Slide 1

TOPOLOGY

Slide 2

TOPOLOGY

$\texttt{PO} \mapsto \texttt{TA}$

$\texttt{OG} \mapsto \texttt{IT}$

Slide 3

TOTALITY

Termination and Guardedness Checking with Continuous Types

Yet Another Approach to General Recursion

Andreas Abel

TYPES Workshop 2002
Berg en Dal, Nijmegen
April 26, 2002

Slide 4

1. Survey: Type Systems for Structural Recursion
2. Corecursion
3. Continuous Types
4. Further Work

Work supported by: PhD Programme Logic in Computer Science, Munich (DFG)

Totality of Recursive Definitions

Slide 5

1. Measure-based
   - Miculan (ordered uniformities)
   - Xi 2001 [8] (DML)
2. Totality proof (supported by tactics)
   - Bove (proof of accessibility)
   - Bertot (proof of well-definedness)
3. Syntactic (fully automatic)
   - By static analysis of code (Coquand 1992 [3], Gimenez 1994 [4], Abel/Altenkirch [2])
   - *By type-checking*

Inductive Datatypes

Slide 6

- E.g. natural numbers and lists

$$\begin{aligned} \mathsf{NatF}(X) &= \mathsf{Zero} + \mathsf{Succ}\, X & \mathsf{ListF}(A)(X) &= \mathsf{Nil} + \mathsf{Cons}\, A \times X \\ \mathsf{Nat} &= \mu \mathsf{NatF} & \mathsf{List}(A) &= \mu(\mathsf{ListF}(A)) \\ &= \mu X.\, \mathsf{Zero} + \mathsf{Succ}\, X & &= \mu X.\, \mathsf{Nil} + \mathsf{Cons}\, A \times X \end{aligned}$$

- Least fixed-point constructed from below via *approximations*.
- $[\![F]\!]^\alpha(\emptyset)$ contains trees of height $< \alpha$.

$$\emptyset = [\![F]\!]^0(\emptyset) \subseteq \dots \underset{\substack{| \\ Y}}{[\![F]\!]^\alpha(\emptyset)} \subseteq \underset{\substack{| \\ F(Y)}}{[\![F]\!]^{\alpha+1}(\emptyset)} \subseteq \dots \underset{\substack{| \\ \mu F}}{[\![F]\!]^{\omega_1}(\emptyset)}\ [\![F]\!]^\alpha(\emptyset)$$

- Basic principle: To show $f \in [\![\mu F]\!] \to T$, fix some α, assume $f \in [\![F]\!]^\alpha(\emptyset) \to T$ and show $f \in [\![F]\!]^{\alpha+1}(\emptyset) \to T$.

Mendler (1987) [7]: Iteration

Slide 7

- Strengthening the iteration rule.

$$\frac{Y\!:\!\mathsf{type},\ g\!:\!Y \to \tau \vdash M : F(Y) \to \tau}{\mathsf{fix}\, g.M : \mu F \to \tau}$$

- Example: map

  $\mathsf{map} : \forall A \forall B.\ (A \to B) \to \mathsf{List}(A) \to \mathsf{List}(B)$

  $\mathsf{map} = \lambda f.\ \mathsf{fix}\, g^{Y \to \mathsf{List}(B)}.\ \lambda l^{\mathsf{ListF}(A)(Y)}.\, \mathsf{case}\, l\ \mathsf{of}$

  $\quad \mathsf{Nil} \Rightarrow \mathsf{Nil}$

  $\quad \mid \mathsf{Cons}(x, xs^{Y}) \Rightarrow \mathsf{Cons}(f\,x, g\,xs^{Y})$

- Typing restricts argument to xs in recursive call.
- Variable xs cannot be used as a list; hence iteration.

Mendler: Primitive Recursion

Slide 8

- Syntax for approximations: Fresh type variables Y.

$$\frac{Y \le \mu F,\ g\!:\!Y \to \tau \vdash M : F(Y) \to \tau}{\mathsf{fix}\, g.M : \mu F \to \tau}$$

- Abbreviation: $Y^n = F^n(Y)$.
- Example: Insertion into sorted list.

