Guarded recursive types

A programming-language perspective

Adrien Guatto[1] & Daniel Gratzer[2]

[1]: Université Paris Cité
[2]: Aarhus University

ÉPIT 2025

Introduction

Daniel

Adrien

Guarded recursion in types and terms

- An alternative to primitive (co)recursion and general recursion [Nakano, 2000]
- Applications to programming with infinite data and to logic and verification

Introduction

Daniel

Adrien

Guarded recursion in types and terms

- An alternative to primitive (co)recursion and general recursion [Nakano, 2000]
- Applications to programming with infinite data and to logic and verification

Introduction

Daniel

Adrien

Guarded recursion in types and terms

- An alternative to primitive (co)recursion and general recursion [Nakano, 2000]
- Applications to programming with infinite data and to logic and verification

Infinite streams in functional programming

Streams as first-class interactions [Kahn, 1974]

- Use streams to represent and manipulate entire, infinite histories of events happening over the unending execution of a program.
- Transfer the benefits of functional programming, such as equational reasoning, to new application domains beyond symbolic computation.

Infinite streams in functional programming

Streams as first-class interactions [Kahn, 1974]

- Use streams to represent and manipulate entire, infinite histories of events happening over the unending execution of a program.
- Transfer the benefits of functional programming, such as equational reasoning, to new application domains beyond symbolic computation.

This idea has (re)appeared and been put into use several times:

- for interactive programs, e.g., GUIs, servers, and games
  - *functional reactive programming* [Elliott and Hudak, 1997] in Haskell
- for reactive programs, e.g., real-time control programs
  - dedicated *synchronous languages* such as Lustre [Caspi et al., 1987]

All of these expose fixpoint operators rather than primitive (co)recursion.

Infinite streams in functional programming

Streams as first-class interactions [Kahn, 1974]

- Use streams to represent and manipulate entire, infinite histories of events happening over the unending execution of a program.
- Transfer the benefits of functional programming, such as equational reasoning, to new application domains beyond symbolic computation.

This idea has (re)appeared and been put into use several times:

- for interactive programs, e.g., GUIs, servers, and games
  - *functional reactive programming* [Elliott and Hudak, 1997] in Haskell
- for reactive programs, e.g., real-time control programs
  - dedicated *synchronous languages* such as Lustre [Caspi et al., 1987]

All of these expose fixpoint operators rather than primitive (co)recursion.

Important questions in safety-critical settings

- *Productivity*: reject unsound cyclic definitions
- *Real-time implementations*: bounded in time and space

Infinite streams in functional programming

Streams as first-class interactions [Kahn, 1974]

- Use streams to represent and manipulate entire, infinite histories of events happening over the unending execution of a program.
- Transfer the benefits of functional programming, such as equational reasoning, to new application domains beyond symbolic computation.

This idea has (re)appeared and been put into use several times:

- for interactive programs, e.g., GUIs, servers, and games
  - *functional reactive programming* [Elliott and Hudak, 1997] in Haskell
- for reactive programs, e.g., real-time control programs
  - dedicated *synchronous languages* such as Lustre [Caspi et al., 1987]

All of these expose fixpoint operators rather than primitive (co)recursion.

Important questions in safety-critical settings

- *Productivity*: reject unsound cyclic definitions (focus of this lecture)
- *Real-time implementations*: bounded in time and space

Infinite streams in functional programming

Streams as first-class interactions [Kahn, 1974]

- Use streams to represent and manipulate entire, infinite histories of events happening over the unending execution of a program.
- Transfer the benefits of functional programming, such as equational reasoning, to new application domains beyond symbolic computation.

This idea has (re)appeared and been put into use several times:

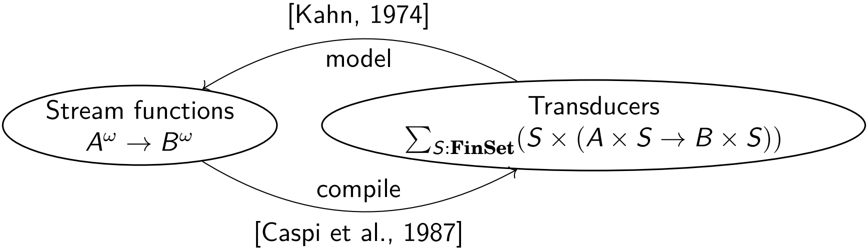
- for interactive programs, e.g., GUIs, servers, and games
  - *functional reactive programming* [Elliott and Hudak, 1997] in Haskell
- for reactive programs, e.g., real-time control programs
  - dedicated *synchronous languages* such as Lustre [Caspi et al., 1987]

All of these expose fixpoint operators rather than primitive (co)recursion.

Important questions in safety-critical settings

- *Productivity*: reject unsound cyclic definitions
- *Real-time implementations*: bounded in time and space

From synchronous languages to synchrony



- Design and study of functional languages compiling to state machines
- Programs have to satisfy specific properties, such as synchrony
- Strongly related to guarded recursion: guarded calculi are all (?) synchronous

Definition (Synchrony, informal and intuitive)

A stream function f is *synchronous* when $\mathrm{xs}|_n = \mathrm{ys}|_n \implies f(\mathrm{xs})|_n = f(\mathrm{ys})|_n$.

This lecture

- A language-oriented reconstruction of guarded recursion starting from

$$\text{types} \leftrightarrow \text{partial orders}$$
$$\text{nonstrict programs} \leftrightarrow \text{monotone maps}$$

- A model of a *guarded* variant of synchronous functional programming

$$\text{types} \leftrightarrow \text{trees}$$
$$\text{synchronous functions} \leftrightarrow \text{height-preserving tree maps}$$

- A syntax suggested by the model.

Inspirations

Birkedal et al. [2012], Pouzet [2002], G. [2016, 2018], Clouston [2018], others.

Caveat

This is a specific view of guarded recursion, coming from programming languages.

Outline

1. Introduction
2. A nonstrict stream language
   - Syntax and execution
   - Modeling nonstrict streams
3. Synchrony in the topos of trees
   - From orders to presheaves
   - Back to syntax
4. Perspectives
   - Limitations
   - Conclusion

Outline

A nonstrict stream language

Syntax of $\mathscr{L}$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \mathsf{fun}(x.t) : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{app}(t, u) : B}$$

$$\frac{(\Gamma \vdash t_i : A_i)_{i \in \{1,2\}}}{\Gamma \vdash \langle t_1, t_2 \rangle : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathsf{proj}_{i \in \{1,2\}}(t) : A_i} \qquad \frac{}{\Gamma \vdash \mathsf{tt}, \mathsf{ff} : \mathsf{Bool}}$$

$$\frac{\Gamma \vdash t : \mathsf{Bool} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash s : A}{\Gamma \vdash \mathsf{if}(t, u, s) : A} \qquad \frac{\Gamma, x : A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A}$$

$$\frac{\Gamma \vdash t : \mathsf{Str}\, A}{\Gamma \vdash \mathsf{head}(t) : A} \qquad \frac{\Gamma \vdash t : \mathsf{Str}\, A}{\Gamma \vdash \mathsf{tail}(t) : \mathsf{Str}\, A} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : \mathsf{Str}\, A}{\Gamma \vdash t :: u : \mathsf{Str}\, A}$$

A nonstrict stream language

Reduction for $\mathscr{L}$

$$V ::= \mathsf{fun}(x.t) \mid \langle t_1, t_2 \rangle \mid \mathsf{tt} \mid \mathsf{ff} \mid V :: t$$
$$E ::= \square \mid \mathsf{app}(E, u) \mid \mathsf{proj}_i(E) \mid \mathsf{if}(E, u, s) \mid E :: t \mid \mathsf{head}(E) \mid \mathsf{tail}(E)$$

$$\begin{aligned}
\mathsf{app}(\mathsf{fun}(x.t), u) &\rightsquigarrow t[u/x] \\
\mathsf{proj}_i(\langle t_1, t_2 \rangle) &\rightsquigarrow t_i \\
\mathsf{if}(\mathsf{tt}, u, s) &\rightsquigarrow u \\
\mathsf{if}(\mathsf{ff}, u, s) &\rightsquigarrow s \\
\mathsf{rec}(x.t) &\rightsquigarrow t[\mathsf{rec}(x.t)/x] \\
\mathsf{head}(V :: t) &\rightsquigarrow V \\
\mathsf{tail}(V :: t) &\rightsquigarrow t
\end{aligned}$$

$$\frac{u \rightsquigarrow u'}{E\{u\} \to E\{u'\}}$$

Summary

A λ-calculus with call-by-name semantics, except for streams which are left-strict.

A nonstrict stream language

Basic metatheory

Lemma (Determinism)

If $t \to t_1$ *and* $t \to t_2$ *then* $t_1 = t_2$.

Lemma (Subject reduction)

If $\Gamma \vdash t : A$ *and* $t \to t'$ *then* $\Gamma \vdash t' : A$.

Write $t \uparrow$ when there exists $(t_i)_{i \in \omega}$ with $t_i \to t_{i+1}$ and $t_0 = t$.

Lemma (Type safety)

If $\Gamma \vdash t : A$ *then either* $t \uparrow$ *or* $t \to^* V \not\to$.

A nonstrict stream language

Productivity

$$tail^0(t) \coloneqq t \qquad\qquad tail^{m+1}(t) \coloneqq \mathsf{tail}(tail^m(t))$$

Definition

A term $t : \mathsf{Str}\, A$ is *productive up to $n \leq \omega$* when $tail^m(t)$ converges to a value for all $m < n$. It is *productive* when it is productive up to ω.

