Towards Intensionally More Expressive Systems for PTime

Stefan Schimanski

Department of Mathematics
Ludwig-Maximilians-Universität München

Types 2006

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ⋆ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ⋆ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ⋆ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ⋆ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ⋆ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ⋆ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ⋆ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ★ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

Motivation

Aim: study extensions of well-known PTime systems LFPL and BC to increase intensional expressiveness

- by relaxing linearity
- by combining different recursion schemes into one system
- by syntactical methods
  - finding decreasing measures for most general reduction sequences
  - considering sharing normalisation techniques
  - pointing out where special reduction strategies are essential
    - ★ Folklore that BC needs special reduction strategy. What happens if we use proper sharing? Weiermann and Beckmann's example breaks down:

$$\begin{aligned} g(;) &= 3 \\ h(x;y) &= c(;y,y,y) \\ f(n;) &= \mathrm{rec}(g,h,h)(n;) \end{aligned}$$

First Example: linearity in LFPL

- LFPL = Hofmann's non-size-increasing term system
  - (affine) linearly typed
  - special $\Diamond$ type, seen as money
  - $\Diamond$ to be payed for list constructors
  - $\Diamond$ cannot be created out of nothing
  - $\Rightarrow$ amount of $\Diamond$ money doesn't increase during normalization
  - non-size-increasing iteration (list {step} base)
- Exactly LinSpace PTime algorithms representable

Hofmann M. : Linear Types and Non-Size-Increasing Polynomial Time Computation. Logic in Computer Science (1998)

Types

$$\sigma, \tau ::= \Diamond \mid B \mid \sigma \multimap \tau \mid \sigma \otimes \tau \mid \sigma \times \tau \mid L(\sigma)$$

Terms

$$s, t ::= x^{\tau} \mid c \mid \lambda x^{\tau}.t \mid \langle t, s \rangle \mid (t\ s) \mid \{t\}$$

Constructors

$\mathbf{tt}\ \mathbf{ff}$	B
$\mathbf{cons}_{\tau}$	$\Diamond \multimap \tau \multimap L(\tau) \multimap L(\tau)$
$\mathbf{nil}_{\tau}$	$L(\tau)$
$\otimes_{\sigma,\tau}$	$\sigma \multimap \tau \multimap \sigma \otimes \tau$

Complexity

Theorem (Aehlig, Schwichtenberg)

For any typed term t there is a polynomial $\vartheta(t)$ such that the length of a (special) reduction sequence is bounded by $\vartheta(t)(|FV(t)|)$.

Proof.

- explicit definition of $\vartheta(t)$ by recursion on t.
- $\vartheta(t)(N)$ decreases every conversion step...
  if $\mathcal{L}(l) \leq N$ for every occurring list l
- linear typing $\Rightarrow$ $|FV(t)|$ doesn't increase
- and $\mathcal{L}(l) \leq |FV(t)|$.

Restricted non-linearity in LFPL

- Insertion Sort

$$\begin{aligned}
&\mathsf{insert}(a, []) = [a] \\
&\mathsf{insert}(a, b :: l) = \mathsf{if}\ a \leq b\ \mathsf{then}\ a :: b :: l\ \mathsf{else}\ b : \mathsf{insert}(a, l) \\
&\mathsf{sort}([]) = [] \\
&\mathsf{sort}(a :: l) = \mathsf{insert}(a, \mathsf{sort}(l))
\end{aligned}$$

- *Not* linear:

$$\mathsf{if}\ p(x)\ \mathsf{then}\ f(x)\ \mathsf{else}\ g(x)$$

- Intuition suggests: $p^{\tau \to B} \in PTIME$ does not harm

Restricted non-linearity in LFPL

- Insertion Sort

$$\begin{aligned}
&\mathsf{insert}(a, []) = [a] \\
&\mathsf{insert}(a, b :: l) = \mathsf{if}\ a \leq b\ \mathsf{then}\ a :: b :: l\ \mathsf{else}\ b : \mathsf{insert}(a, l) \\
&\mathsf{sort}([]) = [] \\
&\mathsf{sort}(a :: l) = \mathsf{insert}(a, \mathsf{sort}(l))
\end{aligned}$$