  $\mathsf{insert} : \mathsf{Nat} \to \mathsf{List(Nat)} \to \mathsf{List(Nat)}$

  $\mathsf{insert} = \lambda a.\ \mathsf{fix}\, g^{Y \to \mathsf{List(Nat)}}.\ \lambda l^{Y^1}.\, \mathsf{case}\, l\ \mathsf{of}$

  $\quad \mathsf{Nil} \Rightarrow \mathsf{Cons}(a, \mathsf{Nil})$

  $\quad \mid \mathsf{Cons}(x, xs^{Y}) \Rightarrow \mathsf{if}\ \ a \le x\ \ \mathsf{then}\ \ \mathsf{Cons}(a, \mathsf{Cons}(x, xs^{Y \le \mathsf{List(Nat)}}))$

  $\quad\quad \mathsf{else}\ \ \mathsf{Cons}(x, g\, xs^{Y})$

- Coercion $Y \le \mu F$ may be used : Stronger than iteration.

Mendler-style Course-of-Value Recursion

Slide 9

- Add subtyping $Y \leq Y^1 = F(Y)$ for $Y \leq \mu F$.
- Example: take every other element of a list.

  half : $\forall A.$ List$(A) \to$ List(A)

  half $=$ fix $g^{Y \to \mathsf{List}(A)}.\ \lambda l^{Y^1}.$ case l of

  $$\begin{array}{ll} \mathsf{Nil} & \Rightarrow \mathsf{Nil} \\ \mid \mathsf{Cons}(x, xs^Y) & \Rightarrow \mathsf{case}\ xs^{Y \leq Y^1}\ \mathsf{of} \\ & \quad \begin{array}{ll} \mathsf{Nil} & \Rightarrow \mathsf{Nil} \\ \mid \mathsf{Cons}(y, ys^Y) & \Rightarrow \mathsf{Cons}(x, g\, y^Y) \end{array} \end{array}$$

- Coercion $Y \leq Y^1$ permits further unwinding.

Giménez 1998 [5]: Size-Change Information

Slide 10

- Use type variable also in result type of recursive functions.

  $$\frac{Y \leq \mu F,\ g{:}Y \to \tau(Y) \vdash M : Y^1 \to \tau(Y^1) \qquad Y \text{ only pos in } \tau(Y)}{\mathsf{fix}\, g.M : \forall Y \leq \mu F.\ Y \to \tau(Y)}$$

- More precise typing. E.g.

  $$\begin{array}{lcl} \mathsf{map} & : & (\mathsf{Nat} \to \mathsf{Nat}) \to \forall Y \leq \mathsf{List}(\mathsf{Nat}).\ Y \to Y \\ \mathsf{insert} & : & \mathsf{Nat} \to \forall Y \leq \mathsf{List}(\mathsf{Nat}).Y \to Y^1 \end{array}$$

- “map” is non-size-increasing, “insert” increases length of list by at most one.
- But: no longer polymorphic.
- Many interesting algorithms are typable, e.g. Euclidean division, filter, quicksort.

- Nesting:

$$\begin{aligned}
&\mathsf{nest} && : \forall Y \leq \mathsf{Nat}.\, Y \to Y \\
&\mathsf{nest}\ \ \mathsf{Zero} && = \mathsf{Zero} \\
&\mathsf{nest}\ \ (\mathsf{Succ}\, n^{Y}) && = (\mathsf{nest}\,(\mathsf{nest}\, n^{Y})^{Y})^{Y \leq Y^{1}}
\end{aligned}$$

- Abel 2002 [1]: SR, SN, bidirectional type-checking.

Slide 11

- Current work: Relax side condition on result types Y only pos in $\tau(Y)$.

