Termination and Guardedness Checking with Continuous Types

Towards a Higher-Order Polymorphic Lambda-Calculus With Sized Types

Andreas Abel

Slide 1

Graduiertenkolleg Logik in der Informatik
Munich, Germany
May 30, 2003

— Work in progress —

Work supported by: GKLI (DFG), TYPES and APPSEM-II

Setting the stage...

- Curry-Howard-Isomorphism:
  proofs by induction = programs with recursion
- Only *terminating* programs constitute valid proofs.
- Design issue: How to integrate terminating recursion into proof/programming language?

Slide 2

One approach: special forms of recursion

Slide 3

- Tame recursion by restricting to special patterns.
- Iteration/catamorphisms
  e.g. Haskell's `List.fold`
- Primitive recursion/paramorphisms
- Problems:
  - Non-trivial operational semantics makes it harder to understand programs.
  - I do not want to write all of my list-processing functions using `fold`.

Another approach: recursion with termination checking

Slide 4

- Use *general recursion*: `letrec`.
- Has "intuitive" meaning through simple operational semantics.
- In general not normalizing, need termination checking.
- Here we used the *sized types* approach [Hughes et al. 1996] [Barthe et al. 2003?].
- View data as trees.
- *Size* = height = # constructors in longest path of tree.
- Height of input data must decrease in each recursive call.
- Termination is ensured by type-checker.

Sized types in a nutshell

Slide 5

- Sizes are *upper bounds*.
- List^a denotes lists of length $< a$.
- List^∞ denotes list of arbitrary (but finite) length.
- Sizes induce *subtyping*: $\mathsf{List}^a \leq \mathsf{List}^b$ if $a \leq b$.
- In general, sizes are *ordinal numbers*, needed e.g. for infinitely branching trees.
- Size expressions:

$$\begin{array}{lcll} a & ::= & i & \text{variable} \\ & | & a+1 & \text{sucessor} \\ & | & \infty & \text{ultimate limit, denoting } \Omega \text{ (first uncountable)} \end{array}$$

Example: list splitting

Slide 6

$$\begin{array}{lll} \mathsf{split}: & \forall A{:}*.\ \mathsf{List}\ A \to \mathsf{List}\ A \times \mathsf{List}\ A & \\ \mathsf{split}\ [\,] & = \langle [\,] & ,[\,] \quad \rangle \\ \mathsf{split}\ (x :: k\ ) & = \mathsf{case}\ k & \mathsf{of} \\ & \quad [\,] & \to \langle (x :: k) \quad ,[\,] \quad \rangle \\ & \mid (y :: l\ ) & \to \mathsf{let}\ \langle xs\ , ys\ \rangle = \mathsf{split}\, l\ \ \mathsf{in} \\ & & \quad \langle (x :: xs) \quad , (y :: ys) \quad \rangle \end{array}$$

- Sized types allow us to express that split denotes a non-size increasing function.

Example: list splitting

Slide 7

$$
\begin{array}{l}
\mathsf{split} : \forall i{:}\mathsf{ord}.\, \forall A{:}*.\ \mathsf{List}^i A \to \mathsf{List}\ A \times \mathsf{List}\ A \\
\mathsf{split}\ [\,] \qquad\quad = \langle [\,] \quad , [\,] \quad \rangle \\
\mathsf{split}\ (x :: k^i)^{i+1} = \mathsf{case}\ k^{i \leq i+1}\ \mathsf{of} \\
\qquad\qquad\qquad\qquad\quad [\,] \qquad \to \langle (x :: k) \quad , [\,] \quad \rangle \\
\qquad\qquad\qquad\quad \mid (y :: l^i) \to \mathsf{let}\ \langle xs\ , ys\ \rangle = \mathsf{split}\, l^i\ \mathsf{in} \\
\qquad\qquad\qquad\qquad\qquad\quad \langle (x :: xs) \quad , (y :: ys) \quad \rangle
\end{array}
$$

- To compute split at stage $i+1$, split is only used at stage i.
- Hence, split is terminating.