- *Not* linear:

$$\mathsf{if}\ p(x)\ \mathsf{then}\ f(x)\ \mathsf{else}\ g(x)$$

- Intuition suggests: $p^{\tau \to B} \in PTIME$ does not harm

Naive extension

- Operator δp of type $\sigma \longrightarrow (B \otimes \sigma)$ for $p^{\sigma \longrightarrow B}$.
- Conversion $(\delta p\ s) \mapsto (p\ s) \otimes s$
- Then

$$\text{if } p(x) \text{ then } f(x) \text{ else } g(x)$$

means:

$$((\delta p\ x)\ \lambda y, z.(y\ \langle (f\ z), (g\ z)\rangle))$$

- This **destroys linearity** $\Rightarrow$ only occurrences of variables count, not their names
- **no easy measure for size anymore** by counting variables, or even occurrences

Naive extension

- Operator δp of type $\sigma \longrightarrow (B \otimes \sigma)$ for $p^{\sigma \longrightarrow B}$.
- Conversion $(\delta p\ s) \mapsto (p\ s) \otimes s$
- Then

$$\text{if } p(x) \text{ then } f(x) \text{ else } g(x)$$

means:

$$((\delta p\ x)\ \lambda y, z.(y\ \langle (f\ z), (g\ z)\rangle))$$

- This **destroys linearity** $\Rightarrow$ only occurrences of variables count, not their names
- **no easy measure for size anymore** by counting variables, or even occurrences

Naive extension

- Operator δp of type $\sigma \longrightarrow (B \otimes \sigma)$ for $p^{\sigma \longrightarrow B}$.
- Conversion $(\delta p\ s) \mapsto (p\ s) \otimes s$
- Then

$$\text{if } p(x) \text{ then } f(x) \text{ else } g(x)$$

means:

$$((\delta p\ x)\ \lambda y, z.(y\ \langle (f\ z), (g\ z)\rangle))$$

- This **destroys linearity** $\Rightarrow$ only occurrences of variables count, not their names
- **no easy measure for size anymore** by counting variables, or even occurrences

Naive extension

- Operator δp of type $\sigma \to (B \otimes \sigma)$ for $p^{\sigma \to B}$.
- Conversion $(\delta p\ s) \mapsto (p\ s) \otimes s$
- Then

$$\text{if } p(x) \text{ then } f(x) \text{ else } g(x)$$

means:

$$((\delta p\ x)\ \lambda y, z.(y\ \langle (f\ z), (g\ z)\rangle))$$

- This **destroys linearity** $\Rightarrow$ only occurrences of variables count, not their names
- **no easy measure for size anymore** by counting variables, or even occurrences

Terms

$$s, t ::= x^\tau \mid c \mid \lambda x^\tau.t \mid \langle t, s \rangle \mid (t\ s) \mid \{t\} \mid \delta f$$

Conversions

$$\begin{array}{rcl}
(\lambda x^\tau.t\ s) & \mapsto & t[s/x] \\
(s \otimes t\ r) & \mapsto & ((r\ s)\ t) \\
((\mathbf{cons}\ d\ v\ x)\ \{h\}\ g) & \mapsto & (h\ d\ v\ (x\ \{h\}\ g)) \\
(\mathbf{nil}\ \{h\}\ g) & \mapsto & g \\
(\mathbf{tt}\ \langle s, t \rangle) & \mapsto & s \\
(\mathbf{ff}\ \langle s, t \rangle) & \mapsto & t \\
(\delta f\ t) & \mapsto & (f\ t) \otimes t
\end{array}$$

Quasi-linear typing rules

$$\frac{}{\Gamma, x^\tau \quad \vdash x^\tau}\,(\text{Var}) \qquad \frac{c \text{ of type } \tau}{\Gamma \quad \vdash c^\tau}\,(\text{Const})$$

$$\frac{\Gamma \quad , x^\sigma \vdash t^\tau}{\Gamma \quad \vdash (\lambda x^\sigma.t)^{\sigma \multimap \tau}}\,(\multimap^+) \qquad \frac{\Lambda_1 \vdash t^{\sigma \multimap \tau} \qquad \Lambda_2 \vdash s^\sigma}{\Lambda_1, \Lambda_2 \vdash (t\ s)^\tau}\,(\multimap^-)$$

$$\frac{\Lambda_1 \vdash t^{\rho \otimes \tau} \qquad \Lambda_2 \vdash s^{\rho \multimap \tau \multimap \sigma}}{\Lambda_1, \Lambda_2 \vdash (t\ s)^\sigma}\,(\otimes^-)$$

$$\frac{\Lambda \vdash t^{L(\tau)} \qquad \emptyset \vdash h^{\Diamond \multimap \tau \multimap \sigma \multimap \sigma}}{\Lambda \vdash (t\ \{h\})^{\sigma \multimap \sigma}}\,(L(\tau)^-)$$

$$\frac{\Gamma \quad \vdash f^{\sigma \multimap \tau}}{\Gamma \quad \vdash \delta f^{\sigma \multimap \tau \otimes \sigma}}\,(\delta^+)$$

