Proof of normalisation using domain theory

Thierry Coquand and Arnaud Spivak

Aug. 24, 2005
Proof of normalisation using domain theory

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 itself simplifies the argument in

W.W. Tait “Normal form theorem for bar recursive functions of finite type”
Proof of normalisation using domain theory

PCF

Introduced by D. Scott in 1969

“A type-theoretical alternative to CUCH, ISWIM and OWHY”


This was the basis of the LCF system
Proof of normalisation using domain theory

PCF

G. Plotkin “LCF considered as a programming language”


Simply typed $\lambda$-calculus with with base types $o, t$ and constants

Basic operations

$tt : o, \  ff : o, \  k_n : t, \  (+1) : t \rightarrow t, \  (−1) : t \rightarrow t, \  Z : t \rightarrow o$

$\supset : o, t, t \rightarrow t, \  \supset : o, o, o \rightarrow o$

$Y_\sigma : (\sigma \rightarrow \sigma) \rightarrow \sigma$
Proof of normalisation using domain theory

Operational semantics

\[
\begin{align*}
\lambda x.t & \Downarrow \lambda x.t & t \Downarrow \lambda x.t' & t'(x = u) \Downarrow v \\
& & t u \Downarrow v \\
a \Downarrow tt & b \Downarrow v & a \Downarrow ff & c \Downarrow v \\
\Downarrow a b c & \Downarrow v \\
& & & f (Y f) \Downarrow v \\
& & & Y f \Downarrow v \\
& & & a \Downarrow k_n \\
& & & (+1) a \Downarrow k_{n+1} \\
& & & a \Downarrow k_{n+1} \\
& & & a \Downarrow k_0 \\
& & & a \Downarrow k_{n+1} \\
& & & Z a \Downarrow tt \\
& & & Z a \Downarrow ff
\end{align*}
\]
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, $[D \to E]$ is the complete lattice of continuous functions, i.e. monotone and such that $f(\bigvee_{i \in I} X_i) = \bigvee_{i \in I} f(X_i)$ for directed families $(X_i)$.

We have natural choices for $D_i$ and $D_o$:

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

We have natural choices for $\llbracket c \rrbracket \in D_{\sigma}$ if $c : \sigma$

$$\llbracket Y \rrbracket f = \bigvee_{n \in \mathbb{N}} f^n \perp \text{ so that } \llbracket Y \rrbracket \in [[D_{\sigma} \to D_{\sigma}] \to D_{\sigma}]]$$
Denotational semantics

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

- $[x]_\rho = \rho(x)$
- $[\lambda x.t]_\rho = u \mapsto [t]_{(\rho, x=u)}$
- $[t u]_\rho = [t]_\rho [u]_\rho$
- $[c]_\rho = [c]$
Proof of normalisation using domain theory

Adequacy theorem

**Theorem:** For any closed term \( t \) of base type \( \iota \) and any value \( k_n \) we have

\[
\llbracket t \rrbracket = n \iff t \downarrow k_n
\]

For instance \( \llbracket t \rrbracket = 0 \iff t \downarrow k_0 \)
Proof of normalisation using domain theory

**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
Proof of normalisation using domain theory

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 \( t \ 0 \) 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 \( d_1 \lor d_2 \)

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

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

\[(d \to e) \ d' = e \text{ if } d \leq d'\]

\[(d \to e) \ d' = \bot \text{ otherwise}\]
Proof of normalisation using domain theory

Finite elements

In set theory \( \iota, \iota \rightarrow \iota, \ldots \) have greater and greater cardinality

For each type \( \sigma \) the finite elements of \( D_\sigma \) form a countable set
Adequacy theorems

S. Abramsky “Domain theory in logical form.”

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.”

