Recursive Functions on Lazy Lists via Domains and Topologies

Andreas Lochbihler

Institute of Information Security
ETH Zurich, Switzerland

Johannes Hölzl

Institut für Informatik
TU München, Germany

ITP 2014

Running example: filtering lazy lists

Task: Given a **codatatype**

define a recursive function

and **prove** properties.

Running example: filtering lazy lists

Task: Given a **codatatype** $\alpha\ llist = [\,] \mid \alpha \cdot \alpha\ llist$

define a recursive function

$$\begin{aligned} lfilter\ P\ [\,] &= [\,] \\ lfilter\ P\ (x \cdot xs) &= (if\ P\ x\ then\ x \cdot lfilter\ P\ xs\ else\ lfilter\ P\ xs) \end{aligned}$$

and **prove** properties.

$$lfilter\ P\ (lfilter\ Q\ xs) = lfilter\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Running example: filtering lazy lists

Task: Given a **codatatype** $\alpha\ \mathit{llist} = [\,] \mid \alpha \cdot \alpha\ \mathit{llist}$

finite and **infinite** lists

define a recursive function

$$\mathit{lfilter}\ P\ [\,] = [\,]$$
$$\mathit{lfilter}\ P\ (x \cdot xs) = (\mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \mathit{else}\ \mathit{lfilter}\ P\ xs)$$

and **prove** properties.

$$\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Running example: filtering lazy lists

Task: Given a **codatatype** $\alpha\ \mathit{llist} = [\,] \mid \alpha \cdot \alpha\ \mathit{llist}$

finite and **infinite** lists

define a recursive function

$$\begin{aligned} \mathit{lfilter}\ P\ [\,] &= [\,] \\ \mathit{lfilter}\ P\ (x \cdot xs) &= (\mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \mathit{else}\ \mathit{lfilter}\ P\ xs) \end{aligned}$$

and **prove** properties.

$$\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Usual definition principles

- well-founded recursion
- guarded/primitive corecursion

Running example: filtering lazy lists

Task: Given a **codatatype** α *llist* $= [\,] \mid \alpha \cdot \alpha$ *llist*

finite and **infinite** lists

define a recursive function

$$\begin{aligned} &\mathit{lfilter}\ P\ [\,] &&= [\,] \\ &\mathit{lfilter}\ P\ (x \cdot xs) &&= (\mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \mathit{else}\ \mathit{lfilter}\ P\ xs) \end{aligned}$$

and **prove** properties.

$$\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Usual definition principles

- ~~well-founded recursion~~
- guarded/primitive corecursion

Running example: filtering lazy lists

Task: Given a **codatatype** α *llist* $= [] \mid \alpha \cdot \alpha$ *llist*

finite and **infinite** lists

define a recursive function

$$\begin{aligned} \textit{lfilter}\ P\ [] &= [] \\ \textit{lfilter}\ P\ (x \cdot xs) &= (\textit{if}\ P\ x\ \textit{then}\ x \cdot \textit{lfilter}\ P\ xs\ \textit{else}\ \textit{lfilter}\ P\ xs) \end{aligned}$$

guarded

and **prove** properties.

$$\textit{lfilter}\ P\ (\textit{lfilter}\ Q\ xs) = \textit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Usual definition principles

- ~~well-founded recursion~~
- guarded/primitive corecursion

Running example: filtering lazy lists

Task: Given a **codatatype** α *llist* $= [\,] \mid \alpha \cdot \alpha$ *llist*

finite and **infinite** lists

define a recursive function

guarded unguarded

$$\mathit{lfilter}\ P\ [\,] = [\,]$$
$$\mathit{lfilter}\ P\ (x \cdot xs) = (\mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \mathit{else}\ \mathit{lfilter}\ P\ xs)$$

and **prove** properties.

$$\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Usual definition principles

