Space-efficiency and the Geometry of Interaction

Ulrich Schöpp

LMU Munich

February 16, 2006

Project *Pro.Platz* at LMU Munich

[Hofmann, Johannsen, Schwichtenberg, S]

Programming language aspects of sublinear space complexity classes (L, Polylog-space, NL, …)

Programming languages capturing logarithmic space

Find a convenient programming language capturing FL.

Existing work

- Function algebra (explicit resource bounds) [Lind 1974]
- Function algebra with unary numbers [Bellantoni 1992]
- Tail recursive readonly programs [Jones 1999]
- Function algebra BC_ε^- [Møller-Neergaard & Mairson 2004]

A model for space-efficient computation

We propose to use a version of the Geometry of Interaction as a flexible semantic universe for space-efficient computation.

This talk

1. Function algebra BC_ε^-
2. A version of the Geometry of Interaction
   - 2.1 Interpretation of BC_ε^-
   - 2.2 Extensions of BC_ε^-
3. Towards an intrinsic model capturing FL

The function algebra BC^-_ε

Origin

- BC [Bellantoni & Cook 1992]
- BC^- [Murawski & Ong 2000]
- $BC^- \subseteq$ FL [Ong & Mairson 2003]
- $BC^-_\varepsilon =$ FL [Møller-Neergaard 2004]

Definition

BC^-_ε is a set of functions on $\mathbb{N}_2 = \{0,1\}^*$.

$$BC^-_\varepsilon \subseteq \{f : \mathbb{N}_2^m \times \mathbb{N}_2^n \to \mathbb{N}_2 \mid m, n \in \mathbb{N}\}$$

Notation: $f(\vec{x};\, \vec{y})$
(*normal* arguments $\vec{x}$; *safe* arguments $\vec{y}$)

Functions in BC^-

Base functions

- $\varepsilon(\cdot;\ \cdot) = \varepsilon$
- $\mathrm{s}_0(\cdot;\ y) = y0$
- $\mathrm{s}_1(\cdot;\ y) = y1$
- $\mathrm{p}(\cdot;\ yi) = y$ and $\mathrm{p}(\cdot;\ \varepsilon) = \varepsilon$
- $\mathrm{c}(\cdot;\ y, u, v) = \begin{cases} u & \text{if } y = y'1 \\ v & \text{otherwise} \end{cases}$
- $\pi_i(\vec{x};\ \vec{y}) = x_i$ and $\pi_{m+i}(\vec{x};\ \vec{y}) = y_i$

Composition

Safe arguments are treated *linearly*.

$$f(g_1(x;\ \cdot), g_2(x;\ \cdot);\ \ h_1(x;\ y_1), h_2(x;\ y_2, y_3))$$

Recursion

$f = \mathrm{rec}(g, h_0, h_1)$ satisfies

$$f(\vec{x}, \varepsilon;\ \vec{y}) = g(\vec{x};\ \vec{y})$$
$$f(\vec{x}, xi;\ \vec{y}) = h_i(\vec{x}, x;\ f(\vec{x}, x;\ \vec{y}))$$

Example:

$$parity(\varepsilon;\ \cdot) = s_1(\varepsilon)$$
$$parity(x0;\ \cdot) = parity(x;\ \cdot)$$
$$parity(x1;\ \cdot) = \mathrm{c}(\cdot;\ parity(x;\ \cdot), s_0(\varepsilon), s_1(\varepsilon))$$

Non-example:

$$f(\vec{x}, \varepsilon;\ \vec{y}) = g(\vec{x};\ \vec{y})$$
$$f(\vec{x}, xi;\ \vec{y}) = \mathrm{c}(\cdot;\ f(\vec{x}, x;\ \vec{y}), \mathrm{p}(f(\vec{x}, x; \vec{y})), h(\vec{x}, x;\ f(\vec{x}, x;\ \vec{y})))$$

Functions in BC^-_ε

BC^-_ε extends BC^- by course-of-value recursion.

$f = \mathrm{rec}(g, h_0, h_1, d_0, d_1)$ satisfies

$$f(\vec{x}, \varepsilon;\ \vec{y}) = g(\vec{x};\ \vec{y})$$
$$f(\vec{x}, xi;\ \vec{y}) = h_i(\vec{x}, x;\ f(\vec{x}, x \gg |d_i(\vec{x}, x;\ \cdot)|;\ \vec{y}))$$

Example:

$$ulog(\varepsilon;\ \cdot) = \varepsilon$$
$$ulog(xi;\ \cdot) = s_1(ulog(x \gg |uhalf(x)|;\ \cdot))$$

Evaluating BC_{ε}^{-} in FL

By example

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Naive call-by-value evaluation uses linear space.
FL-evaluation of BC_{ε}^{-} proceeds bit by bit.

