Ultrametric Semantics of Reactive Programs

Neelakantan R. Krishnaswami <neelk@microsoft.com>
Nick Benton <nick@microsoft.com>

LICS 2011
Functional Reactive Programming in a Nutshell

- Goal: Write interactive programs in a pure style
- Idea: Mutable state of type $X$ becomes stream of values $S(X)$ [Eliot & Hudak 1997]
Goal: Write interactive programs in a pure style

Idea: Mutable state of type $X$ becomes *stream* of values $S(X)$
    [Eliot & Hudak 1997]

Interactive program has type $S(In) \rightarrow S(Out)$
Trouble in Paradise

profit :: S(stockprice) → S(order)
profit prices =
  if today < tomorrow
  then cons(Buy, profit (tl prices))
  else cons(Sell, profit (tl prices))
where
  today = hd prices
  tomorrow = hd (tl prices)
Trouble in Paradise

profit :: S(stockprice) → S(order)
profit prices =
  if today < tomorrow
  then cons(Buy, profit (tl prices))
  else cons(Sell, profit (tl prices))
where
  today = hd prices
  tomorrow = hd (tl prices)
Causal Stream Functions

A function \( f : S(A) \rightarrow S(B) \) is causal, when for all \( n, as, as' \):

\[
[as]_n = [as']_n \implies [f \ as]_n = [fas']_n
\]
Causal Stream Functions

A function $f : S(A) \rightarrow S(B)$ is causal, when for all $n, as, as'$:

$$[as]_n = [as']_n \implies [f\ as]_n = [fas']_n$$

- First $n$ outputs of $f$ depend only on first $n$ inputs
Causal Stream Functions

A function $f : S(A) \rightarrow S(B)$ is causal, when for all $n, as, as'$:

$$\lfloor as \rfloor_n = \lfloor as' \rfloor_n \implies \lfloor f \ as \rfloor_n = \lfloor fas' \rfloor_n$$

- First $n$ outputs of $f$ depend only on first $n$ inputs
- tail not causal: element $n$ of tail $xs = element$ $n + 1$ of $xs$
A function $f : S(A) \rightarrow S(B)$ is causal, when for all $n, as, as'$:

$$[as]_n = [as']_n \implies [f \ as]_n = [fas']_n$$

- First $n$ outputs of $f$ depend only on first $n$ inputs
- Tail not causal: element $n$ of tail $xs = element n + 1 of xs$
- But what about higher-order?
The Category of Ultrametric Spaces

A pair \((X, d : X \times X \to [0, 1])\) is a complete 1-bounded ultrametric space:

- \(d(x, y) = 0\) iff \(x = y\)
- \(d(x, y) = d(y, x)\)
- \(d(x, z) \leq \max(d(x, y), d(y, z))\)
- All Cauchy sequences converge
The Category of Ultrametric Spaces

A pair \((X, d : X \times X \to [0, 1])\) is a complete 1-bounded ultrametric space:

- \(d(x, y) = 0\) iff \(x = y\)
- \(d(x, y) = d(y, x)\)
- \(d(x, z) \leq \max(d(x, y), d(y, z))\)
- All Cauchy sequences converge

A function \(f : A \to B\) is nonexpansive, when for all \(a\) and \(a'\)

\[
    d_B(f \ a, f \ a') \leq d_A(a, a')
\]

So \(f\) is non-distance-increasing
Streams as Ultrametric Spaces

Streams $S(X)$ can be equipped with an ultrametric

$$d(xs, xs') = 2^{-\min\{n \in \mathbb{N} \mid xs_n \neq xs'_n\}}$$
Streams $S(X)$ can be equipped with an ultrametric

$$d(xs, xs') = 2^{-\min\{n \in \mathbb{N} \mid xs_n \neq xs'_n}\}$$

Distance increases, as $xs$ and $xs'$ differ sooner:

- Differ at time 0 — distance 1
- Differ at time 1 — distance $\frac{1}{2}$
- Differ at time 2 — distance $\frac{1}{4}$
- Never differ — distance 0
Nonexpansive Functions and Causality

**Theorem**
The nonexpansive functions $S(X) \rightarrow S(Y)$ are exactly the causal functions.
Nonexpansive Functions and Causality

Theorem
The nonexpansive functions $S(X) \rightarrow S(Y)$ are exactly the causal functions

Idea as follows:
- Suppose $xs$ and $xs'$ first differ at position $n$
Nonexpansive Functions and Causality

Theorem
The nonexpansive functions $S(X) \to S(Y)$ are exactly the causal functions

Idea as follows:
- Suppose $xs$ and $xs'$ first differ at position $n$
- Then $d(xs, xs') = 2^{-n}$
Nonexpansive Functions and Causality

Theorem
The nonexpansive functions $S(X) \rightarrow S(Y)$ are exactly the causal functions

Idea as follows:
- Suppose $xs$ and $xs'$ first differ at position $n$
- Then $d(xs, xs') = 2^{-n}$
- So $d(f \ xs, f \ xs') \leq 2^{-n}$
Nonexpansive Functions and Causality

Theorem
The nonexpansive functions $S(X) \rightarrow S(Y)$ are exactly the causal functions

Idea as follows:
- Suppose $xs$ and $xs'$ first differ at position $n$
- Then $d(xs, xs') = 2^{-n}$
- So $d(f xs, f xs') \leq 2^{-n}$
- So at least the first $n$ positions of $f xs$ and $f xs'$ agree
What are the consequences of this abstract view?
What are the consequences of this abstract view?

- The category of complete 1-bounded ultrmetric spaces is *Cartesian closed*
The Payoff, Part 1

What are the consequences of this abstract view?

- The category of complete 1-bounded ultrmetric spaces is \textit{Cartesian closed}
- The lambda calculus can be interpreted in any CCC...
What are the consequences of this abstract view?

- The category of complete 1-bounded ultrmetric spaces is *Cartesian closed*
- The lambda calculus can be interpreted in any CCC…
- …so a good DSL for reactive programming is functional programming!
Banach’s Contraction Map Theorem

Every strictly contractive function $f : A \rightarrow A$ on a nonempty metric space $A$ has a unique fixed point.
Banach’s Contraction Map Theorem
Every strictly contractive function $f : A \rightarrow A$ on a nonempty metric space $A$ has a unique fixed point.

- “Strictly contractive” = “well-founded feedback”
The Payoff, Part 2

Banach’s Contraction Map Theorem
Every strictly contractive function $f : A \rightarrow A$ on a nonempty metric space $A$ has a unique fixed point.

- “Strictly contractive” = “well-founded feedback”
- So $\mu(\lambda xs. 0 :: \text{map succ } xs) = 0, 1, 2, 3, \ldots$
Banach’s Contraction Map Theorem

Every strictly contractive function $f : A \rightarrow A$ on a nonempty metric space $A$ has a unique fixed point.

- “Strictly contractive” = “well-founded feedback”
- So $\mu(\lambda xs. 0 :: \text{map succ } xs) = 0, 1, 2, 3, \ldots$
- Semantic interpretation of feedback...
- ...which ensures it is well-founded and deterministic
From Semantics to Type Theory

\[
A ::= P \mid A \to B \mid S(A) \mid \mathbf{\bullet} A \quad \text{Types}
\]
\[
e ::= x \mid \lambda x. e \mid e e' \mid \text{cons}(e, e') \mid \text{hd}(e) \mid \text{tl}(e) \mid \mathbf{\bullet} e \mid \text{await}(e) \mid \text{fix } x : A. e \quad \text{Terms}
\]
From Semantics to Type Theory

| A ::= | P | A → B | S(A) | • A | Types |
| e ::= | x | λx. e | e e' |
|       | cons(e, e') | hd(e) | tl(e) |
|       | •e | await(e) | fix x : A. e | Terms |