- ~~well-founded recursion~~
- ~~guarded/primitive corecursion~~

Running example: filtering lazy lists

Task: Given a **codatatype** α *llist* $= [\,] \mid \alpha \cdot \alpha$ *llist*

finite and **infinite** lists

define a recursive function

guarded   unguarded

$$\mathit{lfilter}\ P\ [\,] = [\,]$$
$$\mathit{lfilter}\ P\ (x \cdot xs) = (\mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \mathit{else}\ \mathit{lfilter}\ P\ xs)$$

and **prove** properties.

$$\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Usual

- ~~well~~
- ~~gua~~

lfilter is **underspecified:**

$$\mathit{lfilter}\ (\leq 0)\ (1 \cdot [1, 1, 1, \ldots]) = \mathit{lfilter}\ (\leq 0)\ [1, 1, 1, \ldots]$$

Beyond well-founded and guarded corecursion

$$\begin{aligned} lfilter\ P\ [\,] &= [\,] \\ lfilter\ P\ (x \cdot xs) &= (if\ P\ x\ then\ x \cdot lfilter\ P\ xs\ else\ lfilter\ P\ xs) \end{aligned}$$

$$lfilter\ P\ (lfilter\ Q\ xs) = lfilter\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Previous approaches:

Beyond well-founded and guarded corecursion

$$\begin{aligned} \mathit{lfilter}\ P\ [\,] &= [\,] \\ \mathit{lfilter}\ P\ (x \cdot xs) &= (\mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \mathit{else}\ \mathit{lfilter}\ P\ xs) \end{aligned}$$

$$\mathit{lfinite}\ xs \lor (\forall n.\ \exists x \in \mathit{lset}\ (\mathit{ldrop}\ n\ xs).\ P\ x \land Q\ x) \longrightarrow$$
$$\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \land Q\ x)\ xs$$

Previous approaches:

Partiality leave unspecified for infinite lists w/o satisfying elements

- ⊕ close to specification
- ⊖ properties need preconditions
- ⊖ no proof principles

Beyond well-founded and guarded corecursion

$$\mathit{lfilter}\ P\ [\,] = [\,]$$
$$\mathit{lfilter}\ P\ (x \cdot xs) = (\mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \underbrace{\mathit{else}\ \mathit{lfilter}\ P\ xs}_{\mathit{if}\ \neg\ \mathit{find}\ P\ xs\ \mathit{then}\ [\,]\ \mathit{else}})$$

$$\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$$

Previous approaches:

Partiality leave unspecified for infinite lists w/o satisfying elements

- ⊕ close to specification
- ⊖ properties need preconditions
- ⊖ no proof principles

Search function check whether there are more elements

- ⊕ total function, no preconditions
- ⊖ additional lemmas about search function necessary
- ⊖ ad hoc solution

Two views on *lfilter*

$$\mathit{lfilter} :: (\alpha \Rightarrow \mathit{bool}) \Rightarrow \alpha\ \mathit{llist} \Rightarrow\ \alpha\ \mathit{llist}$$

Two views on *lfilter*

$lfilter :: (\alpha \Rightarrow bool) \Rightarrow \alpha\ llist \Rightarrow \alpha\ llist$

1. produces a list corecursively

- $lfilter :: \beta \Rightarrow \alpha\ llist$
- find chain-complete partial order on α *llist*
- take the least fixpoint for *lfilter*

Two views on *lfilter*

$lfilter :: (\alpha \Rightarrow bool) \Rightarrow \alpha\ llist \Rightarrow \alpha\ llist$

1. produces a list corecursively

- $lfilter :: \beta \Rightarrow \alpha\ llist$
- find chain-complete partial order on α *llist*
- take the least fixpoint for *lfilter*

proof principles

$\rightsquigarrow$ domain theory

fixpoint induction
structural induction

Two views on *lfilter*

$$\textit{lfilter} :: (\alpha \Rightarrow \textit{bool}) \Rightarrow \alpha\ \textit{llist} \Rightarrow \alpha\ \textit{llist}$$

