7. Domain Theory II: Recursively Defined Functions

Wolfgang Schreiner RISC-Linz

Semantics of Loops

B: Boolean-expression

C: Command

...

C ::= ... | **while** B **do** C | ...

...

C[[**while** B **do** C]] =

$\underline{\lambda}s.$ **B**[[B]]$s \rightarrow$ **C**[[**while** B **do** C]](**C**[[C]]s) [] s

Problem: meaning of a syntax phrase may be defined only in terms of its proper sub-parts.

C[[**while** B **do** C]] = w

where w: $Store_\perp \rightarrow Store_\perp$

$w = \underline{\lambda}s.$**B**[[B]]$s \rightarrow w($**C**[[C]]$s)$ [] s

Recursion in syntax exchanged for recursion in function notation!

Recursive Function Definitions

- Function Definition

  $$q = \lambda n.n \textit{ equals zero } \rightarrow \textit{one } [] \; q(n \textit{ plus one})$$

- Possible Function Graphs:
  - $\{(\textit{zero, one})\}$
    $= \{(\textit{zero, one}), (\textit{one}, \bot), (\textit{two}, \bot), \ldots\}$
  - $\{(\textit{zero, one}), (\textit{one, six}), (\textit{two, six}), \ldots\}$
  - $\{(\textit{zero, one}), (\textit{one}, k), (\textit{two}, k), \ldots\}$

Several functions satisfy specification; which one shall we choose?

Least Fixed Point Semantics

Theory that establishes meaning of recursive specifications:

1. Guarantees that every specification has a function satisfying it.
2. Provides means for choosing the "best" function out of the set of possibilities.
3. Ensures that the selected function corresponds to the conventional operational treatment of recursion.

   Argument is mapped to defined answer iff simplification of the specification yields a result in a finite number of recursive invocations.

The Factorial Function

$$\textit{fac(n)} = \textit{n equals zero} \rightarrow \textit{one}$$
$$\textit{[] n times (fac(n minus one))}$$

Only one function satisfies specification:

graph(factorial) =

$\{$*(zero, one), (one, one),*
(two, two), (three, six),
$\ldots$, $(i, i!)$, $\ldots\}$

Simplification

fac(three)
$\Rightarrow$ *three equals zero*
$\quad \rightarrow$ *one [] three times fac(three minus one)*
$=$ *three times fac(three minus one)*
$=$ *three times fac(two)*
$\Rightarrow$ *three times (two equals zero*
$\quad \rightarrow$ *one [] two times fac(two minus one))*
$=$ *three times (two times fac(one))*
$\Rightarrow$ *three times (two times (one equals zero*
$\quad \rightarrow$ *one [] one times fac(one minus one)))*
$=$ *three times (two times one (times fac(zero)))*
$\Rightarrow$ *three times (two times (one times (zero equals zero*
$\quad \rightarrow$ *one [] zero times fac(zero minus one))))*
$=$ *three times (two times one (times one))*
$=$ *six*

Partial Functions

- Answer is produced in a *finite* number of unfolding steps.
- Idea: place limit on number of unfoldings and investigate resulting graphs
  - *zero:* $\{\}$
  - *one:* $\{(\mathit{zero}, \mathit{one})\}$
  - *two:* $\{(\mathit{zero}, \mathit{one}), (\mathit{one}, \mathit{one})\}$
  - $i + 1$: $\{(\mathit{zero}, \mathit{one}), (\mathit{one}, \mathit{one}), \ldots (i, i!)\}$
- Graph at stage i defines function fac_i.
  - *Consistency with each other:*

$$\mathit{graph}(\mathit{fac}_i) \subseteq \mathit{graph}(\mathit{fac}_{i+1})$$

  - *Consistency with ultimate solution:*

$$\mathit{graph}(\mathit{fac}_i) \subseteq \mathit{graph}(\mathit{factorial})$$

  - *Consequently*

$$\bigcup_{i=0}^{\infty} \mathit{graph}(\mathit{fac}_i) \subseteq \mathit{graph}(\mathit{factorial})$$

Partial Functions

- Any result is computed in a finite number of unfoldings.

  $(a, b) \in \mathit{graph}(\mathit{factorial})$
  $\Rightarrow (a, b) \in \mathit{graph}(\mathit{fac}_i)$ *(for some i)*

- Consequently

$$\mathit{graph}(\mathit{factorial}) \subseteq \bigcup_{i=0}^{\infty} \mathit{graph}(\mathit{fac}_i)$$

- Thus

$$\mathit{graph}(\mathit{factorial}) = \bigcup_{i=0}^{\infty} \mathit{graph}(\mathit{fac}_i)$$

Factorial function can be totally understood in terms of the finite subfunctions fac_i!

