Domain-theoretic Modelling of Functional Programming Languages

Tie Hou    Ulrich Berger

Department of Computer Science
Swansea University, UK

29 August, 2011
Roadmap

- Motivation
Roadmap

- Motivation
  - Why are we interested in this problem?
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
- Curry-style System
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal

- Curry-style System
  - Syntax of Curry-style terms
Roadmap

- **Motivation**
  - Why are we interested in this problem?
  - Our goal

- **Curry-style System**
  - Syntax of Curry-style terms
  - $\mu \nu$-types
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu-$types
- Church-style System
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal

- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu-$types

- Church-style System
  - Syntax of Church-style terms
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu$-types
- Church-style System
  - Syntax of Church-style terms
  - Recursive types
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu-$types
- Church-style System
  - Syntax of Church-style terms
  - Recursive types
- Denotational Semantics
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal

- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu$-types

- Church-style System
  - Syntax of Church-style terms
  - Recursive types

- Denotational Semantics
  - Semantics of Curry-style terms and set of types
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu-$types
- Church-style System
  - Syntax of Church-style terms
  - Recursive types
- Denotational Semantics
  - Semantics of Curry-style terms and set of types
  - Soundness theorem for Curry-style terms
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu-$types
- Church-style System
  - Syntax of Church-style terms
  - Recursive types
- Denotational Semantics
  - Semantics of Curry-style terms and set of types
  - Soundness theorem for Curry-style terms
  - Semantics of Church-style terms and set of types
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal

- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu$-types

- Church-style System
  - Syntax of Church-style terms
  - Recursive types

- Denotational Semantics
  - Semantics of Curry-style terms and set of types
  - Soundness theorem for Curry-style terms
  - Semantics of Church-style terms and set of types
  - Soundness theorem for Church-style terms
Roadmap

- Motivation
  - Why are we interested in this problem?
  - Our goal
- Curry-style System
  - Syntax of Curry-style terms
  - $\mu\nu-$types
- Church-style System
  - Syntax of Church-style terms
  - Recursive types
- Denotational Semantics
  - Semantics of Curry-style terms and set of types
  - Soundness theorem for Curry-style terms
  - Semantics of Church-style terms and set of types
  - Soundness theorem for Church-style terms
- Coincidence of Two Systems
Motivation

- PhD project: program extraction from proofs

Correctness of programs
Proof system should be extendible by more and more powerful (co)inductive principles
This requires more and more powerful (co)inductive types and (co)recursion schemes
Want a correctness proof that works for all of these extensions
Want systems to have arbitrary fixed point types and general recursion
Haskell is such a system, but its semantics is problematic

Hou, Berger (Swansea)
Domain-theoretic Modelling
29 August, 2011
Motivation

- PhD project: program extraction from proofs
- Correctness of programs
PhD project: program extraction from proofs
Correctness of programs
Proof system should be extendible by more and more powerful (co)inductive principles
Motivation

- PhD project: program extraction from proofs
- Correctness of programs
- Proof system should be extendible by more and more powerful (co)inductive principles
- This requires more and more powerful (co)inductive types and (co)recursion schemes
Motivation

- PhD project: program extraction from proofs
- Correctness of programs
- Proof system should be extendible by more and more powerful (co)inductive principles
- This requires more and more powerful (co)inductive types and (co)recursion schemes
- Want a correctness proof that works for all of these extensions
Motivation

- PhD project: program extraction from proofs
- Correctness of programs
- Proof system should be extendible by more and more powerful (co)inductive principles
- This requires more and more powerful (co)inductive types and (co)recursion schemes
- Want a correctness proof that works for all of these extensions
- Want systems to have arbitrary fixed point types and general recursion
Motivation

- PhD project: program extraction from proofs
- Correctness of programs
- Proof system should be extendible by more and more powerful (co)inductive principles
- This requires more and more powerful (co)inductive types and (co)recursion schemes
- Want a correctness proof that works for all of these extensions
- Want systems to have arbitrary fixed point types and general recursion
- Haskell is such a system, but its semantics is problematic
Question on semantics of the functional programming language Haskell, which has

- Curry-style $\lambda$-abstraction, e.g. $\lambda x.x$
- unrestricted fixed point types
- unrestricted recursive functions
Question on semantics of the functional programming language Haskell, which has

- Curry-style $\lambda$-abstraction, e.g. $\lambda x.x$
- unrestricted fixed point types
- unrestricted recursive functions

Does it make sense denotationally?
Question on semantics of the functional programming language Haskell, which has

- Curry-style $\lambda$-abstraction, e.g. $\lambda x. x$
- unrestricted fixed point types
- unrestricted recursive functions

Does it make sense denotationally?