2. consumes a list recursively

- *lfilter* :: α *llist* $\Rightarrow \beta$
- find topology on α *llist*
- define *lfilter* on finite lists by well-founded recursion
- take the limit for infinite lists

1. produces a list corecursively

- *lfilter* :: $\beta \Rightarrow \alpha$ *llist*
- find chain-complete partial order on α *llist*
- take the least fixpoint for *lfilter*

proof principles

$\leadsto$ domain theory

fixpoint induction
structural induction

Two views on *lfilter*

$lfilter :: (\alpha \Rightarrow bool) \Rightarrow \alpha\ llist \Rightarrow \alpha\ llist$

2. consumes a list recursively

- $lfilter :: \alpha\ llist \Rightarrow \beta$
- find topology on α *llist*
- define *lfilter* on finite lists by well-founded recursion
- take the limit for infinite lists

1. produces a list corecursively

- $lfilter :: \beta \Rightarrow \alpha\ llist$
- find chain-complete partial order on α *llist*
- take the least fixpoint for *lfilter*

proof principles

$\rightsquigarrow$ topology

convergence on closed sets
uniqueness of limits

$\rightsquigarrow$ domain theory

fixpoint induction
structural induction

Proof principles pay off

Isabelle proofs of $\mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$

Paulson's

```
subsection {* Numerous lemmas required to prove @{text lfilter conj} *}

lemma findRel conj lemma [rule format]:
     "(l,l') ∈ findRel q
      ==> l' = LCons x l'' --> p x --> (l,l') ∈ findRel (%x. p x & q x)"
by (erule findRel.induct, auto)

lemmas findRel conj = findRel conj lemma [OF   refl]

lemma findRel not conj Domain [rule format]:
     "(l,l'') ∈ findRel (%x. p x & q x)
      ==> (l, LCons x l') ∈ findRel q --> ~ p x -->
          l' ∈ Domain (findRel (%x. p x & q x))"
by (erule findRel.induct, auto)

lemma findRel conj2 [rule format]:
     "(l,lxx) ∈ findRel q
      ==> lxx = LCons x lx --> (lx,lz) ∈ findRel(%x. p x & q x) --> ~ p x
          --> (l,lz) ∈ findRel (%x. p x & q x)"
by (erule findRel.induct, auto)

lemma findRel lfilter Domain conj [rule format]:
     "(lx,ly) ∈ findRel p
      ==> ∀l. lx = lfilter q l --> l ∈ Domain (findRel(%x. p x & q x))"
apply (erule findRel.induct)
 apply (blast dest!: sym [THEN lfilter eq LCons] intro: findRel conj, auto)
apply (drule sym [THEN lfilter eq LCons], auto)
apply (drule spec)
apply (drule refl [THEN rev mp])
apply (blast intro: findRel conj2)
done


lemma findRel conj lfilter [rule format]:
     "(l,l'') ∈ findRel(%x. p x & q x)
      ==> l'' = LCons y l' -->
          (lfilter q l, LCons y (lfilter q l')) ∈ findRel p"
by (erule findRel.induct, auto)

lemma lfilter conj lemma:
     "(lfilter p (lfilter q l), lfilter (%x. p x & q x) l)
      ∈ llistD Fun (range (%u. (lfilter p (lfilter q u),
                               lfilter (%x. p x & q x) u)))"
apply (case tac "l ∈ Domain (findRel q)")
 apply (subgoal tac [2] "l ~: Domain (findRel (%x. p x & q x))")
  prefer 3 apply (blast intro: rev subsetD [OF   Domain findRel mono])
 txt{*There are no @{text qs} in @{text l}: both lists are @{text LNil}*}
 apply (simp all add: Domain findRel iff, clarify)
txt{*case @{text "q x"}*}
apply (case tac "p x")
 apply (simp all add: findRel conj [THEN findRel imp lfilter])
 txt{*case @{text "q x"} and @{text "~(p x)"} *}
apply (case tac "l' ∈ Domain (findRel (%x. p x & q x))")
 txt{*subcase: there is no @{text "p & q"} in @{text l'} and therefore none in @{text l}*}
 apply (subgoal tac [2] "l ~: Domain (findRel (%x. p x & q x))")
  prefer 3 apply (blast intro: findRel not conj Domain)
 apply (subgoal tac [2] "lfilter q l' ~: Domain (findRel p) ")
  prefer 3 apply (blast intro: findRel lfilter Domain conj)
 txt{*    {\dots} and therefore too, no @{text p} in @{text "lfilter q l'"}.
   Both results are @{text LNil}*}
 apply (simp all add: Domain findRel iff, clarify)
txt{*subcase: there is a @{text "p & q"} in @{text l'} and therefore also one in @{text l} *}
apply (subgoal tac " (l, LCons xa l'a) ∈ findRel (%x. p x & q x) ")
 prefer 2 apply (blast intro: findRel conj2)
apply (subgoal tac " (lfilter q l', LCons xa (lfilter q l'a)) ∈ findRel p")
 apply simp
apply (blast intro: findRel conj lfilter)
done


lemma lfilter conj: "lfilter p (lfilter q l) = lfilter (%x. p x & q x) l"
apply (rule tac l = "l" in llist fun equalityI, simp all)
apply (blast intro: lfilter conj lemma rev subsetD [OF   llistD Fun mono])
done
```