Presentation with Indexed Types

- Stage expressions and their interpretation:

r, s	$::=$	i	variable	$[\![i]\!]$	$=$	α for some ordinal α
	$\mid$	$s+1$	sucessor	$[\![s+1]\!]$	$=$	$[\![s]\!] + 1$
	$\mid$	∞	ultimate limit	$[\![\infty]\!]$	$=$	ω_1 (first uncountable)

Slide 12

- Translation of previous system:

$$\begin{aligned}
Y &= \mu^{i} F \\
Y^{n} &= \mu^{i+n} F \\
\mu F &= \mu^{\infty} F
\end{aligned}$$

- Subtyping for approximations:

$$\mu^{i} F \leq \mu^{i+1} F \leq \ldots \leq \mu^{\infty} F$$

Slide 13

Presentation with Indexed Types (II)

- Recursion rule:

$$\frac{\Gamma,\ i,\ g : \mu^{i} F \to \tau(i)\ \vdash\ M : \mu^{i+1} F \to \tau(i+1) \qquad i \text{ only pos in } \tau(i)}{\Gamma \vdash \mathsf{fix}\, g.M : \forall i. \mu^{i} F \to \tau(i)}$$

- Reconciled with polymorphism.

$$\mathsf{map} \quad : \quad \forall X \forall Y. (X \to Y) \to \forall i. \mathsf{List}^{i}(X) \to \mathsf{List}^{i}(Y)$$

- Forthcoming article: Barthe/Frade/Gimenez/Pinto/Uustalu.
- For natural number expressions already in Hughes/Pareto/Sabry 1996 [6].

Slide 14

Example: Quick Sort

pivot : Int $\to$ List $\to$ List $\times$ List

pivot a [] $=$ ([] , [])
pivot a $(x :: xs\ )^{i+1}$ $=$ let $(l\ ,\ r\ )$ $=$ pivot a xs in
if $x < a$ then $((x :: l)\ ,\ r\ )$
else $(l\ ,\ (x :: r)\ )$

qsapp : List $\to$ List $\to$ List

qsapp [] ys $= ys$
qsapp $(x :: xs\ )$ ys $=$ let $(l\ ,\ r\ )$ $=$ pivot x xs in
qsapp l $(x ::$ qsapp r $ys)$

quicksort : List $\to$ List

quicksort l $=$ qsapp l []

Example: Quick Sort

Slide 15

$$\text{pivot} \;:\; \text{Int} \rightarrow \forall i.\, \text{List}^{i} \rightarrow \text{List}^{i} \times \text{List}^{i}$$

$$\begin{aligned}
\text{pivot}\;\; a\; [\,]^{i+1} &= ([\,]^{i+1},\; [\,]^{i+1}) \\
\text{pivot}\;\; a\; (x :: xs^{i})^{i+1} &= \text{let}\; (l^{i},\; r^{i}) = \text{pivot}\;\; a\; xs^{i}\;\; \text{in} \\
&\;\; \text{if}\;\; x < a\;\; \text{then}\; ((x :: l)^{i+1},\; r^{i \le i+1}) \\
&\;\; \text{else}\; (l^{i \le i+1},\; (x :: r)^{i+1})
\end{aligned}$$

$$\text{qsapp} : \forall i.\, \text{List}^{i} \rightarrow \text{List}^{\infty} \rightarrow \text{List}^{\infty}$$

$$\begin{aligned}
\text{qsapp}\; [\,]^{i+1}\; ys &= ys \\
\text{qsapp}\; (x :: xs^{i})^{i+1}\; ys &= \text{let}\; (l^{i},\; r^{i}) = \text{pivot}\;\; x\; xs^{i}\;\; \text{in} \\
&\;\; \text{qsapp}\;\; l^{i}\; (x :: \text{qsapp}\;\; r^{i}\; ys)
\end{aligned}$$

$$\text{quicksort} : \text{List}^{\infty} \rightarrow \text{List}^{\infty}$$

$$\text{quicksort}\;\; l = \text{qsapp}\;\; l\; [\,]$$

Guarded Corecursion

Slide 16

- Let $[\![\text{Stream}]\!]^{n}$ denote the set of integer streams which have goodness n, i.e., can be unrolled n times.
- A recursively defined stream $g = M(g)$ surely is of goodness ω, if

  $$g \in [\![\text{Stream}]\!]^{n} \Longrightarrow M(g) \in [\![\text{Stream}]\!]^{n+1}.$$

