CS 611
Advanced Programming Languages

Andrew Myers
Cornell University

Lecture 20
Domain Constructions
17 Oct 01

Administration

- Homework 3 due Friday
- Scribes needed

Fixed points

- Denotational semantics for IMP rely on taking fixed point to define $\mathcal{C}[\![\textbf{while}]\!]$
- Fixed points occur in most language definitions: needed to deal with loops
  - control flow loops: while
  - data loops: recursive functions, recursive data structures, recursive types
- Only know how to find least fixed pts for continuous functions f
- *Need easy way to ensure continuity*

Meta-language

- Idea: define restricted language for expressing mathematical functions
- All functions expressible in this language are continuous
- Looks like a programming language (uF, ML)
  - not executed: just mathematical notation
  - can talk about non-termination!
  - "evaluation" is lazy (vs. eager in ML)
  - "typed" (vs. untyped in uF & variants)

"Types" for Meta-language

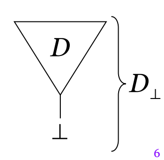
- Meta-language contains domain declarations indicating the set of values meta-variables can take on, e.g.

  $\lambda f \in \Sigma \to \Sigma_\perp.\ \lambda \sigma \in \Sigma.\mathit{if}\ \neg \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ \sigma\ \mathit{else}\ \mathit{strict}(f, \mathcal{C}[\![c]\!]\sigma)$
- Domains will function as types for meta-language
  - but with precisely defined meaning, ordering relation, etc.
  - $T_1 * T_2$ is not necessarily modeled by $T_1 \times T_2$!
- Meta-language consists of domains and associated operations

Lifting

- If D is a domain (for now: cpo), can "lift" by adding new bottom element to form pointed cpo $D_\perp$
- cpo defined by underlying set plus complete ordering relation $\sqsubseteq$
- Elements of $D_\perp$ are $\lfloor d_i \rfloor, \perp$ where $d_i \in D$
- Ordering relation:

$\lfloor d_i \rfloor \sqsubseteq \lfloor d'_i \rfloor \iff d_i \sqsubseteq d_i$
$\perp \sqsubseteq \lfloor d_i \rfloor$



- Complete?

Discrete cpos

- Various discrete cpos: booleans (**B**), natural numbers (ω), integers (**Z**), ...
- Corresponding functions over discrete cpos exist: $+ : \mathbf{Z}\times\mathbf{Z}\to\mathbf{Z}$, $\wedge : \mathbf{B}\times\mathbf{B}\to\mathbf{B}$
- Often want to lift discrete cpos to take fixed points; helpful to extend fcns to pointed cpos
- If $f\in D\to E$, then $f_\perp\in D_\perp\to E_\perp$, $f^*\in D_\perp\to E$ are
  $f_\perp = \lambda d\in D_\perp.\ \mathit{if}\ d{=}\perp\ \mathit{then}\ \perp\ \mathit{else}\ f(d)$
  $f^* = \lambda d\in D_\perp.\ \mathit{if}\ d{=}\perp\ \mathit{then}\ \perp_E\ \mathit{else}\ f(d)$ (if E pointed)
- $2 +_\perp 2 = 4$, $3 +_\perp \perp = \perp$, $\perp \wedge_\perp \mathit{true} = \perp$
- If f continuous, are $f_\perp$, f^* ?

let

- Useful syntax: given $d\in D_\perp$
  $\mathit{let}\ x = d\ \mathit{in}\ e \quad\equiv\quad (\lambda x\in D.e)^* d$
- Expresses evaluation of e that is *strict* in d

Unit

- Simplest cpo: empty set ($\emptyset$)
- Next simplest: *unit domain* (**U**)
  - single element: u
  - ordering relation: reflexive
  - complete: only directed set is {u}

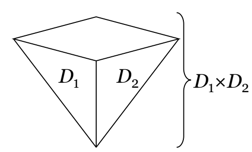
Hasse diagram

·u

- Used to represent computations that terminate but do not produce a value, argument for functions that need no argument
- Also building block for other domains

Products

- If D_1, D_2 are domains, then $D_1\times D_2$ is a product domain
- Underlying set: pairs $\langle d_1, d_2\rangle$ where $d_i\in D_i$
- Ordering:
  $\langle d_1, d_2\rangle \sqsubseteq \langle d'_1, d'_2\rangle$ iff $d_1\sqsubseteq d'_1$ & $d_2\sqsubseteq d'_2$



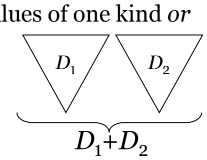
- Extends to n-tuples
- Operations:
  - tupling: $\langle d_1,\ldots,d_m\rangle$
  - projection: $\pi_i\,\langle d_1,\ldots,d_m\rangle = d_i$

CPO?