Our goal is

- to study and compare the denotational semantics of Curry- (i.e. untyped) and Church-style (i.e. typed) $\lambda$-terms
Motivation Continued

Question on semantics of the functional programming language Haskell, which has

- Curry-style $\lambda$-abstraction, e.g. $\lambda x.x$
- unrestricted fixed point types
- unrestricted recursive functions

Does it make sense denotationally?

Our goal is

- to study and compare the denotational semantics of Curry- (i.e. untyped) and Church-style (i.e. typed) $\lambda$-terms

What do we mean by a functional programming language?
Motivation Continued

Question on semantics of the functional programming language Haskell, which has

- Curry-style $\lambda$-abstraction, e.g. $\lambda x.x$
- unrestricted fixed point types
- unrestricted recursive functions

Does it make sense denotationally?

Our goal is

- to study and compare the denotational semantics of Curry- (i.e. untyped) and Church-style (i.e. typed) $\lambda$-terms

What do we mean by a functional programming language?

- typed $\lambda$-calculus + fixed point types + recursion
Curry-style System

Syntax of Curry-style terms

\[ M, N, R_i ::= x \mid \lambda x. M \mid MN \mid \text{rec } x. M \mid C(M_1, \ldots, M_n) \mid \text{case } M \text{ of } \{ C_i(x_i) \rightarrow R_i \}_{i \in \{1, \ldots, n\}} \]

the constructors \( C, C_i \) are: Nil, Left, Right, Pair and In
Curry-style System

Syntax of Curry-style terms

\[ M, N, R_i ::= x \mid \lambda x. M \mid MN \mid \text{rec } x. M \mid C(M_1, \ldots, M_n) \mid \]
\[ \text{case } M \text{ of } \{ C_i(x_i) \rightarrow R_i \}_{i\in\{1,\ldots,n\}} \]

the constructors \( C, C_i \) are: Nil, Left, Right, Pair and In

\( \mu\nu \)-types

\[ \text{Type } \exists \rho, \sigma, \tau ::= \alpha \mid \rho \rightarrow \sigma \mid 1 \mid \rho \times \sigma \mid \rho + \sigma \mid \mu\alpha.\rho \mid \nu\alpha.\rho \]

where in \( \mu\alpha.\rho \) and \( \nu\alpha.\rho \), \( \rho \) is positive in \( \alpha \)
Curry-style Typing Rules

- Expressions of the form
\[
\Gamma = x_1 : \rho_1, \ldots, x_n : \rho_n
\]

<table>
<thead>
<tr>
<th>Table: Typing Rules for Curry-style</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \text{Nil} : 1 )</td>
</tr>
<tr>
<td>( \frac{\Gamma, x : \rho \vdash M : \sigma}{\Gamma \vdash \lambda x. M : \rho \to \sigma} )</td>
</tr>
<tr>
<td>( \frac{\Gamma \vdash M : \rho \to \sigma}{\Gamma \vdash MN : \sigma} )</td>
</tr>
<tr>
<td>( \frac{\Gamma \vdash M : \rho}{\Gamma \vdash \text{Left}(M) : \rho + \sigma} )</td>
</tr>
</tbody>
</table>
Table: Typing Rules for Curry-style continued

\[
\Gamma \vdash M : \rho + \sigma \quad \Gamma, x_1 : \rho \vdash L : \tau \quad \Gamma, x_2 : \sigma \vdash R : \tau \\
\Gamma \vdash \text{case } M \text{ of } \{\text{Left}(x_1) \to L; \text{Right}(x_2) \to R\} : \tau
\]

(9)

\[
\Gamma \vdash M : \rho \times \sigma \quad \Gamma, x : \rho, y : \sigma \vdash N : \tau \\
\Gamma \vdash \text{case } M \text{ of } \{\text{Pair}(x, y) \to N\} : \tau
\]

(10)

\[
\Gamma \vdash M : \rho'[\mu\alpha.\rho'/\alpha] \quad \Gamma \vdash \text{In}(M) : \mu\alpha.\rho' \\
\Gamma \vdash M : \rho'[\nu\alpha.\rho'/\alpha] \quad \Gamma \vdash \text{In}(M) : \nu\alpha.\rho'
\]