The terms ffs and tts below are productive.

$$\begin{aligned}
\mathit{ffs} &\coloneqq \mathsf{rec}(\mathsf{xs}.\mathsf{ff} :: \mathsf{xs}) && : \mathsf{Str\ Bool} \\
\mathit{notb} &\coloneqq \mathsf{fun}(\mathsf{x}.\mathsf{if}(\mathsf{x}, \mathsf{ff}, \mathsf{tt})) && : \mathsf{Bool} \to \mathsf{Bool} \\
\mathit{nots} &\coloneqq \mathsf{rec}(\mathsf{F}.\mathsf{fun}(\mathsf{xs}.\mathsf{app}(\mathit{notb}, \mathsf{head}(\mathsf{xs})) :: \mathsf{app}(\mathsf{F}, \mathsf{tail}(\mathsf{xs})))) && : \mathsf{Str\ Bool} \to \mathsf{Str\ Bool} \\
\mathit{tts} &\coloneqq \mathsf{app}(\mathit{nots}, \mathit{ffs}) && : \mathsf{Str\ Bool}
\end{aligned}$$

A nonstrict stream language

Productivity and time

Here are two non-productive terms, not even productive up to 1.

$$\begin{aligned} \mathit{loop} &\coloneqq \mathsf{rec}(\mathsf{xs}.\mathsf{xs}) && : \mathsf{Str}\,\mathsf{Bool} \\ \mathit{weird} &\coloneqq \mathsf{rec}(\mathsf{xs}.\mathsf{head}(\mathsf{tail}(\mathsf{xs})) :: (\mathsf{tt} :: \mathsf{xs})) && : \mathsf{Str}\,\mathsf{Bool} \end{aligned}$$

The case of weird is the most interesting one.

$$\begin{aligned} & \mathit{tail}^0(\mathit{weird}) \\ \to\ & \mathit{weird} \\ \to\ & \mathsf{head}(\mathsf{tail}(\mathit{weird})) :: (\mathsf{tt} :: \mathit{weird}) \\ \to\ & \mathsf{head}(\mathsf{tail}(\mathsf{head}(\mathsf{tail}(\mathit{weird})) :: (\mathsf{tt} :: \mathit{weird}))) :: (\mathsf{tt} :: \mathit{weird}) \\ \to\ & \ldots \end{aligned}$$

The reduction of streams reflects the temporal intuition of Kahn [1974].

A nonstrict stream language

Synchrony

Definition (Synchrony, formal)

A term $t : \mathsf{Str}\,A \to \mathsf{Str}\,A$ is *synchronous* when, for all $u : \mathsf{Str}\,A$ and $n \leq \omega$, u productive up to n implies $\mathsf{app}(t, u)$ productive up to n.

The term *nots* is synchronous, *ands* is not, *stut* imprecisely so.

$$
\begin{aligned}
\mathit{nots} &\coloneqq \mathsf{rec}(\mathsf{F}.\mathsf{fun}(\mathsf{xs}.\mathsf{app}(\mathit{notb}, \mathsf{head}(\mathsf{xs})) :: \mathsf{app}(\mathsf{F}, \mathsf{tail}(\mathsf{xs})))) && : \mathsf{Str\,Bool} \to \mathsf{Str\,Bool} \\
\mathit{andb} &\coloneqq \mathsf{fun}(\mathsf{x}.\mathsf{fun}(\mathsf{y}.\mathsf{if}(\mathsf{x}, \mathsf{if}(\mathsf{y}, \mathsf{tt}, \mathsf{ff}), \mathsf{ff}))) && : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} \\
\mathit{and1} &\coloneqq \mathsf{fun}(\mathsf{xs}.\mathsf{app}(\mathsf{app}(\mathit{andb}, \mathsf{head}(\mathsf{xs})), \mathsf{head}(\mathsf{tail}(\mathsf{xs})))) && : \mathsf{Str\,Bool} \to \mathsf{Bool} \\
\mathit{ands} &\coloneqq \mathsf{rec}(\mathsf{F}.\mathsf{fun}(\mathsf{xs}.\mathsf{app}(\mathit{and1}, \mathsf{xs}) :: \mathsf{app}(\mathsf{F}, \mathsf{tail}(\mathsf{tail}(\mathsf{xs}))))) && : \mathsf{Str\,Bool} \to \mathsf{Str\,Bool} \\
\mathit{stut} &\coloneqq \mathsf{rec}(\mathsf{F}.\mathsf{fun}(\mathsf{xs}.\mathsf{head}(\mathsf{xs}) :: \mathsf{head}(\mathsf{xs}) :: \mathsf{app}(\mathsf{F}, \mathsf{tail}(\mathsf{xs})))) && : \mathsf{Str\,Bool} \to \mathsf{Str\,Bool}
\end{aligned}
$$

Remark

Synchrony is a stronger condition than *totality*: to be total at type $\mathsf{Str\,Bool} \to \mathsf{Str\,Bool}$ a term is only required to preserve productivity at ω.

A nonstrict stream language

Approximation and equivalence

Let $\Gamma \vdash t, u : A$.

Definition (Approximation)

We say that t *approximates* u, denoted $\Gamma \vdash t \sqsubseteq_{\mathsf{obs}} u : A$, when

$$\forall (\square : (\Gamma \vdash A) \vdash K : (\ \vdash \mathsf{Bool})), K\{t\} \to^* \mathsf{tt} \Rightarrow K\{u\} \to^* \mathsf{tt}.$$

Definition (Equivalence)

We say that t is *equivalent* to u, denoted $\Gamma \vdash t \equiv_{\mathsf{obs}} u : A$, when

$$\Gamma \vdash t \sqsubseteq_{\mathsf{obs}} u : A \text{ and } \Gamma \vdash t \sqsubseteq_{\mathsf{obs}} u : A.$$

Difficulties

The unwieldy nature of these definitions can motivate the study of models.

Outline

A model for $\mathscr{L}$

Setting

A model of $\mathscr{L}$ is a category $\mathcal{C}$ together with

- for each type A or context Γ, an object $[\![A]\!]$ or $[\![\Gamma]\!]$ of $\mathcal{C}$
- for each term $\Gamma \vdash t : A$, a morphism $[\![t]\!] : [\![\Gamma]\!] \to [\![A]\!]$ of $\mathcal{C}$

In addition, $[\![-]\!]$ should be functorial, i.e., commute with substitution.

Official goals: soundness and adequacy

For all $\Gamma \vdash t, u : A$, we expect the model to verify

- *soundness*: if $t \to u$ then $[\![t]\!] = [\![u]\!]$, and
- *adequacy*: if $[\![t]\!] = [\![u]\!]$ then $\Gamma \vdash t \equiv_{\mathsf{obs}} u : A$.

Actual goal: insight

We are looking for an analysis of the language grounded in the model.

A model for $\mathscr{L}$

Requirements

The model must have enough structure to interpret $\mathscr{L}$, mostly:

1. function types with currying and evaluation, i.e., cartesian closure,
2. a fixpoint operator at each type to interpret recursion,
3. an interpretation of recursive types to model streams.

Those classic requirements lead us to various kinds of partial orders.

I will omit much of the details and focus on building intuitions at this stage.

A model for $\mathscr{L}$

The categories $\mathbf{CPO}$ and $\mathbf{PCPO}$

Definition

- A poset P is *complete* when all suprema of directed sets exist.
- It is *pointed* when it has a least element, denoted $\perp_P$ or $\perp$.

Definition

Let P, Q be complete posets. Then $f : P \to Q$ is *Scott-continuous* when:

$$\bigvee f(D) = f\left(\bigvee D\right) \text{ for all } D \subseteq P \text{ directed.}$$

In addition, if P and Q are furthermore pointed, f is *strict* when $f(\perp) = \perp$.

- Complete posets and Scott-continuous maps form a category $\mathbf{CPO}$.
- Pointed complete posets and strict Scott-cont. maps form a category $\mathbf{PCPO}$.

A model for $\mathscr{L}$

Type formers in $\mathbf{PCPO}$

The category $\mathbf{PCPO}$ is closed under various type formers, including:

- cartesian products $P \times Q$, ordered componentwise;
- smashed products $P \otimes Q$, obtained by identifying $\bot_P$ and $\bot_Q$;
- strict function types $P \to_{\mathrm{s}} Q$, ordered pointwise;
- lifting $\uparrow P$, adding a new least element to P;
- unit I, the one-element pcpo, neutral for both $\otimes$ and $\times$;
- etc.

Remark

The object I is both terminal and initial in $\mathbf{PCPO}$. I will write $\iota_P : I \to P$ and $\pi_P : A \to P$, or simply ι and π, for the corresponding unique maps.

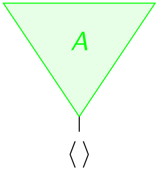
A model for $\mathscr{L}$

Lifting

Given a cpo A, we define a pcpo $\uparrow A$ as follows.

$$\boldsymbol{El}(\uparrow A) = \{\langle x \rangle \mid x \in \boldsymbol{El}(A)\} \cup \{\langle\,\rangle\} \qquad \frac{}{\langle\,\rangle \leq_{\uparrow A} \alpha} \qquad \frac{x \leq_A x'}{\langle x \rangle \leq_{\uparrow A} \langle x' \rangle}$$

Visually:



Remark for the categorically-minded

- The endofunctor $\uparrow$ of $\mathbf{CPO}$ can be given the structure of a monad $(\uparrow, \eta, \mu)$.
- The category $\mathbf{PCPO}$ is (equivalent to) the Eilenberg-Moore category $\mathbf{CPO}_{\uparrow}$.

A model for $\mathscr{L}$

Cartesian and smash product

Given two pcpos P and Q, define their cartesian products $P \times Q$ as for posets.

$$\boldsymbol{El}(P \times Q) = \boldsymbol{El}(P) \times \boldsymbol{El}(Q) \qquad \frac{x \leq_P x' \qquad y \leq_Q y'}{(x, y) \leq_{P \times Q} (x', y')}$$

The *smash product* of pcpos P and Q, is the pcpo $P \otimes Q \coloneqq {\uparrow}({\downarrow} P \times {\downarrow} Q)$.

- Here ${\downarrow} X$ is the sub-cpo of X formed of non-$\perp$ elements.

A model for $\mathscr{L}$

Recursion in $\mathbf{PCPO}$

Theorem (Kleene, Scott)

Every map $f : \uparrow A \to_{\mathrm{s}} A$ of $\mathbf{PCPO}$ has a least "fixpoint" given by

$$\mathit{fix}(f) = \bigsqcup \mathit{iter} \text{ where } \mathit{iter} : \omega \to A = n \mapsto \begin{cases} \bot & \text{if } n = 0 \\ f(\langle \mathit{iter}(n-1) \rangle) & \text{otherwise.} \end{cases}$$

By "fixpoint" we mean that it satisfies $f(\langle \mathit{fix}(f) \rangle) = \mathit{fix}(f)$.

Theorem (Scott, Adámek...)