Structural induction

```
lemma lfilter lfilter: "lfilter P (lfilter Q xs) = lfilter (λx. P x ∧ Q x) xs"
by(induction xs) simp all
```

Fixpoint induction

```
lemma lfilter lfilter: "lfilter P (lfilter Q xs) = lfilter (λx. P x ∧ Q x) xs"
proof -
  have "∀xs. lfilter P (lfilter Q xs) ⊑ lfilter (λx. P x ∧ Q x) xs"
    by(rule lfilter.fixp induct)(auto split: llist.split)
  moreover have "∀xs. lfilter (λx. P x ∧ Q x) xs ⊑ lfilter P (lfilter Q xs)"
    by(rule lfilter.fixp induct)(auto split: llist.split)
  ultimately show "lfilter P (lfilter Q xs) = lfilter (λx. P x ∧ Q x) xs"
    by(blast intro: lprefix antisym)
qed
```

Continuous extension

```
lemma lfilter' lfilter': "lfilter' P (lfilter' Q xs) = lfilter' (λx. Q x ∧ P x) xs"
  by (rule tendsto closed[of   xs]) (auto intro!: closed Collect eq isCont lfilter)
```

The producer view: least fixpoints

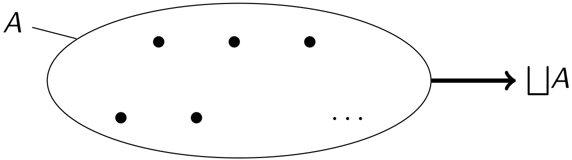
- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$

The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

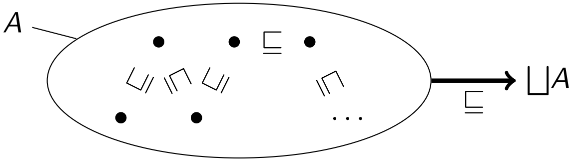
$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$



The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

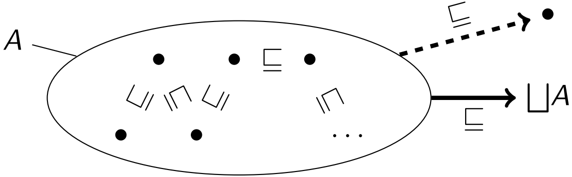
$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$



The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

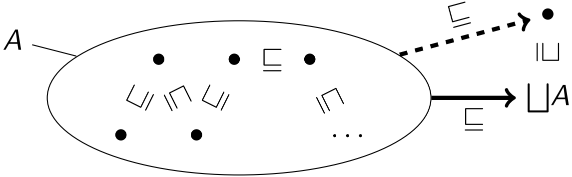
$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$