(11) \hspace{1cm} (12)

\[
\Gamma \vdash M : \mu\alpha.\rho' \quad \Gamma, x : \rho'[\mu\alpha.\rho'/\alpha] \vdash N : \sigma \\
\Gamma \vdash \text{case } M \text{ of } \{\text{In}(x) \to N\} : \sigma
\]

(13)

\[
\Gamma \vdash M : \nu\alpha.\rho' \quad \Gamma, x : \rho'[\nu\alpha.\rho'/\alpha] \vdash N : \sigma \\
\Gamma \vdash \text{case } M \text{ of } \{\text{In}(x) \to N\} : \sigma
\]

(14)
Church-style System

Syntax of Church-style terms

\[ M, N, R_i ::= x \mid \lambda x : \rho. M \mid MN \mid \text{rec } x : \tau. M \mid C(M_1, \ldots, M_n) \mid \]
\[ \text{case } M \text{ of } \{ C_i(x_i) \to R_i \}_{i \in \{1,\ldots,n\}} \]

Recursive types

\[ \textbf{Type } \ni \rho, \sigma, \tau ::= \alpha \mid \rho \to \sigma \mid 1 \mid \rho \times \sigma \mid \rho + \sigma \mid \text{fix } \alpha. \rho \]
## Church-style Typing Rules

**Table: Typing Rules for Church-style**

<table>
<thead>
<tr>
<th>Rule Number</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\Gamma \vdash \text{Nil} : 1$</td>
</tr>
<tr>
<td>2</td>
<td>$\Gamma, x : \rho \vdash x : \rho$</td>
</tr>
<tr>
<td>3</td>
<td>$\Gamma, x : \rho \vdash M : \sigma$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash \lambda x : \rho. M : \rho \rightarrow \sigma$</td>
</tr>
<tr>
<td>4</td>
<td>$\Gamma \vdash \text{rec } x : \tau. M : \tau$</td>
</tr>
<tr>
<td>5</td>
<td>$\Gamma \vdash M : \rho \rightarrow \sigma$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash N : \rho$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash M \cdot N : \sigma$</td>
</tr>
<tr>
<td>6</td>
<td>$\Gamma \vdash M : \rho$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash N : \sigma$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash \text{Pair}(M, N) : \rho \times \sigma$</td>
</tr>
<tr>
<td>7</td>
<td>$\Gamma \vdash M : \rho$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash \text{Left}(M) : \rho + \sigma$</td>
</tr>
<tr>
<td>8</td>
<td>$\Gamma \vdash M : \sigma$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash \text{Right}(M) : \rho + \sigma$</td>
</tr>
</tbody>
</table>
Table: Typing Rules for Church-style continued

\[
\begin{align*}
\Gamma \vdash M : \rho + \sigma & \quad \Gamma, x_1 : \rho \vdash L : \tau & \quad \Gamma, x_2 : \sigma \vdash R : \tau \\
\Gamma \vdash \text{case } M \text{ of } \{\text{Left}(x_1) \rightarrow L; \text{Right}(x_2) \rightarrow R\} : \tau
\end{align*}
\]  
(9)

\[
\begin{align*}
\Gamma \vdash M : \rho \times \sigma & \quad \Gamma, x : \rho, y : \sigma \vdash N : \tau \\
\Gamma \vdash \text{case } M \text{ of } \{\text{Pair}(x, y) \rightarrow N\} : \tau
\end{align*}
\]  
(10)

\[
\begin{align*}
\Gamma \vdash M : \rho' & \quad \Gamma \vdash \text{In}(M) : \text{fix } \alpha.\rho' \\
\Gamma \vdash \text{In}(M) : \text{fix } \alpha.\rho'
\end{align*}
\]  
(11)

\[
\begin{align*}
\Gamma \vdash M : \text{fix } \alpha.\rho' & \quad \Gamma, x : \rho'[\text{fix } \alpha.\rho'/\alpha] \vdash N : \sigma \\
\Gamma \vdash \text{case } M \text{ of } \{\text{In}(x) \rightarrow N\} : \sigma
\end{align*}
\]  
(12)
Curry vs. Church

