CS611 Lecture 13 Domain Constructions 9/22/00
Scribe: Daniel Kifer, Yanling Wang Lecturer: Andrew Myers

1 Meta Language

We can take fixed points of *continuous* functions in a cpo. Fixed points are needed for definitions of structures that we can think of as loops or cyclical graphs:

- control flow: while loop
- recursive functions
- recursive data types

Because loop-like constructs appear in programming languages so often, we need to take fixed points. In addition, because we are defining meaning functions inductively, most or all of the definitions for individual expression forms need to produce continuous functions. Therefore, we need to find an easy way to show that functions are continuous.

We will define a restricted language so that all functions expressible in it are continuous. It's a syntactic restriction on what functions we can write down. Functions (and other domain elements) defined in this language will look as though they are being defined by a programming language. However, there is no notion of evaluation for this language: the functions being defined simply *are*.

The Meta-Language will look like a typed language, because we will explicitly indicate what domains meta-variables are members of. It's not quite the same as the types that will show up in ordinary languages. For example, the ML pair type $T_1{*}T_2$ is not modeled by the product domain $T_1{\times}T_2$. The types in the source language are syntactic elements that can be mapped to domains; these domains have ordering relations defined on them. The Meta-language has a set of constructions that we can use to create new domains from existing ones.

2 Lifting

For now, the domains we talk about are cpo's. We are going to define several different ways to construct a cpo. To specify a cpo, we need the following:

- the set of underlying elements
- the ordering on the elements
- a demonstration that the ordering is complete: any chain in the new domain will have a supremum in the new domain
- to make the construction useful for the meta-language, we also define operations associated with the domains

The lifting construction is perhaps the simplest way to construct a new CPO given an existing one. If we have some cpo domain D, we can add new bottom element to form a pointed cpo $D_\perp$. We modify the ordering relation so that $\forall d \in D_\perp,\ \perp \sqsubseteq d$. Clearly if D is not pointed then $D_\perp$ is. If D is pointed, the new bottom element is even smaller than the original $\perp$ in D.

To distinguish between an element in D and in $D_\perp$, we use the notation $\lfloor d_i \rfloor \in D_\perp$ for $d_i \in D$. For example, if D is pointed, $D_\perp$ has a $\perp$ from the old D. We write it as $\lfloor \perp \rfloor$ to distinguish it from the new $\perp$ we added.

What is the ordering relation in $D_\perp$? We need to:

We preserve the ordering relation in D, so $\lfloor d_i \rfloor \sqsubseteq \lfloor d_j \rfloor \Leftrightarrow d_i \sqsubseteq d_j$ and we have $\perp \sqsubseteq \lfloor d_i \rfloor$ for $\forall d_i \in D$. $D_\perp$ is complete because chain in $D_\perp$ has a supremum in $D\ \perp$ since it either looks like a chain in D or like a chain in D with a $\perp$ in the beginning.

3 Discrete cpos

The ordering relation for discrete cpos (booleans, natural numbers, integers, etc.) is equality. The functions over discrete cpos are continuous because the chains consist of only one element. We can only take fixed points of functions that map pointed cpos to pointed cpos. Therefore we must lift discrete cpos to take fixed points.

We also need to extend functions to pointed cpos:

If $f{\in}D{\to}E$, then $f_\perp{\in}D_\perp{\to}E_\perp$, $f^*{\in}D_\perp{\to}E$ (assuming E is pointed) are

$f_\perp = \lambda d{\in}D_\perp$, if $d =\perp$ then $\perp$ else $f(d)$

$f^* = \lambda d{\in}D_\perp$, if $d = \perp$ then $\perp$ else $f(d)$ (if E is pointed)

If f is continuous, then so is $f_\perp$ and f^*.

4 Let

We define let to be a useful syntax in the following way:

$$let\ x = d.e \ \equiv\ (\lambda x{\in}D.e)^* d$$

This function will return $\perp$ when d is $\perp$. Remember that x is an element of D but not $D_\perp$.

