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

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

and **prove** properties.

\[
\text{lfilter } P \ (\text{lfilter } Q \ xs) = \text{lfilter } (\lambda x. \ P \ x \land Q \ x) \ xs
\]
Task: Given a codatatype \( \alpha \text{llist} = [] \mid \alpha \cdot \alpha \text{llist} \)

define a recursive function

\[ \text{lfilter} \; P \; [] = [] \]
\[ \text{lfilter} \; P \; (x \cdot xs) = (\text{if} \; P \; x \; \text{then} \; x \cdot \text{lfilter} \; P \; xs \; \text{else} \; \text{lfilter} \; P \; xs) \]

and prove properties.

\[ \text{lfilter} \; P \; (\text{lfilter} \; Q \; xs) = \text{lfilter} \; (\lambda x. \; P \; x \land Q \; x) \; xs \]
Running example: filtering lazy lists

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

**define** a recursive function
\[
lfilter \ P \ [ ] = [ ]
\]
\[
lfilter \ P \ (x \cdot xs) = (if \ P \ x \ then \ x \cdot lfilter \ P \ xs \ else \ lfilter \ P \ xs)
\]

and **prove** properties.
\[
lfilter \ P \ (lfilter \ Q \ xs) = lfilter \ (\lambda x. \ P \ x \land Q \ x) \ xs
\]

**Usual definition principles**
- well-founded recursion
- guarded/primitive corecursion
Running example: filtering lazy lists

**Task:** Given a **codatatype** \( \alpha\ \text{llist} = [] \mid \alpha \cdot \alpha\ \text{llist} \)

**define** a recursive function

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

and **prove** properties.

\[
\text{lfilter } P (\text{lfilter } Q xs) = \text{lfilter } (\lambda x. P x \land Q x) xs
\]

**Usual definition principles**

- well-founded recursion
- guarded/primitive corecursion
Running example: filtering lazy lists

**Task:** Given a codatatype \( \alpha \ llist = [] | \alpha \cdot \alpha \ llist \) define a recursive function

\[
\text{lfilter} \ P \ [\ ] = [\ ] \quad \text{guarded}
\]
\[
\text{lfilter} \ P \ (x \cdot xs) = ( \text{if} \ P \ x \ \text{then} \ x \cdot \text{lfilter} \ P \ xs \ \text{else} \ \text{lfilter} \ P \ xs)
\]

and prove properties.

\[
\text{lfilter} \ P \ (\text{lfilter} \ Q \ xs) = \text{lfilter} \ (\lambda x. \ P \ x \land Q \ x) \ xs
\]

**Usual definition principles**
- well founded recursion
- guarded/primitive corecursion
Running example: filtering lazy lists

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

Define a recursive function

$$lfilter \ P \ [] = [] \quad \text{guarded} \quad \text{unguarded}$$

$$lfilter \ P \ (x \cdot xs) = (if \ P \ x \ then \ x \cdot lfilter \ P \ xs \ else \ lfilter \ P \ xs)$$

and prove properties.

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

Usual definition principles

- well-founded recursion
- guarded/primitive corecursion
Running example: filtering lazy lists

**Task:** Given a codatatype \( \alpha \text{l} \text{list} = [] | \alpha \cdot \alpha \text{l} \text{list} \)

**define** a recursive function

\[
\text{lfilter } P \ [ ] = [] \quad \text{guarded} \quad \text{unguarded}
\]

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

and **prove** properties.

\[
\text{lfilter } P \ (\text{lfilter } Q \ xs) = \text{lfilter } (\lambda x. \ P x \land Q x) \ xs
\]

