Adequacy of PCF

Natalie Ravenhill

July 14, 2016

Aim of the Project

- Study the operational and denotational semantics of the programming language PCF
- Prove that these semantics are equivalent at base type, by proving a theorem called Adequacy

Proposal up to now...

Week	Date	Task
1	06/06/16	Domains
2	13/06/16	Domains
3	20/06/16	Denotational Semantics
4	27/06/16	Denotational Semantics
5	04/07/16	Operational Semantics
6	11/07/16	Operational Semantics *(Inspection Week)*

Domain Theory

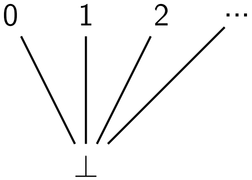
A domain is a set X, an element $\bot \in X$ and a relation $\sqsubseteq \subseteq X \times X$ such that:

- $\forall x \in X.\ \bot \sqsubseteq X$
- $\sqsubseteq$ is a partial order
- For all chains $\underset{x}{\rightharpoonup}$, $\underset{x}{\rightharpoonup}$ has a limit. To prove this there are two properties we must prove
  - $\exists z \in X.\ \forall i.x_i \sqsubseteq z$ *(upper bound)*
  - $\exists z \in X.\ \forall y.(\forall i.x_i \sqsubseteq y) \Rightarrow\ z \sqsubseteq y$ *(least upper bound)*

Domain Theory

Flat Domain of Natural Numbers

$\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$



$$\sqsubseteq \; = \{(\perp, \perp)\} \cup \{(\perp, n), (n, n) \mid n \in \mathbb{N}\}$$

Proved this is a domain

Domain Theory

Continuous Functions

For two domains, $\mathbb{X} = (X, \bot_X, \sqsubseteq_X)$ and $\mathbb{Y} = (Y, \bot_Y, \leq_y)$ The set $Cont(X, Y) = \{f : X \to Y\}$ where:

- $\forall x, x' \in X.\ x \sqsubseteq_X x' \Rightarrow f(x) \leq_Y f(x')$
- $x \in Chain(X) \Rightarrow f(\sqcup x_i) = \sqcup f(x_i)$

The bottom element of this set is $\lambda x.x$.

The relation $\sqsubseteq_{Cont(X,Y)}$ is defined as

$$\sqsubseteq_{Cont(X,Y)} = \{(f, g) \,|\, f, g \in Cont(X, Y) \wedge \forall x \in X.\ f(x) \leq_Y g(x)\}$$

Proved this is a domain.

Fixpoint Theorem

Continuous Functions are used to model fixpoint recursion.

Theorem

Every continuous function $f : X \to X$ has a least fixpoint, which is the limit of the chain $\perp \sqsubseteq f(\perp) \sqsubseteq f^2(\perp) \sqsubseteq \dots$

Proof.

in 2 steps:

1. Prove limit of chain is a fixpoint
2. Limit $\sqsubseteq$ any other fixpoint

□

PCF

Syntax of PCF:

$$
\begin{aligned}
&A ::= \mathrm{Nat} \mid A \to B \\
&e ::= \lambda x : A.e \mid e\ e \mid x \\
&\mid z \mid s(e) \mid case\ (e, z \to e_0\ ,\ s(n) \to e_s \\
&\mid \mathrm{fix}\ x : A\ .\ e
\end{aligned}
$$

Operational Semantics

Small step operational semantics:

$$\frac{}{(\lambda x : A.\ e)\ e' \mapsto [e'/x]e}$$

$$\frac{}{case\ (z, z \to e_0, s(x) \to e_S) \mapsto e_0}$$

$$\frac{}{case\ (s(v), z \to e_0, s(x) \to e_S) \mapsto [v/x]e_S}$$

$$\frac{}{fix\ x : A.e \mapsto [fix\ x : A.e/x]e}$$

Operational Semantics

Coherence Rules:

$$\frac{e_0 \mapsto e_0'}{e_0\ e_1 \mapsto e_0'\ e_1} \qquad \frac{e \mapsto e'}{s(e) \mapsto s(e')}$$

$$\frac{e \mapsto e'}{case\ (e, z \to e_0, s(x) \to e_S) \mapsto case\ (e', z \to e_0, s(x) \to e_S)}$$

Type Safety

i.e. "well typed terms can't go wrong"

Assumed from following two theorems:

Type Preservation

If $\Gamma \vdash e : A$ *and* $e \mapsto e'$, *then* $\Gamma \vdash e' : A$

Type Progress

If $\vdash e : A$ *then* $e \mapsto e'$ *or* e *is a value.*

Typing Rules

We need a typing rule for each expression in PCF:

$$\text{VARIABLES}\ \frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \text{ZERO}\ \frac{}{\Gamma \vdash z : Nat} \qquad \text{SUCC}\ \frac{\Gamma \vdash e : Nat}{\Gamma \vdash s(e) : Nat}$$

$$\text{CASE}\ \frac{\Gamma \vdash e : Nat \qquad \Gamma \vdash e_0 : A \qquad \Gamma, x : Nat \vdash e_s : A}{\Gamma \vdash \ case\ (e, z \to e_0, s(x) \to e_S) \ : A}$$

Typing Rules

We need a typing rule for each expression in PCF:

$$\text{APPLICATION}\ \frac{\Gamma \vdash e : A \to B \qquad \Gamma \vdash e' : A}{\Gamma \vdash e\ e' : B}$$

$$\text{ABSTRACTION}\ \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A.e : A \to B}$$

$$\text{FIX}\ \frac{\Gamma, (x : A) \vdash e : A}{\Gamma \vdash \ \mathit{fix}\ x : A.\ e : A}$$

Type Lemmas

Lemmas we proved for use in Type Safety:

Type Weakening

If $\Gamma \vdash e : A$ *then* $\Gamma, x : C \vdash e : A$

If we add a new variable to the context, this does not chsnge the type of our given expression.

Proof.

By induction on value of e □

Substitution Rules

Variables

$$[e/x]x = e$$

$$[e/x]y = y$$

Zero

$$[e/x]z = z$$

Successor

$$[e/x]s(e') = s([e/x]e')$$

Case

$$[e/x]\ (case\ (e', z \to e_0, s(x) \to e_s)) =$$

$$case\ ([e/x]e', z \to [e/x]e_0, s(x) \to [e/x]e_s)$$

Substitution Rules

Application

$$[e/x]e_0\ e_1 = [e/x]e_0\ ([e/x]e_1)$$

λ **Abstraction**

$$[e/x](\lambda x : A.\ e') = \lambda e : A.\ e' =_\alpha \lambda x : A.\ e'$$

$$[e/x](\lambda y : A.\ e') = \lambda y : A.\ [e/x]e'$$

Type Lemmas

Type Substitution

If $\Gamma \vdash e : A$ *and* $\Gamma, x : A \vdash e' : C$ *then* $\Gamma \vdash [e/x]e' : C$

If we have an expression of type C in a context extended with $x : A$, then we can replace x with this in an expression e' of type C.

Proof.

By induction on e' □

Proof uses Weakening, for example in the case for case

Preservation

Type Preservation

If $\Gamma \vdash e : A$ *and* $e \mapsto e'$, *then* $\Gamma \vdash e' : A$

This says that if $\Gamma \vdash e : A$ and there is an evaluation rule mapping the expression e to another one in one step, then the new expression has the same type.

Proof.

By induction on on evaluation rules $e \mapsto e'$ for each possible value of e $\square$

Proof uses Substitution for example, in the case of $(\lambda x : A.e)e'$

Progress

Type Progress

If $\vdash e : A$ then $e \mapsto e'$ or e is a value.

This says that for any closed term (one with an empty typing context), it either evaluates to another expression or is a value. A value is zero, successor of a number, or a function $\lambda x : A.e$.

Proof.

By induction on evaluation rules $e \mapsto e'$ for each possible value of e $\square$

Denotational Semantics

Denotational Semantics use the **Scott Model**, defined as a function from a typing context to a domain:

$$[\![-]\!] : \mathit{Type} \to \mathit{Domain}$$

Defined inductively as:

$$[\![\mathit{Nat}]\!] = \mathbb{N}_\perp$$

$$[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]$$

Denotation of Typing Contexts

On typing contexts the function is:

$$\llbracket - \rrbracket_{Ctx} : Context \to Domain$$

Defined inductively as:

$$\llbracket \cdot \rrbracket_{Ctx} = \mathbb{1}$$

$$\llbracket \Gamma, x : A \rrbracket_{Ctx} = \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket$$

Where $\mathbb{1}$ is the single element domain and non empty contexts are products of domains. I still need to prove that these are domains!