Every "continuous" functor $F : \mathbf{PCPO} \to \mathbf{PCPO}$ has an initial algebra

$$\mathit{FIX}(F) = \varprojlim \mathit{ITER}^{+}$$

where $\mathit{ITER}^{+} : \omega \to \mathbf{PCPO}$ is the diagram below.

$$I \xrightarrow{\iota} F(I) \xrightarrow{F(\iota)} F^2(I) \xrightarrow{F^2(\iota)} F^3(I) \longrightarrow \dots$$

A model for $\mathscr{L}$

Constructing boolean streams in $\mathbf{PCPO}$

The object $[\![\mathsf{Str\,Bool}]\!]$ can be constructed as the initial algebra of

$$\begin{aligned} F : \mathbf{PCPO} &\to \mathbf{PCPO} \\ F(A) &= [\![\mathsf{Bool}]\!] \otimes {\uparrow} A \\ &= {\uparrow}\mathbb{B} \otimes {\uparrow} A \\ &= {\uparrow}(\mathbb{B} \times A). \end{aligned}$$

Iterating this functor gives rise to the diagram below, up to $A \times I \cong A$.

$$I \xrightarrow{\iota} {\uparrow}\mathbb{B} \xrightarrow{F(\iota)} {\uparrow}(\mathbb{B} \times {\uparrow}\mathbb{B}) \xrightarrow{F^2(\iota)} {\uparrow}(\mathbb{B} \times {\uparrow}(\mathbb{B} \times {\uparrow}\mathbb{B})) \longrightarrow \dots$$

Thus, $F^n(I)$ consists in words of length at most n ordered by prefix, connected by what ought to be thought of as inclusion maps.

Difficulty

This colimit in $\mathbf{PCPO}$ is not so easy to present explicitly.

A model for $\mathscr{L}$

An alternative construction

For general reasons, it is equivalent to consider the diagram $\boldsymbol{ITER}^{-}$ below

$$I \xleftarrow{\pi} \uparrow\mathbb{B} \xleftarrow{F(\pi)} \uparrow(\mathbb{B} \times \uparrow\mathbb{B}) \xleftarrow{F^2(\pi)} \uparrow(\mathbb{B} \times \uparrow(\mathbb{B} \times \uparrow\mathbb{B})) \longleftarrow \dots$$

and compute its limit, which is easier to describe explicitly.

$$\boldsymbol{El}(\llbracket \mathsf{Str\,Bool} \rrbracket) = \left\{ \prod_{n<\omega} F^n(I) \;\middle|\; \forall n < \omega, F^n(p)(x_{n+1}) = x_n \right\}.$$

The coherence requirement force the sequences to be strictly-increasing up to the point at which they become constant (if ever). This is isomorphic to

$$\llbracket \mathsf{Str\,Bool} \rrbracket := (\mathbb{B}^* \cup \mathbb{B}^\omega, \sqsubseteq) \text{ where } u \sqsubseteq v \text{ iff } u \text{ is a prefix of } v.$$

Remark

Divergence arises from the fact that $F^n(I)$ contains words of length $\leq n$.

A model for $\mathscr{L}$

Time and streams

So, streams are "recursive left-strict pairs," à la Kahn [1974].

$$[\![\mathsf{Str}\,A]\!] \cong [\![A]\!] \otimes {\uparrow}[\![\mathsf{Str}\,A]\!]$$

But the temporal intuition breaks down quickly, e.g., $[\![\mathsf{Str}\,\mathsf{Str}\,\mathsf{Bool}]\!]$ contains

$$((b_0^0, (b_1^0, (b_2^0, \bot))), ((b_0^1, \bot), ((b_0^2, (b_1^2, \bot)), \bot)))$$

where clearly the "degrees of productivity" are almost unrelated.

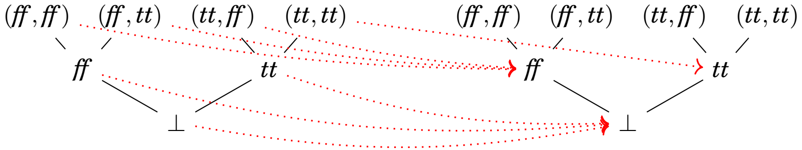
Observation

Synchrony would require a much "stricter" notion of cartesian product.

A model for $\mathscr{L}$

Time and continuous maps

As expected, most continuous maps are not synchronous, e.g., $\color{red}{ands}$.



This is by design since $\mathbf{PCPO}$ models general recursion.

A model for $\mathscr{L}$

Putting it all together

$$
\begin{aligned}
[\![\_]\!] &: \mathscr{L} \to \mathbf{PCPO} \\
[\![\mathsf{Bool}]\!] &= \uparrow \mathbb{B} \\
[\![A \times B]\!] &= [\![A]\!] \times [\![B]\!] \\
[\![A \to B]\!] &= \uparrow [\![A]\!] \to_{\mathrm{s}} [\![B]\!] \\
[\![\mathsf{Str}\ A]\!] &= [\![A]\!] \otimes \uparrow [\![\mathsf{Str}\ A]\!]
\end{aligned}
$$

The interpretation map

$$[\![\_]\!] : \mathscr{L}(\Gamma, A) \to \mathbf{PCPO}(\uparrow [\![\Gamma]\!], [\![A]\!])$$

interprets $\mathscr{L}$ into the Kleisli category $\mathbf{PCPO}^{\uparrow}$ of the lift comonad on $\mathbf{PCPO}$.

Observation

The syntax does not have to mention $\uparrow$, thanks to $\mathsf{force}_A : \uparrow A \to A$ in particular.

Outline

Summing up the inadequacies of $\mathbf{PCPO}$ from our perspective:

1. Scott-continuous stream functions are obviously not synchronous (nor total),
2. the definition of streams is not "right" one, beyond scalars.

These problems stem from the interpretations of $\to$ and $\otimes/\times$, respectively.

A possible solution

Refine the base model with a logical relation [G., 2016].

The rest of this lecture

Describe a model whose objects have an *intrinsic* temporal character.

Towards the topos of trees

Streams without limits

Let us go back to streams computed as the limit of the diagram below.

$$I \xleftarrow{\pi} \uparrow\mathbb{B} \xleftarrow{F(\pi)} \uparrow(\mathbb{B} \times \uparrow\mathbb{B}) \xleftarrow{F^2(\pi)} \uparrow(\mathbb{B} \times \uparrow(\mathbb{B} \times \uparrow\mathbb{B})) \longleftarrow \dots$$

We remove words of length $< n$ at stage n. The ordering becomes useless.

$$1 \xleftarrow{!} \mathbb{B} \xleftarrow{\pi_1} \mathbb{B} \times \mathbb{B} \xleftarrow{\pi_1} (\mathbb{B} \times \mathbb{B}) \times \mathbb{B} \longleftarrow \dots$$

The limit of this diagram in $\mathbf{Set}$ is $\mathbb{B}^\omega$, losing all temporal information.

Towards the topos of trees

Streams without limits

Let us go back to streams computed as the limit of the diagram below.

$$I \xleftarrow{\pi} \uparrow\mathbb{B} \xleftarrow{F(\pi)} \uparrow(\mathbb{B} \times \uparrow\mathbb{B}) \xleftarrow{F^2(\pi)} \uparrow(\mathbb{B} \times \uparrow(\mathbb{B} \times \uparrow\mathbb{B})) \longleftarrow \dots$$

We remove words of length $< n$ at stage n. The ordering becomes useless.

$$1 \xleftarrow{!} \mathbb{B} \xleftarrow{\pi_1} \mathbb{B} \times \mathbb{B} \xleftarrow{\pi_1} (\mathbb{B} \times \mathbb{B}) \times \mathbb{B} \longleftarrow \dots$$

The limit of this diagram in $\mathbf{Set}$ is $\mathbb{B}^\omega$, losing all temporal information.

Key idea

Instead of computing the limit, keep the entire diagram.

Towards the topos of trees

Dropping the now useless initial stage, we have a diagram of sets

$$\mathbb{B} \xleftarrow{\pi_1} \mathbb{B}^2 \xleftarrow{\pi_1} \mathbb{B}^3 \xleftarrow{\pi_1} \mathbb{B}^4 \longleftarrow \dots$$

for interpreting Str Bool. Intuitively, what should its "elements" be? Full streams:

$$\left\{ s \in \prod_{n \in \omega} \mathbb{B}^n \;\middle|\; \forall n \in \omega, s_n = \pi_1(s_{n+1}) \right\}.$$

Yet an "element" of A should be the same thing as a morphism $1 \to A$.

$$\begin{tikzcd}
\{*\} \arrow[d, "s_0"] & \{*\} \arrow[l, "id"'] \arrow[d, "s_1"] & \{*\} \arrow[l, "id"'] \arrow[d, "s_2"] & \{*\} \arrow[l, "id"'] \arrow[d, "s_3"] & \dots \arrow[l] \\
\mathbb{B} & \mathbb{B}^2 \arrow[l, "\pi_1"'] & \mathbb{B}^3 \arrow[l, "\pi_1"'] & \mathbb{B}^4 \arrow[l, "\pi_1"'] & \dots \arrow[l]
\end{tikzcd}$$

This suggests using natural transformations as maps between diagrams.

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad := \quad [\omega^{op}, \mathbf{Set}]$$

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad := \quad [\omega^{op}, \mathbf{Set}]$$

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad := \quad [\omega^{op}, \mathbf{Set}]$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \ldots$$

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad := \quad [\omega^{op}, \mathbf{Set}]$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \ldots$$

Γ

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad \coloneqq \quad [\omega^{op}, \mathbf{Set}]$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \ldots$$

Γ

$$\Gamma(0) \xleftarrow{r_0^\Gamma} \Gamma(1) \xleftarrow{r_1^\Gamma} \Gamma(2) \xleftarrow{r_2^\Gamma} \Gamma(3) \xleftarrow{r_3^\Gamma} \Gamma(4) \qquad \ldots$$

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad := \quad [\omega^{op}, \mathbf{Set}]$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \quad \quad \ldots$$

Γ

$$\Gamma(0) \xleftarrow{r_0^\Gamma} \Gamma(1) \xleftarrow{r_1^\Gamma} \Gamma(2) \xleftarrow{r_2^\Gamma} \Gamma(3) \xleftarrow{r_3^\Gamma} \Gamma(4) \quad \ldots$$

A

$$A(0) \xleftarrow{r_0^A} A(1) \xleftarrow{r_1^A} A(2) \xleftarrow{r_2^A} A(3) \xleftarrow{r_3^A} A(4) \quad \ldots$$