5 Unit

The simplest nontrivial domain is called the Unit domain; it has one element (u). The ordering relation only needs to be reflexive: u is related only to itself. The only ω-chain we can get is $u \sqsubseteq u \sqsubseteq u \sqsubseteq u \sqsubseteq u \sqsubseteq \ldots$. Calculating the LUB of this chain is left to the reader, but it should be clear that a unit domain with a reflexive ordering relation is a cpo.

Why do we need this unit domain? We can use it do distinguish between termination and nontermination when a computation does not produce a value. We use u to present termination. It is also used as argument for functions that need no argument. For example, $U{\to}Z$ is a constant function. Unit domains can be used to build more complicated domains (for example, booleans).

6 Products

If we have two domains D_1 and D_2, then $D_1{\times}D_2$ is a product domain. The underlying elements in this domain are tuples that consist of one element from each domain:

$\langle d_1, d_2\rangle$ where $d_i{\in}D_i$.

The ordering relation on the product domain is:

$\langle d_1, d_2\rangle{\sqsubseteq}\langle d_1', d_2'\rangle$ iff $d_1{\sqsubseteq}d_1'\&d_2{\subseteq}d_2'$

Is this a cpo? Given the ordering relation, we first have to show that it is a partially ordered domain. It is quite obvious that the ordering relation is reflexive, transitive and antisymemtric. We still have to show that it is complete: any chain in the product domain has an LUB that is also in this domain.

For any chain $\langle d_1, d_2\rangle \sqsubseteq \langle d_1', d_2'\rangle \sqsubseteq \ldots \sqsubseteq \langle d_1^{(n)}, d_2^{(n)}\rangle \sqsubseteq \ldots$

Clearly $d_1 \sqsubseteq d_1' \sqsubseteq \ldots \sqsubseteq d_1^{(n)} \sqsubseteq \ldots$ has a least upper bound (call it d_1^∞) and

$d_2 \sqsubseteq d_2' \sqsubseteq \ldots \sqsubseteq d_2^{(n)} \sqsubseteq \ldots$ has a least upper bound (call it d_2^∞)

So $\langle d_1^\infty, d_2^\infty\rangle$ is the least upper bound of the chain $\langle d_1, d_2\rangle \sqsubseteq \langle d_1', d_2'\rangle \sqsubseteq \ldots \sqsubseteq \langle d_1^{(n)}, d_2^{(n)}\rangle \sqsubseteq \ldots$

Since $\langle d_1^\infty, d_2^\infty\rangle \ \in\ D_1 \times D_2$ this new domain is also complete.

This can be extended to n-tuples $\langle d_1, d_2, ..., d_n\rangle$

We have the following two operations on product domains (These functions are clearly continuous and monotonic.):

- tupling: $\langle d_1, d_2, ..., d_n\rangle$

- projection: $\pi_i\langle d_1, d_2, ..., d_n\rangle = d_i$

7 Disjoint Sums

Disjoint sums allow us to have element of one type or another. For example $D_1 + D_2$ lets us have elements of D_1 or D_2. Each element is tagged with an injection function $\{in_i(d_i)|d_i \in D_i\}$ so we know which domain it comes from. The actual tagging function is irrelevant (as long as it works!). One possibility is $\lambda d.\langle i, d\rangle$.

The injection function preserves the original ordering of the individual domains.
$in_i(d_m) \sqsubseteq in_j(d_n)$ iff $i = j$, $d_m \sqsubseteq d_n$ The injection function is clearly continuous. Any chain in $D_1 + D_2$ has to stay in one domain D_i, and the injected version of the supremum in the domain D_i is the LUB of this chain. This can be easily extended to multi-domain sums.

Clearly the sum domain is a cpo, but has no $\perp$ element which is less than everything (otherwise $\perp \in D_i$ for some i, and so if $d \in D_j$ where $j \neq i$, d can't be compared to $\perp$). If we need the fixed point, we must lift the sum. We can lift it by adding a new $\perp$. If D_1 and D_2 both have the $\perp$ element, we can also glue them together and make them share it.