Example: list splitting

Slide 8

$$
\begin{array}{l}
\mathsf{split} : \forall i{:}\mathsf{ord}.\, \forall A{:}*.\ \mathsf{List}^i A \to \mathsf{List}^i A \times \mathsf{List}^i A \\
\mathsf{split}\ [\,]^{i+1} \qquad\quad = \langle [\,]^{i+1}, [\,]^{i+1} \rangle \\
\mathsf{split}\ (x :: k^i)^{i+1} = \mathsf{case}\ k^{i \leq i+1}\ \mathsf{of} \\
\qquad\qquad\qquad\qquad\quad [\,]^{i+1} \quad \to \langle (x :: k)^{i+1}, [\,]^{i+1} \rangle \\
\qquad\qquad\qquad\quad \mid (y :: l^i) \to \mathsf{let}\ \langle xs^i, ys^i \rangle = \mathsf{split}\, l^i\ \mathsf{in} \\
\qquad\qquad\qquad\qquad\qquad\quad \langle (x :: xs)^{i+1}, (y :: ys)^{i+1} \rangle
\end{array}
$$

- We additionally can infer that split is non-size increasing.
- Using split, we can define merge sort...

Example: merge sort

Slide 9

$$
\begin{array}{l}
\mathsf{merge}: \quad \mathsf{List}\ \mathsf{Int} \to \quad \mathsf{List}\ \mathsf{Int} \to \mathsf{List}\ \ \mathsf{Int} \\
\\
\mathsf{msort}: \quad \mathsf{List}\ \mathsf{Int} \to \mathsf{List}\ \ \mathsf{Int} \\
\\
\mathsf{msort}\ [\,] \quad = [\,] \\
\mathsf{msort}\ (x :: k\ ) = \mathsf{case}\ k \quad \mathsf{of} \\
\qquad\qquad\qquad\qquad\qquad\quad [\,] \quad\quad \to x :: [\,] \\
\qquad\qquad\qquad\qquad\quad \mid (y :: l\ ) \to \mathsf{let}\ (xs\ , ys\ ) = \mathsf{split}\, l\ \ \mathsf{in} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{merge}\ (\mathsf{msort}\ (x :: xs) \quad\quad) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (\mathsf{msort}\ (y :: ys) \quad\quad)
\end{array}
$$

Example: merge sort

Slide 10

$$
\begin{array}{l}
\mathsf{merge}: \forall i{:}\mathsf{ord}.\ \mathsf{List}^i\ \mathsf{Int} \to \forall j{:}\mathsf{ord}.\ \mathsf{List}^j\ \mathsf{Int} \to \mathsf{List}^\infty\ \mathsf{Int} \\
\\
\mathsf{msort}: \forall i{:}\mathsf{ord}.\ \mathsf{List}^i\ \mathsf{Int} \to \mathsf{List}^\infty\ \mathsf{Int} \\
\\
\mathsf{msort}\ [\,]^{i+1} \quad = [\,] \\
\mathsf{msort}\ (x :: k^i) = \mathsf{case}\ k^{j+1=i}\ \mathsf{of} \\
\qquad\qquad\qquad\qquad\qquad\quad [\,] \quad\quad \to x :: [\,] \\
\qquad\qquad\qquad\qquad\quad \mid (y :: l^j) \to \mathsf{let}\ (xs^j, ys^j) = \mathsf{split}\, l^j\ \mathsf{in} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{merge}\ (\mathsf{msort}\ (x :: xs)^{j+1=i}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (\mathsf{msort}\ (y :: ys)^{j+1=i})
\end{array}
$$