P. Martin-Löf “Lecture note on the domain interpretation of type theory.”
An untyped programming language

\[ t ::= n \mid t \; t \mid \lambda x.t \quad n ::= x \mid c \mid f \]

Two kind of constants: \textit{defined} \( f, g, \ldots \) and \textit{primitive} \( c, c', \ldots \)

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

\[ f \; x_1 \; \ldots \; x_n \; (c\; y_1 \; \ldots \; y_k) \to u \]

Each constant has an arity \( \text{ar}(f) = n + 1, \text{ar}(c) = k \)

We write \( h, h', \ldots \) for a constant \( f \) or \( c \).
Proof of normalisation using domain theory

Operational semantics

\[
\begin{align*}
\lambda x.t \Downarrow \lambda x.t & \quad c \vec{t} \Downarrow c \vec{t} & \quad |\vec{t}| < ar(h) \\
\hline
h \vec{t} \Downarrow h \vec{t} & \quad t \Downarrow \lambda x.t' & \quad t'(x = u) \Downarrow v \\
\hline
\lambda x.t \Downarrow \lambda x.t & \quad t' & \quad v \\
\hline
\end{align*}
\]

We suppose \( f \vec{t} (c \vec{y}) = u \)
Proof of normalisation using domain theory

Finite elements

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

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

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

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

and $c \vec{U}$ for

$$c \ U_1 \ldots \ U_m$$
Finite elements as set of closed programs

Let $\Lambda$ be the set of all programs

$\Delta$ is $\Lambda$, $\nabla$ 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 $\forall u \in U.\ t\ u \in V$

$U \cap V = \{ t \mid t \in U \land t \in V \}$
Proof of normalisation using domain theory

**Meet-semi lattice**

\[ \nabla \subseteq U \subseteq \Delta \]

\[ c \ U_1 \ldots U_k \cap c \ U'_1 \ldots U'_k = c \ (U_1 \cap U'_1) \ldots (U_k \cap U'_k) \]

\[ c \ U_1 \ldots U_k \cap (U \rightarrow V) = \nabla \quad c \ U_1 \ldots U_k \cap c' \ U'_1 \ldots U'_k = \nabla \]

\[ (U \rightarrow V) \cap (U \rightarrow V') = U \rightarrow (V \cap V') \]

\[ U' \subseteq U, \ V \subseteq V' \Rightarrow (U \rightarrow V) \subseteq U' \rightarrow V' \]
Proof of normalisation using domain theory

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
A *filter* $\alpha$ 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

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

There is a least element $\bot = \uparrow \Delta$ 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 $s\ \perp$, $s\ (s\ \perp)$, 

We have a continuous function $s : D \to D$

$D$ contains the sup of these elements $\omega$ such that $\omega = s\ \omega$

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

We have an application operation on $D$

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

Notice that

$$\bot \beta = \bot$$

$$\top \beta = \top$$
Proof of normalisation using domain theory

**Typing rules**

\[(x:U) \in \Gamma \quad \Gamma, x : U \vdash t : V \quad \Gamma \vdash t : U \rightarrow V \quad \Gamma \vdash u : U \]

\[\Gamma \vdash x : U \quad \Gamma \vdash \lambda x.t : U \rightarrow V \quad \Gamma \vdash t u : V\]

\[\Gamma \vdash t : U \quad \Gamma \vdash t : V \quad \Gamma \vdash t : U \subseteq V \]

\[\Gamma \vdash t : U \cap V \quad \Gamma \vdash t : V \quad \Gamma \vdash t : \Delta\]
Proof of normalisation using domain theory

**Typing rules for constants**

\[
\frac{}{\Gamma \vdash c : \vec{U} \rightarrow c \vec{U}}
\]

\[
\frac{\bar{x} : \vec{U}, \bar{y} : \vec{V} \vdash u : U}{\Gamma \vdash f : \vec{U} \rightarrow (c \vec{V}) \rightarrow U}
\]

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

\[
\frac{}{\Gamma \vdash f : \vec{U} \rightarrow \nabla \rightarrow \nabla}
\]
Proof of normalisation using domain theory