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad := \quad [\omega^{op}, \mathbf{Set}]$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \ldots$$

$$\begin{tikzcd} \Gamma \arrow[d, "f"] & \Gamma(0) & \Gamma(1) \arrow[l, "r_0^\Gamma"'] & \Gamma(2) \arrow[l, "r_1^\Gamma"'] & \Gamma(3) \arrow[l, "r_2^\Gamma"'] & \Gamma(4) \arrow[l, "r_3^\Gamma"'] & \ldots \\ A & A(0) & A(1) \arrow[l, "r_0^A"'] & A(2) \arrow[l, "r_1^A"'] & A(3) \arrow[l, "r_2^A"'] & A(4) \arrow[l, "r_3^A"'] & \ldots \end{tikzcd}$$

The topos of trees

Objects and morphisms: synchrony beyond streams

$$\mathbf{Pr}(\omega) \quad := \quad [\omega^{op}, \mathbf{Set}]$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \dots$$

$$\begin{tikzcd}
\Gamma \arrow[d, "f"] \\
A
\end{tikzcd}
\qquad
\begin{tikzcd}
\Gamma(0) \arrow[d, "f_0"] & \Gamma(1) \arrow[l, "r_0^\Gamma"'] \arrow[d, "f_1"] & \Gamma(2) \arrow[l, "r_1^\Gamma"'] \arrow[d, "f_2"] & \Gamma(3) \arrow[l, "r_2^\Gamma"'] \arrow[d, "f_3"] & \Gamma(4) \arrow[l, "r_3^\Gamma"'] \arrow[d, "f_4"] & \dots \\
A(0) & A(1) \arrow[l, "r_0^A"'] & A(2) \arrow[l, "r_1^A"'] & A(3) \arrow[l, "r_2^A"'] & A(4) \arrow[l, "r_3^A"'] & \dots
\end{tikzcd}$$

Have we lost the ability to write recursive definitions? No. Remember:

Theorem (Kleene)

Every map $f : \uparrow A \to_{\mathrm{s}} A$ of $\mathbf{PCPO}$ *has a least "fixpoint"*

$$\mathit{fix}(f) = \bigsqcup \mathit{iter} \text{ where } \mathit{iter} : \omega \to A = n \mapsto \begin{cases} \bot & \text{if } n = 0 \\ f(\langle \mathit{iter}(n-1) \rangle) & \text{otherwise.} \end{cases}$$

Can we do the same thing in $\mathbf{Pr}(\omega)$, replacing the "completed" supremum with the entire chain, as we just did for types?

Yes, but we need something to play the rôle of lifting.

The topos of trees

Recursion: the "later" modality

$$\triangleright \quad : \quad \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \dots$$

A

$$A(0) \xleftarrow{r_0^A} A(1) \xleftarrow{r_1^A} A(2) \xleftarrow{r_2^A} A(3) \xleftarrow{r_3^A} A(4) \qquad \dots$$

The topos of trees

Recursion: the "later" modality

$$\triangleright \quad : \quad \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \ldots$$

A	$A(0) \xleftarrow{r_0^A}$	$A(1) \xleftarrow{r_1^A}$	$A(2) \xleftarrow{r_2^A}$	$A(3) \xleftarrow{r_3^A}$	$A(4)$	$\ldots$
$\triangleright A$	$\{*\}$	$A(0)$	$A(1)$	$A(2)$	$A(3)$	$\ldots$

The topos of trees

Recursion: the "later" modality

$$\triangleright \quad : \quad \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \quad \quad \ldots$$

A	$A(0) \xleftarrow{r_0^A} A(1) \xleftarrow{r_1^A} A(2) \xleftarrow{r_2^A} A(3) \xleftarrow{r_3^A} A(4) \quad \ldots$
$\triangleright A$	$\{*\} \xleftarrow{!} A(0) \xleftarrow{r_0^A} A(1) \xleftarrow{r_1^A} A(2) \xleftarrow{r_2^A} A(3) \quad \ldots$

The topos of trees

Recursion: the “later” modality

$$\triangleright \quad : \quad \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \ldots$$

$$\begin{array}{c|ccccccccccc}
A & A(0) & \xleftarrow{r_0^A} & A(1) & \xleftarrow{r_1^A} & A(2) & \xleftarrow{r_2^A} & A(3) & \xleftarrow{r_3^A} & A(4) & & \ldots \\
\downarrow \mathsf{delay} & & & & & & & & & & & \\
\triangleright A & \{*\} & \xleftarrow{!} & A(0) & \xleftarrow{r_0^A} & A(1) & \xleftarrow{r_1^A} & A(2) & \xleftarrow{r_2^A} & A(3) & & \ldots
\end{array}$$

The topos of trees

Recursion: the "later" modality

$$\triangleright \quad : \quad \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$$

$$0 \quad \leq \quad 1 \quad \leq \quad 2 \quad \leq \quad 3 \quad \leq \quad 4 \qquad \ldots$$

$$\begin{tikzcd} A \arrow[d, "\mathsf{delay}"] \\ \triangleright A \end{tikzcd}
\qquad
\begin{tikzcd}
A(0) \arrow[d, "!"] & A(1) \arrow[l, "r_0^A"'] \arrow[d, "r_0^A"] & A(2) \arrow[l, "r_1^A"'] \arrow[d, "r_1^A"] & A(3) \arrow[l, "r_2^A"'] \arrow[d, "r_2^A"] & A(4) \arrow[l, "r_3^A"'] \arrow[d, "r_3^A"] & \ldots \\
\{*\} & A(0) \arrow[l, "!"'] & A(1) \arrow[l, "r_0^A"'] & A(2) \arrow[l, "r_1^A"'] & A(3) \arrow[l, "r_2^A"'] & \ldots
\end{tikzcd}$$

The topos of trees

Guarded recursion

Let $f : \triangleright A \to A$ and define $\mathit{fix}(f) : 1 \to A$ by induction as

$$\mathit{fix}(f)_n : \{*\} \to A(n) = \begin{cases} f_0 & \text{if } n = 0 \\ f_n \circ \mathit{fix}(f)_{n-1} & \text{if } n > 0. \end{cases}$$

The topos of trees

Guarded recursion

Let $f : \triangleright A \to A$ and define $\mathit{fix}(f) : 1 \to A$ by induction as

$$\mathit{fix}(f)_n : \{*\} \to A(n) = \begin{cases} f_0 & \text{if } n = 0 \\ f_n \circ \mathit{fix}(f)_{n-1} & \text{if } n > 0. \end{cases}$$

Theorem (Löb)

We have $\mathit{fix}(f) = f \circ \mathsf{delay} \circ \mathit{fix}(f)$ and moreover it is the unique such map.

The topos of trees

Guarded recursion

Let $f : \triangleright A \to A$ and define $\mathit{fix}(f) : 1 \to A$ by induction as

$$\mathit{fix}(f)_n : \{*\} \to A(n) = \begin{cases} f_0 & \text{if } n = 0 \\ f_n \circ \mathit{fix}(f)_{n-1} & \text{if } n > 0. \end{cases}$$

Theorem (Löb)

We have $\mathit{fix}(f) = f \circ \mathsf{delay} \circ \mathit{fix}(f)$ and moreover it is the unique such map.

Proof.

We prove the equation by induction over n.

- Case $n = 0$: we have $\mathit{fix}(f)_0(*) = f_0(*) = f_0(\mathsf{delay}_0(f_0(*)))$.
- Case $n > 0$: we have

$$\begin{aligned} \mathit{fix}(f)_n &= f_n \circ \mathit{fix}(f)_{n-1} \\ &= f_n \circ f_{n-1} \circ r^A_{n-2} \circ \mathit{fix}(f)_{n-1} && \text{(I. H.)} \\ &= f_n \circ r^A_{n-1} \circ f_n \circ \mathit{fix}(f)_{n-1} && \text{(naturality)} \\ &= f_n \circ r^A_{n-1} \circ \mathit{fix}(f)_n. \end{aligned}$$

The uniqueness part of the statement is left as an exercise for the audience. □

The topos of trees

Cartesian-closed structure

The category $\mathbf{Pr}(\omega)$ has cartesian products, defined pointwise.

$$(A \times B)(n) = A(n) \times B(n), \qquad r_n^{A \times B}(x, y) = (r_n^X(x), r_n^Y(y))$$

It also has function objects, which can be described as follows.

$$(A \Rightarrow B)(n) = \left\{ f \in \prod_{i \leq n} A(i) \to B(i) \;\middle|\; \forall i < n, r_n^Y \circ f_{n+1} = f_n \circ r_n^X \right\}$$

$$r_n^{A \Rightarrow B} = (f_i)_{i \leq n+1} \mapsto (f_i)_{i \leq n}$$

The topos of trees

Cartesian-closed structure

The category $\mathbf{Pr}(\omega)$ has cartesian products, defined pointwise.

$$(A \times B)(n) = A(n) \times B(n), \qquad r_n^{A \times B}(x, y) = (r_n^X(x), r_n^Y(y))$$

It also has function objects, which can be described as follows.

$$(A \Rightarrow B)(n) = \left\{ f \in \prod_{i \leq n} A(i) \to B(i) \;\middle|\; \forall i < n, r_n^Y \circ f_{n+1} = f_n \circ r_n^X \right\}$$

$$r_n^{A \Rightarrow B} = (f_i)_{i \leq n+1} \mapsto (f_i)_{i \leq n}$$

Categories of presheaves ($\mathbf{Set}$-valued functors)

- They always have a lot of structure, including bicartesian closure.
  - For example $\mathbf{Pr}(\omega)$ has coproducts, in contrast with $\mathbf{PCPO}^{\uparrow}$.
  - Enough structure to interpret HOL & DTT. See Daniel's part!
- The previous definitions are "unfolded" version of general constructions.

General streams can be defined as

$$[\![\mathsf{Str}\,A]\!] \cong [\![A]\!] \times \triangleright [\![\mathsf{Str}\,A]\!].$$

Again, one can solve this as a colimit in $\mathbf{Pr}(\omega)$, obtaining

$$\begin{aligned} [\![\mathsf{Str}\,A]\!](0) &= [\![A]\!](0) \\ [\![\mathsf{Str}\,A]\!](n+1) &= [\![A]\!](n+1) \times [\![\mathsf{Str}\,A]\!](n). \end{aligned}$$

In particular, $[\![\mathsf{Str}\,\mathsf{Str}\,\mathsf{Bool}]\!]$ looks much better behaved. Can you describe it?

