Proof of normalisation using domain theory

Thierry Coquand and Arnaud Spivak

Aug. 24, 2005

Goal of the presentation

Show an example where computer science helps in simplifying an argument in proof theory

How to prove normalisation for some computation rules introduced in proof theory (variant of bar recursion)

Intuition: if the computation rules make sense, the system should be normalising

Goal of the presentation

This presentation aims to present a simplified version of

Ulrich Berger "Continuous Semantics for Strong Normalisation" LNCS 3526, 23-34, 2005

This work itsef simplifies the argument in

W.W. Tait "Normal form theorem for bar recursive functions of finite type" *Proceedings of the Second Scandinavian Logic Symposium*, North-Holland, 1971

PCF

Introduced by D. Scott in 1969

"A type-theoretical alternative to CUCH, ISWIM and OWHY"

Published in *Theoret. Comput. Sci.* 121 (1993), no. 1-2, 411–440.

This was the basis of the LCF system

PCF

G. Plotkin "LCF considered as a programming language"

Theoretical Computer Science, 5:223-255, 1977

Simply typed λ-calculus with with base types o, ι and constants

Basic operations

$tt : o,\ ff : o,\ k_n : \iota,\ (+1) : \iota \to \iota,\ (-1) : \iota \to \iota,\ Z : \iota \to o$

$\supset_\iota : o, \iota, \iota \to \iota, \quad \supset_o : o, o, o \to o$

$Y_\sigma : (\sigma \to \sigma) \to \sigma$

Operational semantics

$$\frac{}{\lambda x.t \Downarrow \lambda x.t} \qquad \frac{t \Downarrow \lambda x.t' \qquad t'(x = u) \Downarrow v}{t\ u \Downarrow v}$$

$$\frac{a \Downarrow tt \qquad b \Downarrow v}{\supset\ a\ b\ c\ \ \Downarrow v} \qquad \frac{a \Downarrow ff \qquad c \Downarrow v}{\supset\ a\ b\ c\ \ \Downarrow v}$$

$$\frac{f\ (Y\ f) \Downarrow v}{Y\ f \Downarrow v}$$

$$\frac{a \Downarrow k_n}{(+1)\ a \Downarrow k_{n+1}} \qquad \frac{a \Downarrow k_{n+1}}{(-1)\ a \Downarrow k_n} \qquad \frac{a \Downarrow k_0}{Z\ a \Downarrow tt} \qquad \frac{a \Downarrow k_{n+1}}{Z\ a \Downarrow ff}$$

Denotational semantics

A *domain* is a complete partial order D, with a least element $\perp$ and a top element $\top$

If D, E are domains, $[D \rightarrow E]$ is the complete lattice of *continuous* functions, i.e. monotone and such that $f(\vee_{i \in I} X_i) = \vee_{i \in I} f(X_i)$ for *directed* families (X_i)

We have natural choices for D_ι and D_o

$D_{\sigma \rightarrow \tau} = [D_\sigma \rightarrow D_\tau]$

We have natural choices for $[\![c]\!] \in D_\sigma$ if $c : \sigma$

$[\![Y]\!]\ f = \bigvee_{n \in \mathbb{N}} f^n\ \perp$ so that $[\![Y]\!] \in [[D_\sigma \rightarrow D_\sigma] \rightarrow D_\sigma]$

Denotational semantics

Given $\rho : \mathcal{V}_\sigma \to D_\sigma$ and $t : \sigma$ we define $[\![t]\!]_\rho \in D_\sigma$ by induction on t

$$[\![x]\!]_\rho = \rho(x)$$

$$[\![\lambda x.t]\!]_\rho = u \longmapsto [\![t]\!]_{(\rho,x=u)}$$

$$[\![t\ u]\!]_\rho = [\![t]\!]_\rho\ [\![u]\!]_\rho$$

$$[\![c]\!]_\rho = [\![c]\!]$$

Adequacy theorem

Theorem: *For any closed term* t *of base type* ι *and any value* k_n *we have* $[\![t]\!] = n$ *iff* $t \Downarrow k_n$