The injection function is necessary because it can distinguish between identical domains and also domains that may share elements but have different ordering relations. For example: $T = U + U$, $true = in_1(u)$, $false = in_2(u)$. We can NOT simply glue the ordering relations together.

Sums can be unpacked with the *case* construction:
$case\ e\ of\ x_1.e_1|x_2.e_2 \equiv case\ e\ of\ D_1(x_1).e_1|D_2(x_2).e_2$ where $e \in D_1 + D_2$. If $e = in_i(d_i)$, the value of this expression will be evaluated as $(\lambda x_i \in D_i.e_i)d_i$. Case is a continuous function of e if all $f_i \in D_i \to E = (\lambda x_i \in D_i.e_i)$ are continuous.

8 Continuous Functions

For any two cpo's D and E, we define $D \to E$ as the domain of continuous functions that map D to E. We define a pointwise ordering for two functions f,$g \in D \to E$ (a subset of E^D - the set of *all* functions that map $D \to E$):

$f \sqsubseteq g$ iff $\forall d, f(d) \sqsubseteq g(d)$

Given the ordering relations on the functions, the LUB of any chain exists:

$\sqcup_{n\in\omega} f_n = \lambda d \in D. \sqcup_{n\in\omega} f_n(d)$

but it is necessary to show that this LUB is continuous and therefore an element of $D \to E$:

$(\lambda d \in D.\ \sqcup_{n\in\omega} f_n(d))(\sqcup_{m\in\omega} d_m) = \sqcup_{m\in\omega}(\lambda d \in D. \sqcup_{n\in\omega} f_n(d))(d_m)$

Lemma: If f is monotonic and continuous, then

$\sqcup_n \sqcup_m f_n(d_m) = \sqcup_n f_n(d_n) = \sqcup_m \sqcup_n f_n(d_m)$

Let $e_{nm} = f_n(d_m)$
$n \leq n',\ \ m \leq m' \Rightarrow\ e_{nm} \sqsubseteq e_{n'm'}$
$e_{nm} \sqsubseteq e_{n'n'}$ for $n' = max(m, n)$, so $\sqcup_{n,m} e_{nm} \sqsubseteq \sqcup_n e_{nm}$
$e_{nn} \sqcup_m e_{nm}$, so $\sqcup_n e_{nn} \sqsubseteq \sqcup_n \sqcup_m e_{nm}$, $\sqcup_m \sqcup_n e_{nm}$
$\sqcup_m e_{nm} \sqsubseteq \sqcup_{n,m} e_{nm}$, so $\sqcup_n \sqcup_m e_{nm} \sqsubseteq \sqcup_{n,m} e_{nm}$
Proof:
$(\lambda d \in D.\ \sqcup_{n\in\omega} f_n(d))(\sqcup_{m\in\omega} d_m)$
$= \sqcup_{n\in\omega} f_n(\sqcup_{m\in\omega} d_m)$
$= \sqcup_{n\in\omega} \sqcup_{m\in\omega} f_n(d_m)$
$= \sqcup_{n\in\omega} f_n(d_n)$
$= \sqcup_{m\in\omega} \sqcup_{n\in\omega} f_n(d_m)$
$= \sqcup_{m\in\omega}(\lambda d \in D\ .\ \sqcup_{n\in\omega} f_n(d))(d_m)$

9 Operations on functions

We have the following new operations in our Meta-Language:
$apply \in (D \to E) \times D \to E\ \lambda p.(\pi_1 p)(\pi_2 p)$

$curry \ \in ((D \times E) \to F) \to (D \to E \to F)$
$\quad = \lambda f \in (D \times E) \to F.\lambda d \in D.\lambda e \in E.f\langle d, e\rangle$
$compose = \circ \ \in (D \to E) \times (E \to F) \to (D \to F)$
$\quad = \lambda\langle f, g\rangle.\lambda d \in D.f(g(d))$
$fix \in (D \to D) \to D$ (D pointed)
$\quad = \lambda g \in D \to D. \sqcup_n g^n(\bot)$
$\quad = \sqcup_n \lambda g \in D \to D.g^n(\bot)$
fix is a continuous function since it takes the LUB of continuous functions.