Lambda-calculus with subtyping

Slide 11

- Types

$$A, B, C ::= 1 \mid A \times B \mid A + B \mid A \to B$$

- Terms

$$\begin{aligned} r, s, t \ ::= \ & \langle\rangle \mid \langle s, t\rangle \mid \mathsf{fst}\ r \mid \mathsf{snd}\ r \\ & \mid \mathsf{inl}\, t \mid \mathsf{inr}\, t \mid \mathsf{case}\, r\ \mathsf{of}\ \mathsf{inl}\, x \Rightarrow s \mid \mathsf{inr}\, y \Rightarrow t \\ & \mid x \mid \lambda x.t \mid r\, s \mid \mathsf{let}\ x = r\ \mathsf{in}\ t \end{aligned}$$

- Subtyping

$$\frac{}{1 \leq 1} \qquad \frac{A_1 \leq A_2 \qquad B_1 \leq B_2}{A_1 \times B_1 \leq A_2 \times B_2} \qquad \frac{A_1 \leq A_2 \qquad B_1 \leq B_2}{A_1 + B_1 \leq A_2 + B_2}$$

$$\frac{A_2 \leq A_1 \qquad B_1 \leq B_2}{A_1 \to B_1 \leq A_2 \to B_2}$$

Polymorphism

Slide 12

- Types

$$A, B, C ::= \ldots \mid X \mid \forall X.\, A$$

- Typing

$$\frac{\Gamma, X \vdash t : A}{\Gamma \vdash t : \forall X.\, A} \qquad \frac{\Gamma \vdash t : \forall X.\, A}{\Gamma \vdash t : [B/X]A}$$

- Subtyping

$$\frac{}{X \leq X} \qquad \frac{\Gamma \vdash [C/X]A \leq B}{\Gamma \vdash (\forall X.A) \leq B} \qquad \frac{\Gamma, X \vdash A \leq B}{\Gamma \vdash A \leq \forall X.B}$$

Size polymorphism

- Sizes

$$a, b, c ::= i \mid a + 1 \mid \infty$$

- Ordering

$$\frac{}{i \leq i} \qquad \frac{a \leq b}{a + 1 \leq b + 1} \qquad \frac{a \leq b}{a \leq b + 1} \qquad \frac{}{\infty \leq \infty}$$

Slide 13

- Types

$$A, B, C ::= \ldots \mid \forall i.\, A$$

- Typing

$$\frac{\Gamma, i \vdash t : A}{\Gamma \vdash t : \forall i.\, A} \qquad \frac{\Gamma \vdash t : \forall i.\, A}{\Gamma \vdash t : [a/i]A}$$

- Subtyping

$$\frac{\Gamma \vdash [a/i]A \leq B}{\Gamma \vdash (\forall i.A) \leq B} \qquad \frac{\Gamma, i \vdash A \leq B}{\Gamma \vdash A \leq \forall i.B}$$

Inductive types

- Types and terms

$$\begin{array}{lcll} A, B, C & ::= & \ldots \mid \mu^a X.\, A & \text{where } X \text{ only pos in } A \\ r, s, t & ::= & \ldots \mid \mathsf{in}\, t \mid \mathsf{out}\, t & \end{array}$$

- Typing

Slide 14

$$\frac{\Gamma \vdash t : [\mu^a X.A/X]A}{\Gamma \vdash \mathsf{in}\, t : \mu^{a+1} X.A} \qquad \frac{\Gamma \vdash r : \mu^{a+1} X.A}{\Gamma \vdash \mathsf{out}\, r : [\mu^a X.A/X]A}$$

- Subtyping

$$\frac{a \leq b \text{ or } b = \infty \qquad \Gamma, X \vdash A \leq B}{\Gamma \vdash \mu^a X.A \leq \mu^b X.B}$$

- Admissible typing rules

$$\frac{\Gamma \vdash t : [\mu^\infty X.A/X]A}{\Gamma \vdash \mathsf{in}\, t : \mu^\infty X.A} \qquad \frac{\Gamma \vdash r : \mu^\infty X.A}{\Gamma \vdash \mathsf{out}\, r : [\mu^\infty X.A/X]A}$$