For instance $[\![t]\!] = 0$ iff $t \Downarrow k_0$

Application: transformation of programs

Assume we have a program $t = C[u]$ having u as a subprogram

If $[\![u]\!] = [\![u']\!]$ then $[\![t]\!] = [\![C[u]]\!] = [\![C[u']]\!]$

This follows from the *compositionality* property of the denotational semantics

If $t \Downarrow k_0$ then $[\![t]\!] = 0$ hence $[\![C[u']]\!] = 0$

Hence *by the adequacy theorem* $C[u'] \Downarrow k_0$

Elegant way of proving the equivalence of programs (for instance for justification of compiler optimisations)

Avoids messy syntactical details

Adequacy theorem

Plotkin's result is for a *simply typed* language

Proof by induction on the types, reminiscent of reducibility, by introduction of a *computability* predicate

The adequacy result holds for *untyped* languages!

In some sense, untyped λ-calculus has a type structure

Finite elements

$d \in D$ is finite iff $d \leq \bigvee_{i\in I} \alpha_i$ implies $d \leq \bigvee_{i\in K} \alpha_i$ for some finite $K \subseteq I$

The finite elements represent *observable pieces of information* about a program

0: the program t reduces to 0

$0 \rightarrow 0$: if we apply t to 0 the result $t\ 0$ reduces to 0

$\perp \rightarrow 0$: if we apply t to a looping program l the result $t\ l$ reduces to 0

For the last example this means intuitively that the program t does not even look at its argument during the computation

Finite elements

If d_1, d_2 are finite so is $d_1 \vee d_2$

Algebraic domains: any element is the sup of the set of finite elements below it

If D, E are algebraic then $[D \rightarrow E]$ is algebraic: the finite elements are exactly finite sups of step functions $d \rightarrow e$

$(d \rightarrow e)\ d' = e$ if $d \leq d'$

$(d \rightarrow e)\ d' = \perp$ otherwise

Finite elements

In set theory $\iota, \iota \to \iota, \ldots$ have greater and greater cardinality

For each type σ the finite elements of D_σ form a countable set

Adequacy theorems

S. Abramsky "Domain theory in logical form."
Annals of Pure and Applied Logic, 51:1-77, (199)1.

R. Amadio and P.L. Curien *Domains and Lambda-Calculi.*
Cambridge tracts in theoretical computer science, 46, (1997).

H. Barendregt, M. Coppo and M. Dezani-Ciancaglini
"A filter lambda model and the completeness of type assignment."
J. Symbolic Logic 48 (1983), no. 4, 931–940 (1984).

P. Martin-Löf "Lecture note on the domain interpretation of type theory."
Workshop on Semantics of Programming Languages, Chalmers, (1983).

An untyped programming language

$$t \ ::= \ n \mid t\ t \mid \lambda x.t \qquad\qquad n \ ::= \ x \mid c \mid f$$

Two kind of constants: *defined* $f, g, \ldots$ and *primitive* $c, c', \ldots$

f is defined by equations (computation rules) of the form

$$f\ x_1\ \ldots\ x_n\ (c\ y_1\ \ldots\ y_k) \rightarrow u$$

Each constant has an arity $ar(f) = n + 1, ar(c) = k$

We write $h, h', \ldots$ for a constant f or c

Operational semantics

$$\frac{}{\lambda x.t \Downarrow \lambda x.t} \qquad \frac{}{c\ \vec{t} \Downarrow c\ \vec{t}} \qquad \frac{|\vec{t}| < ar(h)}{h\ \vec{t} \Downarrow h\ \vec{t}}$$

$$\frac{t \Downarrow \lambda x.t' \qquad t'(x = u) \Downarrow v}{t\ u \Downarrow v}$$

$$\frac{t \Downarrow c\ \vec{t} \qquad u(\vec{x} = \vec{u}, \vec{y} = \vec{t}) \Downarrow v}{f\ \vec{u}\ t \Downarrow v}$$

We suppose $f\ \vec{x}\ (c\ \vec{y}) = u$

Finite elements

Given a set of constants c with arity $ar(c) \in \mathbb{N}$

$$U, V \ ::= \ \Delta \mid U \to V \mid U \cap V \mid c\,\vec{U} \mid \nabla$$

If $\vec{U}$ is a vector $U_1, \ldots, U_m$ we write $\vec{U} \to U$ for

$$U_1 \to (\cdots \to (U_m \to U) \ldots)$$

and $c\,\vec{U}$ for

$$c\,U_1\ \ldots\ U_m$$

Finite elements as set of closed programs

Let Λ be the set of all programs

Δ is Λ, ∇ is $\emptyset$

$c\ U_1\ \ldots\ U_k = \{t \mid t \Downarrow c\ u_1\ \ldots\ u_k,\ u_i \in U_i\}$