Evaluating BC_{ε}^{-} in FL

$$\begin{aligned} \mathit{shift}(\varepsilon;\, y) &= y \\ \mathit{shift}(xi;\, y) &= \mathrm{p}(\cdot;\, \mathit{shift}(x;\, y)) \end{aligned}$$

$$\mathrm{Bit}(0)? \text{ — } \mathit{shift}(10;\, 1101)$$

Evaluating BC_{ε}^{-} in FL

$$\mathit{shift}(\varepsilon;\, y) = y$$
$$\mathit{shift}(xi;\, y) = \mathrm{p}(\cdot;\, \mathit{shift}(x;\, y))$$

$$\mathrm{Bit}(0)? \multimap \mathit{shift}(10;\, 1101)$$
$$\mathrm{Bit}(1)? \multimap \mathit{shift}(1;\, 1101)$$

Evaluating BC_{ε}^{-} in FL

$$\begin{aligned} \mathit{shift}(\varepsilon;\, y) &= y \\ \mathit{shift}(xi;\, y) &= \mathrm{p}(\cdot;\, \mathit{shift}(x;\, y)) \end{aligned}$$

$$\begin{aligned} &\mathrm{Bit}(0)? \text{ — } \mathit{shift}(10;\, 1101) \\ &\mathrm{Bit}(1)? \text{ — } \mathit{shift}(1;\, 1101) \\ &\mathrm{Bit}(2)? \text{ — } \mathit{shift}(\varepsilon;\, 1101) \end{aligned}$$

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

$$\mathrm{Bit}(0)? \text{ — } shift(10;\ 1101)$$
$$\mathrm{Bit}(1)? \text{ — } shift(1;\ 1101)$$
$$\mathrm{Bit}(2)? \text{ — } shift(\varepsilon;\ 1101)$$

Bit 2 of $shift(\varepsilon;\ 1101)$ is 1

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Bit 2 of $shift(\varepsilon;\ 1101)$ is 1

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Bit 2 of $shift(\varepsilon;\ 1101)$ is 1

$\mathrm{Bit}(0)?$ — $shift(10;\ 1101)$

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Bit 2 of $shift(\varepsilon;\ 1101)$ is 1

$\mathrm{Bit}(0)?$ — $shift(10;\ 1101)$

$\mathrm{Bit}(1)?$ — $shift(1;\ 1101)$

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Bit 2 of $shift(\varepsilon;\ 1101)$ is 1

$\mathrm{Bit}(0)?$ — $shift(10;\ 1101)$
$\mathrm{Bit}(1)?$ — $shift(1;\ 1101)$

Bit 1 of $shift(1;\ 1101)$ is 1

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Bit 1 of $shift(1;\ 1101)$ is 1

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Bit 1 of $shift(1;\ 1101)$ is 1

$\mathrm{Bit}(0)?$ — $shift(10;\ 1101)$

Evaluating BC_{ε}^{-} in FL

$$shift(\varepsilon;\ y) = y$$
$$shift(xi;\ y) = \mathrm{p}(\cdot;\ shift(x;\ y))$$

Bit 1 of $shift(1;\ 1101)$ is 1

$\mathrm{Bit}(0)?$ — $shift(10;\ 1101)$

Bit 0 of $shift(10;\ 1101)$ is 1

Evaluating BC_ε^- in FL

- By linearity, recursion can be evaluated by storing only:
  - Initial question, current question
  - Recursion depth
  - Already computed bit and its depth
- All intermediate values and all questions are polynomially bounded.

$$|f(\vec{x};\ \vec{y})| \leq p(|\vec{x}, \vec{y}|)$$
$$q \leq p'(q_0)$$

$\Rightarrow$ By iterating over the bits of the output, f can be computed in logarithmic space.

Modelling the evaluation of BC^{-}_{ε}

Question/answer dialogues $\Rightarrow$ *Game Semantics*
Need only a special case $\Rightarrow$ *Geometry of Interaction situation*

Geometry of Interaction

- Origin: syntax-free explanation of cut elimination for linear logic [Girard 1988]
- Connection to game semantics [Abramsky, Jagadeesan 92]
- General categorical formulation [Hyland], [Abramsky, Haghverdi, Scott 2000]
- Compilation of programming languages [Mackie 1995]
- Close connections to abstract machines, e.g. Krivine Abstract Machine [Danos, Herbelin, Regnier 1996]
- Used in complexity theory [Baillot], [Pedicini], [Dal Lago] ...

Geometry of Interaction

Objects

$$A = (A^-, A^+)$$

Example:

$$\mathbb{N}^- = \mathbb{N}$$
$$\mathbb{N}^+ = \{0, 1, *\}$$

Natural numbers are represented by functions $\mathbb{N}^- \to \mathbb{N}^+$.