The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

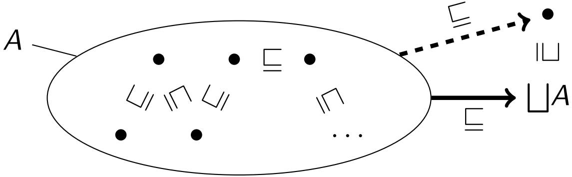
$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = [\,]$



The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$

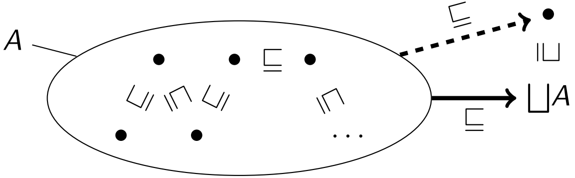


- lift $(\sqsubseteq, \bigsqcup)$ point-wise to function space $\beta \Rightarrow \alpha$ *llist*

The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$



- lift $(\sqsubseteq, \bigsqcup)$ point-wise to function space $\beta \Rightarrow \alpha$ *llist*

Knaster-Tarski theorem:
If f on a ccpo is monotone, then f has a least fixpoint.

The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = [\,]$

partial-function (llist) *lfilter* :: $(\alpha \Rightarrow \mathit{bool}) \Rightarrow \alpha\ \mathit{llist} \Rightarrow \alpha\ \mathit{llist}$ where

$$\mathit{lfilter}\ P\ xs = (\mathit{case}\ xs\ \mathit{of}\ [\,] \Rightarrow [\,] \mid x \cdot xs \Rightarrow \mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ xs\ \mathit{else}\ \mathit{lfilter}\ P\ xs)$$

- lift $(\sqsubseteq, \bigsqcup)$ point-wise to function space $\beta \Rightarrow \alpha\ \mathit{llist}$

Knaster-Tarski theorem:
If f on a ccpo is monotone, then f has a least fixpoint.

The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$

partial-function (llist) *lfilter* :: $(\alpha \Rightarrow \mathit{bool}) \Rightarrow \alpha\ \mathit{llist} \Rightarrow \alpha\ \mathit{llist}$ where

$$\mathit{lfilter}\ P\ \mathit{xs} = (\mathit{case}\ \mathit{xs}\ \mathit{of}\ [] \Rightarrow [] \mid x \cdot \mathit{xs} \Rightarrow \mathit{if}\ P\ x\ \mathit{then}\ x \cdot \mathit{lfilter}\ P\ \mathit{xs}\ \mathit{else}\ \mathit{lfilter}\ P\ \mathit{xs})$$

Light-weight domain theory

- ⊕ [] represents "undefined", no additional values in α *llist*
- ⊕ full function space $\Rightarrow$, no continuity restrictions
- ⊖ less automation
- ⊖ less expressive (no nested or higher-order recursion)

The producer view: induction proofs

- structural induction

$$\frac{adm\ Q \qquad Q\ [] \qquad \forall x\ xs.\ lfinite\ xs \wedge Q\ xs \longrightarrow Q\ (x \cdot xs)}{Q\ xs}$$

- fixpoint induction rule generated for *lfilter*

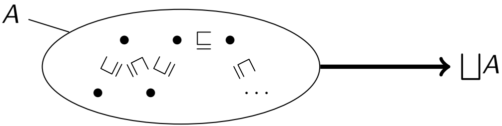
The producer view: induction proofs

- structural induction

$$\frac{adm\ Q \qquad Q\ [] \qquad \forall x\ xs.\ \mathit{lfinite}\ xs \wedge Q\ xs \longrightarrow Q\ (x \cdot xs)}{Q\ xs}$$