Inductive types – example

Slide 15

$$
\begin{array}{lcl}
\mathsf{List}^i(A) & := & \mu^i X.\, 1 + A \times X \\
\mathsf{nil} & : & \forall A \forall i.\, \mathsf{List}^i A \\
 & := & \mathsf{in}(\mathsf{inl}\langle\rangle) \\
\mathsf{cons} & : & \forall A.\, A \to \forall i.\, \mathsf{List}^i(A) \to \mathsf{List}^{i+1}(A) \\
 & := & \lambda a \lambda as.\, \mathsf{in}(\mathsf{inr}\langle a, as\rangle) \\
\mathsf{head} & : & \forall A \forall i.\, \mathsf{List}^{i+1}(A) \to (1 + A) \\
\mathsf{head} & := & \lambda l.\, \mathsf{case}\,(\mathsf{out}\, l)\ \mathsf{of}\ \mathsf{inl}\ \_ \Rightarrow \mathsf{inl}\langle\rangle \mid \mathsf{inr}\, p \Rightarrow \mathsf{inr}(\mathsf{fst}\ p)
\end{array}
$$

Could we also type $\mathsf{head} : \forall A \forall i.\, \mathsf{List}^i(A) \to (1 + A)$?

Case distinction for inductive types

Slide 16

- Case distinction on ordinal i:

$$\frac{\Gamma, i, \Gamma' \vdash r : \mu^i X.A \qquad \Gamma, j, x : \mu^{j+1} X.A \vdash t : C(j+1)}{\Gamma, i, \Gamma' \vdash \mathsf{let}\ x = r\ \mathsf{in}\ t : C(i)}$$

  where i only pos in $C(i)$.

- Better typing for head:

$$
\begin{array}{lcl}
\mathsf{head} & : & \forall A \forall i.\, \mathsf{List}^i(A) \to (1 + A) \\
\mathsf{head} & := & \lambda l.\, \mathsf{let}\ x^{j+1} = l^i\ \mathsf{in} \\
 & & \quad \mathsf{case}\,(\mathsf{out}\, x)\ \mathsf{of}\ \mathsf{inl}\ \_ \Rightarrow \mathsf{inl}\langle\rangle \mid \mathsf{inr}\, p \Rightarrow \mathsf{inr}(\mathsf{fst}\ p)
\end{array}
$$

- Case distinction on ordinal could be integrated into case construct.

Infinite structures

Slide 17

- On infinite objects like streams, we are interested in the *definedness* rather than the size.
- $s : \mathsf{Stream}^a(A)$ means s is defined upto depth a.
- Objects which are defined upto depth ∞ are called *productive*.
- Subtyping for streams:

$$\mathsf{Stream}^\infty(A) \leq \ldots \mathsf{Stream}^{i+1}(A) \leq \mathsf{Stream}^i(A)$$

Coinductive types

Slide 18

- Types

$$A, B, C ::= \ldots \mid \nu^a X.\, A \quad \text{where } X \text{ only pos in } A$$

- Typing

$$\frac{\Gamma \vdash t : [\nu^a X.A/X]A}{\Gamma \vdash \mathsf{in}\, t : \nu^{a+1} X.A} \qquad \frac{\Gamma \vdash r : \nu^{a+1} X.A}{\Gamma \vdash \mathsf{out}\, r : [\nu^a X.A/X]A}$$

- Subtyping

$$\frac{a \leq b \text{ or } b = \infty \qquad \Gamma, X \vdash A \leq B}{\Gamma \vdash \nu^b X.A \leq \nu^a X.B}$$

- Admissible typing rules

$$\frac{\Gamma \vdash t : [\nu^\infty X.A/X]A}{\Gamma \vdash \mathsf{in}\, t : \nu^\infty X.A} \qquad \frac{\Gamma \vdash r : \nu^\infty X.A}{\Gamma \vdash \mathsf{out}\, r : [\nu^\infty X.A/X]A}$$