Syntax of Curry-style terms

\[ M, N, R_i ::= x \mid \lambda x. M \mid MN \mid \text{rec } x. M \mid C(M_1, \ldots, M_n) \mid \]
\[ \text{case } M \text{ of } \{ C_i(x_i) \to R_i \}_{i \in \{1,\ldots,n\}} \]

Syntax of Church-style terms

\[ M, N, R_i ::= x \mid \lambda x : \rho. M \mid MN \mid \text{rec } x : \tau. M \mid C(M_1, \ldots, M_n) \mid \]
\[ \text{case } M \text{ of } \{ C_i(x_i) \to R_i \}_{i \in \{1,\ldots,n\}} \]
Curry vs. Church

- Syntax of Curry-style terms

\[ M, N, R_i ::= x \mid \lambda x.M \mid MN \mid \text{rec } x.M \mid C(M_1, \ldots, M_n) \mid \text{case } M \text{ of } \{ C_i(x_i) \rightarrow R_i \}_{i \in \{1,\ldots,n\}} \]

- Syntax of Church-style terms

\[ M, N, R_i ::= x \mid \lambda x: \rho.M \mid MN \mid \text{rec } x: \tau.M \mid C(M_1, \ldots, M_n) \mid \text{case } M \text{ of } \{ C_i(x_i) \rightarrow R_i \}_{i \in \{1,\ldots,n\}} \]

- \( \mu\nu \)-types

\[ \text{Type} \ni \rho, \sigma, \tau ::= \alpha \mid \rho \rightarrow \sigma \mid 1 \mid \rho \times \sigma \mid \rho + \sigma \mid \mu \alpha.\rho \mid \nu \alpha.\rho \]

where in \( \mu \alpha.\rho \) and \( \nu \alpha.\rho \), \( \rho \) is positive in \( \alpha \)

- Recursive types

\[ \text{Type} \ni \rho, \sigma, \tau ::= \alpha \mid \rho \rightarrow \sigma \mid 1 \mid \rho \times \sigma \mid \rho + \sigma \mid \text{fix } \alpha.\rho \]
Typing Derivation

\[
\frac{\Gamma, x : \rho \vdash M : \sigma}{\Gamma \vdash \lambda x. M : \rho \rightarrow \sigma}
\]

\[
\frac{\Gamma, x : \rho \vdash M : \sigma}{\Gamma \vdash \lambda x : \rho. M : \rho \rightarrow \sigma}
\]
Typing Derivation

\[ \Gamma, x : \rho \vdash M : \sigma \quad \Gamma, x : \rho \vdash M : \sigma \]
\[ \frac{}{\Gamma \vdash \lambda x. M : \rho \rightarrow \sigma} \quad \frac{}{\Gamma \vdash \lambda x : \rho. M : \rho \rightarrow \sigma} \]

Curry-style typing derivation \( \cong \) Church-style term
Denotational Semantics

Why denotational semantics?

- assigns mathematical meaning to systems

\[ D \cong (1 + D + (D + D) + D \times D + [D \to D]) \]

\( D \) is a Scott domain. Its elements are:

- \( \star \)
- \( \text{In}(a) \)
- \( \text{Left}(a) \)
- \( \text{Right}(a) \)
- \( \text{Pair}(a, b) \)
- \( \text{Fun}(f) \)
- \( \perp \)

\(^1\) called "Language of Realiser" because we use it for program extraction.
Why denotational semantics?

- assigns mathematical meaning to systems
- can be used to obtain information about operational semantics

\[ D \xrightarrow{\simeq} (1 + D + (D + D) + D \times D + [D \to D]) \]

\( D \) is a Scott domain. Its elements are:
- \( \star \)
- \( \text{In}(a) \)
- \( \text{Left}(a) \)
- \( \text{Right}(a) \)
- \( \text{Pair}(a, b) \)
- \( \text{Fun}(f) \)
- \( \perp \)

If \( X \) is an entity in \( \text{LoR}_1 \) (Curry or Church), we denote its interpretation by \( JX \) called “Language of Realiser” because we use it for program extraction.
**Denotational Semantics**

Why denotational semantics?
- assigns mathematical meaning to systems
- can be used to obtain information about operational semantics

No written down and complete denotational semantics of Haskell

---

\[ D \simeq (1 + D) \times (D + D + (D \rightarrow D)) + \perp \]