Morphisms

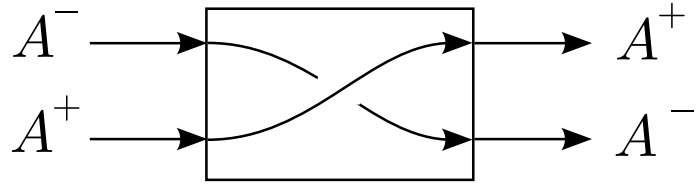
Morphism of type $f : A \to B$ is a partial function

$$f : B^- + A^+ \longrightarrow B^+ + A^-$$

Identity and Composition

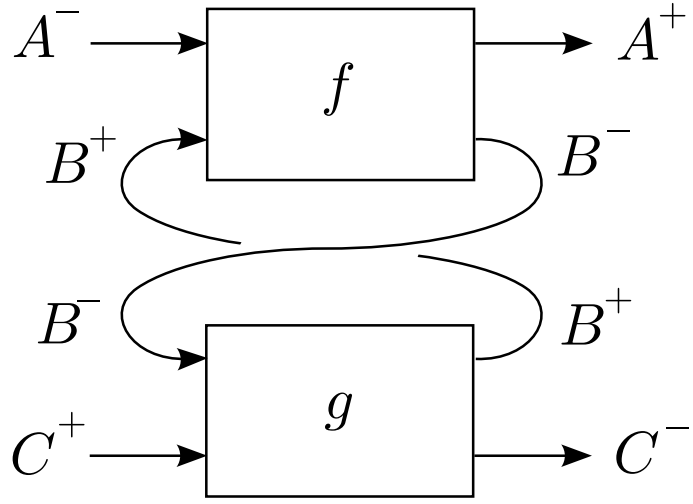
Identity

$$id\colon A \longrightarrow A$$



Composition

$$\frac{f\colon A \longrightarrow B \qquad g\colon B \longrightarrow C}{g \cdot f\colon A \longrightarrow C}$$



Structure

- Pairs

$$A \otimes B = (A^- + B^-, A^+ + B^+)$$

- Functions

$$A \multimap B = (A^+ + B^-, A^- + B^+)$$

- Storage

$$!A = (A^- \times \mathbb{N},\ A^+ \times \mathbb{N})$$

Interpreting BC^-_ε

$f(x_1, \ldots, x_m;\ y_1, \ldots, y_n)$ is interpreted as a morphism

$$\|f\| : !\mathbb{N}^m \otimes !\mathbb{N}^n \longrightarrow \mathbb{N}$$

Conditional

$\|\mathrm{c}\| : !\mathbb{N} \otimes !\mathbb{N} \otimes !\mathbb{N} \longrightarrow \mathbb{N}$ is given by

$$\begin{aligned}
\mathbb{N}^- + !\mathbb{N}^+ + !\mathbb{N}^+ + !\mathbb{N}^+ &\longrightarrow \mathbb{N}^+ + !\mathbb{N}^- + !\mathbb{N}^- + !\mathbb{N}^- \\
in_1(q) &\longmapsto in_2(0, q) \\
in_2(1, s) &\longmapsto in_3(s, 0) \\
in_2(_, s) &\longmapsto in_4(s, 0) \\
in_3(a, s) &\longmapsto in_1(a) \\
in_4(a, s) &\longmapsto in_1(a)
\end{aligned}$$

Interpreting BC_{ε}^{-}

Recursion combinator

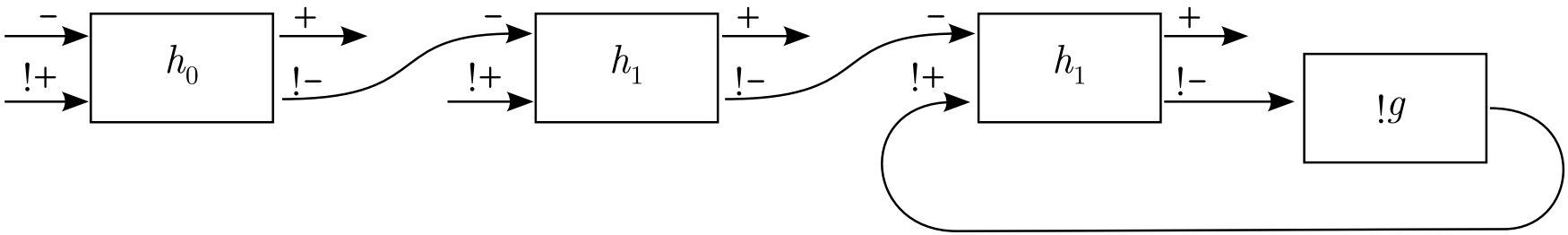
$$\mathrm{rec}\colon !\mathbb{N}\otimes!(!\mathbb{N}\multimap\mathbb{N})\otimes!(!\mathbb{N}\multimap\mathbb{N})\otimes!\mathbb{N}\multimap\mathbb{N}$$