The topos of trees

Streams

General streams can be defined as

$$[\![\mathsf{Str}\,A]\!] \cong [\![A]\!] \times \triangleright [\![\mathsf{Str}\,A]\!].$$

Again, one can solve this as a colimit in $\mathbf{Pr}(\omega)$, obtaining

$$\begin{aligned} [\![\mathsf{Str}\,A]\!](0) &= [\![A]\!](0) \\ [\![\mathsf{Str}\,A]\!](n+1) &= [\![A]\!](n+1) \times [\![\mathsf{Str}\,A]\!](n). \end{aligned}$$

In particular, $[\![\mathsf{Str}\,\mathsf{Str}\,\mathsf{Bool}]\!]$ looks much better behaved. Can you describe it?

Remark

Birkedal et al. [2012] show how to build general guarded recursive types, even allowing for negative self-references (in line with Farzad's lecture this afternoon).

$$\mathcal{W} = \mathsf{Loc} \to_{\mathsf{fin}} \mathcal{T} \qquad\qquad \mathcal{T} = \triangleright \mathcal{W} \to \mathsf{Val} \to \mathsf{Prop}$$

Daniel will develop this example in detail.

Outline

Towards a guarded and synchronous stream language

We have a category where objects are explicit approximation sequences, with a fixpoint theorem, and nice properties. Is there a price to pay?

Towards a guarded and synchronous stream language

We have a category where objects are explicit approximation sequences, with a fixpoint theorem, and nice properties. Is there a price to pay?

The force morphism is no longer with us

The functor $\triangleright$ is not a comonad: there is no map $\triangleright A \to A$ in general.

(Exercise: what is $\triangleright 0$ in $\mathbf{Pr}(\omega)$? What does this imply for $\triangleright 0 \to 0$?)

Towards a guarded and synchronous stream language

We have a category where objects are explicit approximation sequences, with a fixpoint theorem, and nice properties. Is there a price to pay?

The force morphism is no longer with us

The functor $\triangleright$ is not a comonad: there is no map $\triangleright A \to A$ in general.

(Exercise: what is $\triangleright 0$ in $\mathbf{Pr}(\omega)$? What does this imply for $\triangleright 0 \to 0$?)

Consequences

The syntax needs to include $\triangleright$ as a type former.

Towards a guarded and synchronous stream language

We have a category where objects are explicit approximation sequences, with a fixpoint theorem, and nice properties. Is there a price to pay?

The force morphism is no longer with us

The functor $\triangleright$ is not a comonad: there is no map $\triangleright A \to A$ in general.

(Exercise: what is $\triangleright 0$ in $\mathbf{Pr}(\omega)$? What does this imply for $\triangleright 0 \to 0$?)

Consequences

The syntax needs to include $\triangleright$ as a type former.

We want to follow the discipline of *natural deduction*, meaning:

- introduction and elimination forms with a *generic* context in the conclusion
- β/η rules governing the interplay between introduction and elimination forms
  - β rule: elimination-of-introduction simplifies, e.g., $\mathsf{proj}_i(\langle t_1, t_2 \rangle) \equiv t_i$.
  - η rule: terms can be written as intro-of-elim for their type

The Fitch-style guarded language $\mathscr{S}$

Intuitions

$$\left(\frac{\Gamma, x : \triangleright A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A}\right) \qquad \frac{? \vdash t : A}{\Gamma \vdash \mathsf{guard}(t) : \triangleright A} \qquad \frac{? \vdash t : \triangleright A}{\Gamma \vdash \mathsf{open}(t) : A}$$

- We ought to be able to write a simple β-rule: $\mathsf{open}(\mathsf{guard}(t)) \rightsquigarrow t$.

The Fitch-style guarded language $\mathscr{S}$

Intuitions

$$\left(\frac{\Gamma, x : \triangleright A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A}\right) \qquad \frac{\Gamma? \vdash t : A}{\Gamma \vdash \mathsf{guard}(t) : \triangleright A} \qquad \frac{\Gamma? \vdash t : \triangleright A}{\Gamma \vdash \mathsf{open}(t) : A}$$

- We ought to be able to write a simple β-rule: $\mathsf{open}(\mathsf{guard}(t)) \rightsquigarrow t$.
- Can we pick the same context in the premises and in the conclusion?

The Fitch-style guarded language $\mathscr{S}$

Intuitions

$$\left(\frac{\Gamma, x : \triangleright A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A}\right) \qquad \frac{\Gamma? \vdash t : A}{\Gamma \vdash \mathsf{guard}(t) : \triangleright A} \qquad \frac{\Gamma? \vdash t : \triangleright A}{\Gamma \vdash \mathsf{open}(t) : A}$$

- We ought to be able to write a simple β-rule: $\mathsf{open}(\mathsf{guard}(t)) \rightsquigarrow t$.
- Can we pick the same context in the premises and in the conclusion? No.

$$\mathsf{fun}(x.\mathsf{open}(x)) \not: \triangleright A \to A \qquad \mathsf{rec}(x.\mathsf{open}(x)) \not: \mathsf{Str\,Bool}$$

The Fitch-style guarded language $\mathscr{S}$

Intuitions

$$\left(\frac{\Gamma, x : \triangleright A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A}\right) \qquad \frac{? \vdash t : A}{\Gamma \vdash \mathsf{guard}(t) : \triangleright A} \qquad \frac{? \vdash t : \triangleright A}{\Gamma \vdash \mathsf{open}(t) : A}$$

- We ought to be able to write a simple β-rule: $\mathsf{open}(\mathsf{guard}(t)) \rightsquigarrow t$.
- Can we pick the same context in the premises and in the conclusion? No.

$$\mathsf{fun}(x.\mathsf{open}(x)) \not{:} \triangleright A \to A \qquad \mathsf{rec}(x.\mathsf{open}(x)) \not{:} \mathsf{Str}\,\mathsf{Bool}$$

- We need to be able to write interesting terms, e.g.,

$$\mathsf{delay} \coloneqq \mathsf{fun}(x.\mathsf{guard}(x)) : A \to \triangleright A$$

$$\circledast \coloneqq \mathsf{fun}(f.\mathsf{fun}(x.\mathsf{guard}(\mathsf{open}(f)\,\mathsf{open}(x)))) : \triangleright(A \to B) \to \triangleright A \to \triangleright B$$

The Fitch-style guarded language $\mathscr{S}$

Intuitions

$$\left(\frac{\Gamma, x : \triangleright A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A}\right) \qquad \frac{🔒\,\Gamma \vdash t : A}{\Gamma \vdash \mathsf{guard}(t) : \triangleright A} \qquad \frac{🔓\,\Gamma \vdash t : \triangleright A}{\Gamma \vdash \mathsf{open}(t) : A}$$

- We ought to be able to write a simple β-rule: $\mathsf{open}(\mathsf{guard}(t)) \rightsquigarrow t$.
- Can we pick the same context in the premises and in the conclusion? No.

$$\mathsf{fun}(x.\mathsf{open}(x)) \;\not{:}\; \triangleright A \to A \qquad\qquad \mathsf{rec}(x.\mathsf{open}(x)) \;\not{:}\; \mathsf{Str\,Bool}$$

- We need to be able to write interesting terms, e.g.,

$$\begin{aligned} \mathsf{delay} &\coloneqq \mathsf{fun}(x.\mathsf{guard}(x)) : A \to \triangleright A \\ \circledast &\coloneqq \mathsf{fun}(f.\mathsf{fun}(x.\mathsf{guard}(\mathsf{open}(f)\,\mathsf{open}(x)))) : \triangleright(A \to B) \to \triangleright A \to \triangleright B \end{aligned}$$

A simple scope discipline

The term t in $\mathsf{open}(t)$ loses access to the variables bound *after* the last $\mathsf{guard}(-)$.

The Fitch-style guarded language $\mathscr{S}$

More formally

Definition (Typing contexts and un/locking)

- Contexts Γ map variables $x \in \mathsf{dom}(\Gamma)$ to a type $\Gamma(x).\mathsf{ty}$ and a *depth* $\Gamma(x).\mathsf{d}$.
- The operation 🔒 increases the depth of every variable.
- The operation 🔓 decreases the depth of every positive-depth variable and removes variables at depth zero.

$$\frac{x \in \mathsf{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x).\mathsf{ty}} \qquad \dots \qquad \frac{\Gamma, x : \triangleright A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A} \qquad \frac{\Gamma \vdash t : \mathsf{Str}\, A}{\Gamma \vdash \mathsf{head}(t) : A} \qquad \frac{\Gamma \vdash t : \mathsf{Str}\, A}{\Gamma \vdash \mathsf{tail}(t) : \triangleright \mathsf{Str}\, A}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : \triangleright \mathsf{Str}\, A}{\Gamma \vdash t :: u : \mathsf{Str}\, A} \qquad \frac{\text{🔒}\Gamma \vdash t : A}{\Gamma \vdash \mathsf{guard}(t) : \triangleright A} \qquad \frac{\text{🔓}\Gamma \vdash t : \triangleright A}{\Gamma \vdash \mathsf{open}(t) : A}$$

Remark for the categorically-minded

Categorically, $\text{🔒} \dashv \text{🔓} : \mathbb{C} \to \mathbb{C}$ where $\mathbb{C}$ is the category of contexts and renamings.