  We say that g is guarded in M.
- Rules for corecursion, e.g.:

  $$\frac{\Gamma,\; i,\; g : \text{Stream}^{i} \vdash M : \text{Stream}^{i+1}}{\Gamma \vdash \text{fix}^{\nu} g.M \;:\; \forall i.\, \text{Stream}^{i}}$$

  $$\frac{\Gamma,\; i,\; g : \text{Stream}^{i} \rightarrow \text{Stream}^{i} \vdash M : \text{Stream}^{i+1} \rightarrow \text{Stream}^{i+1}}{\Gamma \vdash \text{fix}^{\nu} g.M \;:\; \forall i.\, \text{Stream}^{i} \rightarrow \text{Stream}^{i}}$$

Example: Sequence of Natural Numbers

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

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

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

Slide 17 nats $= (0 :: (\mathsf{mapStr}\ \ {+1}\ \mathsf{nats}^i)^i)^{i+1}$

Subtyping for Stream:

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

Semantics

- Let $\Gamma \vdash \tau$: type and $\theta : \Gamma$ a valuation of the free type and stage variables in τ.
- Semantics $[\![\tau]\!]\theta$:

Slide 18

$$\begin{aligned}
[\![\sigma \to \tau]\!]\theta &= \{M \mid \forall N \in [\![\sigma]\!]\theta.\, M\, N \in [\![\tau]\!]\theta\} \\
[\![\mathsf{List}^i(\tau)]\!]\theta &= \Phi^{[\![i]\!]\theta}_{\mathsf{List}(\tau),\theta}(\emptyset) \\
[\![\mathsf{Stream}^i(\tau)]\!]\theta &= \Phi^{[\![i]\!]\theta}_{\mathsf{Stream}(\tau),\theta}(\mathsf{SN})
\end{aligned}$$

with

$$\begin{aligned}
\Phi_{\mathsf{List}(\tau),\theta}(Q) &= \{\mathsf{nil}, M :: N \mid M \in [\![\tau]\!]\theta, N \in Q\} \\
\Phi_{\mathsf{Stream}(\tau),\theta}(Q) &= \{M \mid \mathsf{fst}\, M \in [\![\tau]\!]\theta, \mathsf{snd}\, M \in Q\}
\end{aligned}$$

Uniform Operator Iteration

Slide 19

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

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

- Lemma: Assume Φ increasing, i.e., $\Phi^\alpha(Q) \subseteq \Phi^\beta(Q)$ for $\alpha \le \beta$.
$$\Phi^\lambda(Q) = \bigcup_{\alpha<\lambda} \Phi^\alpha(Q)$$

- Lemma: Assume Φ decreasing, i.e., $\Phi^\alpha(Q) \supseteq \Phi^\beta(Q)$ for $\alpha \le \beta$.
$$\Phi^\lambda(Q) = \bigcap_{\alpha<\lambda} \Phi^\alpha(Q)$$

Relaxing the Side Condition

Slide 20

- Recursion rule:

$$\frac{\Gamma,\ i,\ g : \mu^i F \to \tau(i)\ \vdash\ M : \mu^{i+1} F \to \tau(i+1) \qquad i\ \cap\text{-cont}\ \tau(i)}{\Gamma \vdash \mathsf{fix}\, g.M : \forall i. \mu^i F \to \tau(i)}$$

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

$$\underline{\lim}_{\alpha\to\lambda} [\![\mu^i F \to \tau(i)]\!]_{i\mapsto\alpha} \subseteq [\![\mu^i F \to \tau(i)]\!]_{i\mapsto\lambda}$$

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

$\cup$-Continuity

- A set-valued function P : On $\to \mathcal{P}$(TM) is called $\cup$-*continuous* if

$$P(\lambda) \subseteq \underline{\lim}_{\alpha \to \lambda} P(\alpha)$$

- Grammar for $\cup$-continuous types i $\cup$-cont τ:

Slide 21

$$\frac{i \text{ only neg in } \tau}{i \ \cup\text{-cont } \tau} \qquad \frac{i \ \cup\text{-cont } \sigma, \tau}{i \ \cup\text{-cont } \sigma + \tau, \sigma \times \tau} \qquad \frac{i \ \cup\text{-cont } \tau}{i \ \cup\text{-cont List}^s(\tau)}$$

- Theorem: If i $\cup$-cont τ then $[\![\tau]\!](i)$ is $\cup$-continuous.
- All polynomial datatypes (finitely branching trees) are $\cup$-continuous.