Coinductive types – example

Slide 19

$$
\begin{array}{lcl}
\mathsf{Stream}^i(A) & := & \nu^i X.\, A \times X \\
\mathsf{s\_cons} & : & \forall A.\, A \to \forall i.\, \mathsf{Stream}^i(A) \to \mathsf{Stream}^{i+1}(A) \\
 & := & \lambda a \lambda as.\, \mathsf{in}\langle a, as\rangle \\
\mathsf{s\_head} & : & \forall A \forall i.\, \mathsf{Stream}^{i+1}(A) \to A \\
\mathsf{s\_head} & := & \lambda s.\ \mathsf{fst}\,(\mathsf{out}\, s) \\
\mathsf{s\_tail} & : & \forall A \forall i.\, \mathsf{Stream}^{i+1}(A) \to \mathsf{Stream}^i(A) \\
\mathsf{s\_tail} & := & \lambda s.\ \mathsf{snd}\,(\mathsf{out}\, s)
\end{array}
$$

Recursion and corecursion

Slide 20

- Terms

$$r, s, t ::= \ldots \mid \mathsf{fix}^\mu\ s \mid \mathsf{fix}^\nu\ s$$

- Typing

$$\frac{\Gamma, i \vdash s : ((\mu^i X.A) \to B(i)) \to (\mu^{i+1} X.A) \to B(i+1)}{\Gamma \vdash \mathsf{fix}^\mu\, s : \forall i.\, (\mu^i X.A) \to B(i)} \quad i \cap\text{-}\mathsf{cont}\ B(i)$$

$$\frac{\Gamma, i \vdash s : A(i) \to A(i+1)}{\Gamma \vdash \mathsf{fix}^\nu\, s : \forall i.\, A(i)} \quad i\ \mathsf{legal}^\nu\ A(i)$$

- Legal types for corecursion

$$\frac{}{i\ \mathsf{legal}^\nu\ \nu^i X.A}\ (i \notin A) \qquad \frac{i\ \mathsf{legal}^\nu\ A \qquad i\ \mathsf{legal}^\nu\ B}{i\ \mathsf{legal}^\nu\ A \times B}$$

$$\frac{i\ \mathsf{only\ pos\ in}\ A \qquad i\ \mathsf{legal}^\nu\ B}{i\ \mathsf{legal}^\nu\ A \to B}$$

Corecursion example: sequence of natural numbers

Slide 21

- Map for streams in sugared recursion syntax:

  $\mathsf{s\_map} : \forall X \forall Y.\, (X \to Y) \to \forall i.\, \mathsf{Stream}^i(X) \to \mathsf{Stream}^i(Y)$

  $\mathsf{s\_map}\ f\ (x :: xs^i)^{i+1} = ((f\, x) :: \mathsf{s\_map}\ f\ xs^i)^{i+1}$

- Stream of natural numbers in orginal recursion syntax:

  $\mathsf{nats} : \forall i.\, \mathsf{Stream}^i(\mathsf{Int})$

  $\mathsf{nats} = \mathsf{fix}^\nu\, \lambda nats.\, (0 :: (\mathsf{s\_map}\ \ +1\ nats^i)^i)^{i+1}$

Reduction

Slide 22

- Special evaluation contexts:

$$E ::= \bullet \mid E\, s \mid \mathsf{fst}\ E \mid \mathsf{snd}\ E$$

- Rules for (co)inductive data and (co)recursion:

$$\begin{array}{lcl} \mathsf{out}\,(\mathsf{in}\, t) & \longrightarrow_\beta & t \\ \mathsf{fix}^\mu\, s\,(\mathsf{in}\, t) & \longrightarrow_\beta & s\,(\mathsf{fix}^\mu\, s)\,(\mathsf{in} t) \\ \mathsf{out}\,(E[\mathsf{fix}^\nu\, s]) & \longrightarrow_\beta & \mathsf{out}\,(E[s\,(\mathsf{fix}^\nu\, s)]) \end{array}$$

- Add β-rules for the lambda-calculus with sums and products plus congruence rules.
- For the resulting reduction relation we can show strong normalization.