Partial Functions

- Representations of sub-functions

  $fac_0 = \lambda n.\ \bot$
  $fac_{i+1} = \lambda n.\ n$ *equals zero* $\rightarrow$ *one*
  [] n *times* fac_i*(*n *minus one)*

- Each definition is *non-recursive*

  Recursive specification can be understood in terms of a family of non-recursive ones.

- Common format can be extracted

  "Functional" F
  F: $(Nat \rightarrow Nat_\bot) \rightarrow (Nat \rightarrow Nat_\bot)$
  $F = \lambda f.\ \lambda n.\ n$ *equals zero* $\rightarrow$ *one*
  [] n *times* f*(*n *minus one)*

Each subfunction is an instance of the functional!

Functional and Fixed Point

- Partial functions

  $fac_{i+1} = F(fac_i) = F^i(\bot)$
  $\bot := (\lambda n.\bot)$

- Function graph

  $$graph(factorial) = \bigcup_{i=0}^{\infty} graph(F^i(\bot))$$

- Fixed point property

  $graph(F(factorial)) = graph(factorial)$
  $F(factorial) = factorial$

The function factorial is a fixed point of the functional F!

q Function

$Q = \lambda q.\lambda n.n$ *equals zero*
$\quad \rightarrow$ *one* [] $q(n$ *plus one*$)$

$Q^0(\bot) = (\lambda n.\bot)$
graph$(Q^0(\bot)) = \{\}$

$Q^1(\bot) = \lambda n.n$ *equals zero*
$\quad \rightarrow$ *one* [] $(\lambda n.\bot)(n$ *plus one*$)$
$=$ $\lambda n.n$ *equals zero* $\rightarrow$ *one* [] $\bot$
graph$(Q^1(\bot)) = \{$*(zero, one)*$\}$

$Q^2(\bot) = Q(Q^1(\bot)) = \lambda n.n$ *equals zero*

$\rightarrow$ *one* [] $((n$ *plus one*$)$ *equals zero* $\rightarrow$ *one* [] $\bot)$
graph$(Q^2(\bot)) = \{$*(zero, one)*$\}$

q Function

- Convergence has occured

  $graph(Q^i(\bot)) = \{(zero,\ one)\},\ i \geq 1$

- Resulting graph

  $$\bigcup_{i=0}^{\infty} graph\big(Q^i(\bot)\big) = \{\big(zero,\ \ one\big)\}$$

- Fix point property

  $$Q\big(qlimit\big) \ = \ qlimit$$

- Still many solutions possible

  $graph(q_k) =$
  $\{\ (zero,\ one),\ (one,\ k),\ \ldots,\ (i,\ k),\ \ldots\}$

- Least fixed point property

  $graph(qlimit) \subseteq graph(q_k)$

The function qlimit is the least fixed point of the functional Q!

Recursive Specifications

The meaning of a recursive specification $f = F(f)$ *is taken to be fix(*F*), the least fixed point of the functional denoted by* F*.*

$$\mathit{graph}(\mathit{fix}\ F) = \bigcup_{i=0}^{\infty} \mathit{graph}(F^i(\bot))$$

- The domain D of F must be a pointed cpo
  - *partial ordering on* D*,*
  - *every chain in* D *has a least upper bound in* D*,*
  - D *has a least element.*
- F must be continuous

  Preserves limits of chains.
- Semantic domains are cpos and their operations are continuous.

  Pointed cpos are created from primitive domains and union domains by lifting.

Factorial Function

$$F = \lambda\ f.\ \lambda\ n.\ \ n\ equals\ zero \rightarrow one$$
$$[]\ n\ times\ (f(n\ minus\ one))$$

Simplification rule

$$fix\ F = F(fix\ F)$$

$$\begin{aligned}
&(fix\ F)(three)\\
&= (F\ (fix\ F))(three)\\
&= (\lambda\ f.\ \lambda\ n.\ \ n\ equals\ zero \rightarrow one\\
&\qquad []\ n\ times\ (f(n\ minus\ one))(fix\ F))(three)\\
&= (\lambda\ n.\ \ n\ equals\ zero \rightarrow one\\
&\qquad []\ n\ times\ (fix\ F)(n\ minus\ one))(three)\\
&= three\ equals\ zero \rightarrow one\\
&\qquad []\ three\ times\ (fix\ F)(three\ minus\ one)\\
&= three\ times\ (fix\ F)(two)\\
&= three\ times\ (F\ (fix\ F))(two)\\
&= \ldots\\
&= three\ times\ (two\ times\ (fix\ F)(two))
\end{aligned}$$

Fixed point property justifies rec. unfolding!