$\cap$-Continuity

- A set-valued function P : On $\to \mathcal{P}$(TM) is called $\cap$-*continuous* if

$$\overline{\lim}_{\alpha \to \lambda} P(\alpha) \subseteq P(\lambda)$$

- Grammar for $\cap$-continuous types i $\cap$-cont τ:

Slide 22

$$\frac{i \text{ only pos in } \tau}{i \ \cap\text{-cont } \tau} \qquad \frac{i \ \cap\text{-cont } \sigma, \tau}{i \ \cap\text{-cont } \sigma + \tau, \sigma \times \tau} \qquad \frac{i \ \cap\text{-cont } \tau}{i \ \cap\text{-cont List}(\tau)}$$

$$\frac{i \ \cup\text{-cont } \sigma \qquad i \ \cap\text{-cont } \tau}{i \ \cap\text{-cont } \sigma \to \tau} \qquad \frac{i \ \cap\text{-cont } \tau}{i \ \cap\text{-cont Stream}^s(\tau)}$$

- Theorem: If i $\cap$-cont τ then $[\![\tau]\!](i)$ is $\cap$-continuous.
- All positive datatypes are $\cap$-continuous.

Strong Normalization

Slide 23

- Unrolling of fixpoints is limited by guards C (constructor) and D (destructor).

$$
\begin{array}{rcl}
(\lambda x.M)\,N & \longrightarrow_\beta & [N/x]M \\
(\mathsf{fix}^\mu g.M)\,(C N) & \longrightarrow_\beta & ([\mathsf{fix}^\mu g.M/g]M)\,(C N) \\
D((\mathsf{fix}^\nu g.M)\,N) & \longrightarrow_\beta & D(([\mathsf{fix}^\nu g.M/g]M)\,N)
\end{array}
$$

- This enables us to prove strong normalization.

Contribution and Further Work

Slide 24

My work so far:

- Strongly normalizing language with fairly liberal general recursion.
- Polymorphism.
- Subtyping.

Further work:

- Lexicographic and simultaneous product of ordinal indices.
- Constructor subtyping.
- Interplay between recursion and corecursion.
- Size inference.
- More termination orderings.

References

[1] Andreas Abel. Termination checking with types. Technical Report 0201, Institut für Informatik, Ludwigs-Maximilians-Universität München, 2002.

[2] Andreas Abel and Thorsten Altenkirch. A predicative analysis of structural recursion. *Journal of Functional Programming*, 12(1):1–41, January 2002.

Slide 25

[3] Thierry Coquand. Infinite objects in type theory. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs (TYPES '93)*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer-Verlag, 1993.

[4] Eduardo Giménez. Codifying guarded definitions with recursive schemes. In Peter Dybjer, Bengt Nordström, and Jan Smith, editors, *Types for Proofs and Programs, International Workshop TYPES´94, Båstad, Sweden, June 6-10, 1994, Selected Papers*, volume 996 of *LNCS*, pages 39–59. Springer, 1995.

[5] Eduardo Giménez. Structural recursive definitions in type theory. In *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *LNCS*, pages 397–408. Springer, 1998.

[6] John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *Symposium on Principles of Programming Languages*, pages 410–423, 1996.

Slide 26

[7] Nax P. Mendler. Recursive types and type constraints in second-order lambda calculus. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science, Ithaca, N.Y.*, pages 30–36. IEEE Computer Society Press, 1987.

[8] Hongwei Xi. Dependent types for program termination verification. In *Proceedings of 16th IEEE Symposium on Logic in Computer Science*, Boston, USA, June 2001.