x : A = t], vs) = A(\Gamma, vs)$$

The type-checker can take as an input such a context with definitions. A good test is provided by the next exercice.

Various optimisations are possible. For instance, one can compile the variables as deBruijn indexes with a pointer to their types, and then simplify the environment to be only a list of values.

Paradox

Check T. Hurkens' paradox for $* : *$ written here in a domain-free style [**?**].

$\perp$	:	$*$		
$\neg$	:	$*\to*$	=	$\lambda X.X\to\perp$
Pow	:	$*\to*$	=	$\lambda X.X\to*$
U	:	$*$	=	$\Pi X : *.(Pow\ (Pow\ X)\to X)\to Pow\ (Pow\ X)$
τ	:	$Pow\ (Pow\ U)\to U$	=	$\lambda t\lambda X\lambda f\lambda p.t\ (\lambda x\ (p\ (f\ (x\ X\ f))))$
σ	:	$U\to Pow\ (Pow\ U)$	=	$\lambda s.s\ U\ \tau$
Q	:	$Pow\ (Pow\ U)$	=	$\lambda p.\Pi x : U.\sigma\ x\ p\to p\ x$
B	:	$Pow\ U$	=	$\lambda y.\neg\ \Pi p : Pow\ U.\sigma\ y\ p\to p\ (\tau\ (\sigma\ y))$
C	:	U	=	$\tau\ Q$
lem_1	:	$Q\ B$	=	$\lambda x\lambda k\lambda l.l\ B\ k\ (\lambda p.l\ (\lambda y.p\ (\tau\ (\sigma\ y))))$
A	:	$*$	=	$\Pi p : Pow\ U.Q\ p\to p\ C$
lem_2	:	$\neg\ A$	=	$\lambda h.h\ B\ lem_1\ (\lambda p.h\ (\lambda y.p\ (\tau\ (\sigma\ y))))$
lem_3	:	A	=	$\lambda p\lambda h.h\ C\ (\lambda x.h\ (\tau\ (\sigma\ x)))$
$loop$	:	$\perp$	=	$lem_2\ lem_3$

Infinity Axiom

Using the normalisation theorem for CC, show that the following context is consistent, it consists of

$$\perp : *, A : *, f : A \to A, a : A, \quad R : A \to A \to *$$

with

$$h_1 : \Pi x : A.R\ x\ x \to \perp,\ h_2 : \Pi x, y, z : A.R\ x\ y \to R\ y\ z \to R\ x\ z,\ h_3 : \Pi x : A.R\ x\ (f\ x)$$

Intuitively this context expresses that the small type A contains an infinite number of elements. Show first by induction on normal forms, that if we have a term in normal form of type A then this term has the form $f^n\ a$ and if a term in normal form is of type $R\ u\ v$ then we have $u = f^p\ a,\ v = f^q\ a$ with $p < q$. See [**?**].

Definitions of booleans

The following argument (due to Herman Geuvers [**?**]) shows intuitively why in $P2$, it is not possible to instantiate the following context

$$B : *,\ 0, 1 : B,\ h : \Pi C : B \to *.C\ 0 \to C\ 1 \to \Pi x : B.C\ x$$

This is surprising since we can instantiate the context

$$B : *,\ 0, 1 : B,\ h : \Pi C : *.C \to C \to B \to C$$

by taking $B = \Pi X : *.X \to X \to X$ and $0 = \lambda X : *.\lambda x, y : X.x,\ 1 = \lambda X : *.\lambda x, y : X.y$ and $h = \lambda C : *.\lambda x, y : C.\lambda b : B.b\ C\ x\ y$.

We reason by contradiction (the same argument would work for CC), supposing that we have a closed instantiation $B, 0, 1, h$ of the context and using a special model, that had already been introduced to give models of CC where the axiom of choice does not hold.

This model is described intuitively. Let us write $A, B, C, \ldots$ terms of type $*$ and $K, L, M, \ldots$ terms of type $\Box$. Let U be the set of all untyped (maybe open) lambda terms. The type $*$ is interpreted by $\{\emptyset, U\}$, so that $[\![A]\!]$ is $\emptyset$ or U. The terms $t : A$ are interpreted by stripping away all types/kind information; for instance $\lambda A : *.\lambda x : A.x$ becomes $\lambda x.x \in U$; so we have $[\![\lambda x : A.t]\!] = \lambda x.[\![t]\!]$ and $[\![\lambda X : K.t]\!] = [\![t]\!]$. We take $[\![\Pi x : A.B]\!]$ to be the set of all terms t such that if $u \in [\![A]\!]$ then $t\ u \in [\![B]\!]_{x=u}$ while $[\![\Pi X : K.B]\!]$ is the set of all terms t such that if $\alpha \in [\![K]\!]$ then $t \in [\![B]\!]_{X=\alpha}$. In both cases this set is either $\emptyset$ or U. Finally $[\![\Pi x : A.L]\!]$ is the set of all functions α such that if $t \in [\![A]\!]$ then $\alpha(t) \in [\![L]\!]_{x=t}$.

In this model we have $[\![B]\!] = U$ because $[\![B]\!]$ is inhabited and we can consider $\alpha : U \to *$

$$\alpha(v) = \{u \in U \mid v = [\![0]\!] \vee v = [\![1]\!]\}$$

$\alpha(v)$ is U whenever it is inhabited. We should have, for all $t, u, v \in U$

$$[\![h]\!]\ \alpha\ t\ u\ v \in \alpha(v)$$

but $\alpha(v)$ may be empty, hence the result.