Quasi-linear typing rules

$$\frac{}{\Gamma, x^\tau; \Lambda \vdash x^\tau}\,(\text{Var}) \qquad \frac{c \text{ of type } \tau}{\Gamma; \Lambda \vdash c^\tau}\,(\text{Const})$$

$$\frac{\Gamma; \Lambda, x^\sigma \vdash t^\tau}{\Gamma; \Lambda \vdash (\lambda x^\sigma.t)^{\sigma \multimap \tau}}\,(\multimap^+) \qquad \frac{\Gamma;\Lambda_1 \vdash t^{\sigma \multimap \tau} \qquad \Gamma;\Lambda_2 \vdash s^\sigma}{\Gamma;\Lambda_1, \Lambda_2 \vdash (t\ s)^\tau}\,(\multimap^-)$$

$$\frac{\Gamma;\Lambda_1 \vdash t^{\rho \otimes \tau} \qquad \Gamma;\Lambda_2 \vdash s^{\rho \multimap \tau \multimap \sigma}}{\Gamma;\Lambda_1, \Lambda_2 \vdash (t\ s)^\sigma}\,(\otimes^-)$$

$$\frac{\Gamma;\Lambda \vdash t^{L(\tau)} \qquad \emptyset;\emptyset \vdash h^{\Diamond \multimap \tau \multimap \sigma \multimap \sigma}}{\Gamma;\Lambda \vdash (t\ \{h\})^{\sigma \multimap \sigma}}\,(L(\tau)^-)$$

$$\frac{\Gamma; \Lambda \vdash f^{\sigma \multimap \tau}}{\Gamma; \Lambda \vdash \delta f^{\sigma \multimap \tau \otimes \sigma}}\,(\delta^+)$$

Quasi-linear typing rules

$$\frac{}{\Gamma, x^\tau; \Lambda \vdash x^\tau}\,(\mathrm{Var}) \qquad \frac{c \text{ of type } \tau}{\Gamma; \Lambda \vdash c^\tau}\,(\mathrm{Const}) \qquad \frac{}{\Gamma; \Lambda, x^\tau \vdash x^\tau}\,(\mathrm{Var}_l)$$

$$\frac{\Gamma; \Lambda, x^\sigma \vdash t^\tau}{\Gamma; \Lambda \vdash (\lambda x^\sigma.t)^{\sigma \multimap \tau}}\,(\multimap^+) \qquad \frac{\Gamma;\Lambda_1 \vdash t^{\sigma \multimap \tau} \qquad \Gamma;\Lambda_2 \vdash s^\sigma}{\Gamma;\Lambda_1, \Lambda_2 \vdash (t\ s)^\tau}\,(\multimap^-)$$

$$\frac{\Gamma;\Lambda_1 \vdash t^{\rho \otimes \tau} \qquad \Gamma;\Lambda_2 \vdash s^{\rho \multimap \tau \multimap \sigma}}{\Gamma;\Lambda_1, \Lambda_2 \vdash (t\ s)^\sigma}\,(\otimes^-)$$

$$\frac{\Gamma;\Lambda \vdash t^{L(\tau)} \qquad \emptyset;\emptyset \vdash h^{\Diamond \multimap \tau \multimap \sigma \multimap \sigma}}{\Gamma;\Lambda \vdash (t\ \{h\})^{\sigma \multimap \sigma}}\,(L(\tau)^-)$$

$$\frac{\Gamma; \Lambda \vdash f^{\sigma \multimap \tau}}{\Gamma; \Lambda \vdash \delta f^{\sigma \multimap \tau \otimes \sigma}}\,(\delta^+)$$

Reduction strategy matters

- Assume $p = \lambda x^{B}.\mathbf{tt}$
- and $t = \lambda l^{L(\tau)}.(l\ \{\lambda\Diamond, v, x^{B}.((\delta p\ x)\ \lambda y, z.y)\}\ \mathbf{tt})$.
- Then the following reduction sequence is possible:

$$(t\ (cons\ \Diamond_1\ d_1\ \ldots (cons\ \Diamond_n\ d_n\ nil)))$$
$$\rightarrow \ldots \rightarrow \underbrace{(\delta p\ (\delta p\ \ldots \mathbf{tt}).1).1}_{\text{depth } n}$$

- Applying all n δp conversions leads to $2^{(n-1)}$ sub-terms of the form $(\lambda x^{B}.\mathbf{tt}\ t)$, i.e. an **exponential complexity**.