$$\begin{aligned}
inl(q) &\longmapsto in_x(0,\langle q,q,0,-1,-1\rangle)\\
in_x(*,s) &\longmapsto in_g(s.q,s)\\
in_x(i,s) &\longmapsto in_{h_i}(s.q,s)\\
in_g(a,s) &\longmapsto \begin{cases} inr(a) & \text{if } s_3=0\\ in_x(q_0,\langle s_1,s_1,0,s_3,a\rangle) & \text{otherwise}\end{cases}\\
in_{h_i}(a,s) &\longmapsto in_g(a,s)\\
in_{r_i}((q,l),s) &\longmapsto \begin{cases} in_{r_i}((s_5,l),s) & \text{if } s_3+1=s_4\\ in_x(s_3+1,\langle s_1,q,s_3+1,s_4,s_5\rangle)) & \text{otherwise}\end{cases}
\end{aligned}$$

Interpreting BC^{-}_{ε}

Recursion combinator

$$\mathrm{rec}\colon\ !\mathbb{N}\otimes!(!\mathbb{N}\multimap\mathbb{N})\otimes!(!\mathbb{N}\multimap\mathbb{N})\otimes!\mathbb{N}\multimap\mathbb{N}$$



Interpreting BC_ε^- in GoI

$f(x_1, \ldots, x_m;\ y_1, \ldots, y_n)$ in BC_ε^- is interpreted as

$$\|f\| : !\mathbb{N}^{\otimes m} \otimes !\mathbb{N}^{\otimes n} \longrightarrow \mathbb{N}.$$

There exist $k \in \mathbb{N}$ and a polynomial p, such that:

1. If $\langle \vec{x}, \vec{y} \rangle \in Space(b(z))$ then

   $$\|f\| \cdot !\langle \vec{x}, \vec{y} \rangle \in Space(\, k \cdot (z + \log |\vec{x}, \vec{y}|) + b(k \cdot (z + \log |\vec{x}, \vec{y}|))\,)$$

2. For all $\langle \vec{x}, \vec{y} \rangle$ and all initial questions q_0, *at most one* question (q, s) may be sent to each safe argument y_i.

   $$q \leq q_0 + p(|\vec{x}, \vec{y}|)$$

What do we gain from the interpretation?

Direct consequences for BC^{-}_{ε}

- Simple compilation of the language
- Function types
- Recursively defined functions
- General framework for modelling data types

More generally

- GoI captures the recomputation pattern often used in FL-computation
- GoI-Realizability for FL

Recursively defined functions

Example

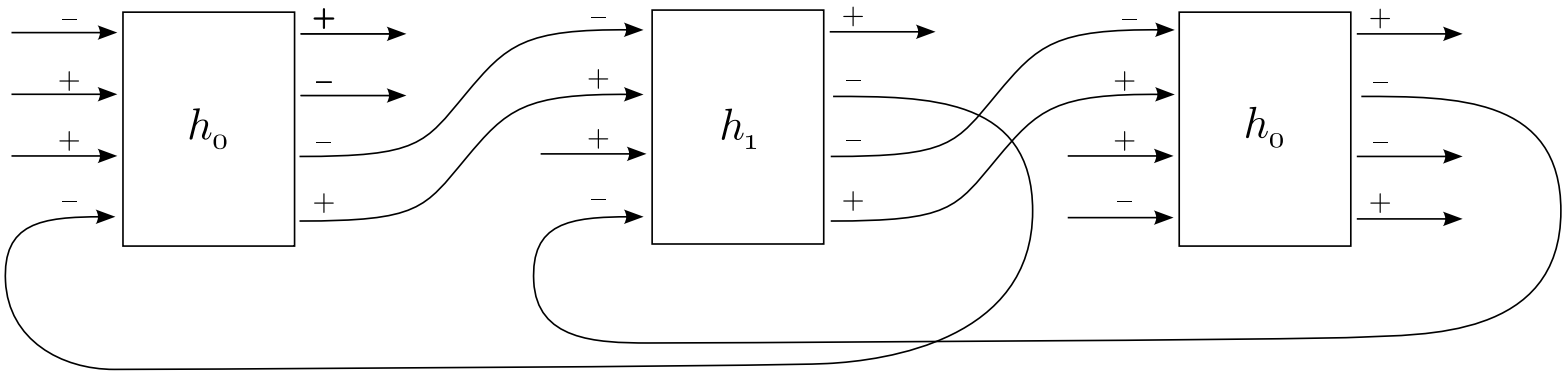
Addition is naturally defined by recursion with result type $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

$$add(\varepsilon) = \lambda y.\, \lambda c.\, \text{if } c \text{ then } inc(y) \text{ else } y$$
$$add(xi) = \lambda y.\, \lambda c.\, s_{(i+y_0+c_0) \bmod 2}(add(x)\ \mathrm{p}(y)\ \lfloor (i + y_0 + c_0)/2 \rfloor)$$