**Typing rules for constants**

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

\[
\text{add } x \ 0 = x \quad \text{add } x \ (s \ y) = s \ (\text{add } x \ y)
\]

then we have the typing rules

\[
\text{add } : U \rightarrow 0 \rightarrow U \\
\frac{x:U, y:W \vdash \text{add } x \ y : V}{\text{add } : U \rightarrow (s \ W) \rightarrow s \ V}
\]
Proof of normalisation using domain theory

Types and finite elements

Δ corresponds to ⊥

U → V corresponds to the step function defined by

[U → V] U' = V if U ≤ U'

[U → V] U' = ⊥ otherwise

∇ corresponds to ⊤, the top element of the domain
Proof of normalisation using domain theory

**Denotational semantics**

\[
[t]_\rho \in D \text{ for } \rho : \mathcal{V} \to D
\]

\[c\] (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 \mapsto [t]_\rho(x=\alpha)
Proof of normalisation using domain theory

**Typing rules and denotational semantics**

**Theorem:** We have $\vdash t : U$ iff $U \leq \llbracket t \rrbracket$

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

$U \leq \llbracket t \rrbracket_{x_1=U_1, \ldots, x_n=U_n}$
Proof of normalisation using domain theory

Denotational semantics

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

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

**Lemma:** $\Gamma \vdash \lambda x.t : U \rightarrow V$ 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 \(\alpha\) then \([\lambda x.t]_\rho = [\lambda y.u]_\nu\)
Denotational semantics

This alternative characterisation of the semantics of $\beta$-conversion is described in

R. Hindley and J. Seldin “Combinators and $\lambda$-calculus”, University Press, 1986

and goes back to G. Berry
Proof of normalisation using domain theory

Adequacy theorem

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

**Corollary:** If $\llbracket t \rrbracket = c \vec{U}$ then there exists $\vec{u}$ such that $t \Downarrow c \vec{u}$
Proof of normalisation using domain theory

Application: Gödel system $T$

Weak version of the normalisation theorem in a semantical way

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

\[
\text{natrec } u \ v \ 0 = u \quad \text{natrec } u \ v \ (s \ m) = v \ m \ (\text{natrec } u \ v \ m)
\]

The base type is $\iota$ and $0 : \iota$, $s : \iota \to \iota$ and $\text{natrec} : \sigma \to (\iota \to \sigma \to \sigma) \to \iota \to \sigma$
To each type $\sigma$ we associate a predicate $\text{Tot}_\sigma$ on $D$

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

$\text{Tot}_{\sigma \rightarrow \tau}(b)$ means that $\text{Tot}_\sigma(a)$ implies $\text{Tot}_\tau(b, a)$

If $\Gamma$ is a context define $\text{Tot}_\Gamma(\rho)$ to mean $\text{Tot}_\sigma(\rho(x))$ for all $x : \sigma$ in $\Gamma$
Proof of normalisation using domain theory

Application: Gödel system $T$

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

**Lemma 2:** If $\text{Tot}_\sigma(a)$ then $a \neq \bot$

**Corollary:** If $\vdash t : \iota$ then $t \downarrow 0$ or there exists $t'$ such that $t \downarrow s t'$
Proof of normalisation using domain theory

**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
Proof of normalisation using domain theory

\textbf{Strong Normalisation}

\( \mathcal{N} \) subset of \textit{strongly normalisable} terms

We write \( w, w' \) for strongly normalisable terms

\textit{Simple} terms

\[ s ::= x \mid s \, w \mid f \, \vec{w} \, s \]
Proof of normalisation using domain theory

Head-reduction

\[(\lambda x. u) \ v \succ u(x = v)\]

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

\[
\frac{u \succ u'}{u \, v \succ u' \, v} \quad \frac{u \succ u'}{f \, \vec{u} \ u \succ f \, \vec{u} \ u'}
\]

We say that \(u\) is of \textit{head-redex form} iff there exists \(u'\) such that \(u \succ u'\)
We let $S \subseteq \mathcal{N}$ be the set of strongly normalisable terms that reduce to a simple term

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

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

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

\[ X \subseteq \Lambda \text{ is saturated iff} \]

(CR1) \( S \subseteq X \subseteq N \)