The Fitch-style guarded language $\mathscr{S}$

Substitution

Writing $\mathsf{Te}(\Gamma\,;A)$ for $\{t \mid \Gamma \vdash t : A\}$, the set of substitutions from Γ to Δ is

$$\mathsf{Sub}(\Delta\,;\Gamma) \coloneqq \prod_{x \in \mathsf{dom}(\Gamma)} \mathsf{Te}(\unlock^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty}).$$

The Fitch-style guarded language $\mathscr{S}$

Substitution

Writing $\mathsf{Te}(\Gamma\,;A)$ for $\{t \mid \Gamma \vdash t : A\}$, the set of substitutions from Γ to Δ is

$$\mathsf{Sub}(\Delta\,;\Gamma) \coloneqq \prod_{x \in \mathsf{dom}(\Gamma)} \mathsf{Te}(\faUnlock^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty}).$$

Locking and unlocking should act on substitutions, sending $\sigma \in \mathsf{Sub}(\Gamma\,;\Delta)$ to

$$\begin{aligned} \faLock\,\sigma &\in \mathsf{Sub}(\faLock\,\Gamma\,;\faLock\,\Delta) \\ \faUnlock\,\sigma &\in \mathsf{Sub}(\faUnlock\,\Gamma\,;\faUnlock\,\Delta) \end{aligned}$$

The Fitch-style guarded language $\mathscr{S}$

Substitution

Writing $\mathsf{Te}(\Gamma\,;A)$ for $\{t \mid \Gamma \vdash t : A\}$, the set of substitutions from Γ to Δ is

$$\mathsf{Sub}(\Delta\,;\Gamma) \coloneqq \prod_{x \in \mathsf{dom}(\Gamma)} \mathsf{Te}(\faUnlock^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty}).$$

Locking and unlocking should act on substitutions, sending $\sigma \in \mathsf{Sub}(\Gamma\,;\Delta)$ to

$$\begin{aligned} \faLock\,\sigma \quad &\in \mathsf{Sub}(\faLock\,\Gamma\,;\faLock\,\Delta) = \mathsf{Sub}(\Gamma\,;\Delta) \\ \faUnlock\,\sigma \quad &\in \mathsf{Sub}(\faUnlock\,\Gamma\,;\faUnlock\,\Delta) \end{aligned}$$

Writing $\mathsf{Te}(\Gamma\,;A)$ for $\{t \mid \Gamma \vdash t : A\}$, the set of substitutions from Γ to Δ is

$$\mathsf{Sub}(\Delta\,;\Gamma) \coloneqq \prod_{x \in \mathsf{dom}(\Gamma)} \mathsf{Te}(🔓^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty}).$$

Locking and unlocking should act on substitutions, sending $\sigma \in \mathsf{Sub}(\Gamma\,;\Delta)$ to

$$\begin{aligned} 🔒\sigma &\in \mathsf{Sub}(🔒\Gamma\,;🔒\Delta) = \mathsf{Sub}(\Gamma\,;\Delta) \\ 🔓\sigma &\in \mathsf{Sub}(🔓\Gamma\,;🔓\Delta) = \prod_{x \in \mathsf{dom}^{+}(\Gamma)} \mathsf{Te}(🔓^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty}) \end{aligned}$$

The Fitch-style guarded language $\mathscr{S}$

Substitution

Writing $\mathsf{Te}(\Gamma\,;A)$ for $\{t \mid \Gamma \vdash t : A\}$, the set of substitutions from Γ to Δ is

$$\mathsf{Sub}(\Delta\,;\Gamma) \coloneqq \prod_{x \in \mathsf{dom}(\Gamma)} \mathsf{Te}(\faUnlock^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty}).$$

Locking and unlocking should act on substitutions, sending $\sigma \in \mathsf{Sub}(\Gamma\,;\Delta)$ to

$$\faLock\,\sigma \coloneqq \sigma \in \mathsf{Sub}(\faLock\,\Gamma\,;\faLock\,\Delta) = \mathsf{Sub}(\Gamma\,;\Delta)$$

$$\faUnlock\,\sigma \coloneqq \sigma|_{\mathsf{dom}^+(\Gamma)} \in \mathsf{Sub}(\faUnlock\,\Gamma\,;\faUnlock\,\Delta) = \prod_{x \in \mathsf{dom}^+(\Gamma)} \mathsf{Te}(\faUnlock^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty})$$

The Fitch-style guarded language $\mathscr{S}$

Substitution

Writing $\mathsf{Te}(\Gamma\,;A)$ for $\{t \mid \Gamma \vdash t : A\}$, the set of substitutions from Γ to Δ is

$$\mathsf{Sub}(\Delta\,;\Gamma) \coloneqq \prod_{x \in \mathsf{dom}(\Gamma)} \mathsf{Te}(🔓^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty}).$$

Locking and unlocking should act on substitutions, sending $\sigma \in \mathsf{Sub}(\Gamma\,;\Delta)$ to

$$🔒\sigma \coloneqq \sigma \in \mathsf{Sub}(🔒\Gamma\,;🔒\Delta) = \mathsf{Sub}(\Gamma\,;\Delta)$$

$$🔓\sigma \coloneqq \sigma|_{\mathsf{dom}^{+}(\Gamma)} \in \mathsf{Sub}(🔓\Gamma\,;🔓\Delta) = \prod_{x \in \mathsf{dom}^{+}(\Gamma)} \mathsf{Te}(🔓^{\Gamma(x).\mathsf{d}}\Delta\,;\Gamma(x).\mathsf{ty})$$

Lemma (Weakening and substitution)

- ***Weakening***: *if $\Gamma \vdash t : A$ then $🔒\Gamma \vdash t : A$. If $🔓\Gamma \vdash t : A$ then $\Gamma \vdash t : A$.*
- ***Substitution***: *if $\Gamma \vdash t : A$ and $\sigma \in \mathsf{Sub}(\Delta\,;\Gamma)$ then $\Delta \vdash t[\sigma] : A$.*

The Fitch-style guarded language $\mathscr{S}$

Back to the interpretation in $\mathbf{Pr}(\omega)$

Interpreting typing contexts

- Define $\lhd : \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$ ("earlier") to be the functor $A \mapsto n \mapsto A_{n+1}$.
- A typing context Γ is interpreted in $\mathbf{Pr}(\omega)$ as the object

$$[\![\Gamma]\!] \coloneqq \prod_{x \in \mathsf{dom}(\Gamma)} \lhd^{\Gamma(x).\mathsf{d}} [\![\Gamma(x).\mathsf{ty}]\!].$$

The Fitch-style guarded language $\mathscr{S}$

Back to the interpretation in $\mathbf{Pr}(\omega)$

Interpreting typing contexts

- Define $\lhd : \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$ ("earlier") to be the functor $A \mapsto n \mapsto A_{n+1}$.
- A typing context Γ is interpreted in $\mathbf{Pr}(\omega)$ as the object

$$[\![\Gamma]\!] := \prod_{x \in \mathsf{dom}(\Gamma)} \lhd^{\Gamma(x).\mathsf{d}} [\![\Gamma(x).\mathsf{ty}]\!].$$

The functor $\lhd$ is left adjoint to $\rhd$.

$$((-)^{\sharp}, (-)_{\flat}) : \mathbf{Pr}(\omega)(-, \rhd =) \cong \mathbf{Pr}(\omega)(\lhd -, =)$$

We have $[\![🔒 \Gamma]\!] = \lhd [\![\Gamma]\!]$ as well as a canonical morphism $\mathsf{w}_\Gamma : [\![\Gamma]\!] \to \lhd [\![🔓 \Gamma]\!]$.

The Fitch-style guarded language $\mathscr{S}$

Back to the interpretation in $\mathbf{Pr}(\omega)$

Interpreting typing contexts

- Define $\lhd : \mathbf{Pr}(\omega) \to \mathbf{Pr}(\omega)$ ("earlier") to be the functor $A \mapsto n \mapsto A_{n+1}$.
- A typing context Γ is interpreted in $\mathbf{Pr}(\omega)$ as the object

$$[\![\Gamma]\!] := \prod_{x \in \mathsf{dom}(\Gamma)} \lhd^{\Gamma(x).\mathsf{d}} [\![\Gamma(x).\mathsf{ty}]\!].$$

The functor $\lhd$ is left adjoint to $\rhd$.

$$((-)^\sharp, (-)_\flat) : \mathbf{Pr}(\omega)(-, \rhd =) \cong \mathbf{Pr}(\omega)(\lhd -, =)$$

We have $[\![🔒\Gamma]\!] = \lhd[\![\Gamma]\!]$ as well as a canonical morphism $\mathsf{w}_\Gamma : [\![\Gamma]\!] \to \lhd[\![🔓\Gamma]\!]$.

Interpreting terms

$$\left[\!\!\left[\frac{🔒\Gamma \vdash t : A}{\Gamma \vdash \mathsf{guard}(t) : \rhd A}\right]\!\!\right] = [\![🔒\Gamma \vdash t : A]\!]_\flat \qquad \left[\!\!\left[\frac{🔓\Gamma \vdash t : \rhd A}{\Gamma \vdash \mathsf{open}(t) : A}\right]\!\!\right] = \mathsf{w}_\Gamma \,;\, [\![🔓\Gamma \vdash t : \rhd A]\!]^\sharp$$

Atomic β reduction

$$\boxed{t \rightsquigarrow t'}$$

$$
\begin{aligned}
\mathsf{app}(\mathsf{fun}(x.t), u) &\rightsquigarrow t[x/u] && (1)\\
\mathsf{if}(b \in \{\mathsf{tt}, \mathsf{ff}\}, t_{\mathsf{tt}}, t_{\mathsf{ff}}) &\rightsquigarrow t_b && (2)\\
\mathsf{head}(t :: u) &\rightsquigarrow t && (3)\\
\mathsf{tail}(t :: u) &\rightsquigarrow u && (4)\\
\mathsf{rec}(x.t) &\rightsquigarrow t[\mathsf{guard}(\mathsf{rec}(x.t))/x] && (5)\\
\mathsf{open}(\mathsf{guard}(t)) &\rightsquigarrow t && (6)
\end{aligned}
$$

Atomic β reduction

$$\boxed{t \rightsquigarrow t'}$$

$$\begin{aligned}
\mathsf{app}(\mathsf{fun}(x.t), u) &\rightsquigarrow t[x/u] && (1)\\
\mathsf{if}(b \in \{\mathsf{tt}, \mathsf{ff}\}, t_{\mathsf{tt}}, t_{\mathsf{ff}}) &\rightsquigarrow t_b && (2)\\
\mathsf{head}(t :: u) &\rightsquigarrow t && (3)\\
\mathsf{tail}(t :: u) &\rightsquigarrow u && (4)\\
\mathsf{rec}(x.t) &\rightsquigarrow t[\mathsf{guard}(\mathsf{rec}(x.t))/x] && (5)\\
\mathsf{open}(\mathsf{guard}(t)) &\rightsquigarrow t && (6)
\end{aligned}$$

Lemma (Subject reduction, atomic case)

If $\Gamma \vdash t : A$ and $t \rightsquigarrow t'$ then $\Gamma \vdash t' : A$.

The proof is the usual one, with clauses 5 and 6 relying on lock/unlock weakening.