\( D \) is a Scott domain. Its elements are:
- \( \star \)
- \( \text{In}(a) \)
- \( \text{Left}(a) \)
- \( \text{Right}(a) \)
- \( \text{Pair}(a, b) \)
- \( \text{Fun}(f) \)
- \( \perp \)

If \( X \) is an entity in \( \text{LoR}^1 \) (Curry or Church), we denote its interpretation by \( J^k X \)

\(^1\) called "Language of Realiser" because we use it for program extraction.
Denotational Semantics

Why denotational semantics?
- assigns mathematical meaning to systems
- can be used to obtain information about operational semantics

No written down and complete denotational semantics of Haskell

Based on the domain

\[ D \simeq (1 + D + (D + D) + D \times D + [D \to D]) \bot \]

\( D \) is a Scott domain. Its elements are:
- \( \star, \text{In}(a), \text{Left}(a), \text{Right}(a), \text{Pair}(a, b), \text{Fun}(f), \bot \)

If \( X \) is an entity in \( \text{LoR}^1 \) (Curry or Church), we denote its interpretation by \( \llbracket X \rrbracket \)

\(^1\)called "Language of Realiser" because we use it for program extraction
### Value of Curry-style terms

For every environment $\eta : \text{Var} \rightarrow D$ and every Curry-style LoR term $M$, we define the value $[M]\eta \in D$ as follows

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\text{Nil}]\eta$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>$[x]\eta$</td>
<td>$\eta(x)$</td>
</tr>
<tr>
<td>$[\text{Pair}(M, N)]\eta$</td>
<td>$\text{Pair}([M]\eta, [N]\eta)$</td>
</tr>
<tr>
<td>$[\text{Left}(M)]\eta$</td>
<td>$\text{Left}([M]\eta)$</td>
</tr>
<tr>
<td>$[\text{Right}(M)]\eta$</td>
<td>$\text{Right}([M]\eta)$</td>
</tr>
<tr>
<td>$[\text{In}(M)]\eta$</td>
<td>$\text{In}([M]\eta)$</td>
</tr>
<tr>
<td>$[MN]\eta$</td>
<td>$\begin{cases} f([N]\eta) &amp; \text{if } [M]\eta = \text{Fun}(f) \ \perp &amp; \text{otherwise} \end{cases}$</td>
</tr>
</tbody>
</table>
### Value of Curry-style terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\lambda x. M) (\eta)</td>
<td>(\text{Fun}(f)) where (f \in [D \to D]) s.t. (f(a) := [M]_\eta[x := a](a \in D))</td>
</tr>
<tr>
<td>(\text{rec } x. M) (\eta)</td>
<td>(\text{LFP}(f)) where (f \in [D \to D]) s.t. (f(a) := [M]_\eta[x := a](a \in D))</td>
</tr>
<tr>
<td>(\text{case } M \text{ of } { C_i(x_i) \to R_i }) \eta)</td>
<td>(\begin{cases} [R_i]_\eta[\vec{x}<em>i := \vec{a}] &amp; \text{if } [M]</em>\eta = C_i(\vec{a}) \ \bot &amp; \text{otherwise} \end{cases})</td>
</tr>
</tbody>
</table>

where LFP stands for least fixed point defined by

\[
\text{LFP}(f) = \bigcup_{n \in \mathbb{N}} f^n(\bot)
\]
Value of $\mu\nu$-types

For every environment $\xi : \mathbf{TVar} \rightarrow \wp(D)$ and every type $\rho$, we define the value $[\rho]_\xi \subseteq D$ as follows

$$
\begin{align*}
[1]_\xi & := \{\ast\} \\
[\alpha]_\xi & := \xi(\alpha) \\
[\rho + \sigma]_\xi & := \{\text{Left}(a) \mid a \in [\rho]_\xi\} \cup \{\text{Right}(a) \mid a \in [\sigma]_\xi\} \\
[\rho \times \sigma]_\xi & := \{\text{Pair}(d_1, d_2) \mid d_1 \in [\rho]_\xi, d_2 \in [\sigma]_\xi\} \\
[\rho \rightarrow \sigma]_\xi & := \{\text{Fun}(f) \mid f \in [D \rightarrow D] \text{ s.t. } f([\rho]_\xi) \subseteq [\sigma]_\xi\} \\
[\mu\alpha.\rho]_\xi & := \text{LFP}(\wedge X \subseteq D.\text{In}([\rho]_\xi[\alpha := X])) \\
[\nu\alpha.\rho]_\xi & := \text{GFP}(\wedge X \subseteq D.\text{In}([\rho]_\xi[\alpha := X]))
\end{align*}
$$
<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $\Gamma \vdash M : \rho$, $\eta \in [\Gamma]\xi$ and $M$ contains no recursion, then $[M]\eta \in [\rho]\xi$</td>
</tr>
</tbody>
</table>
Soundness For Curry-style terms