(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

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

**Lemma:** If $X$ and $Y$ are saturated then so is $X \rightarrow Y$
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 $t \in S$ then $t \in c \vec{X}$

if $t$ is of head-redex form and $\to (t) \subseteq c \vec{X}$ then $t \in c \vec{X}$
Finite elements as saturated sets

We consider the new set of finite elements (types)

\[ U ::= \Delta \mid W \mid W, V ::= c \vec{W} \mid W \cap W \mid W \rightarrow W \mid \nabla \]

Each finite element \( W \) can be interpreted as a saturated set

Notice that if \( c \vec{u} \in W \) then \( |\vec{u}| = ar(c) \)
Proof of normalisation using domain theory

Meet-semi lattice

\[ \nabla \subseteq U \subseteq \Delta \]

\[ c \ W_1 \ldots W_k \cap c \ W'_1 \ldots W'_k = c \ (W_1 \cap W'_1) \ldots (W_k \cap W'_k) \]

\[ c \ W_1 \ldots W_k \cap \ (W \rightarrow V) = \nabla \quad c \ W_1 \ldots W_k \cap c' \ W'_1 \ldots W'_l = \nabla \]

\[ (W \rightarrow V) \cap (W \rightarrow V') = W \rightarrow (V \cap V') \]

\[ W' \subseteq W, \ V' \subseteq V' \Rightarrow (W \rightarrow V) \subseteq W' \rightarrow V' \]
Proof of normalisation using domain theory

Meet-semi lattice

The filters over this lattice define a new domain E

As before we have an application

\[ \alpha \beta = \{ \Delta \} \cup \{ W \mid \exists V. V \in \beta \land (V \rightarrow W) \in \alpha \} \]

Notice that \( \alpha \perp = \perp \) for all \( \alpha \)
Proof of normalisation using domain theory

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 \( W \)
Proof of normalisation using domain theory

Typing rules

\[
\begin{align*}
(x:W) & \in \Gamma & \Gamma, x : W \vdash t : V & \Gamma \vdash t : W \rightarrow V & \Gamma \vdash u : W \\
\Gamma \vdash x : W & & \Gamma \vdash \lambda x.t : W \rightarrow V & & \Gamma \vdash t \ u : V \\
\Gamma \vdash t : W & & \Gamma \vdash t : V & & \Gamma \vdash t : W \cap V \\
\Gamma \vdash t : W & & W \subseteq V & & \Gamma \vdash t : V
\end{align*}
\]
Proof of normalisation using domain theory

Typing rules for constants

\[ \vdash c : \overrightarrow{W} \rightarrow c \overrightarrow{W} \]

\[ \overrightarrow{x} : \overrightarrow{W}, \overrightarrow{y} : \overrightarrow{V} \vdash u : W \]

\[ \vdash f : \overrightarrow{W} \rightarrow (c \overrightarrow{V}) \rightarrow W \]

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

\[ \vdash f : \overrightarrow{W} \rightarrow \nabla \rightarrow \nabla \]
Proof of normalisation using domain theory

Strict semantics

We define \( [t]_\rho \in \mathbb{E} \) 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 \( W_i \in \rho(x_i) \)
Proof of normalisation using domain theory

Strict semantics

**Theorem:** We have

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

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

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

If \([t]_{\rho, x = \alpha} = [u]_{\nu, y = \alpha} \text{ for all } \alpha \neq \bot\) then \([\lambda x. t]_\rho = [\lambda y. u]_\nu\)
Proof of normalisation using domain theory

\textbf{Strict semantics}

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

If $[u]_\rho \neq \bot$ then

$$[(\lambda x.t) \ u]_\rho = [t(x = u)]_\rho$$
Proof of normalisation using domain theory

Application: Gödel’s system $T$

**Theorem:** If $\Gamma \vdash t : \sigma$ and $\text{Tot}_\Gamma(\rho)$ then $\text{Tot}_\sigma([t]_\rho)$

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

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

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

The special element $\top \in D$ satisfies

$$\top \beta = \top$$

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

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

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