$U \rightarrow V$ is the set of programs t such that t computes to $\lambda x.t'$ or to $h\ \vec{t}$, $|\vec{t}| < ar(h)$ and $\forall u \in U.\ t\ u \in V$

$U \cap V = \{t \mid t \in U\ \wedge\ t \in V\}$

Meet-semi lattice

$$\nabla \subseteq U \subseteq \Delta$$

$$c\ U_1\ \ldots\ U_k \cap c\ U_1'\ \ldots\ U_k' = c\ (U_1 \cap U_1')\ \ldots\ (U_1 \cap U_k')$$

$$c\ U_1\ \ldots\ U_k \cap (U \rightarrow V) = \nabla \qquad c\ U_1\ \ldots\ U_k \cap c'\ U_1'\ \ldots\ U_l' = \nabla$$

$$(U \rightarrow V) \cap (U \rightarrow V') = U \rightarrow (V \cap V')$$

$$U' \subseteq U,\ V \subseteq V' \Rightarrow (U \rightarrow V) \subseteq U' \rightarrow V'$$

Key property

Lemma: *We have* $\cap_{i\in I}(U_i \rightarrow V_i) \subseteq U \rightarrow V$ *iff* $(\cap_{i\in L} V_i) \subseteq V$ *where* $L = \{i \in I \mid U \subseteq U_i\}$

This holds only, a priori, for the *formal* inclusion relation

Decidability

Given U, V we can decide whether $U \subseteq V$ or not

Filters

A *filter* α is a set of types such that

(1) $\Delta \in \alpha$

(2) if $U, V \in \alpha$ then $U \cap V \in \alpha$

(3) if $U \in \alpha$ and $U \subseteq V$ then $V \in \alpha$

These elements are ordered by inclusion

$\uparrow (U \cap V) = \uparrow U \vee \uparrow V$

There is a least element $\perp = \uparrow \Delta$ and a top element $\top = \uparrow \nabla$

We identify U and $\uparrow U$

Filters

The poset of all these filters is a complete lattice D

This poset is *algebraic*: any element is the directed sup of all finite elements below it

Notice that the greatest element $\top$ is finite!

The finite elements of D are *exactly* the types

Filters

This domain D contains 0, s 0, but also $\mathsf{s}\ \perp,\ \mathsf{s}\ (\mathsf{s}\ \perp), \ldots$

We have a continuous function $\mathsf{s} : \mathsf{D} \rightarrow \mathsf{D}$

D contains the sup of these elements ω such that $\omega = \mathsf{s}\ \omega$

$$\omega = \{\perp, \mathsf{s}\ \perp,\ \mathsf{s}\ (\mathsf{s}\ \perp), \ldots\}$$

Filters

We have an application operation on D

$$\alpha\ \beta = \{\Delta\} \cup \{V \mid \exists U.\ [U \rightarrow V] \in \alpha\ \wedge\ U \in \beta\}$$

Notice that

$\perp\ \ \beta = \perp$

$\top\ \beta = \top$

Typing rules

$$\frac{(x{:}U) \in \Gamma}{\Gamma \vdash x : U} \qquad \frac{\Gamma, x : U \vdash t : V}{\Gamma \vdash \lambda x.t : U \to V} \qquad \frac{\Gamma \vdash t : U \to V \qquad \Gamma \vdash u : U}{\Gamma \vdash t\ u : V}$$

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash t : V}{\Gamma \vdash t : U \cap V} \qquad \frac{\Gamma \vdash t : U \qquad U \subseteq V}{\Gamma \vdash t : V} \qquad \frac{}{\Gamma \vdash t : \Delta}$$

Typing rules for constants

$$\frac{}{\vdash c : \vec{U} \rightarrow c\ \vec{U}}$$

$$\frac{\vec{x} : \vec{U}, \vec{y} : \vec{V} \vdash u : U}{\vdash f : \vec{U} \rightarrow (c\ \vec{V}) \rightarrow U}$$