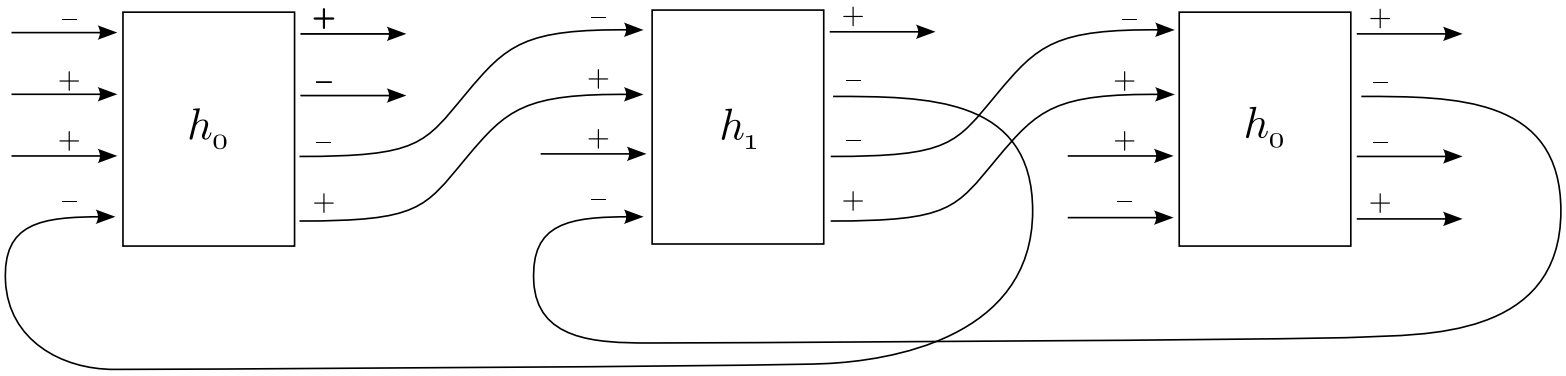
Simplest case

$$g : \mathbb{N} \multimap \mathbb{N} \qquad f = \mathrm{rec}(g, h_0, h_1)$$
$$h_i : (\mathbb{N} \multimap \mathbb{N}) \multimap (\mathbb{N} \multimap \mathbb{N}) \qquad f\ \varepsilon = g$$
$$f : \mathbb{N} \to \mathbb{N} \multimap \mathbb{N} \qquad f\ xi = h_i\ (f\ x)$$

Recursively defined functions



Recursively defined functions

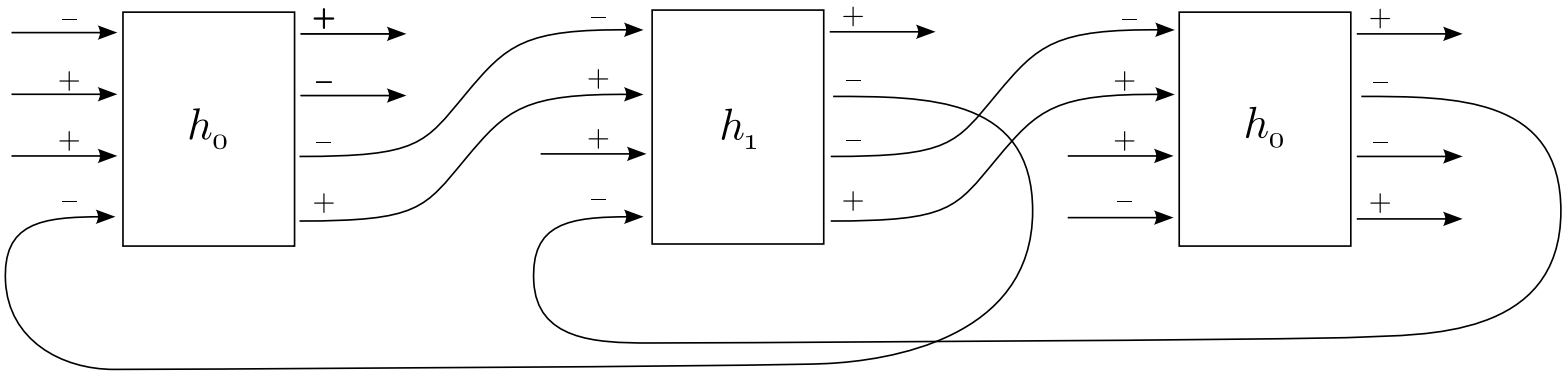


Exploit *linearity* of h_0 and h_1 [e.g. Hofmann 1997].