Semantics of size expressions

Slide 23

Let θ be a mapping of size variables into ordinals $< \Omega + \omega$.

$$\begin{aligned} [\![i]\!]\theta &= \theta(i) \\ [\![a+1]\!]\theta &= [\![a]\!] + 1 \\ [\![\infty]\!]\theta &= \Omega \end{aligned}$$

Semantics of types

Slide 24

- Let $\Gamma \vdash A : *$ and $\theta : \Gamma$ a valuation of the free type and size variables in A.
- Semantics $[\![A]\!]\theta \subseteq \mathsf{SN}$ (saturated set):

$$\begin{aligned} [\![A \to B]\!]\theta &= \{r \mid r\,s \in [\![B]\!]\theta \text{ for all } s \in [\![A]\!]\theta\} \\ [\![\mathsf{List}^i(A)]\!]\theta &= \Phi^{\alpha}_{\mathsf{List}(A),\theta}(\emptyset) \\ [\![\mathsf{Stream}^i(A)]\!]\theta &= \Phi^{\alpha}_{\mathsf{Stream}(A),\theta}(\mathsf{SN}) \end{aligned}$$

with ordinal $\alpha = [\![i]\!]\theta$ and

$$\begin{aligned} \Phi_{\mathsf{List}(A),\theta}(Q) &= \{\mathsf{nil}, \mathsf{cons}\,s\,t \mid s \in [\![A]\!]\theta,\ t \in Q\} \\ \Phi_{\mathsf{Stream}(A),\theta}(Q) &= \{r \mid \mathsf{s\_head}\,r \in [\![A]\!]\theta,\ \mathsf{s\_tail}\,r \in Q\} \end{aligned}$$

Uniform Operation Iteration

Slide 25

- Iterates Φ^α can be defined uniformly for least and greatest fixed-points using limes inferior. Let $P : \mathsf{On} \to \mathcal{P}(\mathsf{SN})$.

$$\begin{array}{lcl} \underline{\lim}_{\alpha\to\lambda} P(\alpha) & = & \bigcup_{\alpha_0<\lambda} \bigcap_{\alpha_0\leq\alpha<\lambda} P(\alpha) \\ \Phi^0(Q) & = & Q \\ \Phi^{\alpha+1}(Q) & = & \Phi(\Phi^\alpha(Q)) \\ \Phi^\lambda(Q) & = & \underline{\lim}_{\alpha\to\lambda} \Phi^\alpha(Q) \end{array}$$

- Lemma: Assume Φ increasing, i.e., $\Phi^\alpha(Q) \subseteq \Phi^\beta(Q)$ for $\alpha \leq \beta$.

$$\Phi^\lambda(Q) = \bigcup_{\alpha<\lambda} \Phi^\alpha(Q)$$

- Lemma: Assume Φ decreasing, i.e., $\Phi^\alpha(Q) \supseteq \Phi^\beta(Q)$ for $\alpha \leq \beta$.

$$\Phi^\lambda(Q) = \bigcap_{\alpha<\lambda} \Phi^\alpha(Q)$$

On the side condition on recursion

Slide 26

- Recall the recursion rule:

$$\frac{\Gamma, i \vdash s : ((\mu^i X.A) \to B(i)) \to (\mu^{i+1} X.A) \to B(i+1)}{\Gamma \vdash \mathsf{fix}^\mu\, s : \forall i.\, (\mu^i X.A) \to B(i)} \quad i \cap\text{-}\mathsf{cont}\ B(i)$$

- Soundness is proven by transfinite induction on the ordinal $[\![i]\!]$. For the limit case to hold, B must admit

$$\begin{array}{rcl} \underline{\lim}_{\alpha\to\lambda} & [\![(\mu^i X.A) \to B(i)]\!](i \mapsto \alpha) \\ \subseteq & [\![(\mu^i X.A) \to B(i)]\!](i \mapsto \lambda) \end{array}$$

- Thus, result type of this fix^μ-construction must be *continuous* in i.
- We distinguish two kinds of continuity.