Stratified β reduction

A *context* is a term with a unique occurrence of a formal "hole" denoted $\Box$.

$$\mathsf{Kx} \ni K ::= \Box \mid \mathsf{app}(K, u) \mid \mathsf{app}(t, K) \mid \mathsf{fun}(x.K) \mid \ldots$$

For every $K \in \mathsf{Kx}$ and $n \in \omega$ we define $K(n)$ as follows.

$$\begin{aligned}
\Box(n) &= n \\
\mathsf{guard}(K)(n) &= K(n+1) \\
\mathsf{open}(K)(n) &= K(n-1) \\
op(\ldots, K, \ldots)(n) &= K(n) \text{ otherwise}
\end{aligned}$$

Stratified β reduction

A *context* is a term with a unique occurrence of a formal "hole" denoted $\square$.

$$\mathrm{Kx} \ni K ::= \square \mid \mathsf{app}(K, u) \mid \mathsf{app}(t, K) \mid \mathsf{fun}(x.K) \mid \ldots$$

For every $K \in \mathrm{Kx}$ and $n \in \omega$ we define $K(n)$ as follows.

$$\begin{aligned} \square(n) &= n \\ \mathsf{guard}(K)(n) &= K(n+1) \\ \mathsf{open}(K)(n) &= K(n-1) \\ op(\ldots, K, \ldots)(n) &= K(n) \text{ otherwise} \end{aligned}$$

We can define a family of reduction relations for each $m \in \omega + 1$.

$$\boxed{t \rightarrow_k t'} \qquad\qquad \frac{u \rightsquigarrow u' \qquad K(0) < m}{K\{u\} \rightarrow_m K\{u'\}}$$

Lemma (Subject reduction)

If $\Gamma \vdash t : A$ and $t \rightarrow_m t'$ then $\Gamma \vdash t' : A$.

Metatheoretical results

Classic results

Theorem

The relation $\rightarrow_\omega$ is confluent.

Proof.

By the method of Tait and Martin-Löf. $\square$

Metatheoretical results

Classic results

Theorem

The relation $\to_\omega$ is confluent.

Proof.

By the method of Tait and Martin-Löf. □

Theorem (G., Tasson, Vienot)

The relations $\to_m$ for $m < \omega$ are strongly normalizing.

Proof.

By a step-indexed adaptation of Girard's reducibility candidates. □

(The theorem above has been proved for a slightly different variant of $\to_m$.)

Metatheoretical results

Erasing the modality

Target language

- Let $\mathscr{V}$ be *call-by-value* STLC with general rec. and $\mathsf{Str}\,A \cong A \times \mathsf{Unit} \to \mathsf{Str}\,A$.

$$\frac{\Gamma, x : \mathsf{Unit} \to A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : \mathsf{Unit}\,\mathsf{Str}\,A}{\Gamma \vdash t :: u : \mathsf{Str}\,A} \qquad \ldots$$

- Its model in $\mathbf{PCPO}$ [Amadio and Curien, 1998] is s.t. $[\![\mathsf{Unit} \to A]\!] \cong {\uparrow}[\![A]\!]$.

Metatheoretical results

Erasing the modality

Target language

- Let $\mathscr{V}$ be *call-by-value* STLC with general rec. and $\mathsf{Str}\,A \cong A \times \mathsf{Unit} \to \mathsf{Str}\,A$.

$$\frac{\Gamma, x : \mathsf{Unit} \to A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : \mathsf{Unit}\,\mathsf{Str}\,A}{\Gamma \vdash t :: u : \mathsf{Str}\,A} \qquad \ldots$$

- Its model in $\mathbf{PCPO}$ [Amadio and Curien, 1998] is s.t. $[\![\mathsf{Unit} \to A]\!] \cong \uparrow[\![A]\!]$.

Define an erasure function $\lceil - \rceil$ on types and terms.

$$\begin{aligned} \lceil \mathsf{Bool} \rceil &= \mathsf{Bool} \\ \lceil \mathsf{Str}\,A \rceil &= \mathsf{Str}\,\lceil A \rceil \\ \lceil A \to B \rceil &= \lceil A \rceil \to \lceil B \rceil \\ \lceil \triangleright A \rceil &= \mathsf{Unit} \to \lceil A \rceil \end{aligned} \qquad \begin{aligned} &\ldots \\ \lceil \mathsf{rec}(x.t) \rceil &= \mathsf{rec}(x.\lceil t \rceil) \\ \lceil \mathsf{guard}(t) \rceil &= \mathsf{fun}(().\lceil t \rceil) \\ \lceil \mathsf{open}(t) \rceil &= \lceil t \rceil\,() \end{aligned}$$

Metatheoretical results

Erasing the modality

Target language

- Let $\mathscr{V}$ be *call-by-value* STLC with general rec. and $\mathsf{Str}\,A \cong A \times \mathsf{Unit} \to \mathsf{Str}\,A$.

$$\frac{\Gamma, x : \mathsf{Unit} \to A \vdash t : A}{\Gamma \vdash \mathsf{rec}(x.t) : A} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : \mathsf{Unit}\,\mathsf{Str}\,A}{\Gamma \vdash t :: u : \mathsf{Str}\,A} \qquad \dots$$

- Its model in $\mathbf{PCPO}$ [Amadio and Curien, 1998] is s.t. $[\![\mathsf{Unit} \to A]\!] \cong \uparrow[\![A]\!]$.

Define an erasure function $\lceil - \rceil$ on types and terms.

$$\begin{aligned} \lceil \mathsf{Bool} \rceil &= \mathsf{Bool} & \dots \\ \lceil \mathsf{Str}\,A \rceil &= \mathsf{Str}\,\lceil A \rceil & \lceil \mathsf{rec}(x.t) \rceil &= \mathsf{rec}(x.\lceil t \rceil) \\ \lceil A \to B \rceil &= \lceil A \rceil \to \lceil B \rceil & \lceil \mathsf{guard}(t) \rceil &= \mathsf{fun}(().\lceil t \rceil) \\ \lceil \triangleright A \rceil &= \mathsf{Unit} \to \lceil A \rceil & \lceil \mathsf{open}(t) \rceil &= \lceil t \rceil\,() \end{aligned}$$

Theorem (G., Jafarrahmani, Tasson)

If $t : \mathsf{Str}\,\mathsf{Bool}$ then $\lceil t \rceil$ has the same elements as t. In particular, $\lceil t \rceil$ is productive.

Outline

Limitations

Results

$\mathscr{S}$ is the simplest interesting guarded language I can think of.

Disappointment

$\mathscr{S}$ is unsatisfactory compared to existing synchronous or guarded languages.

What is lacking or unpleasant in $\mathscr{S}$?

Limitations

Failure of confluence in $\mathscr{S}$

The reduction relations $\to_k$ for $0 < k < \omega$ fail to be confluent. Witness, for $k = 1$:

$$\mathsf{fun}(x.\mathsf{guard}(x))\,(I\,y)$$

Limitations

Failure of confluence in $\mathscr{S}$

The reduction relations $\to_k$ for $0 < k < \omega$ fail to be confluent. Witness, for $k = 1$:

$$\begin{tikzcd}[column sep=small] \mathsf{fun}(x.\mathsf{guard}(x))\,(I\,y) \arrow[dr, "1"'] & \\ & \mathsf{fun}(x.\mathsf{guard}(x))\,y \end{tikzcd}$$

Limitations

Failure of confluence in $\mathscr{S}$

The reduction relations $\rightarrow_k$ for $0 < k < \omega$ fail to be confluent. Witness, for $k = 1$:

$$
\begin{tikzcd}
\mathsf{fun}(x.\mathsf{guard}(x))\,(I\,y) \arrow[dr, "1"'] & & \mathsf{guard}(y) \\
& \mathsf{fun}(x.\mathsf{guard}(x))\,y \arrow[ur, "1"'] &
\end{tikzcd}
$$

Limitations

Failure of confluence in $\mathscr{S}$

The reduction relations $\to_k$ for $0 < k < \omega$ fail to be confluent. Witness, for $k = 1$:

$$\begin{tikzcd}[column sep=small]
& \mathsf{guard}(I\,y) & \\
\mathsf{fun}(x.\mathsf{guard}(x))\,(I\,y) \arrow[ur, "1"] \arrow[dr, "1"'] & & \mathsf{guard}(y) \\
& \mathsf{fun}(x.\mathsf{guard}(x))\,y \arrow[ur, "1"'] &
\end{tikzcd}$$

Limitations

Failure of confluence in $\mathscr{S}$

The reduction relations $\to_k$ for $0 < k < \omega$ fail to be confluent. Witness, for $k = 1$:

$$
\begin{tikzcd}
 & \mathsf{guard}(I\,y) \arrow[dr, "2"] & \\
\mathsf{fun}(x.\mathsf{guard}(x))\,(I\,y) \arrow[ur, "1"] \arrow[dr, "1"'] & & \mathsf{guard}(y) \\
 & \mathsf{fun}(x.\mathsf{guard}(x))\,y \arrow[ur, "1"'] &
\end{tikzcd}
$$

Limitations

$\triangleright$ is not enough

One can write simple synchronous functions in $\mathscr{S}$.

$$\begin{aligned} \mathit{nots} &: \mathsf{Str\,Bool} \to \mathsf{Str\,Bool} \\ \mathit{nots} &\coloneqq \mathsf{rec}(\mathsf{F}.\mathsf{fun}(\mathsf{xs}.\mathsf{app}(\mathit{notb}, \mathsf{head}(\mathsf{xs})) :: (\mathsf{F} \circledast \mathsf{tail}(\mathsf{xs})))) \end{aligned}$$

However, some very reasonable functions cannot be written.

$$\begin{aligned} \mathit{ands} &\coloneqq \mathsf{rec}(\mathsf{F}.\mathsf{fun}(\mathsf{xs}.\mathsf{app}(\mathit{and1}, \mathsf{xs}) :: \mathsf{app}(\mathsf{F}, \mathsf{tail}(\mathsf{tail}(\mathsf{xs}))))) \\ \mathit{stut} &\coloneqq \mathsf{rec}(\mathsf{F}.\mathsf{fun}(\mathsf{xs}.\mathsf{head}(\mathsf{xs}) :: \mathsf{head}(\mathsf{xs}) :: \mathsf{app}(\mathsf{F}, \mathsf{tail}(\mathsf{xs})))) \end{aligned}$$

Worse from a synchronous perspective, mutual recursion is rejected as well!

$$\begin{aligned} \mathit{natpos} &: \mathsf{Str\,Nat} \times \mathsf{Str\,Nat} && (*\ \mathit{In}\ \mathscr{L}!\ *) \\ \mathit{natpos} &\coloneqq \mathsf{rec}(\mathsf{NP}.\langle 0 :: \mathsf{proj}_2(\mathsf{NP}), \mathsf{app}(\mathsf{sucs}, \mathsf{proj}_1(\mathsf{NP}))\rangle) \end{aligned}$$

A possible solution (G. [2018])

Add new modalities beyond $\triangleright$, corresponding to other time transforms.

Limitations

Simple types are not enough

The historical interest in guarded recursion from the type-theoretical side was to replace the positivity criterion used in proof assistants (see Damien's lecture).