**Usual**

- well
- guarded

\textbf{Ifilter is underspecified:}

\[
\text{lfilter } (\leq 0) \ (1 \cdot [1, 1, 1, \ldots]) = \text{lfilter } (\leq 0) \ [1, 1, 1, \ldots]
\]
Beyond well-founded and guarded corecursion

\[\text{lfilter } P \; [] = []\]
\[\text{lfilter } P \; (x \cdot xs) = (\text{if } P \; x \; \text{then } x \cdot \text{lfilter } P \; xs \; \text{else } \text{lfilter } P \; xs)\]

\[\text{lfilter } P \; (\text{lfilter } Q \; xs) = \text{lfilter } (\lambda x. \; P \; x \land Q \; x) \; xs\]

Previous approaches:
Beyond well-founded and guarded corecursion

\[
\text{lfilter } P \; \emptyset = \emptyset
\]
\[
\text{lfilter } P \; (x \cdot xs) = (\text{if } P \; x \; \text{then } x \cdot \text{lfilter } P \; xs \; \text{else } \text{lfilter } P \; xs)
\]

\[
\text{lfinite } xs \lor (\forall n. \exists x \in \text{lset} \; (\text{ldrop } n \; xs). \; P \; x \land Q \; x) \longrightarrow
\]
\[
\text{lfilter } P \; (\text{lfilter } Q \; xs) = \text{lfilter} \; (\lambda x. \; P \; x \land Q \; x) \; xs
\]

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

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

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 \textit{ifilter}

\begin{center}
\texttt{ifilter :: (\alpha \Rightarrow bool) \Rightarrow \alpha llist \Rightarrow \alpha llist}
\end{center}
Two views on *lfilter*

\[
\text{lfilter} :: (\alpha \rightarrow \text{bool}) \Rightarrow \alpha \text{ llist} \Rightarrow \alpha \text{ llist}
\]

1. produces a list corecursively
   - \text{lfilter} :: \beta \Rightarrow \alpha \text{ llist}
   - find chain-complete partial order on \(\alpha \text{ llist}\)
   - take the least fixpoint for \text{lfilter}

\[\mapsto\] domain theory
  - fixpoint induction
  - structural induction

\[\mapsto\] topology
  - convergence on closed sets
  - uniqueness of limits
Two views on \textit{lfilter}

\[ \text{lfilter} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \text{ llist} \Rightarrow \alpha \text{ llist} \]

1. produces a list corecursively

- \text{lfilter} :: \beta \Rightarrow \alpha \text{ llist}
- find chain-complete partial order on \(\alpha \text{ llist}\)
- take the least fixpoint for \text{lfilter}

\[ \leadsto \text{proof principles} \]

\[ \leadsto \text{domain theory} \]
- fixpoint induction
- structural induction
Two views on \textit{lfilter}

\texttt{lfilter} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \text{llist} \Rightarrow \alpha \text{llist}

1. produces a list corecursively
   - \texttt{lfilter} :: \beta \Rightarrow \alpha \text{llist}
   - find chain-complete partial order on \alpha \text{llist}
   - define \texttt{lfilter} on finite lists by well-founded recursion
   - take the least fixpoint for \texttt{lfilter}

2. consumes a list recursively
   - \texttt{lfilter} :: \alpha \text{llist} \Rightarrow \beta
   - find topology on \alpha \text{llist}
   - take the limit for infinite lists

\textbf{proof principles}

\Rightarrow \text{domain theory}

\begin{align*}
\text{fixpoint induction} \\
\text{structural induction}
\end{align*}
Two views on \textit{lfilter}

\[ lfilter :: (\alpha \Rightarrow bool) \Rightarrow \alpha \ llist \Rightarrow \alpha \ llist \]

1. produces a list corecursively
   - \( lfilter :: \alpha \ llist \Rightarrow \beta \)
   - find chain-complete partial order on \( \alpha \ llist \)
   - define \( lfilter \) on finite lists by well-founded recursion
   - take the least fixpoint for \( lfilter \)

2. consumes a list recursively
   - \( lfilter :: \beta \Rightarrow \alpha \ llist \)
   - find topology on \( \alpha \ llist \)
   - define \( lfilter \) on finite lists by well-founded recursion
   - take the limit for infinite lists

