Fixed Points and elements of Domain theory -continuation

1 Examples - Context free Grammars

1. $E = \epsilon \mid aE$
2. $E = a \mid bEb$
3. $E = \epsilon \mid aaEE$

One can use fixed point theorem to provide the semantics for these grammars.

Notations: Σ - alphabet; Σ^* - the set of all strings over Σ; $P(\Sigma^*)$ - the set of all subsets of Σ^*, i.e., the set of all languages over Σ.

Notes:

1. $P(\Sigma^*)$ partially ordered by inclusion is an ω-CPO.
2. Concatenation: $P(\Sigma^*) \times P(\Sigma^*) \longrightarrow P(\Sigma^*)$ is continuous.
3. Union - $\bigcup$: $P(\Sigma^*) \times P(\Sigma^*) \longrightarrow P(\Sigma^*)$ is continuous.

Hence, $X \rightarrowtail \epsilon \bigcup \{a\} X$ is continuous; $X \rightarrowtail \{a\} \bigcup \{b\} X \{b\}$ is continuous. Therefore, these functions have the least fixed points.

Example: Find the language defined by $E = \epsilon \mid aE$.

Let $F(X)$ be $\epsilon \bigcup \{a\} X$.

$F(\emptyset) = \epsilon$.

$F(F(\emptyset)) = \epsilon \bigcup \{a\}$.

$F^n(\emptyset) = ?$

Does the least fixed point of F is the same as the language defined by the grammar in Computational Models course?

$E = \epsilon \mid aE \mid bC$ and $C = a \mid CE$?

Here we associate a function $G : P(\Sigma^*) \times P(\Sigma^*) \longrightarrow P(\Sigma^*) \times P(\Sigma^*)$.

Compute $G^n(\emptyset, \emptyset)$ - the n-th approximation to the least fixed point. Show this is the set of strings obtained by derivation trees of depth at most n.

2 Additional fixed point theorems

A subset A of a partial order $\langle D, \sqsubseteq \rangle$ is **directed** if for every $a, b \in A$ there is $c \in A$ which uis an upper bound for $a and b$, i.e., $a \sqsubseteq c$ and $b \sqsubseteq c$.

A **directed** CPO is a structure $\langle D, \sqsubseteq \rangle$, where D is a set and $\sqsubseteq$ is a partial order on D such that every directed subset of D has the least upper bound.

Theorem 1 *1. Let $\langle D, \sqsubseteq \rangle$ be a directed CPO and let F be a continuous function from D to D. Then F has the least fixed point.*

2. Let $\langle D, \sqsubseteq \rangle$ be a directed CPO and let F be a monotone function from D to D. Then F has the least fixed point.

Note: in (2) Lfp of F might differ from $\bigsqcup\{F^n(\bot) \ : \ n \in \mathcal{N}\}$.

If we start from flat (or from directed) CPOs and define the full monotone frame, then all the CPO in this frame are directed. In this frame the least fixed point operator can be interpreted, because it is monotone.

A complete lattice is a structure $\langle D, \sqsubseteq \rangle$, where D is a set and $\sqsubseteq$ is a partial order on D such that

1. Every subset of D has the least upper bound.
2. Every subset of D has the greatest lower bound.

Theorem 2 (Knaster-Tarski) *Let $\langle D, \sqsubseteq \rangle$ be a complete lattice and let F be a monotone function from D to D. Then F has the least fixed point and the greatest fixed point.*

3 Simultaneous Fixed point vs iterative fixed point

<u>Simultaneous fixed point</u>

letrec (x be $f(x,z)$ and z be $g(x,z)$) in x.

Here we have to find the least solution $\langle x_0, z_0 \rangle$ of the system of equations below and return x_0.

$$\begin{cases} x = f(x,z) \\ z = g(x,z) \end{cases}$$

<u>Iterated fixed point</u>

letrec z be $g(x,z)$) in z.

Here x is free and the solution depends on the value of x.

We can consider, letrec $Z(x)$ be $g(x, Z(x))$ in $Z(x)$.

The last expression is a shorthand for letrec Z be $(\lambda x.\ g(x, Z(x))\ )$ in $Z(x)$.

Let Z_0 be the least solution of $Z = (\lambda x.\ g(x, Z(x))$, i.e., Z_0 be the leat fixed point of $\lambda Z \lambda x.\ g(x, Z(x))$.

Consider now letrec x be $f(x, Z_0(x))$ in x.