- Is product domain a cpo if D_1, D_2 are?
- Any chain $\langle d_0, d'_0\rangle \sqsubseteq \langle d_1, d'_1\rangle \sqsubseteq \langle d_2, d'_2\rangle \sqsubseteq \ldots$ must have LUB in $D_1\times D_2$
- Definition of $\sqsubseteq$: $d_0\sqsubseteq d_1\sqsubseteq d_2\sqsubseteq\ldots$ is chain in D_1, $d'_0\sqsubseteq d'_1\sqsubseteq d'_2\sqsubseteq\ldots$ is chain in D_2
- LUB is $\bigsqcup\langle d_n, d'_n\rangle = \langle \bigsqcup d_n, \bigsqcup d'_n\rangle$
- Operations continuous?
  $\pi_i \bigsqcup_{n\in\omega} x_n = \bigsqcup \pi_i x_n = \bigsqcup d_{in}$
  $\bigsqcup\langle x_{1n}, \ldots, x_{mn}\rangle = \langle \bigsqcup d_{1n}, \ldots, \bigsqcup d_{mn}\rangle$

Sums

- Sometimes want to allow values of one kind *or* another: D_1+D_2
- Elements of domain are elements of D_1 or D_2 tagged with origin: $\{in_i(d)\mid d\in D_i\}$



- Form of in_i is irrelevant (could be $\lambda d.\langle i, d\rangle$)
- Preserves ordering of individual domains:
  $in_i(d) \sqsubseteq in_j(d')$ iff $i{=}j$, $d\sqsubseteq d'$
- Injection function in_i is continuous
- Extends naturally to multi-domain sum
- not pointed

Sums, cont'd

- Why tag? Distinguishes identical domains
  - $\mathbf{B} = \mathbf{U} + \mathbf{U}$, true = in_1(u), false = in_2(u)
- Sums unpacked with *case* construction:

  $\textit{case } e \textit{ of } in_i(x_1).e_1 \mid in_i(x_2).e_2$ *or*: $\textit{case } e \textit{ of } D_1(x_1).e_1 \mid D_2(x_2).e_2$
- Given $e = in_i(d_i)$, has value $f_i(d_i) \in E$ where $f_i \in D_i \to E = (\lambda x_i \in D_i\,.e_i)$
- Continuous function of e if all f_i continuous:

$$\sqcup \textit{case } e_n \textit{ of } \ldots = \textit{case } \sqcup e_n \textit{ of } \ldots\ ?$$

$$\sqcup f_i(d_{in}) = f_i(\sqcup d_{in})$$

- Also continuous function of each f_i

$$\sqcup \textit{case } e \textit{ of } f_{1n} \mid f_2 = \textit{case } e \textit{ of } \sqcup f_{1n} \mid f_2 = \sqcup f_{1n}(d_1)$$

Continuous functions

- Given cpos D, E, define $D \to E$ as domain of *continuous* functions mapping D to E (subset of E^D)
- Pointwise ordering: $f \sqsubseteq g$ iff $f(d) \sqsubseteq g(d)$
- Complete?

$$\sqcup_{n\in\omega} f_n = \lambda d \in D\,.\,\sqcup_{n\in\omega} f_n(d)\ \Big\}\ \textit{continuous?}$$

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

Continuity of LUB

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

need exchange lemma

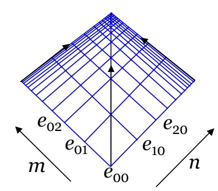
Exchange Lemma

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

Let $e_{nm} = f_n(d_m)$

$n \le n'$, $m \le m' \Rightarrow e_{nm} \sqsubseteq e_{n'm'}$

Lemma holds for any such e_{nm}



$e_{nn} \sqsubseteq \sqcup_m e_{nm}$, so $\sqcup_n e_{nn} \sqsubseteq \sqcup_n \sqcup_m e_{nm}, \sqcup_m \sqcup_n e_{nm}$

$\sqcup_n e_{nm} \sqsubseteq \sqcup_n e_{nn}$, so $\sqcup_m \sqcup_n e_{nm} \sqsubseteq \sqcup_n e_{nn}$

Operations on functions

- *apply* $\in (D \to E) \times D \to E = \lambda p.(\pi_1 p)(\pi_2 p)$
- *curry* $\in ((D \times E) \to F) \to (D \to E \to F)$
  $= \lambda f \in (D \times E) \to F\,.\,\lambda d \in D.\lambda e \in E.f\,\langle d,e\rangle$
- *compose* $= \cdot \circ \cdot \in (D \to E) \times (E \to F) \to (D \to F)$
  $= \lambda f \in D \to E, g \in E \to F.\lambda d \in D.f\,(g(d))$
- *fix* $\in (D \to D) \to D$ (D pointed)
  $= \lambda g \in D \to D.\ \sqcup_n g^n(\bot)$
  $= \sqcup_n \lambda g \in D \to D.\ g^n(\bot)$ (defn of $\sqcup$)

  is continuous: LUB of continuous fcns $\lambda g \in D \to D.\ g^n(\bot)$

Meta-Language

- Have defined various constructs that we can use to define continuous functions
- Constructs are a syntax for a meta-language in which only continuous functions can be defined
- How do we know when expression $\lambda x.e$ is continuous?
- Idea: use structural induction on form of e so every syntactically valid e can be abstracted over any variable to produce continuous function
- Problem: structural induction $\Rightarrow$ need to consider open terms e

Continuity in variables

- Idea: consider a meta-language expression e to be implicitly function of its free variables
- e is continuous in variable x if $\lambda x.e$ is continuous for arbitrary values of other (non-x) free variables in e
- e is continuous in variables not free in e
- structural induction: for each syntactic form, show that term is continuous in variables assuming sub-terms are