Several authors [Birkedal et al., 2012, Birkedal and Møgelberg, 2013, Bizjak et al., 2016, Bahr et al., 2017, Gratzer, 2025, ...] have developed dependent type theories featuring "later"-like modalities.

Daniel will touch upon this line of work in his lecture.

Outline

Conclusion

Summary

- Start from a run-of-the-mill nonstrict language $\mathscr{L}$ with streams.
- Build a very classic denotational semantics, with synchrony in mind.
- Contrast this model with a category where all maps are synchronous.
- Transfer back features from the latter to the syntax, obtaining $\mathscr{S}$.

Some open questions

- What is the relationship between general and guarded recursion?
  - Study functors between $\mathbf{Pr}(\omega)$ and some well-chosen category of domains.
- Can we have a proper λ-calculus with guarded recursion?
  - Make $\to_k$ confluent for all k.
  - Make $\to_\omega$ strongly normalizing via infinitary rewriting?
- Design a similar calculus for other temporal modalities.

References I

R. Amadio and P. Curien. *Domains and Lambda-Calculi*. Cambridge University Press, 1998.

P. Bahr, H. Bugge Grathwohl, and R. E. Møgelberg. The Clocks Are Ticking: No More Delays! Reduction Semantics for Type Theory with Guarded Recursion. In *Logic in Computer Science (LICS'17)*. Springer, 2017. URL http://www.itu.dk/people/mogel/papers/lics2017.pdf.

L. Birkedal and R. E. Møgelberg. Intensional Type Theory with Guarded Recursive Types qua Fixed Points on Universes. *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, 6 2013. doi: 10.1109/lics.2013.27. URL http://www.itu.dk/people/mogel/papers/lics2013.pdf.

L. Birkedal, R. E. Møgelberg, J. Schwinghammer, and K. Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012. URL https://arxiv.org/pdf/1208.3596.pdf.

A. Bizjak, H. Bugge Grathwohl, R. Clouston, R. E. Møgelberg, and L. Birkedal. Guarded Dependent Type Theory with Coinductive Types. In *Foundations of Software Science and Computation Structures (FoSSaCS'16)*. Springer, 2016. URL https://arxiv.org/pdf/1601.01586v1.

References II

P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *Principles of Programming Languages (POPL'87)*, 1987. URL `http://www-verimag.imag.fr/~halbwach/SCAN/lustre-popl87.pdf`.

R. Clouston. Fitch-Style Modal Lambda Calculi. In *Foundations of Software Science and Computation Structures (FoSSaCS'18)*, 2018. URL `https://arxiv.org/pdf/1710.08326`.

C. Elliott and P. Hudak. Functional Reactive Animation. In *International Conference on Functional Programming (ICFP'97)*. ACM, 1997. URL `http://conal.net/papers/icfp97/icfp97.pdf`.

D. Gratzer. A modal deconstruction of löb induction. *Proc. ACM Program. Lang.*, 9(POPL):864–892, 2025. doi: 10.1145/3704866. URL `https://doi.org/10.1145/3704866`.

A. Guatto. *A Synchronous Functional Language with Integer Clocks*. PhD thesis, École normale supérieure, 2016. URL `http://www.di.ens.fr/~guatto/papers/thesis_guatto.pdf`.

References III

A. Guatto. A Generalized Modality for Recursion. In *Logic in Computer Science (LICS'18)*, 2018. URL `https://arxiv.org/pdf/1805.11021`.

H. Huwig and A. Poigné. A note on inconsistencies caused by fixpoints in a cartesian closed category. *Theoretical Computer Science*, 73(1):101–112, 1990. ISSN 0304-3975. doi: **https://doi.org/10.1016/0304-3975(90)90165-E**. URL `https://www.sciencedirect.com/science/article/pii/030439759090165E`.

G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing Congress (IFIP'74)*. IFIP, 1974. URL `https://www.cs.princeton.edu/courses/archive/fall07/cos595/kahn74.pdf`.

H. Nakano. A Modality for Recursion. In *Logic in Computer Science (LICS'00)*. IEEE, 2000. URL `http://www602.math.ryukoku.ac.jp/~nakano/papers/modality-lics00.ps`.

M. Pouzet. Lucid Synchrone: un langage synchrone d'ordre supérieur, 11 2002. URL `https://www.di.ens.fr/~pouzet/bib/habilitation-pouzet02.ps.gz`. Habilitation à diriger des recherches.

Appendix

Additional slides

Mapping the CPO landscape

$$\begin{tikzcd}
\mathbf{CPO} \arrow[r, bend left, "F_{\uparrow}"] & \mathbf{CPO}_{\uparrow} \cong \mathbf{PCPO} \arrow[l, bend left, "U_{\uparrow}"] \arrow[r, bend left, "F^{\uparrow}"] & \mathbf{PCPO}^{\uparrow} \arrow[l, bend left, "U^{\uparrow}"]
\end{tikzcd}$$

Category	Pointed?	Strict?	CCC?	Fixpoints?	Coproducts?
$\mathbf{CPO}$	N	N	Y	N	Y
$\mathbf{PCPO}/\mathbf{CPO}_{\uparrow}$	Y	Y	N	Y*	Y
$\mathbf{PCPO}^{\uparrow}$	Y	N	Y	Y	N

*: "lift-guarded" fixpoints, in the sense that $\mathit{fix}_A : (\uparrow A \to A) \to A$.

Huwig and Poigné [1990]'s incompatibility result

A CCC w/ coproducts and general fixpoints is equivalent to the terminal category.

Additional slides

Proof of the uniqueness of the Löb fixed-point

Recall that given $f : \triangleright A \to A$, the map $\mathit{fix}(f) : 1 \to A$ is defined as

$$\mathit{fix}(f)_n : \{*\} \to A(n) = \begin{cases} f_0 & \text{if } n = 0 \\ f_n \circ \mathit{fix}(f)_{n-1} & \text{if } n > 0. \end{cases}$$

Show that every $g : 1 \to A$ satisfying $g = f \circ \mathsf{delay} \circ g$ is $\mathit{fix}(f)$ by induction.

- Case $n = 0$: immediate since $(\mathit{delay} \circ g)_n = !$.
- Case $n > 0$: we have

$$\begin{aligned} g_n &= f_n \circ r^A_{n-1} \circ g_n \\ &= f_n \circ g_{n-1} \circ r^A_n && \text{(naturality)} \\ &= f_n \circ \mathit{fix}(f)_{n-1} \circ r^A_n && \text{(I. H.)} \\ &= f_n \circ r^A_n \circ \mathit{fix}(f)_n && \text{(naturality)} \\ &= \mathit{fix}(f)_n. \end{aligned}$$

Additional slides

Contexts and renaming for the Fitch-style syntax

The set of types Ty is given by $\mathsf{Ty} \ni A, B ::= \mathsf{Bool} \mid A \to B \mid \mathsf{Str}\, A \mid \triangleright A$.

Definition (The category $\mathbb{C}$ of contexts and renamings)

- Objects are finite families $\Gamma = (\mathsf{dom}(\Gamma) : \mathbf{FinSet}, \underline{\Gamma} : \mathsf{dom}(\Gamma) \to \mathsf{Ty} \times \omega)$.
- Morphisms are type-preserving, non-depth-decreasing maps

$$\mathbb{C}(\Gamma\,;\Delta) \coloneqq \{\rho : \mathsf{dom}(\Gamma) \to \mathsf{dom}(\Delta) \mid \forall x \in \mathsf{dom}(\Gamma), \underline{\Gamma}(x) \le \underline{\Delta}(\rho(x))\}.$$

Additional slides

Contexts and renaming for the Fitch-style syntax

The set of types Ty is given by $\mathsf{Ty} \ni A, B ::= \mathsf{Bool} \mid A \to B \mid \mathsf{Str}\, A \mid \triangleright A$.

Definition (The category $\mathbb{C}$ of contexts and renamings)

- Objects are finite families $\Gamma = (\mathsf{dom}(\Gamma) : \mathbf{FinSet}, \underline{\Gamma} : \mathsf{dom}(\Gamma) \to \mathsf{Ty} \times \omega)$.
- Morphisms are type-preserving, non-depth-decreasing maps
$$\mathbb{C}(\Gamma\,;\Delta) := \{\rho : \mathsf{dom}(\Gamma) \to \mathsf{dom}(\Delta) \mid \forall x \in \mathsf{dom}(\Gamma), \underline{\Gamma}(x) \leq \underline{\Delta}(\rho(x))\}.$$

Given Γ in $\mathsf{obj}(\mathbb{C})$, write:

- $\mathsf{dom}^+(\Gamma)$ for $\{x \in \mathsf{dom}(\Gamma) \mid \Gamma(x).\mathsf{d} > 0\}$,
- for $f : \omega \to \omega$ monotone, set $\mathsf{dom}(f_*\Gamma) := \mathsf{dom}(\Gamma)$ and $\underline{f_*\Gamma}(x) := (id \times f) \circ \underline{\Gamma}$.

Definition (The locking and unlocking functors)

Define two functors 🔒, 🔓 $: \mathbb{C} \to \mathbb{C}$ by their actions on objects.

$$🔒\Gamma := (+1)_*\Gamma \qquad 🔓\Gamma := (\mathsf{dom}^+(\Gamma), (-1)_*\Gamma)$$

Their action on morphisms is morally the identity.

Additional slides

No resource guarantees

Synchronous languages such as Lustre compile to finite state machines.

$$A \to_{sync} B \qquad \hookrightarrow \qquad \sum_{S:\mathbf{FinSet}} (S \times (\partial A \times S \to \partial B \times S))$$

The input and output "alphabets" ∂A and ∂B should arguably be finite.

However, types such as $\mathsf{Str}\,\mathsf{Str}\,\mathsf{Bool}$ seem to be intrinsically non-real-time.

$$\begin{aligned}\partial\, \mathsf{Str}\,\mathsf{Str}\,\mathsf{Bool}_n &\cong \sum_{\mathsf{xs}:\mathsf{Str}\,\mathsf{Str}\,\mathsf{Bool}_n} \{\mathsf{xs}' \in \mathsf{Str}\,\mathsf{Str}\,\mathsf{Bool}_{n+1} \mid \mathsf{xs} = \mathsf{delay}(\mathsf{xs}')_n\} \\ &\cong \mathbb{B}^n\end{aligned}$$

It should however be possible to adapt the state-passing transform to the general guarded-recursive setting, and to reject non-finite-state programs.