# Domain Theory I

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

# 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 subparts.*

**C**[[**while** B **do** C]] $= w$
  where $w$: $\mathit{Store}_\perp \rightarrow \mathit{Store}_\perp$
  $w = \underline{\lambda} s.$**B**[[B]]$s \rightarrow w($**C**[[C]]$s)$ [] $s$

*Recursion in syntax exchanged for recursion in function notation!*

# Recursive Function Definitions

- Function Definition

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

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

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

Only one function satisfies specification:

graph(factorial) =
$\{$(zero, one), (one, one),
(two, two), (three, six),
$\ldots$, $(i$, $i!)$, $\ldots\}$

# Simplification

fac(three)
$\longrightarrow$ three equals zero
    $\longrightarrow$ one [] three times fac(three minus one)
$=$ three times fac(three minus one)
$=$ three times fac(two)
$\longrightarrow$ three times (two equals zero
    $\longrightarrow$ one [] two times fac(two minus one))
$=$ three times (two times fac(one))
$\longrightarrow$ three times (two times (one equals zero
    $\longrightarrow$ one [] one times fac(one minus one)))
$=$ three times (two times one (times fac(zero)))
$\longrightarrow$ three times (two times (one times (zero equals zero
    $\longrightarrow$ 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: $\{$(zero, one)$\}$
  - two: $\{$(zero, one), (one, one)$\}$
  - $i+1$: $\{$(zero, one), (one, one), $\ldots(i,\ i!)\}$
- Graph at stage $i$ defines function *fac*$_i$.
  - Consistency with each other:

    graph(fac$_i$) $\subseteq$ *graph*(fac$_{i+1}$)
  - Consistency with ultimate solution:

    graph(fac$_i$) $\subseteq$ *graph*(factorial)
  - Consequently

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

# Partial Functions

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

  $(a, b) \in \mathsf{graph}(\mathsf{factorial})$
  $\rightarrow (a, b) \in \mathsf{graph}(\mathsf{fac}_i)$ (for some $i$)

- Consequently

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

- Thus

  $\mathsf{graph}(\mathsf{factorial}) = \cup_{i=0}^{\infty} \mathsf{graph}(\mathsf{fac}_i)$

*Factorial function can be totally understood in terms of the finite subfunctions fac$_i$!*

# Partial Functions

- Representations of sub-functions

  $\mathsf{fac}_0 = \lambda n.\ \perp$
  $\mathsf{fac}_{i+1} = \lambda n.\ n$ equals zero $\rightarrow$ one
  [] $n$ times $\mathsf{fac}_i(n$ 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$_\perp$) $\rightarrow$ (Nat $\rightarrow$ Nat$_\perp$)
  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

  $\mathsf{fac}_{i+1} = \mathsf{F}(\mathsf{fac}_i) = \mathsf{F}^i(\bot)$
  $\bot := (\lambda n.\bot)$

- Function graph

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

- Fixed point property

  $\mathsf{graph}(\mathsf{F}(\mathsf{factorial})) = \mathsf{graph}(\mathsf{factorial})$
  $\mathsf{F}(\mathsf{factorial}) = \mathsf{factorial}$

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

# q Function

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

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

$Q^1(\perp) = \lambda n.n \textit{ equals zero}$
$\quad \rightarrow \textit{one} \;[]\; (\lambda n.\perp)(n \textit{ plus one})$
$= \lambda n.n \textit{ equals zero} \rightarrow \textit{one} \;[]\; \perp$
$\text{graph}(Q^1(\perp)) = \{(\text{zero, one})\}$

$Q^2(\perp) = Q(Q^1(\perp)) = \lambda n.n \textit{ equals zero}$
$\quad \rightarrow \textit{one} \;[]\; ((n \textit{ plus one}) \text{ equals zero} \rightarrow \textit{one} \;[]\; \perp)$
$\text{graph}(Q^2(\perp)) = \{(\text{zero, one})\}$

# q Function

- Convergence has occured

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

- Resulting graph

  $\cup_{i=0}^{\infty}$ graph$(Q^i(\perp)) = \{($zero, one$)\}$

- Fix point property

  $Q($qlimit$)$ $=$ qlimit

- Still many solutions possible

  graph(q$_k$) $=$

  $\{$ (zero, one), (one, k), . . . , (i, k), . . . $\}$

- 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 $F$.

*graph*(*fix* $F$) $= \cup_{i=0}^{\infty}$ 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

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

Simplification rule

$$\mathsf{fix\ F} = \mathsf{F(fix\ F)}$$

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

*Fixed point property justifies rec. unfolding!*

# Double Recursion

$$\begin{aligned}\mathsf{g} = \lambda\mathsf{n}.\ &\mathsf{n\ equals\ zero} \longrightarrow \mathsf{one}\\ &[]\ (\mathsf{g(n\ minus\ one)\ plus}\\ &\qquad \mathsf{g(n\ minus\ one))\ minus\ one}\end{aligned}$$

graph($F^0(\perp)$) = $\{\}$
graph($F^1(\perp)$) = $\{$(zero, one)$\}$
graph($F^1(\perp)$) = $\{$(zero, one), (one, one)$\}$
graph($F^2(\perp)$) = $\{$(zero, one), (one, one), (two, one)$\}$
...
graph($F^{i+1}(\perp)$) = $\{$(zero, one), ..., ($i$, 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).(. . . , . . . )
F$^0$($\perp$) = ({}, {})
F$^1$($\perp$) = ({}, {(zero, zero)})
F$^2$($\perp$) = ({(zero,zero)}, {(zero, zero)})
. . .
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*$_\perp \longrightarrow$ *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($F^0(\perp)$)={}
graph($F^1(\perp)$)={
( { ([[A]],zero), ([[B]],zero), ...},
{ ([[A]],zero), ([[B]],zero), ...} ),
...
( { ([[A]],zero), ([[B]],four), ...},
{ ([[A]],zero), ([[B]],four), ...} ), ...}
graph($F^2(\perp)$)={
( { ([[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

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

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

# Reasoning about Least Fixed Points

- Fixed Point Induction Principle:

  To prove $P(\text{fix } F)$, 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 $F : D \rightarrow D$, and inclusive predicate $P : D \rightarrow \mathbf{B}$.

- 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]]$s' \rightarrow s'$ [] $(f s')$)

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

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

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

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

   (a) $F(f)($**C**[[C]]$\perp) = \perp = G(g)(\perp)$.
   (b) $s \neq \perp$:
      i. **C**[[C]]$s = \perp$: $F(f)(\perp)$=$\perp$=(let $s' = \perp$ in **B**[[B]]$s' \rightarrow s'$ [] $(g s')) = G(g)(\perp)$.
      ii. **C**[[C]]$s = s_0 \neq \perp$: $F(f)(s_0)$=**B**[[¬ B]]$s_0 \rightarrow f($**C**[[C]]$s_0)$[]$s_0$ = **B**[[B]]$s_0 \rightarrow s_0$ [] $f($**C**[[C]]$s_0)$ = **B**[[B]]$s_0 \rightarrow s_0$ [] $f(g s_0) = G(g)(s)$