• A is the delay modality
From Semantics to Type Theory

\[
A ::= \ P \mid A \rightarrow B \mid S(A) \mid \bullet A \quad \text{Types}
\]

\[
e ::= \ x \mid \lambda x.\ e \mid e\ e' \\
\quad \mid \text{cons}(e, e') \mid \text{hd}(e) \mid \text{tl}(e) \\
\quad \mid \bullet e \mid \text{await}(e) \mid \text{fix} \ x : A. \ e \quad \text{Terms}
\]

- $\bullet A$ is the delay modality
  - $\bullet (A, d) = (A, d')$ where
  - $d'(a, a') = \frac{1}{2} \cdot d(a, a')$
From Semantics to Type Theory

$A ::= P \mid A \to B \mid S(A) \mid \bullet A$  \hspace{1em} \text{Types}

$e ::= x \mid \lambda x. e \mid e e'$

\hspace{1em} \mid \text{cons}(e, e') \mid \text{hd}(e) \mid \text{tl}(e)$

\hspace{1em} \mid \bullet e \mid \text{await}(e) \mid \text{fix } x : A. e$  \hspace{1em} \text{Terms}$

- $\bullet A$ is the delay modality
  - $\bullet(A, d) = (A, d')$ where
  - $d'(a, a') = \frac{1}{2} \cdot d(a, a')$

- $\bullet S(A)$ are “streams starting on the next time step”
From Semantics to Type Theory

\[
A ::= \ P \mid A \to B \mid S(A) \mid \bullet A \quad \text{Types}
\]

\[
e ::= x \mid \lambda x.\ e \mid e \ e' \\
\mid \ \text{cons}(e, \ e') \mid \text{hd}(e) \mid \text{tl}(e) \\
\mid \bullet e \mid \text{await}(e) \mid \text{fix } x : A.\ e \quad \text{Terms}
\]

- \bullet A is the delay modality
  - \bullet(A, d) = (A, d') where
  - d'(a, a') = \frac{1}{2} \cdot d(a, a')
- \bullet S(A) are “streams starting on the next time step”
- \epsilon : \bullet A \to B \simeq \bullet A \to \bullet B \quad \text{and} \quad \zeta : \bullet A \times B \simeq \bullet A \times \bullet B
From Semantics to Type Theory

\[
A ::= P \mid A \to B \mid S(A) \mid \bullet A \quad \text{Types}
\]

\[
e ::= x \mid \lambda x. \; e \mid e \; e' \mid \text{cons}(e, e') \mid \text{hd}(e) \mid \text{tl}(e) \mid \bullet e \mid \text{await}(e) \mid \text{fix } x : A. \; e \quad \text{Terms}
\]

- \(\bullet A\) is the *delay* modality
  - \(\bullet (A, d) = (A, d')\) where
    - \(d'(a, a') = \frac{1}{2} \cdot d(a, a')\)
- \(\bullet S(A)\) are “streams starting on the next time step”
- \(\epsilon : \bullet A \to B \simeq \bullet A \to \bullet B\) and \(\zeta : \bullet A \times B \simeq \bullet A \times \bullet B\)
- Banach’s theorem has type \((\bullet A \to A) \to A\)
The Typing Judgement

- Key judgement $\Gamma \vdash e :_j A$
The Typing Judgement

- Key judgement $\Gamma \vdash e : \cdot A$
- Read $e$ has type $A$ at time $i$
The Typing Judgement