Theorem

If $\Gamma \vdash M : \rho$, $\eta \in \llbracket \Gamma \rrbracket \xi$ and $M$ contains no recursion, then $\llbracket M \rrbracket \eta \in \llbracket \rho \rrbracket \xi$

Proof

By induction on the structure of $\Gamma \vdash M : \rho$
Soundness For Curry-style terms

**Theorem**

If $\Gamma \vdash M : \rho$, $\eta \in [\Gamma]\xi$ and $M$ contains no recursion, then $[M]\eta \in [\rho]\xi$

**Proof**

By induction on the structure of $\Gamma \vdash M : \rho$

**Remark**

Special forms of recursion can be added, e.g. guarded (co)recursion, or structural recursion.
General Recursion?

Take \( \vdash \text{rec } x.x : 1 \) as an example

- We know

\[
\llbracket \text{rec } x.x \rrbracket \eta = \text{LFP}(f) \text{ where } f(a) = \llbracket x \rrbracket \eta[x := a] = a
= \bigcup_{n \in \mathbb{N}} f^n(\bot) \text{ where } f(a) = a
= \bot
\]

- But \( \llbracket 1 \rrbracket \xi = \{ \star \} \), and \( \bot \not\in \{ \star \} \)

- That is \( \llbracket \text{rec } x.x \rrbracket \eta \not\in \llbracket 1 \rrbracket \xi \)

- Hence soundness does not hold
Value of Church-style terms

For every environment \( \zeta : [D \rightarrow D]^\text{TVar} \), \( \eta : \text{Var} \rightarrow D \), and every Church-style LoR term \( M \), we define the value \([M]^{\zeta \eta} \in D\) as follows

- \([\text{Nil}]^{\zeta \eta} := \star\)
- \([x]^{\zeta \eta} := \eta(x)\)
- \([\text{Pair}(M, N)]^{\zeta \eta} := \text{Pair}([M]^{\zeta \eta}, [N]^{\zeta \eta})\)
- \([\text{Left}(M)]^{\zeta \eta} := \text{Left}([M]^{\zeta \eta})\)
- \([\text{Right}(M)]^{\zeta \eta} := \text{Right}([M]^{\zeta \eta})\)
- \([\text{In}(M)]^{\zeta \eta} := \text{In}([M]^{\zeta \eta})\)
- \([MN]^{\zeta \eta} := \begin{cases} f([N]^{\zeta \eta}) & \text{if } [M]^{\zeta \eta} = \text{Fun}(f) \\ \bot & \text{otherwise} \end{cases}\)
Value of Church-style terms

\[
\begin{align*}
\llbracket \lambda x : \rho.M \rrbracket^\zeta \eta &= \text{Fun}(f) \text{ where } f \in [D \to D] \text{ s.t. } f(a) := \llbracket M \rrbracket^\zeta \eta[x := \langle \rho \rangle \zeta(a)] \\
\llbracket \text{rec } x : \tau.M \rrbracket^\zeta \eta &= \text{LFP}(f) \text{ where } f \in [D \to D] \text{ s.t. } f(a) := \llbracket M \rrbracket^\zeta \eta[x := \langle \tau \rangle \zeta(a)] \\
\llbracket \text{case } M \text{ of } \{C_i(x_i) \to R_i\} \rrbracket^\zeta \eta &= \begin{cases} 
\llbracket R_i \rrbracket^\zeta \eta[x_i := \vec{a}] & \text{if } \llbracket M \rrbracket^\zeta \eta = C_i(\vec{a}) \\
\bot & \text{otherwise}
\end{cases}
\end{align*}
\]

- Semantics of types, \langle \rho \rangle \zeta, is defined by means of finitary projections (Amadio et al. [1986])
Value of recursive types

For every recursive type $\rho$ we define $\langle \rho \rangle : [[D \to D]^{\text{TVar}} \to [D \to D]]$ as follows.