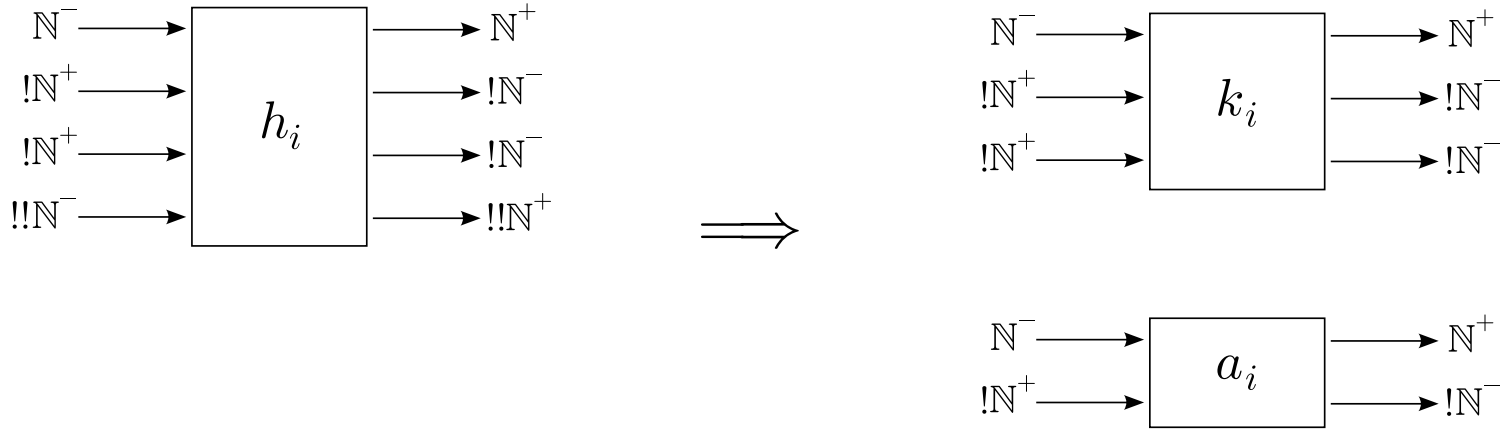
A linear function $h_i : !(!\mathbb{N} \multimap \mathbb{N}) \multimap (!\mathbb{N} \multimap \mathbb{N})$ decomposes in $k_i : !\mathbb{N} \multimap !\mathbb{N} \multimap \mathbb{N}$ and $a_i : !\mathbb{N} \multimap \mathbb{N}$.