- Key judgement $\Gamma \vdash e : j A$
- Read $e$ has type $A$ at time $i$
- Context annotated with times $\Gamma ::= \cdot \mid \Gamma, x :: i A$
Typing Rules

\[
\frac{i \leq j}{\Gamma, x : i A \vdash x : j A}
\]

\[
\frac{\Gamma, x : i A \vdash e : i B}{\Gamma \vdash \lambda x. e : i A \rightarrow B}
\]

\[
\frac{\Gamma \vdash e : i A}{\Gamma \vdash \text{fix } x : A. e : i A}
\]

\[
\frac{\Gamma \vdash e : i A \rightarrow B \quad \Gamma \vdash e' : i A}{\Gamma \vdash e \, e' : i B}
\]

\[
\frac{\Gamma \vdash e : i A \quad \Gamma \vdash e' : i_1 S(A)}{\Gamma \vdash \text{cons}(e, e') : i S(A)}
\]

\[
\frac{\Gamma \vdash e : i S(A)}{\Gamma \vdash \text{hd}(e) : i A}
\]

\[
\frac{\Gamma \vdash e : i S(A)}{\Gamma \vdash \text{tl}(e) : i_1 S(A)}
\]

\[
\frac{\Gamma \vdash e : i_1 A}{\Gamma \vdash \bullet e : i \bullet A}
\]

\[
\frac{\Gamma \vdash e : i \bullet A}{\Gamma \vdash \text{await}(e) : i_1 A}
\]
Interpreting the Syntax

\[ \begin{align*}
\Gamma & \vdash e : i \ A \\
\Gamma & \vdash x : i \ A \\
\Gamma & \vdash \lambda x. \ e : i \ A \rightarrow B \\
\Gamma & \vdash \text{cons}(e, e') : i \ S(A) \\
\Gamma & \vdash \text{hd}(e) : i \ A \\
\Gamma & \vdash \text{tl}(e) : i_+1 \ S(A) \\
\Gamma & \vdash \bullet e : i \ \bullet A \\
\Gamma & \vdash \text{await}(e) : i_+1 \ A \\
\Gamma & \vdash \text{fix} \ x : A. \ e : i_+1 \ A
\end{align*} \]

\[ \begin{align*}
\in & \begin{align*}
\Gamma & \rightarrow \bullet^i(\llbracket A \rrbracket)
\end{align*} \\
= & \begin{align*}
\delta^{i-j} \circ \pi_x \text{ when } x : j \ A \in \Gamma
\end{align*} \\
& \begin{align*}
\epsilon^i \circ \lambda(\llbracket \Gamma, x : i \ A \vdash e : i \ B \rrbracket)
\end{align*} \\
& \begin{align*}
\bullet^i(\text{cons}) \circ \zeta^i \circ \langle \llbracket e \rrbracket, \llbracket e' \rrbracket \rangle
\end{align*} \\
& \begin{align*}
\bullet^i(\text{head}) \circ \llbracket \Gamma \vdash e : i \ S(A) \rrbracket
\end{align*} \\
& \begin{align*}
\bullet^i(\text{tail}) \circ \llbracket \Gamma \vdash e : i \ S(A) \rrbracket
\end{align*} \\
= & \begin{align*}
\llbracket \Gamma \vdash e : i_+1 \ A \rrbracket
\end{align*} \\
= & \begin{align*}
\llbracket \Gamma \vdash e : i \ \bullet A \rrbracket
\end{align*} \\
= & \begin{align*}
\lambda \gamma. \mu(\lambda v. \llbracket \Gamma, x : i_+1 \ A \vdash e : i \ A \rrbracket (\gamma, v))
\end{align*}
\]
Example 1: fibs

fibs : $S(\mathbb{N})$          // $t = 0$
fibs = fix $xs:S(\mathbb{N})$.   // $t = 1$
      let $ys = \text{tl}(xs)$ in // $t = 2$
      cons(1, cons(1, map (+) (zip $xs$ $ys$))))
Example 1: fibs

\[
fibs : S(\mathbb{N}) \quad // t = 0
fibs = \text{fix } x\text{s}:S(\mathbb{N}). \quad // t = 1
\quad \text{let } y\text{s} = \text{tl}(x\text{s}) \text{ in } \quad // t = 2
\quad \text{cons}(1, \text{cons}(1, \text{map } (+) (\text{zip } x\text{s } y\text{s})))
\]