$$
\begin{align*}
\langle 1 \rangle \zeta(a) & := \begin{cases} 
\star & \text{if } a = \star \\
\perp & \text{otherwise}
\end{cases} \\
\langle \alpha \rangle \zeta(a) & := \zeta(\alpha)(a) \\
\langle \rho + \sigma \rangle \zeta(a) & := \begin{cases} 
\text{Left}(\langle \rho \rangle \zeta(b)) & \text{if } a = \text{Left}(b) \\
\text{Right}(\langle \sigma \rangle \zeta(b)) & \text{if } a = \text{Right}(b) \\
\perp & \text{otherwise}
\end{cases} \\
\langle \rho \times \sigma \rangle \zeta(a) & := \begin{cases} 
\text{Pair}(\langle \rho \rangle \zeta(b_1), \langle \sigma \rangle \zeta(b_2)) & \text{if } a = \text{Pair}(b_1, b_2) \\
\perp & \text{otherwise}
\end{cases}
\end{align*}
$$
Semantics of Church-style LoR’s Set of Types
(Continued)

Value of recursive types

\[
\langle \rho \rightarrow \sigma \rangle \zeta (a) := \begin{cases} 
\text{Fun}(g) \text{ where } g : [D \rightarrow D] \text{ s.t. } g = \langle \sigma \rangle \zeta \circ f \circ \langle \rho \rangle \zeta & \text{if } a = \text{Fun}(f) \\
\bot & \text{otherwise}
\end{cases}
\]

\[
\langle \text{fix } \alpha.\rho \rangle \zeta := \text{LFP}(\lambda p.\lambda a.\text{case}_{\text{In}} a (\lambda b.\text{In}(\langle \rho \rangle \zeta [\alpha := p](b))))
\]

Set \([\rho] \zeta := (\langle \rho \rangle \zeta)(D)\)

where \(\text{case}_{c_1, \ldots, c_n} : D \rightarrow [D^{\text{arity}(C_i)} \rightarrow D] \rightarrow D\) s.t.

\[
\text{case}_{c_1, \ldots, c_n} a f_1 \ldots f_2 := \begin{cases} 
 f_i(b_i) & \text{if } a = C_i(b_i) \\
\bot & \text{otherwise}
\end{cases}
\]
Soundness For Church-style terms

Let $\eta \in \llbracket \Gamma \rrbracket \zeta$ mean $\Gamma(x_i) = \rho_i \land \eta(x_i) \in \llbracket \rho_i \rrbracket \zeta$

**Theorem**

If $\Gamma \vdash M : \rho$ and $\eta \in \llbracket \Gamma \rrbracket \zeta$, then $\llbracket M \rrbracket \hat{\zeta} \eta \in \llbracket \rho \rrbracket \zeta$
Soundness For Church-style terms

Let $\eta \in \llbracket \Gamma \rrbracket \varsigma$ mean $\Gamma(x_i) = \rho_i \land \eta(x_i) \in \llbracket \rho_i \rrbracket \varsigma$

**Theorem**

If $\Gamma \vdash M : \rho$ and $\eta \in \llbracket \Gamma \rrbracket \varsigma$, then $\llbracket M \rrbracket \hat{\varsigma} \eta \in \llbracket \rho \rrbracket \varsigma$

**Proof**

By induction on the structure of $\Gamma \vdash M : \rho$
Main Result

- Limited recursion for Curry-style languages
- Unrestricted recursion for Church-style ones

How are these two related for programs in both languages?

Let $M$ be a Church-style term, $M'$ the corresponding Curry-style term and $\rho$ a regular recursive type (i.e. $\text{fix}_\alpha \tau$ is only used if $\tau$ is positive in $\alpha$).

Theorem (Coincidence)

If $\Gamma \vdash M : \rho$ and $\eta \in J_{\Gamma K} \zeta$, then $J_M K \zeta \eta = \langle \rho \rangle \zeta (J_{M'} K \eta)$.
Main Result

- Limited recursion for Curry-style languages
- Unrestricted recursion for Church-style ones

How are these two related for programs in both languages?

