CS 611
Advanced Programming Languages
Andrew Myers
Cornell University
Lecture 20
Domain Constructions
17 Oct 01

**Fixed points**
- Denotational semantics for IMP rely on taking fixed point to define $\sem{\text{while}}$
- 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.
  \[
  \lambda f \in \Sigma \to \Sigma, \lambda \sigma \in \Sigma. if \neg \sem{\sigma} then \sigma else strict(f, \sem{\sigma})
  \]
- Domains will function as types for meta-language
  - but with precisely defined meaning, ordering relation, etc.
  - $T_1 \times T_2$ 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 $D_\bot$
- cpo defined by underlying set plus complete ordering relation $\sqsubseteq$
- Elements of $D_\bot$ are $[d_i], \bot$ where $d_i \in D$
- Ordering relation:
  \[
  [d_i] \sqsubseteq [d_i'] \iff d_i \sqsubseteq d_i
  \]
- Complete?

**Administration**
- Homework 3 due Friday
- Scribes needed
Discrete cpos
- Various discrete cpos: booleans (B), natural numbers (ω), integers (Z), ...
- Corresponding functions over discrete cpos exist: + : Z × Z → Z, ∧ : B × B → B
- Often want to lift discrete cpos to take fixed points; helpful to extend functions to pointed cpos
- If f : D → E, then f ∈ D, f' : D → E are
  f₁ = λd ∈ D₁ if d = ⊥ then ⊥ else f(d)
  f'₁ = λd ∈ D₁ if d = ⊥ then ⊥ else f(d) (if E pointed)
- 2 +₁ 2 = 4, 3 +₁ ⊥ = ⊥, ⊥ ∧₁ true = ⊥
- If f/continuous, are f₁, f'₁?

Unit
- Simplest cpo: empty set (∅)
- Next simplest: unit domain (U)
- Hasse diagram
  - single element: u
  - ordering relation: reflexive
  - complete: only directed set is {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

CPO?
- Is product domain a cpo if D₁, D₂ are?
- Any chain (d₀, d'₀) ≤ (d₁, d'₁) ≤ (d₂, d'₂) ≤ ... must have LUB in D₁ × D₂
- Definition of ≤: d₀ ≤ d₁ ≤ d₂ ... is chain in D₁, d'₀ ≤ d'₁ ≤ d'₂ ... is chain in D₂
- LUB is ∪(d₀, d'₀) = (∪d₀, ∪d'₀)
- Operations continuous?
  \( \piᵢ \cup_{n ∈ ω} xₙ = \cup \piᵢ.xₙ = \cup dᵢ \)
  \( \cup(x_{m₁}, ..., x_{mᵢ}) = \cup \cup d_{m₁}, ..., \cup d_{mᵢ} \)

let
- Useful syntax: given de D₁
  let x = d in e = (λx ∈ D₁.e)d
- Expresses evaluation of e that is strict in d

Products
- If D₁, D₂ are domains, then D₁ × D₂ is a product domain
- Underlying set: pairs (d₁, d₂) where d₁ ∈ D₁
- Ordering:
  \( \langle d₁, d₂ \rangle \sqsubseteq \langle d₁', d₂' \rangle \) iff
  \( d₁ \sqsubseteq d₁' \) & \( d₂ \sqsubseteq d₂' \)
- Extends to n-tuples
- Operations:
  - tupling: \( \langle d₁, ..., dₙ \rangle \)
  - projection: \( πᵢ(d₁, ..., dₙ) = dᵢ \)

Sums
- Sometimes want to allow values of one kind or another: D₁ + D₂
- Elements of domain are elements of D₁ or D₂ tagged with origin:
  \( \langle \text{in}(d) \sqsubseteq d \rangle \)
- Form of inᵢ is irrelevant (could be λd.(i, d))
- Preserves ordering of individual domains:
  \( \text{in}(d) \sqsubseteq \text{in}(d') \) iff \( i = j \) & \( d \sqsubseteq d' \)
- Injection function \( \text{in}_i \) is continuous
- Extends naturally to multi-domain sum
- not pointed
Sums, cont’d
- Why tag? Distinguishes identical domains
  - $B = U + U$, true = $i_1(u)$, false = $i_2(u)$
- Sums unpacked with case construction:
  \[ \text{case } e \text{ of } i_1(x_1), e_1 \mid i_2(x_2), e_2 \text{ or: case } e \text{ of } D(x_1), e_1 \mid D(x_2), e_2 \]
- Given $e = i_1(d_1)$, has value $f(d_1) \in E$ where $f \in D \to E = (\lambda x \in D, e)$
- Continuous function of $e$ if all $f_i$ continuous:
  \[ \bigcup_{n \in \omega} f_i \text{ of } ... = \bigcup_{n \in \omega} e_i \text{ of } ... \]
  \[ f(d_m) = f(\bigcup_{n \in \omega} d_m) \]
- Also continuous function of each $f_i$
  \[ \bigcup_{n \in \omega} f_{i_n} | f_2 = \bigcup_{n \in \omega} f_{i_f} | f_2 = \bigcup_{n \in \omega} f_{i_2}(d_i) \]

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?
  \[ \bigcup_{n \in \omega} f_n = \lambda d \in D \cdot \bigcup_{n \in \omega} f_n(d) \]

Exchange Lemma
\[ \bigcup_{n \leq n'} m f_n(d_m) = \bigcup_{n \leq n'} m f_n(d_m) = \bigcup_{n \leq n'} m f_n(d_m) \]

Operations on functions
- \text{apply} $\in (D \to E) \times D \to E = \lambda p.(\pi \cdot p)(\pi \cdot p)$
- \text{curry} $\in ((D \times E) \to F) \to (D \to E \to F)$
  \[ \lambda f \in D \to E \cdot \lambda x \in E \cdot f(d, e) \]
- \text{compose} $\circ \in (E \to (E \to F)) \to (D \to F)$
  \[ \lambda f \in E \to E \cdot g \in E \to F \cdot \lambda d \in D \cdot f(g(d)) \]
- \text{fix} $\in (D \to D) \to D$
  \[ \lambda g \in D \to D \cdot \bigcup_{n \in \omega} g^n(\bot) \]
  \[ \text{is continuous: LUB of continuous functions } \lambda g \in D \to D \cdot g^n(\bot) \]

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 syntacally 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.