Copyout function

Interactive text editor

copyout: Openfile $\rightarrow$ *File*$_\perp$
copyout $= \lambda$*(front, back). null front* $\rightarrow$ *back*
[] copyout((tl front), ((hd front) cons back))

- Functional C, copyout = fix C.
- With i unfoldings, list pairs of length $i-1$ can be appended.
- Codomain is lifted because least fixed point semantics requires that codomain of any recursively defined function be pointed.

  ispointed$(A \rightarrow B)$ = *ispointed*(B)
  min$(A \rightarrow B) = \lambda a.$*min*$(B)$

Double Recursion

g = $\lambda n.$ *n equals zero* $\rightarrow$ *one*
[] (g(n minus one) plus
g(n minus one)) minus one

$graph(F^0(\bot)) = \{\}$
$graph(F^1(\bot)) = \{(zero, one)\}$
$graph(F^1(\bot)) = \{(zero, one), (one, one)\}$
$graph(F^2(\bot)) = \{(zero, one), (one, one), (two, one)\}$
...
$graph(F^{i+1}(\bot)) = \{(zero, one), \ldots, (i, one)\}$

fix F = $\lambda n.$ *one*

Stepwise construction of graph yields insight!

Simultaneous Definitions

f, g: $Nat \rightarrow Nat_\perp$

$f = \lambda x.$ *x equals zero* $\rightarrow$ *g(zero)*

[] *f(g(x minus one)) plus two*

$g = \lambda y.$ *y equals zero* $\rightarrow$ *zero*

[] *y times f(y minus one)*

$T = Nat \rightarrow Nat_\perp$

F: $(T \times T) \rightarrow (T \times T)$

$F = \lambda(f,g).(\ldots, \ldots)$

$F^0(\perp) = (\{\}, \{\})$

$F^1(\perp) = (\{\}, \{(zero, zero)\})$

$F^2(\perp) = (\{(zero,zero)\}, \{(zero, zero)\})$

$\ldots$

$F^5(\perp) = (\{(zero,zero), (one, two), (two, two)\},$

$\{(zero, zero), (one, zero),$

$(two, four), (three, six)\})$

$F^i(\perp) = F^5(\perp), i > 5$

$fix(F) = (f,g)$

While Loops

C[[**while** B **do** C]] =

fix($\lambda f.\underline{\lambda} s.$ **B**[[B]]$s \rightarrow f($**C**[[C]]$s)$ [] s)

Function: $Store_\perp \rightarrow Store_\perp$

Example:

C[[**while** A>0 **do** (A:=A-1; B:=B+1)]]
= fix F where

F = $\lambda f.\underline{\lambda} s.$ *test* $s \rightarrow f($*adjust* $s)$ [] s

test = **B**[[A > 0]]

adjust = **C**[[A:=A-1; B:=B+1]]

Partial function graphs:

- Each pair in graph shows store prior to loop entry and after loop exit.
- Each graph $F^{i+1}(\perp)$ contains those pairs whose input stores finish processing in at most i iterations.

Example

graph($F^0(\bot)$)=$\{\}$
graph($F^1(\bot)$)=$\{$
($\{$ ([[A]],*zero*), ([[B]],*zero*), ... $\}$,
$\{$ ([[A]],*zero*), ([[B]],*zero*), ... $\}$),
...
($\{$ ([[A]],*zero*), ([[B]],*four*), ... $\}$,
$\{$ ([[A]],*zero*), ([[B]],*four*), ... $\}$), ... $\}$
graph($F^2(\bot)$)=$\{$
($\{$ ([[A]],*zero*), ([[B]],*zero*), ... $\}$,
$\{$ ([[A]],*zero*), ([[B]],*zero*), ... $\}$),
...
($\{$ ([[A]],*zero*), ([[B]],*four*), ... $\}$,
$\{$ ([[A]],*zero*), ([[B]],*four*), ... $\}$),
...
($\{$ ([[A]],*one*), ([[B]],*zero*), ... $\}$,
$\{$ ([[A]],*zero*), ([[B]],*one*), ... $\}$),
...
($\{$ ([[A]],*one*), ([[B]],*four*), ... $\}$,
$\{$ ([[A]],*zero*), ([[B]],*five*), ... $\}$), ... $\}$