**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 \( \text{lfilter } P \ (\text{lfilter } Q \ xs) = \text{lfilter } (\lambda x. \ P \ x \land Q \ x) \ xs \)

Paulson's

**subsection** (Numerous lemmas required to prove \( \text{lfilter } \text{lfilter} \text{conj} \))

```isabelle
lemma \text{lfilter} \text{lfilter} \text{conj} \text{rule}: 
  \((\text{lfilter } P) \ (\text{lfilter } Q \ \text{xs}) \rightarrow \text{lfilter } P \ (\lambda x. \ P \ x \land Q \ x) \ \text{xs}) 
by \text{erule \text{lfilter} \text{lfilter} \text{induct}, \text{auto})
```

**lemma** \text{lfilter} \text{lfilter} \text{conj} \text{lemmas} \text{OP} \text{refl}:

```isabelle
lemma \text{lfilter} \text{lfilter} \text{conj} \text{Domain \rule}: 
  \((\text{lfilter } P) \ (\text{lfilter } Q \ \text{xs}) \rightarrow \text{lfilter } P \ (\lambda x. \ P \ x \land Q \ x) \ \text{xs}) 
by \text{erule \text{lfilter} \text{lfilter} \text{induct}, \text{auto})
```

**lemma** \text{lfilter} \text{lfilter} \text{conj} \text{rule}: 

```isabelle
lemma \text{lfilter} \text{lfilter} \text{conj} \text{rule}: 
  \((\text{lfilter } P) \ (\text{lfilter } Q \ \text{xs}) \rightarrow \text{lfilter } P \ (\lambda x. \ P \ x \land Q \ x) \ \text{xs}) 
by \text{erule \text{lfilter} \text{lfilter} \text{induct}, \text{auto})
```

**Structural induction**

```isabelle
lemma \text{lfilter} \text{lfilter}: 
  \text{lfilter } P \ (\text{lfilter } Q \ \text{xs}) = \text{lfilter } (\lambda x. \ P \ x \land Q \ x) \ \text{xs} 
by \text{induction \text{xs}) \ \text{simp all}
```

**Fixpoint induction**

```isabelle
lemma \text{lfilter} \text{lfilter}: 
  \text{lfilter } P \ (\text{lfilter } Q \ \text{xs}) = \text{lfilter } (\lambda x. \ P \ x \land Q \ x) \ \text{xs} 
```

**Continuous extension**

```isabelle
lemma \text{lfilter} \text{lfilter}: 
  \text{lfilter } P \ (\text{lfilter } Q \ \text{xs}) = \text{lfilter } (\lambda x. \ P \ x \land Q \ x) \ \text{xs} 
by \text{rule \text{tendsto \text{closed}) \ \text{x\text{all}) \ \text{auto \text{closed \text{Collect \text{eq \text{isCont \text{filter}})}}}
```

Lochbihler (ETHZ), Hölz (TUM)

Recursively functions on lazy lists

ITP 2014 5 / 11
The producer view: least fixpoints

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

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

Light-weight domain theory

$[\ ]$ represents "undefined", no additional values in $\alpha \text{llist}$

full function space $\Rightarrow$, no continuity restrictions

Recursive functions on lazy lists

ITP 2014 6 / 11
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 $\bot = []$

Light-weight domain theory

\[ \text{\texttt{\textit{lfilter}} :: } (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \text{llist} \Rightarrow \alpha \text{llist} \]

\[ \text{lfilter } P \text{ } xs = \begin{cases} & [], \text{ if } P x \text{ is not defined} \\ & \text{case } xs \text{ of } \[] \Rightarrow \[] \\ & x \cdot xs \Rightarrow \text{if } P x \text{ then } x \cdot \text{lfilter } P \text{ } xs \text{ else } \text{lfilter } P \text{ } xs \\ \end{cases} \]