- fixpoint induction rule generated for *lfilter*

Induction is sound only
for **admissible** statements Q



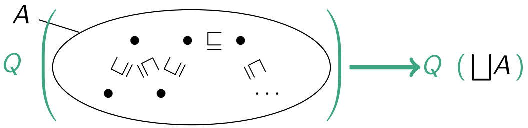
The producer view: induction proofs

- structural induction

$$\frac{\mathit{adm}\ Q \qquad Q\ [] \qquad \forall x\ xs.\ \mathit{lfinite}\ xs \wedge Q\ xs \longrightarrow Q\ (x \cdot xs)}{Q\ xs}$$

- fixpoint induction rule generated for *lfilter*

Induction is sound only
for **admissible** statements Q



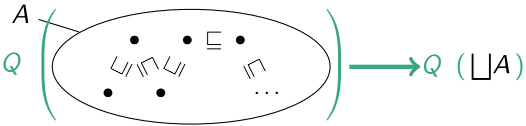
The producer view: induction proofs

- structural induction

$$\frac{\mathit{adm}\ Q \qquad Q\ [] \qquad \forall x\ xs.\ \mathit{lfinite}\ xs \wedge Q\ xs \longrightarrow Q\ (x \cdot xs)}{Q\ xs}$$

- fixpoint induction rule generated for *lfilter*

Induction is sound only
for **admissible** statements Q



lemma *lfilter P* (*lfilter Q xs*) = *lfilter* ($\lambda x.\ P\ x \wedge Q\ x$) *xs*
by(induction *xs*) simp_all

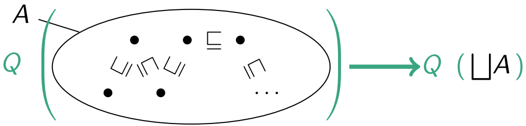
The producer view: induction proofs

- structural induction

$$\frac{adm\ Q \qquad Q\ [\,] \qquad \forall x\ xs.\ \mathit{lfinite}\ xs \wedge Q\ xs \longrightarrow Q\ (x \cdot xs)}{Q\ xs}$$

- fixpoint induction rule generated for *lfilter*

Induction is sound only
for **admissible** statements Q



proof automation via syntactic decomposition rules for admissibility

$$adm\ (\lambda xs.\ \mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ \ xs\ ) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ \ xs\ \ )$$

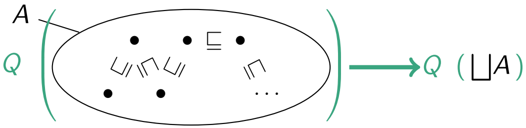
The producer view: induction proofs

- structural induction

$$\frac{adm\ Q \qquad Q\ [\,] \qquad \forall x\ xs.\ \mathit{lfinite}\ xs \wedge Q\ xs \longrightarrow Q\ (x \cdot xs)}{Q\ xs}$$

- fixpoint induction rule generated for *lfilter*

Induction is sound only
for **admissible** statements Q



proof automation via syntactic decomposition rules for admissibility

$$adm\ (\lambda xs.\ \mathit{lfilter}\ P\ (\mathit{lfilter}\ Q\ xs) = \mathit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs)$$

atomic predicate **continuous contexts**

The consumer view: continuous extensions

datatype α *list* = [] | $\alpha \cdot \alpha$ *list*
filter :: ($\alpha \Rightarrow$ *bool*) $\Rightarrow \alpha$ *list* $\Rightarrow \alpha$ *list*

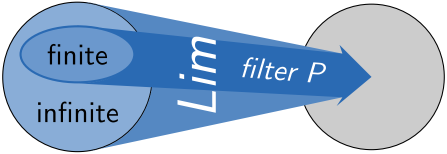
1. Define *filter* recursively on **finite** lists.