Reduction strategy matters

- Assume $p = \lambda x^B.\mathbf{tt}$
- and $t = \lambda l^{L(\tau)}.(l\ \{\lambda\Diamond, v, x^B.((\delta p\ x)\ \lambda y, z.y)\}\ \mathbf{tt})$.
- Then the following reduction sequence is possible:

$$(t\ (cons\ \Diamond_1\ d_1\ \ldots (cons\ \Diamond_n\ d_n\ nil))) \\ \rightarrow \ldots \rightarrow \underbrace{(\delta p\ (\delta p\ \ldots \mathbf{tt}).1).1}_{\text{depth } n}$$

- Applying all n δp conversions leads to $2^{(n-1)}$ sub-terms of the form $(\lambda x^B.\mathbf{tt}\ t)$, i.e. an **exponential complexity**.

Reduction strategy matters

- Assume $p = \lambda x^{B}.\mathbf{tt}$
- and $t = \lambda l^{L(\tau)}.(l\ \{\lambda\Diamond, v, x^{B}.((\delta p\ x)\ \lambda y, z.y)\}\ \mathbf{tt})$.
- Then the following reduction sequence is possible:

$$(t\ (cons\ \Diamond_1\ d_1\ \ldots (cons\ \Diamond_n\ d_n\ nil))) \\ \rightarrow \ldots \rightarrow \underbrace{(\delta p\ (\delta p\ \ldots \mathbf{tt}).1).1}_{\text{depth } n}$$

- Applying all n δp conversions leads to $2^{(n-1)}$ sub-terms of the form $(\lambda x^{B}.\mathbf{tt}\ t)$, i.e. an **exponential complexity**.

"Healthy" reduction strategy

- Reason for exponential growth: duplication of remaining "work"
- "one δp after the other"
- $\Rightarrow$ Only **convert δp if the argument is in normal form**
- or **using sharing** no special reduction strategy at all

"Healthy" reduction strategy

- Reason for exponential growth: duplication of remaining "work"
- "one δp after the other"
- $\Rightarrow$ Only **convert δp if the argument is in normal form**
- or **using sharing** no special reduction strategy at all

"Healthy" reduction strategy

- Reason for exponential growth: duplication of remaining "work"
- "one δp after the other"
- $\Rightarrow$ Only **convert δp if the argument is in normal form**
- or **using sharing** no special reduction strategy at all

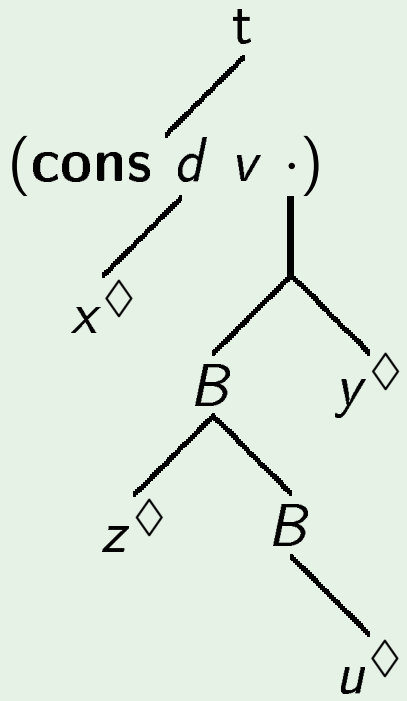
Syntactical analysis

- Inductive predicates to render **interaction of variables**
  - $c_t(s)$ smallest "passive" super-term or t itself
  - $v \succ_t x$ between subterms $v \trianglelefteq t$ and free variables $x \in FV(t)$
  - $x \prec\!\succ_t y$ between $x, y \in FV(t)$

1. $c_t(x) \succ_t x$
2. $z \prec\!\succ_s x \wedge \lambda z^\tau.s \trianglelefteq t \rightarrow c_t(\lambda z^\tau.s) \succ_t x$
3. $\exists c(c \succ_t x \wedge c \succ_t y) \rightarrow x \prec\!\succ_t y.$