Lochbihler (ETHZ), Hörl (TUM)
The producer view: least fixpoints

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

$(\sqsubseteq, \biguplus)$ forms a **chain-complete partial order (CCPO)** with $\bot = []$
The producer view: least fixpoints

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

$(\sqsubseteq, \biguplus)$ forms a chain-complete partial order (CCPO) with $\bot = []$
The producer view: least fixpoints

- prefix order \( \sqsubseteq \) defined coinductively
- least upper bound \( \bigcup \) defined by primitive corecursion

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

- lift \((\sqsubseteq, \bigcup)\) point-wise to function space \( \beta \Rightarrow \alpha \) \( \text{llist} \)
The producer view: least fixpoints

- prefix order \( \sqsubseteq \) defined coinductively
- least upper bound \( \bigcup Y \) defined by primitive corecursion

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

- lift \((\sqsubseteq, \bigcup)\) point-wise to function space \( \beta \Rightarrow \alpha \text{ llist} \)

**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 $\biguplus Y$ defined by primitive corecursion

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

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

\[
\text{lfilter} \ P \ \text{xs} = (\text{case} \ \text{xs} \ \text{of} \ [] \Rightarrow [] \\
| \ x \cdot \text{xs} \Rightarrow \text{if} \ P \ x \ \text{then} \ x \cdot \text{lfilter} \ P \ \text{xs} \ \text{else} \ \text{lfilter} \ P \ \text{xs})
\]

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

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

**Light-weight domain theory**

- \( [\ ] \) represents “undefined”, no additional values in \( \alpha llist \)
- full function space \( \Rightarrow \), no continuity restrictions
- less automation
- less expressive (no nested or higher-order recursion)
The producer view: induction proofs

- structural induction

\[
\text{adm } Q \quad Q [ ] \quad \forall x. \text{INFINITE } xs \land Q xs \quad \rightarrow \quad Q (x \cdot xs)
\]

- fixpoint induction rule generated for \textit{Ifilter}
The producer view: induction proofs

- **structural induction**

\[
\frac{\text{adm } Q \quad Q []}{\forall x \ \text{lfinite } xs \land Q \ xs \rightarrow Q (x \cdot xs)}
\]

- **fixpoint induction** rule generated for \textit{lfilter}

Induction is sound only for \textit{admissible} statements \( Q \)
The producer view: induction proofs

- structural induction

\[
\text{adm } Q \quad Q [] \quad \forall x \; xs. \; \text{lfinite } xs \land Q \; xs \implies Q \; (x \cdot xs)
\]

- fixpoint induction rule generated for \textit{lfilter}

Induction is sound only for \textit{admissible} statements \( Q \).
The producer view: induction proofs

- **structural induction**

\[
\text{adm } Q \quad Q [] \quad \forall x \; xs. \ lfinite \; xs \land Q \; xs \quad \rightarrow \quad Q \; (x \cdot xs)
\]

- **fixpoint induction** rule generated for \(lfilter\)

Induction is sound only for **admissible** statements \(Q\)

**lemma** \(lfilter\; P\; (lfilter\; Q\; xs) = lfilter\; (\lambda x. \; P\; x \land Q\; x)\; xs\)

by (induction \(xs\)) simp_all
The producer view: induction proofs

- **structural induction**

  \[
  \text{adm } Q \quad Q \[\quad \forall x \; x s. \; \text{lfinite} \; x s \land Q \; x s \quad \rightarrow \quad Q \; (x \cdot x s) \quad Q \; x s
  \]

- **fixpoint induction** rule generated for *lfilter*

  \[
  \text{Induction is sound only for admissible statements } Q
  \]

proof automation via syntactic decomposition rules for admissibility

\[
\text{adm } (\lambda x s. \; \text{lfilter} \; P \; (\text{lfilter} \; Q \; x s) \; ) = \text{lfilter} \; (\lambda x. \; P \; x \land Q \; x) \; x s
\]
The producer view: induction proofs