We suppose $f\ \ \vec{x}\ \ (c\ \vec{y}) = u$

$$\frac{}{\vdash f : \vec{U} \rightarrow \nabla \rightarrow \nabla}$$

Typing rules for constants

If we have $0, \mathsf{s}, \mathsf{add}$ with the equations

$$\mathsf{add}\ x\ 0 = x \qquad \mathsf{add}\ x\ (\mathsf{s}\ y) = \mathsf{s}\ (\mathsf{add}\ x\ y)$$

then we have the typing rules

$$\frac{}{\mathsf{add} : U \to 0 \to U} \qquad \frac{x{:}U, y{:}W \vdash \mathsf{add}\ x\ y : V}{\mathsf{add} : U \to (\mathsf{s}\ W) \to \mathsf{s}\ V}$$

Types and finite elements

Δ corresponds to $\bot$

$U \rightarrow V$ corresponds to the step function defined by

$[U \rightarrow V]\ U' = V$ if $U \leq U'$

$[U \rightarrow V]\ U' = \bot$ otherwise

∇ corresponds to $\top$, the top element of the domain

Denotational semantics

$[\![t]\!]_\rho \in \mathsf{D}$ for $\rho : \mathcal{V} \rightarrow \mathsf{D}$

$[\![c]\!]$ (res. $[\![f]\!]$) is the filter of all types U such that $\vdash d : U$ (resp. $\vdash f : U$)

$[\![x]\!]_\rho = \rho(x)$

$[\![t\ u]\!]_\rho = [\![t]\!]_\rho\ [\![u]\!]_\rho$

$[\![\lambda x.t]\!]_\rho = \alpha \longmapsto [\![t]\!]_{(\rho, x=\alpha)}$

Typing rules and denotational semantics

Theorem: We have $\vdash t : U$ iff $U \leq [\![t]\!]$

More generally, we have $x_1{:}U_1, \ldots, x_n{:}U_n \vdash t : U$ iff

$$U \leq [\![t]\!]_{x_1=U_1,\ldots,x_n=U_n}$$

Denotational semantics

An alternative approach is to define directly $[\![t]\!]_\rho \in \mathsf{D}$ by

$$[\![t]\!]_\rho = \{U \mid x_1{:}U_1, \ldots, x_n{:}U_n \vdash t : U,\ U_i \in \rho(x_i)\}$$

Lemma: $\Gamma \vdash \lambda x.t : U \to V$ *iff* $\Gamma, x{:}U \vdash t : V$

Denotational semantics

Theorem: *We have*

$[\![x]\!]_\rho = \rho(x)$

$[\![t\ u]\!]_\rho = [\![t]\!]_\rho\ [\![u]\!]_\rho$

$[\![\lambda x.t]\!]_\rho\ \alpha = [\![t]\!]_{(\rho,x=\alpha)}$

if $[\![t]\!]_{\rho,x=\alpha} = [\![u]\!]_{\nu,y=\alpha}$ *for all* α *then* $[\![\lambda x.t]\!]_\rho = [\![\lambda y.u]\!]_\nu$

Denotational semantics

This alternative characterisation of the semantics of β-conversion is described in

R. Hindley and J. Seldin "*Combinators and λ-calculus*", University Press, 1986

and goes back to G. Berry

Adequacy theorem

Theorem: *If* $\vdash t : U$ *then* $t \in U$

Corollary: *If* $[\![t]\!] = c\ \vec{U}$ *then there exists* $\vec{u}$ *such that* $t \Downarrow c\ \vec{u}$

Application: Gödel system T

Weak version of the normalisation theorem in a semantical way

The constants of Gödel system T are $0, \mathsf{s}, \mathsf{natrec}$

$\mathsf{natrec}\ u\ v\ 0 = u$ $\qquad$ $\mathsf{natrec}\ u\ v\ (\mathsf{s}\ m) = v\ m\ (\mathsf{natrec}\ u\ v\ m)$

The base type is ι and $0 : \iota,\ \mathsf{s} : \iota \rightarrow \iota$ and $\mathsf{natrec} : \sigma \rightarrow (\iota \rightarrow \sigma \rightarrow \sigma) \rightarrow \iota \rightarrow \sigma$