Let $M$ be a Church-style term, $M^\rightarrow$ the corresponding Curry-style term and $\rho$ a regular recursive type (i.e. $\text{fix } \alpha.\tau$ is only used if $\tau$ is positive in $\alpha$)

Theorem (Coincidence)

If $\Gamma \vdash M : \rho$ and $\eta \in \llbracket \Gamma \rrbracket_\zeta$, then $\llbracket M \rrbracket_\zeta \eta = \langle \rho \rangle_\zeta (\llbracket M^\rightarrow \rrbracket \eta)$
Hybrid Logical Relation

Definition

\[ \sim_{R, \zeta}^1 := \{ (\bot, \bot), (\ast, \ast) \} \]

\[ \sim_{R, \zeta}^\alpha := R(\alpha) \]

\[ \sim_{R, \zeta}^{\rho_1 \times \rho_2} := \{ (\bot, \bot) \} \cup \{ (\text{Pair}(a_1, a_2), \text{Pair}(b_1, b_2)) \mid a_i \sim_{\rho_i}^{R, \zeta} b_i (i = 1, 2) \} \]

\[ \sim_{R, \zeta}^{\rho_1 + \rho_2} := \{ (\bot, \bot) \} \cup \{ (\text{Left}(a_1), \text{Left}(b_1)) \mid a_1 \sim_{\rho_1}^{R, \zeta} b_1 \} \]
\[ \quad \cup \{ (\text{Right}(a_2), \text{Right}(b_2)) \mid a_2 \sim_{\rho_2}^{R, \zeta} b_2 \} \]

\[ \sim_{R, \zeta}^{\rho \to \sigma} := \{ (\bot, \bot) \} \cup \{ (\text{Fun}(f), \text{Fun}(g)) \mid \forall a, b \in D (a \sim_{\rho}^{R, \zeta} b \Rightarrow f(a) \sim_{\sigma}^{R, \zeta} g(b)) \}
\]
\[ \wedge \langle \sigma \rangle \zeta \circ f \circ \langle \rho \rangle \zeta = \langle \sigma \rangle \zeta \circ g \circ \langle \rho \rangle \zeta \}

\[ \sim_{R, \zeta}^{\text{fix } \alpha . \rho} := \text{LFP}((\forall r \subseteq D \times D. \{ (\text{In}(a), \text{In}(b)) \mid a \sim_{\rho}^{R[\alpha := r], \zeta[\alpha := \text{LFP}(\lambda \rho \in [D \to D]. \langle \rho \rangle \zeta[\alpha := \rho])] b \})} \]
Proof Sketch

Assume $\text{FV}(M) \subseteq \text{dom}(\Gamma)$

Let $\eta \sim_{\Gamma}^{R,\zeta} \eta'$ denote the following: for all $x \in \text{dom}(\Gamma)$, if $\Gamma(x) = \sigma$, then $\eta(x) \sim_{\sigma}^{R,\zeta} \eta'(x)$

Lemma

If $\Gamma \vdash M : \rho$ and $\eta \sim_{\Gamma}^{R,\zeta} \eta'$, then $[M]_{\zeta}^{\epsilon} \eta \sim_{\rho}^{R,\zeta} [M^-] \eta'$
Proof Sketch

Assume $\text{FV}(M) \subseteq \text{dom}(\Gamma)$

Let $\eta \sim_{R, \zeta}^{\Gamma} \eta'$ denote the following: for all $x \in \text{dom}(\Gamma)$, if $\Gamma(x) = \sigma$, then $\eta(x) \sim_{\sigma, \zeta}^{R} \eta'(x)$

Lemma

If $\Gamma \vdash M : \rho$ and $\eta \sim_{\Gamma}^{R, \zeta} \eta'$, then $[M]^{\zeta} \eta \sim_{\rho, \zeta}^{R} [M^{-}] \eta'$

Proof

By induction on the structure of $\Gamma \vdash M : \rho$
We have given a domain-theoretic semantics for the Curry- and Church-style systems
We have given a domain-theoretic semantics for the Curry- and Church-style systems

We have proved the coincidence of these two systems
We have given a domain-theoretic semantics for the Curry- and Church-style systems

We have proved the coincidence of these two systems

This research is part of a project on Interactive Theorem Proving and Program Extraction

make (semi-)automated theorem proving a real possibility
Abramsky, S., and Jung, A.  
Domain theory.  

Amadio, R. M., Bruce, K. B., and Longo, G.  
The finitary projection model for second order lambda calculus and solutions to higher order domain equations.  

Schwichtenberg, H.  
Realizability interpretation of proofs in constructive analysis.  

Stoltenberg-Hansen, V., Lindström, I., and Griffor, E. R.  
Mathematical Theory of Domains.  
Thank You!