- **structural induction**

  \[
  \text{adm } Q \quad Q \; [] \quad \forall x \; xs. \; \text{lfinite } xs \land Q \; xs \rightarrow Q \; (x \cdot xs)
  \]

  \[Q \; xs\]

- **fixpoint induction** rule generated for \texttt{lfilter}

  Induction is sound only for admissible statements \(Q\)

  proof automation via syntactic decomposition rules for admissibility

  \[
  \text{adm } (\lambda xs. \text{lfilter } P \; (\text{lfilter } Q \; xs)) = \text{lfilter } (\lambda x. \; P \; x \land Q \; x) \; xs
  \]

  atomic predicate  continuous contexts
The consumer view: continuous extensions

**datatype** $\alpha \ list = [] | \alpha \cdot \alpha \ list$

**filter** :: $(\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \ list \Rightarrow \alpha \ list$

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

---

Lochbihler (ETHZ), Hölzl (TUM)  Recursive functions on lazy lists  ITP 2014 8 / 11
The consumer view: continuous extensions

**datatype** $\alpha$ *list* = $\mathbb{[]}$ | $\alpha \cdot \alpha$ *list*

*filter* :: $(\alpha \rightarrow \text{bool}) \rightarrow \alpha$ *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** \( \alpha \) list = [] | \( \alpha \cdot \alpha \) list

\[ \text{filter} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \text{ list} \Rightarrow \alpha \text{ list} \]

\[ l\text{filter} \, P \, xs = \text{Lim} (\text{filter} \, P) \, xs \]

1. Define \( \text{filter} \) recursively on finite lists.
2. Take the limit.

Introduce **CCPO topology**

\( \sim \) define the open sets

Lochbihler (ETHZ), Hölzl (TUM)
The consumer view: continuous extensions

**datatype** $\alpha \ list = [] \mid \alpha \cdot \alpha \ list$

```
define filter recursively on finite lists.
```

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

1. Define $\text{filter}$ recursively on finite lists.

2. Take the limit.

```
introduce CCPO topology
```

```
\sim define the open sets
```

```
non-empty overlap $open S$
```

Lochbihler (ETHZ), Hölzl (TUM)
The consumer view: continuous extensions

```plaintext
datatype α list = [] | α · α list
```

```plaintext
filter :: (α ⇒ bool) ⇒ α list ⇒ α list
```

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

2. Take the limit.

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

**Properties of a CCPO topology**

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

```
\[ \begin{array}{c}
\text{not the Scott topology!} \\
\end{array} \]
```

introduce **CCPO topology**

```
\[ \begin{array}{c}
\text{define the open sets} \\
\end{array} \]
```

```
\[ \begin{array}{c}
\text{non-empty overlap} \quad \text{open} \ S \\
\end{array} \]
```

Lochbihler (ETHZ), Hölzl (TUM)
The consumer view: proving

1. Prove that \( \text{filter } P \) is continuous!
   follows from monotonicity of \( \text{filter} \)

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

\[
\begin{align*}
\text{closed } \{ xs \mid Q \; xs \} & \quad \forall ys. \text{ lfinite } ys \land ys \sqsubseteq xs \rightarrow Q \; ys \\
& \quad Q \; xs
\end{align*}
\]

**lemma** \( \text{lfilter } P \; (\text{lfilter } Q \; xs) = \text{lfilter } (\lambda x. P \; x \land Q \; x) \; xs \)

**by** (rule converge_closed[of _ xs]) (auto intro!: closed_eq isCont_lfilter )
1. Prove that \( \text{filter } P \) is continuous! It follows from monotonicity of \( \text{filter} \).