The consumer view: continuous extensions

datatype α *list* $=$ [] | $\alpha \cdot \alpha$ *list*
filter :: $(\alpha \Rightarrow$ *bool*$) \Rightarrow \alpha$ *list* $\Rightarrow \alpha$ *list*

lfilter P xs $=$ *Lim* (*filter P*) *xs*

1. Define *filter* recursively on **finite** lists.
2. Take the limit.

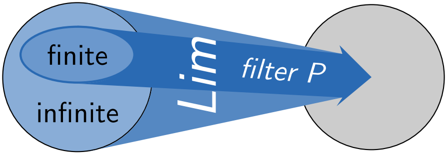


The consumer view: continuous extensions

datatype α *list* = [] | $\alpha \cdot \alpha$ *list*
filter :: ($\alpha \Rightarrow$ *bool*) $\Rightarrow \alpha$ *list* $\Rightarrow \alpha$ *list*

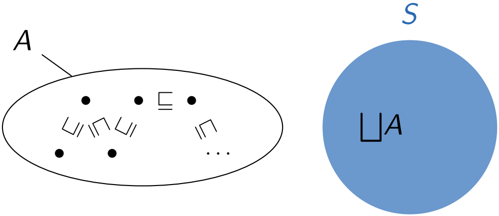
lfilter P xs = *Lim* (*filter P*) *xs*

1. Define *filter* recursively on **finite** lists.
2. Take the limit.



introduce **CCPO topology**

$\rightsquigarrow$ define the open sets

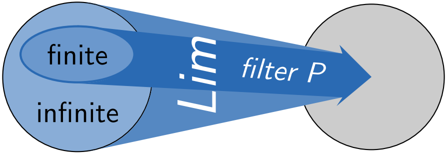


The consumer view: continuous extensions

datatype α *list* = [] | $\alpha \cdot \alpha$ *list*
filter :: ($\alpha \Rightarrow$ *bool*) $\Rightarrow \alpha$ *list* $\Rightarrow \alpha$ *list*

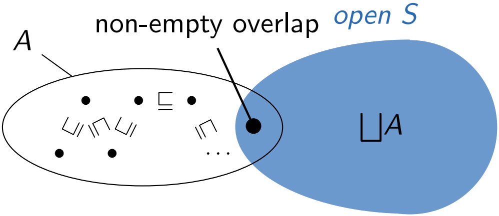
lfilter P xs = *Lim* (*filter P*) *xs*

1. Define *filter* recursively on **finite** lists.
2. Take the limit.



introduce **CCPO topology**

$\rightsquigarrow$ define the open sets



The consumer view: continuous extensions

datatype α *list* = [] | $\alpha \cdot \alpha$ *list*
filter :: $(\alpha \Rightarrow bool) \Rightarrow \alpha$ *list* $\Rightarrow \alpha$ *list*

lfilter P xs = *Lim* (*filter P*) *xs*

1. Define *filter* recursively on **finite** lists.
2. Take the limit.

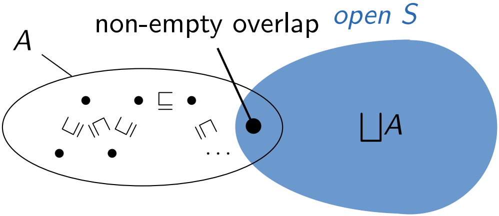
Properties of a CCPO topology

- ⊕ limits are unique
- ⊕ finite elements are discrete, i.e., *open* {*xs*}

} not the Scott topology!

introduce **CCPO topology**

⇝ define the open sets



The consumer view: proving

1. Prove that *filter P* is continuous!
   follows from monotonicity of *filter*

2. Proof rule **convergence on a closed set** (specialised for α *llist*):

$$\frac{\mathit{closed}\ \{xs \mid Q\ xs\} \qquad \forall ys.\ \mathit{lfinite}\ ys \wedge ys \sqsubseteq xs \longrightarrow Q\ ys}{Q\ xs}$$

lemma *lfilter P* (*lfilter Q xs*) = *lfilter* ($\lambda x.\ P\ x \wedge Q\ x$) *xs*
by (rule converge_closed[of _ *xs*]) (auto intro!: closed_eq isCont_lfilter)