$$(h_i\ f)\ n = k_i\ n\ (f\ (a_i\ n))$$

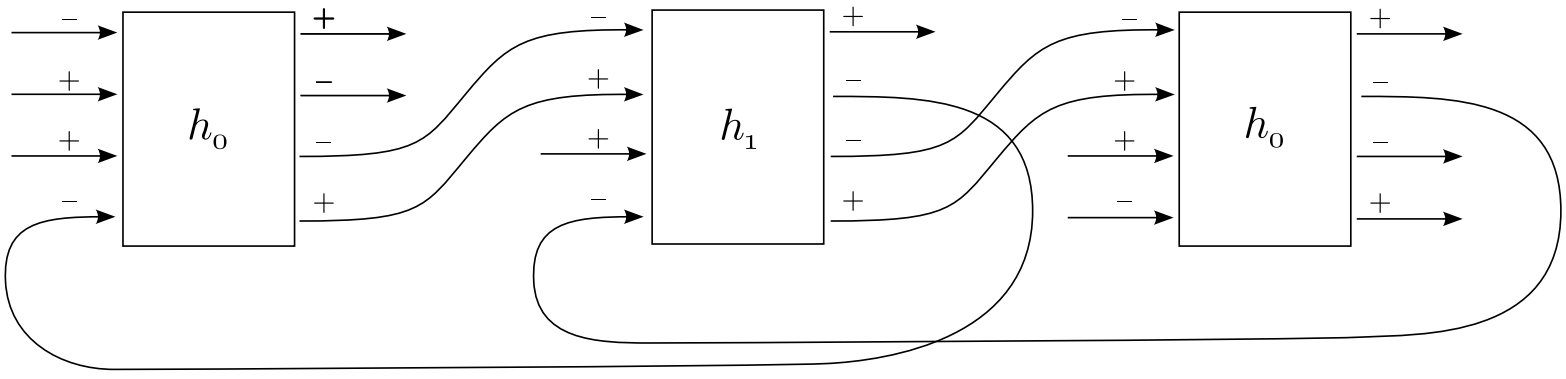
Recursively defined functions



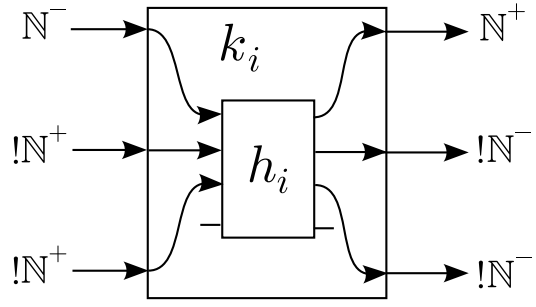
A linear function $h_i : !(!\mathbb{N} \multimap \mathbb{N}) \multimap (!\mathbb{N} \multimap \mathbb{N})$ decomposes in $k_i : !\mathbb{N} \multimap !\mathbb{N} \multimap \mathbb{N}$ and $a_i : !\mathbb{N} \multimap \mathbb{N}$.

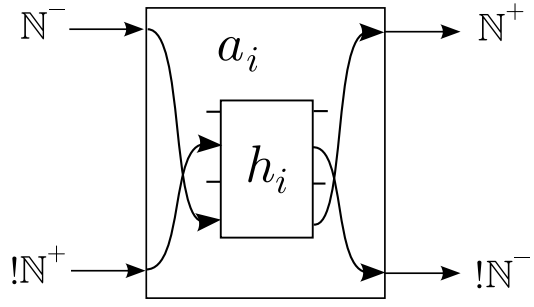


Recursively defined functions

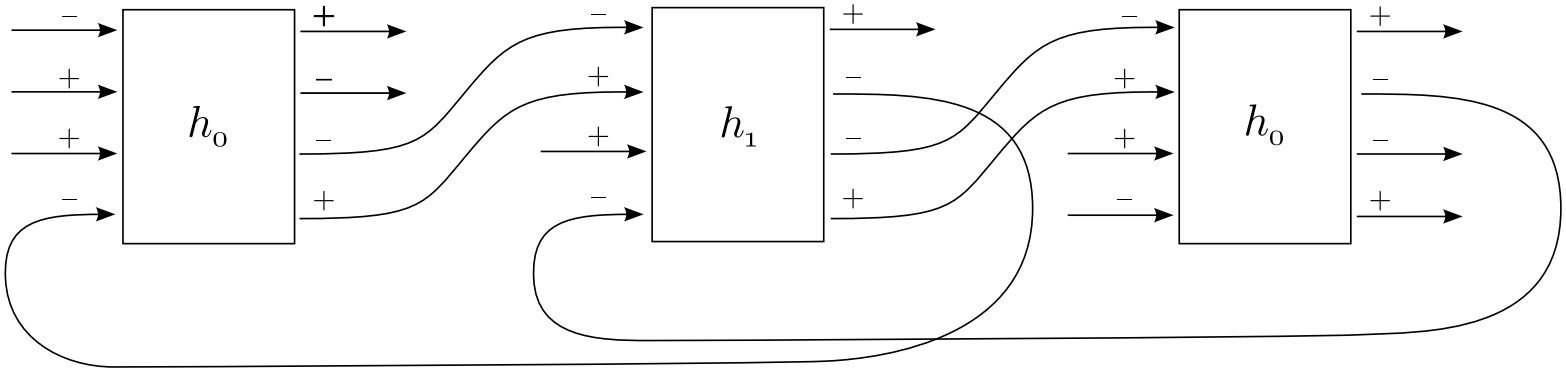


A linear function $h_i : !(!\mathbb{N} \multimap \mathbb{N}) \multimap (!\mathbb{N} \multimap \mathbb{N})$ decomposes in $k_i : !\mathbb{N} \multimap !\mathbb{N} \multimap \mathbb{N}$ and $a_i : !\mathbb{N} \multimap \mathbb{N}$.



