Project so far...

Natalie Ravenhill

July 12, 2016

This was what I had planned to do up to this point, from my project proposal:

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)*

I have mostly followed the plan in my proposal, apart from switching around operational and denotational semantics.

1 Domains

in the first two weeks of the project I studied Domain Theory. In our first meeting, Neel and I discussed how to prove that a given set with partial ordering and bottom element is a domain, describing the least upper bound of a chain (as opposed to using directed sets).

Then I proved two structures are domains:

- The **flat domain of natural numbers**, which is the natural numbers and a bottom element $\bot$, with no other ordering than $\bot \sqsubseteq n$ for any $n \in \mathbb{N}$
- The domain of **continuous functions** between two domains $(X, \sqsubseteq_X, \bot_X)$ and $(Y, \leq_Y, \bot_Y)$ where the bottom element is a function that loops on all inputs $\lambda x.x$ and the ordering is

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

After this we discussed the syntax of PCF and the fixpoint operator, which is used for recursion. In the domain model for PCF, the fixpoint is modelled by a continuous function, so our next task in Domain Theory was to prove the following theorem:

Theorem 1. *Every continuous function $f : X \to X$ has a least fixpoint, which is the limit of the chain $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot) \sqsubseteq \ldots$*

To prove this, first I had to prove that the limit of the chain is a fixpoint, then prove that it is less than any other fixpoint in the ordering.

2 Operational Semantics

In the second two weeks, I developed the operational semantics of PCF. First I discussed the evaluation rules for PCF with Neel. Then we decided our overall goal for our Operational Semantics was that it must be type safe.

Therefore my first task was to define the *typing rules* for PCF, so we could know what every possible PCF-definable expression is.

2.1 Type Weakening and Type Substitution

Type Safety is known to hold when two sub theorems hold, which are Type Preservation and Type Progress. However before we proved these theorems, there were two Lemmas I could use to make these proofs easier, which were the following:

- Type Weakening:

  Lemma 1. *If $\Gamma \vdash e : A$ then $\Gamma, x : C \vdash e : A$*

  For an expression e rhat is derivable in a typing context Γ, we can add any variable of any type to the context and Γ wlll still be derivable

- Type Substitution

  Lemma 2. *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.

Both of these theorems were proved by induction on derivation trees. For weakening we use induction on the derivation tree of $\Gamma \vdash e : A$.

For substitution, we first need to actually define the rules for substituting an expression for a variable of the same type in another expression. Then the proof was completed by induction on derivation trees of $\Gamma, x : A \vdash: e' : C$.

We use Weakening in the proof of Substitution, for example, the derivation case statement has the following typing rule:

$$\frac{\Gamma \vdash e : Nat \qquad \Gamma \vdash e_0 : C \qquad \Gamma, y : Nat \vdash e_S : C}{\Gamma \vdash \ case\ (e', z \to e_0, s(y) \to e_S)\ : C}\ \text{CASE}$$

Assuming we have a derivation tree for $\Gamma, x : A \vdash \mathit{case}\ (e', z \to e_0, s(y) \to e_S)\ : C$, we have derivation trees for $\Gamma, x : A \vdash e : Nat$, $\Gamma, x : A \vdash e_0 : C$ and $\Gamma, x : A, y : Nat \vdash e_S : C$

We want to get a derivation tree for $\Gamma \vdash \mathit{case}\ ([e/x]e', z \to [e/x]e_0, s(y) \to [e/x]e_S)\ : C$, so we can substitute e' for x in each tree we have. To use substitution for $\Gamma, y : Nat, x : Nat \vdash e' : A$, we need to add $y : Nat$ to the context of e. This is done with Weakening. Now we get $\Gamma \vdash [e/x]e_S : C$ and can use the typing rule of case to get the substitution in the whole expression.

2.2 Type Preservation

Next I proved Type Preservation, which is the following theorem:

Theorem 2. *If* $\Gamma \vdash e : A$ *and* $e \mapsto e'$*, then* $\Gamma \vdash e' : A$

This says that is $\Gamma \vdash e : A$ and there is an evaluation rule mapping the expression e to another one in one step, then the new expression has the same type. This was also proved by induction, on the the derivation trees of each expression.

Some of the cases involve the use of Substitution, for example, the rule for function application where the function is a λ abstraction:

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

To prove type progress in this case, we have

$$\frac{\dfrac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x : A).e : A \to B} \qquad \Gamma \vdash e' : A}{\Gamma \vdash (\lambda x : A).e\ e' : B}$$

And the tree for $(\lambda x : A).e\ e' \mapsto [e'/x]e$ for E. We have subtrees $\Gamma\ \vdash e' : A$ and $\Gamma, x : A \vdash e : B$, so we use these as parameters to the Substitution Lemma to get the tree for $\Gamma \vdash [e'/x]e : B$.

3 Type Progress

Then I proved Type Progress, which is the following theorem:

Theorem 3. *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 $\lambda x : A.e$.

This was also proved by induction on the evaluation rules on each expression.

4 Denotational Semantics

Neel and myself defined the denotational semantics in a meeting, as a function from a typing context to a domain:

$$[\![-]\!] : Type \to Domain$$

We have two possible ways to define a type, so there are two domains we use:

1. The type of Natural numbers is the ground type, so they are modelled by a single domain. We use the flat domain of Natural numbers, where $\bot$ represents a term that loops forever.

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

2. Function types are formed of other types. We model them using the domain of continuous functions.

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

On typing contexts this is:

$$[\![-]\!]_{Ctx} : Context \to Domain$$

that maps a typing context to a domain. The domain will be a nested tuple, the size of which depends on the number of variables in Γ. *Therefore I must prove separately that products of domains are also domains.*

Denotational Semantics of Contexts The empty context is given by

$$[\![\cdot]\!]_{Ctx} = \mathbb{1}$$

the single element set. *We must also prove separately that this is a domain.*

Adding a variable to a context Γ gives us the following:

$$[\![\Gamma, x : A]\!]_{Ctx} = [\![\Gamma]\!] \times [\![A]\!]$$

The products of the domains give all combinations of all possible values of each variable. If we want a specific valuation of the variables, we can refer to $\gamma \in [\![\Gamma]\!]_{Ctx}$.

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

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

So $[\![\Gamma \vdash e : A]\!]\gamma$ gives us an element of $[\![A]\!]$. We then defined this on each possible value of e individually.

5 Correctness

Now we have both semantics, we can prove correctness which is the following theorem:

Theorem 4. *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.

We prove this theorem by induction on $e \mapsto e'$, so the cases are on the evaluation rules. We can use the fact that $f(fix(f)) = fix(f)$ and the following substitution lemma to help us prove this:

Lemma 3. *If* $\Gamma \vdash e : A$ *and* $\Gamma, x : A \vdash e' : C$ *and* $\gamma \in [\![\Gamma]\!]$*, then* $[\![\Gamma \vdash [e/x]e' : C]\!]\gamma = [\![\Gamma, x : A \vdash e' : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

This was proved by induction on e'