- Looks like a lazy functional program
Example 1: fibs

fibs : $S(\mathbb{N})$                // t = 0
fibs = fix xs:$S(\mathbb{N})$.        // t = 1
     let ys = tl(xs) in               // t = 2
     cons(1, cons(1, map (+) (zip xs ys)))

- Looks like a lazy functional program
- Note $xs$ and $ys$ are at different times
Example 2: map

\[
\text{map} : (A \rightarrow B) \rightarrow S(A) \rightarrow S(B)
\]
\[
\text{map} = \lambda f : A \rightarrow B.
\]
\[
\qquad \text{fix } r : S(A) \rightarrow S(B).
\]
\[
\qquad \lambda xs : S(A). \ \text{cons}(f (\text{hd } xs), r (\text{tl } xs))
\]
Example 2: \texttt{map}

\[
\begin{align*}
\text{map} : & (A \rightarrow B) \rightarrow S(A) \rightarrow S(B) \\
\text{map} & = \lambda f : A \rightarrow B. \\
& \quad \text{fix } r : S(A) \rightarrow S(B). \\
& \quad \quad \lambda x:\!S(A). \ cons(f \ (\text{hd } x), \ r \ (\text{tl } x))
\end{align*}
\]

\[\]

\begin{itemize}
  \item Looks like a functional program
\end{itemize}
Example 2: map

\[
\text{map} : (A \rightarrow B) \rightarrow S(A) \rightarrow S(B)
\]

\[
\text{map} = \lambda f : A \rightarrow B. \\
\text{fix } r : S(A) \rightarrow S(B). \\
\lambda xs : S(A). \text{cons}(f (\text{hd } xs), r (\text{tl } xs))
\]

- Looks like a functional program
- Note higher-type recursion
Non-Example: illfounded

\[
\text{illfounded} \neq S(\mathbb{N}) \\
\text{illfounded} = \text{fix } xs : S(\mathbb{N}). \; xs
\]
Non-Example: illfounded

\[ \text{illfounded} \not\in S(\mathbb{N}) \]
\[ \text{illfounded} = \text{fix } xs:S(\mathbb{N}). \; xs \]

- Ill-founded feedback
Non-Example: illfounded

illfounded / $S(\mathbb{N})$
illfounded = fix xs:$S(\mathbb{N})$. xs

- Ill-founded feedback
- “Time-checking” error: xs is at time 1, not 0
Non-Example: illfounded

\[
\text{illfounded} \neq S(\mathbb{N})
\]
\[
\text{illfounded} = \text{fix } xs:S(\mathbb{N}). \; xs
\]

- Ill-founded feedback
- “Time-checking” error: \(xs\) is at time 1, not 0
- Typing rules block unguarded recursion
How Can We Implement This

- Imperative implementation based on dataflow propagation [TLDI 2010]
How Can We Implement This

- Imperative implementation based on dataflow propagation [TLDI 2010]
- Idea similar to spreadsheets, self-adjusting computation [Acar et al.], subject-observer
How Can We Implement This

- Imperative implementation based on dataflow propagation [TLDI 2010]
- Idea similar to spreadsheets, self-adjusting computation [Acar et al.], subject-observer
- Correctness proof via logical relation between imperative code and ultrametric semantics
How Can We Implement This

- Imperative implementation based on dataflow propagation [TLDI 2010]
- Idea similar to spreadsheets, self-adjusting computation [Acar et al.], subject-observer
- Correctness proof via logical relation between imperative code and ultrametric semantics
- Proof uses ideas similar to Dreyer and Hur [POPL 2011]
Demo

Demo!
Conclusions

- Ultrametric semantics give a simple denotational semantics to reactive programs
- The lambda calculus is the correct DSL for this domain
- We can implement this!
- Look at our upcoming LICS 2011 paper
- Look at our upcoming ICFP 2011 paper
  <http://research.microsoft.com/~nick/guisemantics.pdf>