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

\[
\text{closed } \{xs \mid Q \ x \} \quad \forall ys. \ \text{Infinite } ys \land ys \subseteq xs \quad \rightarrow \quad Q \ ys
\]

**lemma** \( \text{Ifilter } P \ ( \text{Ifilter } Q \ xs) = \text{Ifilter } (\lambda x. \ P \ x \land Q \ x) \ xs \)

**by** (rule converge\_closed\[of \_ \ xs]) (auto intro!: \text{closed\_eq \ isCont\_Ifilter} )

**decomposition rules** for closedness
### Summary

<table>
<thead>
<tr>
<th>Comparison</th>
<th>least fixpoint</th>
<th>continuous extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccpo</td>
<td>on result type</td>
<td>on parameter type</td>
</tr>
<tr>
<td>monotonicity</td>
<td>of the functional</td>
<td>of the function</td>
</tr>
<tr>
<td>proof principles</td>
<td>structural induction (=) convergence on a closed set</td>
<td>fixpoint induction</td>
</tr>
</tbody>
</table>

Available in the AFP entry Coinductive
## Summary

<table>
<thead>
<tr>
<th>Comparison</th>
<th>least fixpoint</th>
<th>continuous extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccpo</td>
<td>on result type</td>
<td>on parameter type</td>
</tr>
<tr>
<td>monotonicity</td>
<td>of the functional</td>
<td>of the function</td>
</tr>
<tr>
<td>proof principles</td>
<td>structural induction $\Rightarrow$ convergence on a closed set</td>
<td>fixpoint induction</td>
</tr>
</tbody>
</table>

Available in the AFP entry Coinductive

Which codatatypes can be turned into *useful* ccpos?

- extended naturals $\textit{enat} = 0 \mid e\textit{Suc enat}$
- $n$-ary trees $\alpha \textit{tree} = \textit{Leaf} \mid \textit{Node} \alpha (\alpha \textit{tree}) (\alpha \textit{tree})$
- streams $\alpha \textit{stream} = \textit{Stream} \alpha (\alpha \textit{stream})$

\{ finite truncations \[ no finite elements \]
Two views on `lfilter`

\[
lfilter :: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \text{l}l\text{ist} \Rightarrow \alpha \text{l}l\text{ist}
\]

1. produces a list corecursively
   - `lfilter :: \beta \Rightarrow \alpha \text{l}l\text{ist}
   - find topology on `\alpha \text{l}l\text{ist}`
   - define `lfilter` on finite lists by well-founded recursion
   - take the least fixpoint for `lfilter`

2. consumes a list recursively
   - `lfilter :: \alpha \text{l}l\text{ist}
   - find chain-complete partial order on `\alpha \text{l}l\text{ist}
   - define `lfilter` on finite lists by well-founded recursion
   - take the limit for infinite lists

Proof principles pay off

Isabelle proofs of `lfilter P \ (lfilter Q xs) = lfilter (\lambda x. P x \land Q x ) \ xs`

Paulson's
- Structural induction
- Fixpoint induction
- Continuous extension
- Uniqueness of limits

The consumer view: continuous extensions

```
datatype \alpha list = [] | \alpha \cdot \alpha list
filter :: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha list \Rightarrow \alpha list
lfilter P xs = \text{Lim} \ (filter P) \ xs
```

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

The producer view: least fixpoints

- prefix order `\leq` defined coinductively
- least upper bound `\bigcup Y` defined by primitive corecursion
- `(\leq, \bigcup)` forms a chain-complete partial order (CCPO) with `\bot = []`

```
partial-function (\text{l}l\text{ist}) lfilter :: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \text{l}l\text{ist} \Rightarrow \alpha \text{l}l\text{ist}
where
lfilter P xs = (\text{case} \ xs \ \text{of} \ [\ ] \Rightarrow [\ ] \\
               x \cdot \ xs \Rightarrow \text{if} \ P \ x \ \text{then} \ x \cdot \ lfilter P \ xs \ \text{else} \ lfilter P \ xs)
```

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