Interacting variables

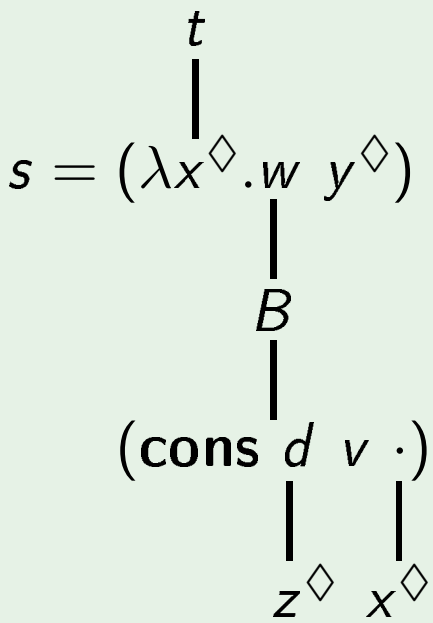
Example



- x and y interact.
- z and u do not interact.

Interacting variables through λ

Example



Syntactical analysis

- Inductive predicates to render **interaction of variables**
  - $c_t(s)$ smallest "passive" super-term or t itself
  - $v \succ_t x$ between subterms $v \trianglelefteq t$ and free variables $x \in FV(t)$
  - $x \prec\succ_t y$ between $x, y \in FV(t)$

  1. $c_t(x) \succ_t x$
  2. $z \prec\succ_s x \wedge \lambda z^\tau.s \trianglelefteq t \rightarrow c_t(\lambda z^\tau.s) \succ_t x$
  3. $\exists c(c \succ_t x \wedge c \succ_t y) \rightarrow x \prec\succ_t y.$

- Induces **equivalence relation** over variables which reflects non-linearity
- every list resides completely in one class
- **classes do not grow**, only new classes are created
- new size measure: size of those equivalence classes
- $\Rightarrow$ strong normalisation with sharing in PTime

Syntactical analysis

- Inductive predicates to render **interaction of variables**
  - $c_t(s)$ smallest "passive" super-term or t itself
  - $v \succ_t x$ between subterms $v \trianglelefteq t$ and free variables $x \in FV(t)$
  - $x \prec\!\succ_t y$ between $x, y \in FV(t)$

  1. $c_t(x) \succ_t x$
  2. $z \prec\!\succ_s x \wedge \lambda z^\tau.s \trianglelefteq t \rightarrow c_t(\lambda z^\tau.s) \succ_t x$
  3. $\exists c(c \succ_t x \wedge c \succ_t y) \rightarrow x \prec\!\succ_t y.$

- Induces **equivalence relation** over variables which reflects non-linearity
- every list resides completely in one class
- **classes do not grow**, only new classes are created
- new size measure: size of those equivalence classes
- $\Rightarrow$ strong normalisation with sharing in PTime

Syntactical analysis

- Inductive predicates to render **interaction of variables**
  - $c_t(s)$ smallest "passive" super-term or t itself
  - $v \succ_t x$ between subterms $v \trianglelefteq t$ and free variables $x \in FV(t)$
  - $x \prec\!\succ_t y$ between $x, y \in FV(t)$

  1. $c_t(x) \succ_t x$
  2. $z \prec\!\succ_s x \wedge \lambda z^\tau.s \trianglelefteq t \rightarrow c_t(\lambda z^\tau.s) \succ_t x$
  3. $\exists c(c \succ_t x \wedge c \succ_t y) \rightarrow x \prec\!\succ_t y.$

- Induces **equivalence relation** over variables which reflects non-linearity
- every list resides completely in one class
- **classes do not grow**, only new classes are created
- new size measure: size of those equivalence classes
- $\Rightarrow$ strong normalisation with sharing in PTime

Second Example: non-size-increasing iteration and BC

- LFPL originally developed to inject new base functions into BC
- What happens when adding nsi-It to BC directly?
- i.e. nsi-Iteration over incomplete/safe terms
- Naive:

$$
\begin{aligned}
\text{cat} &= \lambda a, b.It(a)(\lambda x, \Diamond, t, y.(\textbf{cons}\ x\ \Diamond\ y))b \\
\exp_1 &= \lambda a.Rec((\textbf{cons}\ 1\ \Diamond\ \textbf{nil}))(\lambda x, t, y.(\text{cat}\ y\ y))a \\
\exp_2 &= \lambda a.Rec(\lambda z.(z, z)(\textbf{cons}\ 1\ \Diamond\ \textbf{nil})) \\
&\quad (\lambda x, t, y.(\lambda z.(z, z)\ (\text{cat}\ (\mathit{fst}\ y)\ (\mathit{snd}\ y))))a
\end{aligned}
$$