Let x_1 be the least solution of $x = f(x, Z_0(x))$, i.e., x_1 is the least fixed point of $\lambda x.\ f(x, Z_0(x))$.

<u>Simultaneous vs iterated fixed point</u>

It can be proved that the solution $\langle x_0, z_0 \rangle$ of the simultaneous fixed point is the same as $\langle x_1, Z_0(x_1) \rangle$ which was obtained without use of simultaneous fixed point.

Hence we have the following (denotational semantics) equivalence:

letrec (x be M_1 and z be M_2) in x= letrec x be (letrec z be M_2 in M_1) in x.

4 Comparing let and letrec declarations

$$\text{letrec } x \text{ be } N(\cdots, x, \cdots, x, \cdots) \text{ in } M(\cdots, x, \cdots, x, \cdots, x, \cdots).$$
$$\text{let } x \text{ be } N(\cdots, x, \cdots, x, \cdots) \text{ in } M(\cdots, x, \cdots, x, \cdots, x, \cdots).$$

Notes:

1. In the case $x \notin \mathtt{free}(N)$ the binding is the same for let and for letrec; also the meaning is the same.

2. Preliminary renaming may ensure that $x \notin \mathtt{free}(N)$ for let declarations, but not for letrec declarations.

3. The intended meaning of leterc is that within the body M , x denotes a solution (the least one) of the equation $x = N(\cdots, x, \cdots, x, \cdots)$. The equation makes sense only x and N have the same type.

4. let declaration can be transformed into semantically equivalent letrec declaration.

5. Enriching the language with an appropriate primitives (fixed point constants) one can transform letrec declaration into let declaration.

6. The extension from λ to let declaration amounts to <u>pure</u> syntactical sugaring (no additional primitives are needed). However, the further extension to letrec was achieved at the price of additional primitives. One can prove that lletrec declaration are not pure syntactical sugaring in the case of typed λ. Yet for type free λ-languages there is a term which behaves like fixed point: the paradoxical combinator $Y =_{\mathrm{def}} \lambda f.((\lambda x.f(xx))(\lambda x.f(xx)))$.

$$Yf \to_{\beta}^{*} f(Yf)$$

5 Equivalent (denotation) transformations

1. letrec x be N in $M = M$, provided $x \notin \mathtt{free}(M)$.
2. Full expansion:

   letrec x be N in M=letrec x be N in $M[N/x]$.
3. Partial expansion: instead of $M[N/x]$ one substitutes N only for some of the free occurrences of x in M.
4. Denesting: Let x_1 be (let x_2 be N_2 in N_1) in M=let (x_1 be N_1 and x_2 be N_2) in M, assuming x_2 is not free in M.

There are a lot of other important and useful equivalence. We discussed already the relationship between simultaneous and iterated letrec declarations.

6 Relationship between denotational and operational semantics

Definition 1 *Adequacy of operational and denotational semantics: For every program (i.e., a closed term of type int)* P*:*

$$P \rightarrow_{op} \bar{n} \quad \textit{if and only if} \quad [\![P]\!] = n.$$

Theorem 3 *The continuous model is adequate for lazy PCF and normal order PCF.*

Usually we want to reason about subprograms which are not programs.

Definition 2 (Soundness) *For any terms* M *and* N

$$M =_{oper} N \quad \textit{if and only if} \quad \forall \rho.\ [\![M]\!]\rho = [\![N]\!]\rho.$$

Theorem 4 *The continuous model is sound for lazy PCF and normal order PCF.*

What about the other direction?

The denotational semantics has the following congruence property:

$[\![N_1]\!] = [\![N_2]\!]$ implies that for any term M, the term obtained from M by **replacing** of x by N_1 is equivalent to the term obtained from M by **replacing** of x by N_2.

Observational congruence is the minimal equivalence $\sim$ on terms such that $N_1 \sim N_2$ if for all $P : \mathtt{int}$

$$P[x := N_1] = P[x := N_2].$$

Definition 3 *Semantics is fully abstract if*

$$[\![N_1]\!] = [\![N_2]\!] \Leftrightarrow N_1 \sim N_2$$

The continuous model is not fully abstract for PCF.

Hint: (1) Parallel If is not definable in PCF, i.e., there is no term M such that its meaning is the parallel if. (2) there are denotationally different terms N_1 and N_2 such that $[\![N_1]\!]a = [\![N_2]\!]a$ for every $a \in D^{bool \to int \to int \to int}$ such that $a \neq PIF$. However,

Theorem 5 *The continuous model is fully abstract for PCF with parallel IF.*