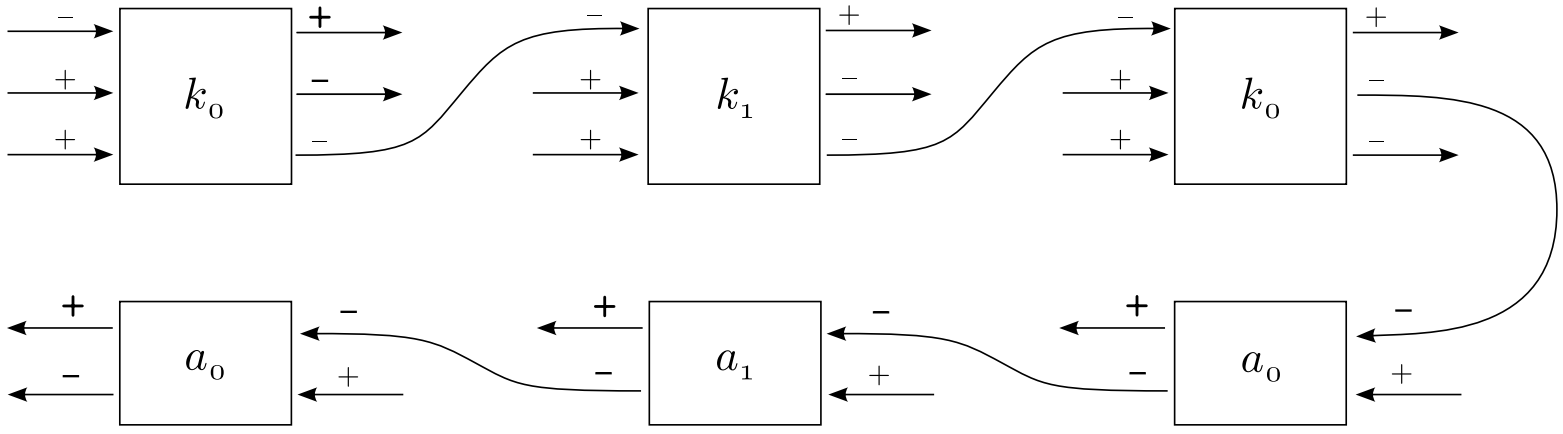


Recursively defined functions



A linear function $h_i : !(!\mathbb{N} \multimap \mathbb{N}) \multimap (!\mathbb{N} \multimap \mathbb{N})$ decomposes in $k_i : !\mathbb{N} \multimap !\mathbb{N} \multimap \mathbb{N}$ and $a_i : !\mathbb{N} \multimap \mathbb{N}$.

$\Rightarrow$ Recursion decomposes in two nested recursions on $\mathbb{N}$



Recursively defined functions — Example

Write

$$f(\varepsilon) = \lambda y.\, \varepsilon$$
$$f(xi) = \lambda y.\, \text{if } y \text{ then } s_i(f(x)\ 0) \text{ else } (f(x)\ 1)$$

as $f(x) = \lambda y.\, f'(x, \varepsilon;\ y)$.

$$a_i(\varepsilon;\ y) = y$$
$$a_i(x_r i;\ y) = \text{if } a_i(x_r;\ y) \text{ then } 0 \text{ else } 1$$

$$f'_i(\varepsilon, x_r;\ y) = \varepsilon$$
$$f'_i(xi, x_r;\ y) = \text{if } a_i(x_r;\ y) \text{ then } s_i(f(x, ix_r;\ y)) \text{ else } (f(x, ix_r;\ y))$$

Towards an intrinsic model for FL

Instead of establishing resource bounds afterwards,
we want a model that captures FL intrinsically.

Size-restricted Geometry of Interaction

Restrict to morphisms $f : B^- + A^+ \longrightarrow B^+ + A^-$ that are

- non-size-increasing and
- computable in linear space.

Realizability

Restrict to functions that are realised by size-restricted GoI-morphisms.

Realizability

Objects

Set $|A|$, size-restricted GoI object $\|A\|$, realizability relation $\Vdash_A$

$$\langle i,l,k\rangle, e \Vdash_A a \qquad \begin{cases} a \in |A| \\ e\colon \|A\|^- \to \|A\|^+ \\ i,l,k \in \mathbb{N} \end{cases}$$

Example: Natural numbers

$$|\mathbb{N}| = \mathbb{N} \qquad \|\mathbb{N}\| = (\mathbf{L}(\mathbf{B}) \otimes \mathbf{L}(\Diamond),\ \{*,0,1\} \otimes \mathbf{L}(\Diamond))$$

$\langle i,l,k\rangle, e \Vdash n$ iff

- $l \geq |n|$
- On question $\langle b, \Diamond^{k\cdot\log l}\rangle$, the function e returns Bit b of n and gives back the memory

Realizability

Morphisms

Morphism $f : A \to B$ is a function $f : |A| \to |B|$, for which there exist $r : !\|A\| \to \|B\|$ and φ, such that

$$\alpha, e \Vdash x \implies \alpha + \varphi, r \cdot !e \Vdash f(x)$$

holds.
(plus a condition restricting the use of the !-modality)

Proposition. The underlying function of every morphism $f : \mathbb{N} \to \mathbb{N}$ is FL-computable.

Realizability — Work in progress

Structure and constructions

- Monoidal closure: $\otimes$, $\multimap$
- Question-linearity and modality $\Box$
- Interpretation of BC_{ε}^{-}

Programming language for FL on this basis

- Resource type $\Diamond$ for size-control
- Modality $\Box$ to control question-linearity