The consumer view: proving

1. Prove that *filter P* is continuous!
   follows from monotonicity of *filter*

2. Proof rule **convergence on a closed set** (specialised for α *llist*):

$$\frac{\textit{closed}\ \{xs \mid Q\ xs\} \qquad \forall ys.\ \textit{lfinite}\ ys \wedge ys \sqsubseteq xs \longrightarrow Q\ ys}{Q\ xs}$$

lemma $\textit{lfilter}\ P\ (\textit{lfilter}\ Q\ xs) = \textit{lfilter}\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$
by (rule converge_closed[of _ *xs*]) (auto intro!: closed_eq isCont_lfilter)

decomposition rules
for closedness

Summary

Comparison	**least fixpoint**		**continuous extension**
ccpo	on result type		on parameter type
monotonicity	of the functional		of the function
proof principles	structural induction fixpoint induction	=	convergence on a closed set

Available in the AFP entry Coinductive

Summary

Comparison	**least fixpoint**	**continuous extension**
ccpo	on result type	on parameter type
monotonicity	of the functional	of the function
proof principles	structural induction = fixpoint induction	convergence on a closed set

Available in the AFP entry Coinductive

Which codatatypes can be turned into *useful* ccpos?

- ⊕ extended naturals $enat = 0 \mid eSuc\ enat$
- ⊕ n-ary trees $\alpha\ tree = Leaf \mid Node\ \alpha\ (\alpha\ tree)\ (\alpha\ tree)$

} **finite truncations**

- ⊖ streams $\alpha\ stream = Stream\ \alpha\ (\alpha\ stream)$ **no finite elements**

Two views on *lfilter*

lfilter :: $(\alpha \Rightarrow bool) \Rightarrow \alpha\ llist \Rightarrow \alpha\ llist$

2. consumes a list recursively

- *lfilter* :: $\alpha\ llist \Rightarrow \beta$
- find topology on α *llist*
- define *lfilter* on finite lists by well-founded recursion
- take the limit for infinite lists

1. produces a list corecursively

- *lfilter* :: $\beta \Rightarrow \alpha\ llist$
- find chain-complete partial order on α *llist*
- take the least fixpoint for *lfilter*

proof principles

$\leadsto$ topology

convergence on closed sets
uniqueness of limits

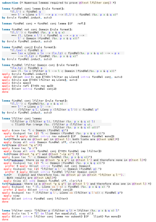
$\leadsto$ domain theory

fixpoint induction
structural induction

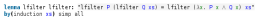
Proof principles pay off

Isabelle proofs of *lfilter* P (*lfilter* Q xs) = *lfilter* $(\lambda x.\ P\ x \wedge Q\ x)$ xs

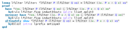
Paulson's



Structural induction



Fixpoint induction



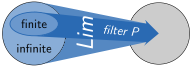
Continuous extension

The consumer view: continuous extensions

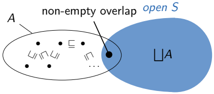
datatype α *list* = [] | $\alpha \cdot \alpha$ *list*
filter :: $(\alpha \Rightarrow bool) \Rightarrow \alpha\ list \Rightarrow \alpha\ list$

lfilter P xs = *Lim* (*filter* P) xs

1. Define *filter* recursively on **finite** lists.
2. Take the limit.



introduce **CCPO topology**
$\leadsto$ define the open sets



The producer view: least fixpoints

- prefix order $\sqsubseteq$ defined coinductively
- least upper bound $\bigsqcup Y$ defined by primitive corecursion

$(\sqsubseteq, \bigsqcup)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$

partial-function (llist) *lfilter* :: $(\alpha \Rightarrow bool) \Rightarrow \alpha\ llist \Rightarrow \alpha\ llist$ where
lfilter P xs = (*case* xs *of* [] $\Rightarrow$ []
| $x \cdot xs \Rightarrow$ *if* P x *then* $x \cdot$ *lfilter* P xs *else* *lfilter* P xs)

- lift $(\sqsubseteq, \bigsqcup)$ point-wise to function space $\beta \Rightarrow \alpha\ llist$

Knaster-Tarski theorem:
If f on a ccpo is monotone, then f has a least fixpoint.