Application: Gödel system T

To each type σ we associate a predicate Tot_σ on D

$a \in$ D is a *total* integer iff $a = \mathsf{s}^k\ 0$ for some $k \in \mathbb{N}$

$Tot_{\sigma \to \tau}(b)$ means that $Tot_\sigma(a)$ implies $Tot_\tau(b\ a)$

If Γ is a context define $Tot_\Gamma(\rho)$ to mean $Tot_\sigma(\rho(x))$ for all $x{:}\sigma$ in Γ

Application: Gödel system T

Lemma 1: *If* $\Gamma \vdash t : \sigma$ *and* $Tot_\Gamma(\rho)$ *then* $Tot_\sigma([\![t]\!]_\rho)$. *In particular, if* $\vdash t : \sigma$ *then* $Tot_\sigma([\![t]\!])$.

Lemma 2: *If* $Tot_\sigma(a)$ *then* $a \neq \perp$

Corollary: *If* $\vdash t : \iota$ *then* $t \Downarrow \mathsf{0}$ *or there exists* t' *such that* $t \Downarrow \mathsf{s}\ t'$

Strong Normalisation

As explained in the talk of Benjamin Grégoire for the (total) correctness of the type-checking algorithm we need a (strong) normalisation theorem

B. Grégoire and X. Leroy
A compiled implementation of strong reduction, *ICFP* 2002, 235-246.

Strong Normalisation

$\mathcal{N}$ subset of *strongly normalisable* terms

We write w, w' for strongly normalisable terms

Simple terms

$$s \ ::= \ x \mid s \ w \mid f \ \vec{w} \ s$$

Head-reduction

$$(\lambda x.u)\ v \succ u(x = v)$$

$$f\ \vec{u}\ (c\ \vec{v}) \succ u(\vec{x} = \vec{u}, \vec{y} = \vec{v})$$

$$\frac{u \succ u'}{u\ v \succ u'\ v} \qquad \frac{u \succ u'}{f\ \vec{u}\ u \succ f\ \vec{u}\ u'}$$

We say that u is of *head-redex form* iff there exists u' such that $u \succ u'$

Head-reduction and reduction

We let $\mathcal{S} \subseteq \mathcal{N}$ be the set of strongly normalisable terms that reduce to a simple term

$\mathcal{S} \subseteq \mathcal{N} \subseteq \Lambda$

We write $u \rightarrow u'$ ordinary reduction and

$$\rightarrow (u) = \{u' \mid u \rightarrow u'\}$$

Saturated set

$X \subseteq \Lambda$ is *saturated* iff

(CR1) $\mathcal{S} \subseteq X \subseteq \mathcal{N}$

(CR2) if $t \in X$ then $\rightarrow (t) \subseteq X$

(CR3) if t is of head-redex form and $\rightarrow (t) \subseteq X$ then $t \in X$

Saturated subsets

lemma: *If $I \neq \emptyset$ and X_i saturated then $\cap_{i \in I} X_i$ are saturated*

If $X, Y \subseteq \Lambda$ then we define

$$X \to Y = \{t \in \Lambda \mid \forall u \in X.\ t\ u \in Y\}$$

lemma: *If X and Y are saturated then so is $X \to Y$*

Saturated subsets

If $X_1, \ldots, X_k \subseteq \Lambda$ then $c\ X_1\ \ldots\ X_k$ is the set of terms defined inductively as follows

if $t_1 \in X_1, \ldots, t_k \in X_k$ then $c\ \vec{t} \in c\ \vec{X}$

if $t \in \mathcal{S}$ then $t \in c\ \vec{X}$

if t is of head-redex form and $\rightarrow (t) \subseteq c\ \vec{X}$ then $t \in c\ \vec{X}$

Finite elements as saturated sets

We consider the new set of finite elements (types)

$$U \;::= \Delta \mid W \qquad W, V \;::=\; c\,\vec{W} \mid W \cap W \mid W \to W \mid \nabla$$

Each finite element W can be interpreted as a saturated set

Notice that if $c\,\vec{u} \in W$ then $|\vec{u}| = ar(c)$

Meet-semi lattice

$\nabla \subseteq U \subseteq \Delta$

$c\ W_1\ \ldots\ W_k \cap c\ W_1'\ \ldots\ W_k' = c\ (W_1 \cap W_1')\ \ldots\ (W_1 \cap W_k')$