  $\Rightarrow$ in conflict with duplication of safe variables
- **destroys sharing and separation** of safe lists
- more precise analysis (e.g. using GoI) can characterise sane cases

Second Example: non-size-increasing iteration and BC

- LFPL originally developed to inject new base functions into BC
- What happens when adding nsi-It to BC directly?
- i.e. nsi-Iteration over incomplete/safe terms
- Naive:

$$
\begin{aligned}
\text{cat} &= \lambda a, b.It(a)(\lambda x, \Diamond, t, y.(\mathbf{cons}\ x\ \Diamond\ y))b \\
\text{exp}_1 &= \lambda a.Rec((\mathbf{cons}\ 1\ \Diamond\ \mathbf{nil}))(\lambda x, t, y.(\text{cat}\ y\ y))a \\
\text{exp}_2 &= \lambda a.Rec(\lambda z.(z, z)(\mathbf{cons}\ 1\ \Diamond\ \mathbf{nil})) \\
&\quad (\lambda x, t, y.(\lambda z.(z, z)\ (\text{cat}\ (\mathit{fst}\ y)\ (\mathit{snd}\ y))))a
\end{aligned}
$$

  $\Rightarrow$ in conflict with duplication of safe variables

- **destroys sharing and separation** of safe lists
- more precise analysis (e.g. using GoI) can characterise sane cases

Second Example: non-size-increasing iteration and BC

- LFPL originally developed to inject new base functions into BC
- What happens when adding nsi-It to BC directly?
- i.e. nsi-Iteration over incomplete/safe terms
- Naive:

$$
\begin{aligned}
\text{cat} &= \lambda a, b.\mathit{It}(a)(\lambda x, \Diamond, t, y.(\mathbf{cons}\ x\ \Diamond\ y))b \\
\text{exp}_1 &= \lambda a.\mathit{Rec}((\mathbf{cons}\ 1\ \Diamond\ \mathbf{nil}))(\lambda x, t, y.(\text{cat}\ y\ y))a \\
\text{exp}_2 &= \lambda a.\mathit{Rec}(\lambda z.(z, z)(\mathbf{cons}\ 1\ \Diamond\ \mathbf{nil})) \\
&\quad (\lambda x, t, y.(\lambda z.(z, z)\ (\text{cat}\ (\mathit{fst}\ y)\ (\mathit{snd}\ y))))a
\end{aligned}
$$

$\Rightarrow$ in conflict with duplication of safe variables

- **destroys sharing and separation** of safe lists
- more precise analysis (e.g. using GoI) can characterise sane cases

Second Example: non-size-increasing iteration and BC

- LFPL originally developed to inject new base functions into BC
- What happens when adding nsi-It to BC directly?
- i.e. nsi-Iteration over incomplete/safe terms
- Naive:

$$
\begin{aligned}
\text{cat} &= \lambda a, b.It(a)(\lambda x, \Diamond, t, y.(\textbf{cons}\ x\ \Diamond\ y))b \\
\text{exp}_1 &= \lambda a.Rec((\textbf{cons}\ 1\ \Diamond\ \textbf{nil}))(\lambda x, t, y.(\text{cat}\ y\ y))a \\
\text{exp}_2 &= \lambda a.Rec(\lambda z.(z, z)(\textbf{cons}\ 1\ \Diamond\ \textbf{nil})) \\
&\quad\ (\lambda x, t, y.(\lambda z.(z, z)\ (\text{cat}\ (\mathit{fst}\ y)\ (\mathit{snd}\ y))))a
\end{aligned}
$$

  $\Rightarrow$ in conflict with duplication of safe variables

- **destroys sharing and separation** of safe lists
- more precise analysis (e.g. using GoI) can characterise sane cases

Conclusion

- Known systems can be extended
  ... but **syntactical methods** get **much more complicated**.
- Forcing reduction strategies is a tool to avoid "bad" behaviour
  ... though often **sharing can remove those constraints**.
- **Hence:** How essential are sharing or reduction strategies?
  LLL does not need that, more careful about duplication.
- **Combination of recursion schemes** not explored very much yet
- But fruitful to better understand dynamics of normalisation in PTime systems
  ... e.g. importance of separation of safe data in BC.