∪-Continuity

- A set-valued function $P : \mathsf{On} \to \mathcal{P}(\mathsf{SN})$ is called ∪-continuous if

$$P(\lambda) \subseteq \varinjlim_{\alpha \to \lambda} P(\alpha)$$

  Hughes, Pareto & Sabry (1996) call P *overshooting*.

Slide 27

- Grammar for ∪-continuous types i ∪-cont A:

$$\frac{i \text{ only neg in } A}{i \cup\text{-cont } A} \qquad \frac{i \cup\text{-cont } A, B}{i \cup\text{-cont } A + B, A \times B} \qquad \frac{i \cup\text{-cont } A}{i \cup\text{-cont } \mathsf{List}^a(A)}$$

- Theorem: If i ∪-cont A then $[\![A]\!](i)$ is ∪-continuous.

∩-Continuity

- A set-valued function $P : \mathsf{On} \to \mathcal{P}(\mathsf{SN})$ is called ∩-continuous if

$$\varinjlim_{\alpha \to \lambda} P(\alpha) \subseteq P(\lambda)$$

  Hughes, Pareto & Sabry (1996) call P *undershooting*.

Slide 28

- Grammar for ∩-continuous types i ∩-cont A:

$$\frac{i \text{ only pos in } A}{i \cap\text{-cont } A} \qquad \frac{i \cap\text{-cont } A, B}{i \cap\text{-cont } A + B, A \times B} \qquad \frac{i \cap\text{-cont } A}{i \cap\text{-cont } \mathsf{List}^a(A)}$$

$$\frac{i \cup\text{-cont } A \qquad i \cap\text{-cont } B}{i \cap\text{-cont } A \to B} \qquad \frac{i \cap\text{-cont } A}{i \cap\text{-cont } \mathsf{Stream}^a(A)}$$

- Theorem: If i ∩-cont A then $[\![A]\!](i)$ is ∩-continuous.

Work in progress: F^ω with sized types

Slide 29

- Kinds.

$$\begin{array}{rcll} \kappa & ::= & * & \text{types} \\ & | & \mathsf{ord} & \text{ordinal sizes} \\ & | & \kappa \xrightarrow{+} \kappa' & \text{covariant type constructors} \\ & | & \kappa \xrightarrow{-} \kappa' & \text{contravariant type constructors} \\ & | & \kappa \xrightarrow{0} \kappa' & \text{invariant type constructors} \end{array}$$

- "Subconstructors" $F \leq G : \kappa$. E.g.,

$$\frac{X \leq Y : \kappa \vdash F\,X \leq G\,Y : \kappa'}{F \leq G : \kappa \xrightarrow{+} \kappa'}$$

- Well-kindedness definable by $F : \kappa \iff F \leq F : \kappa$

Inductive constructors

Slide 30

- Inductive constructors.

$$\mu_\kappa : \mathsf{ord} \xrightarrow{+} (\kappa \xrightarrow{+} \kappa) \xrightarrow{+} \kappa$$

- Example for an inductive type:

$$\mathsf{List} = \lambda i \lambda A.\ \mu_* i\,(\lambda X.\, 1 + A \times X)$$

- Inductive functors: μ_κ for $\kappa = * \to *$.
- E.g., $\mathsf{Term}\,A$, de Bruijn terms with free variables in A:

$$\mathsf{Term} = \mu_{*\to*} \infty \lambda T \lambda A.\ A + T(1 + A) + TA \times TA$$

Conclusions

Sized types:

- Conceptually *lean* way of ensuring termination.
- Well-typedness ensures termination.
- No external static analysis required.

Slide 31

System F^{ω}:

- Size expressions can be integrated into constructors.
- Sized types scale to higher-order polymorphism.

Goal: extend to dependent types.

Related Work

- Hughes, Pareto, Sabry (1996)
  Proving the correctness of reactive system using sized types
- Barthe, Frade, Giménez, Pinto, Uustalu (2003?)
  Type-based termination of recursive definitions
- Buchholz (2003?)
  Recursion on nonwellfounded trees

Slide 32