$c\ W_1\ \ldots\ W_k \cap (W \to V) = \nabla \qquad c\ W_1\ \ldots\ W_k \cap c'\ W_1'\ \ldots\ W_l' = \nabla$

$(W \to V) \cap (W \to V') = W \to (V \cap V')$

$W' \subseteq W,\ V \subseteq V' \Rightarrow (W \to V) \subseteq W' \to V'$

Meet-semi lattice

The filters over this lattice define a new domain E

As before we have an application

$$\alpha\ \beta = \{\Delta\} \cup \{W \mid \exists V.\ V \in \beta \wedge (V \to W) \in \alpha\}$$

Notice that $\alpha\ \perp = \perp$ for all α

Strict semantics

We consider the new typing system with only judgements of the form $\Gamma \vdash t : W$

Lemma: If $\vdash t : W$ then t belongs to the saturated set W

Typing rules

$$\frac{(x{:}W) \in \Gamma}{\Gamma \vdash x : W} \qquad \frac{\Gamma, x : W \vdash t : V}{\Gamma \vdash \lambda x.t : W \to V} \qquad \frac{\Gamma \vdash t : W \to V \qquad \Gamma \vdash u : W}{\Gamma \vdash t\ u : V}$$

$$\frac{\Gamma \vdash t : W \qquad \Gamma \vdash t : V}{\Gamma \vdash t : W \cap V}$$

$$\frac{\Gamma \vdash t : W \qquad W \subseteq V}{\Gamma \vdash t : V}$$

Typing rules for constants

$$\frac{}{\vdash c : \vec{W} \rightarrow c\ \vec{W}}$$

$$\frac{\vec{x} : \vec{W}, \vec{y} : \vec{V} \vdash u : W}{\vdash f : \vec{W} \rightarrow (c\ \vec{V}) \rightarrow W}$$

We suppose $f\ \vec{x}\ (c\ \vec{y}) = u$

$$\frac{}{\vdash f : \vec{W} \rightarrow \nabla \rightarrow \nabla}$$

Strict semantics

We define $[t]_\rho \in \mathsf{E}$ to be the following filter: $U \in [t]_\rho$ iff

(1) $U = \Delta$, or

(2) $x_1{:}W_1, \ldots, x_n{:}W_n \vdash t : U$ in the new system, with $W_i \in \rho(x_i)$

Strict semantics

Theorem: *We have*

$[x]_\rho = \rho(x)$

$[t\ u]_\rho = [t]_\rho\ [u]_\rho$

$[\lambda x.t]_\rho\ \alpha = [t]_{(\rho,x=\alpha)}$ if $\alpha \neq \perp$

if $[t]_{\rho,x=\alpha} = [u]_{\nu,y=\alpha}$ *for all* $\alpha \neq \perp$ *then* $[\lambda x.t]_\rho = [\lambda y.u]_\nu$

Strict semantics

Theorem: If $[t] \neq \perp$ then t is strongly normalisable

If $[\![u]\!]_\rho \neq \perp$ then

$$[\![(\lambda x.t)\ u]\!]_\rho = [\![t(x = u)]\!]_\rho$$

Application: Gödel's system T

Theorem: *If* $\Gamma \vdash t : \sigma$ *and* $Tot_\Gamma(\rho)$ *then* $Tot_\sigma([t]_\rho)$

The crucial case is the application: if $\vdash t : \sigma \rightarrow \tau$ and $u : \sigma$ then by induction $Tot_{\sigma\rightarrow\tau}([t])$ and $Tot_\tau([u])$. Hence $[u] \neq \perp$ and

$$[t\ u] = [t]\ [u]$$

Corollary: *If* $\vdash t : \sigma$ *then* t *is strongly normalisable*

Interpretation of $\top$

The special element $\top \in \mathsf{D}$ satisfies

$$\top\ \beta = \top$$

if $\beta \neq \perp$, but also

$$f\ \alpha_1\ \ldots\ \alpha_n\ \top = \top$$

if $\alpha_1 \neq \perp,\ \ldots,\ \alpha_n \neq \perp$