Denotational Semantics of Terms

Given a well typed term $\Gamma \vdash e : A$ we have

$$[\![\Gamma \vdash e : A]\!] \in [\![\Gamma]\!]_{Ctx} \to [\![A]\!]$$

We can now inductively define this for every e:

Denotational Semantics of Terms

Variables

$$[\![\Gamma \vdash x_i : A_i]\!] = \lambda\gamma \in [\![\Gamma]\!].\pi_i(\gamma)$$

Zero

$$[\![\Gamma \vdash z : Nat]\!]\gamma = 0$$

Successor

$$[\![\Gamma \vdash s(e) : Nat]\!]\gamma = \text{Let } v = [\![\Gamma \vdash e : Nat]\!]\gamma \text{ in}$$

$$\begin{cases} v+1 & \text{if } v \neq \bot \\ \bot & \text{if } v = \bot \end{cases}$$

Denotational Semantics of Terms

Case

$[\![\Gamma \vdash case\ (e, z \mapsto e_0, s(y) \mapsto e_S) : C]\!]\gamma = \text{Let } v = [\![\Gamma \vdash e : Nat]\!]\gamma \text{ in}$

$$\begin{cases} [\![\Gamma \vdash e_0 : C]\!]\gamma & \text{if } v = 0 \\ [\![\Gamma \vdash e_S : C]\!]\gamma & \text{if } v = n+1 \\ \perp & \text{if } v = \perp \end{cases}$$

Denotational Semantics of Terms

Application

$$\llbracket \Gamma \vdash e\ e' : B \rrbracket \gamma = \text{Let } f = \llbracket \Gamma \vdash e : A \to B \rrbracket \gamma \text{ in}$$
$$\text{Let } v = \llbracket \Gamma \vdash e' : A \rrbracket \gamma$$
$$\text{in } f(v)$$

λ abstraction

$$\llbracket \Gamma \vdash \lambda x : A.e : A \to B \rrbracket \gamma = \lambda a \in \llbracket A \rrbracket . \llbracket \Gamma, x : A \vdash e : A \rrbracket (\gamma, a/x)$$

Fixpoint

$$\llbracket \Gamma \vdash \mathit{fix}\, x : A.e : A \rrbracket \gamma = \mathit{fix}_{\llbracket A \rrbracket} (\lambda a \in \llbracket A \rrbracket . \llbracket \Gamma, x : A \vdash e : A \rrbracket (\gamma, a/x)$$

Correctness

Now we can relate the Operational Semantics to the Denotational semantics with the following theorem:

Theorem

If $\Gamma \vdash e : A$ *and* $e \mapsto e'$ *and* $\gamma \in [\![\Gamma]\!]$, *then* $[\![\Gamma \vdash e : A]\!]\gamma = [\![\Gamma' \vdash e' : A]\!]\gamma$

which says that for a well typed expression e, if it maps to another expression e', then its denotation will be equal to that of the new expression in its new context.

Proof.

by induction on $e \mapsto e'$, so the cases are on the evaluation rules. We can use the fact that $f(\mathit{fix}(f)) = \mathit{fix}(f)$ and a substitution lemma $\square$

Substitution Lemma

Lemma

If $\Gamma \vdash e : A$ *and* $\Gamma, x : A \vdash e' : C$ *and* $\gamma \in [\![\Gamma]\!]$, *then* $[\![\Gamma \vdash [e/x]e' : C]\!]\gamma = [\![\Gamma, x : A \vdash e' : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

Proof.

By induction on e' $\square$

Adequacy

Adequacy is the following theorem:

Theorem

If $\vdash e : Nat$ (ie. e is a closed term of type Nat) and $[\![e]\!] = n$ then $[\![e]\!] = n \Leftrightarrow e \mapsto^* n$

Correctness is one part of this. If we wanted to prove the opposite direction, we cannot do this just using induction.

So we need Logical Relations for the other direction.

Rest of Proposal

Week	Date	Task
7	18/07/16	Logical Relations
8	25/07/16	Logical Relations
9	01/08/16	Adequacy
10	08/08/16	Adequacy
11	15/08/16	Presentation preparation
12	22/08/16	*(Presentation Week)*
13	29/08/16	Writing report
14	05/09/16	Writing report