While Loops

Representation by finite subfunctions

$$
\begin{aligned}
&\mathbf{C}[\![\textbf{while B do C}]\!] = \sqcup\{ \\
&\quad \underline{\lambda} s.\bot, \\
&\quad \underline{\lambda} s.\mathbf{B}[\![\mathrm{B}]\!]s \to \bot \;[]\; s, \\
&\quad \underline{\lambda} s.\mathbf{B}[\![\mathrm{B}]\!]s \to (\mathbf{B}[\![\mathrm{B}]\!](\mathbf{C}[\![\mathrm{C}]\!]s) \to \bot \;[] \\
&\mathbf{C}[\![\mathrm{C}]\!]s) \\
&\qquad\quad [] \; s, \\
&\quad \underline{\lambda} s.\mathbf{B}[\![\mathrm{B}]\!]s \to (\mathbf{B}[\![\mathrm{B}]\!](\mathbf{C}[\![\mathrm{C}]\!]s) \to \\
&\qquad (\mathbf{B}[\![\mathrm{B}]\!](\mathbf{C}[\![\mathrm{C}]\!](\mathbf{C}[\![\mathrm{C}]\!]s)) \to \bot \\
&\qquad [] \; \mathbf{C}[\![\mathrm{C}]\!](\mathbf{C}[\![\mathrm{C}]\!]s)) \; [] \; \mathbf{C}[\![\mathrm{C}]\!]s) \; [] \; s, \ldots\} \\
&= \sqcup\{ \\
&\quad \mathbf{C}[\![\textbf{diverge}]\!], \\
&\quad \mathbf{C}[\![\textbf{if B then diverge else skip}]\!], \\
&\quad \mathbf{C}[\![\textbf{if B then (C; if B then diverge else skip)} \\
&\qquad \textbf{else skip}]\!], \\
&\quad \mathbf{C}[\![\textbf{if B then (C; if B then} \\
&\qquad\qquad \textbf{(C; if B then diverge else skip)} \\
&\qquad \textbf{else skip) else skip}]\!], \ldots\}
\end{aligned}
$$

Loop iteration can be understood by sequence of non-iterating programs.

Reasoning about Least Fixed Points

- Fixed Point Induction Principle:

  To prove $P(\textit{fix}\ F)$*, it suffices to prove*

  1. $P(\bot)$
  2. $P(d) \Rightarrow P(F(d))$*, for arbitrary* $d \in D$

  for pointed cpo D*, continuous functional* $F : D \to D$*, and inclusive predicate* $P : D \to B$*.*

- Inclusiveness of predicates

  If predicate holds for every element of chain, it also holds for its least upper bound.

- All universally quantified combinations of conjunctions/disjunctions that use only $\sqsubseteq$ over functional expressions are inclusive.

Mainly useful for showing equivalences of program constructs.

Reasoning about Least Fixed Points

C[[**repeat** C **until** B]] = fix($\lambda f.\underline{\lambda} s.$
let $s' =$ **C**[[C]]s in **B**[[B]]$s' \to s'$ [] $(f s')$)

C[[C; **while** ¬B **do** C]] $\stackrel{?}{=}$ **C**[[**repeat** C **until** B]]

Proof: $P(f,g) = \forall s. f(\mathbf{C}[\![\mathsf{C}]\!]s) = (gs)$

1. P($\bot$, $\bot$) holds obviously.
2. Prove $P(F(f), G(g))$

   $F = (\lambda f.\underline{\lambda} s.$ **B**[[¬ B]]$s \to f($**C**[[C]]$s)$ [] $s)$
   $G = (\lambda f.\underline{\lambda} s.$ let $s' =$ **C**[[C]]s in **B**[[B]]s'
   $\to s'$ [] $(f s'))$

   (a) $F(f)($**C**[[C]]$\bot) = \bot = G(g)(\bot)$.

   (b) $s \neq \bot$:

   i. **C**[[C]]$s = \bot$: $F(f)(\bot) = \bot =$(let $s' = \bot$ in **B**[[B]]$s' \to s'$ [] $(gs')) = G(g)(\bot)$.

   ii. **C**[[C]]$s = s_0 \neq \bot$: $F(f)(s_0) =$**B**[[¬ B]]s_0 $\to$ $f($**C**[[C]]$s_0)[]s_0$ = **B**[[B]]$s_0 \to s_0$ [] $f($**C**[[C]]$s_0)$ = **B**[[B]]$s_0 \to s_0$ [] $f(gs_0) = G(g)(s)$