Type-Based Termination of Functional Programs

Andreas Abel

Department of Computer Science
Ludwig-Maximilians-University Munich

Kolloquium Programmiersprachen und Grundlagen der Programmierung
KPS'07
11 October 2007
Timmendorfer Strand, Germany

Termination

- Question: Will the run of a program eventually halt?
- Undecidable for Turing-complete programming languages (Halteproblem).
- No termination checker can give a definitive answer for all programs.
- Problem still interesting for:
  - optimization and program specialization
  - total correctness of programs
  - proof assistants like Agda, Coq, Epigram, LEGO

Type-based termination

- View data (natural numbers, lists, binary trees) as trees.
- Type of data is equipped with a size.
- Size = upper bound on height of tree.
- Size must decrease in each recursive call.
- Termination is ensured by type-checker.

Sized types in a nutshell

- 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 $a \leq b$.
- Size expressions a, b.

$$
\begin{array}{rcll}
a & ::= & i & \text{variable} \\
 & | & a + 1 & \text{successor} \\
 & | & \infty & \omega
\end{array}
$$

Splitting: definition

$$\mathsf{split} : \quad \forall A.\ \mathsf{List}\ A \to \mathsf{List}\ A \times \mathsf{List}\ A$$

$$\begin{array}{ll}
\mathsf{split}\ [\,] & = ([\,] \quad , [\,] \quad) \\
\mathsf{split}\ (y :: l\ ) & = \mathsf{let}\ (xs\ , ys\ ) = \mathsf{split}\ l\ \ \mathsf{in} \\
 & \quad ((y :: ys) \quad , xs \qquad)
\end{array}$$

Splitting: termination

$$\mathsf{split} : \forall i.\, \forall A.\ \mathsf{List}^{i} A \to \mathsf{List}\ A \times \mathsf{List}\ A$$

$$\begin{array}{lll}
\mathsf{split}\ [\,] & = & ([\,] \quad , [\,] \quad) \\
\mathsf{split}\ (y :: l^{i})^{i+1} & = & \mathsf{let}\ (xs\ , ys\ ) = \mathsf{split}\ l^{i}\ \mathsf{in} \\
 & & ((y :: ys) \quad , xs \qquad)
\end{array}$$

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

Splitting: bounded output

$$\mathsf{split} : \forall i.\, \forall A.\, \mathsf{List}^{i} A \to \mathsf{List}^{i} A \times \mathsf{List}^{i} A$$

$$\begin{array}{lcl} \mathsf{split}\ [\,]^{i+1} & = & ([\,]^{i+1}, [\,]^{i+1}) \\ \mathsf{split}\ (y :: l^{i})^{i+1} & = & \mathsf{let}\ (xs^{i}, ys^{i}) = \mathsf{split}\ l^{i}\ \mathsf{in} \\ & & ((y :: ys)^{i+1}, xs^{i \le i+1}) \end{array}$$

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

Merging: definition

merge produces a sorted list from two sorted input lists.

$$
\begin{array}{l}
\mathsf{merge} : \quad \mathsf{List}\ \ \mathsf{Int} \to \mathsf{List}\ \ \mathsf{Int} \to \mathsf{List}\ \ \mathsf{Int} \\[1ex]
\begin{array}{lcl}
\mathsf{merge}\ [\,]\qquad\qquad l & = & l \\
\mathsf{merge}\ (x :: xs\ )\quad l & = & \mathsf{merge}'\ l
\end{array} \\[1ex]
\qquad \mathsf{where}\ \ \mathsf{merge}' : \quad \mathsf{List}\ \ \mathsf{Int} \to \mathsf{List}\ \ \mathsf{Int} \\[1ex]
\qquad\qquad \begin{array}{lcl}
\mathsf{merge}'\ [\,] & = & x :: xs \\
\mathsf{merge}'\ (y :: ys\ ) & = & \mathsf{if}\ \ x \le y\ \ \mathsf{then} \\
 & & \qquad x :: \mathsf{merge}\ xs\ \ (y :: ys) \\
 & & \mathsf{else}\ \ y :: \mathsf{merge}'\ ys
\end{array}
\end{array}
$$

Merging: termination

merge terminates by lexicographic ordering.

$$\mathsf{merge} : \forall i.\, \mathsf{List}^{i}\ \mathsf{Int} \to \mathsf{List}^{\infty}\ \mathsf{Int} \to \mathsf{List}^{\infty}\ \mathsf{Int}$$

$$\begin{array}{lcl}
\mathsf{merge}\ [\,]\ \ l & = & l \\
\mathsf{merge}\ (x :: xs^{i})^{i+1}\ \ l & = & \mathsf{merge}'\ l
\end{array}$$

$$\mathsf{where}\ \ \mathsf{merge}' : \forall j.\, \mathsf{List}^{j}\ \mathsf{Int} \to \mathsf{List}^{\infty}\ \mathsf{Int}$$

$$\begin{array}{lcl}
\mathsf{merge}'\ [\,] & = & x :: xs \\
\mathsf{merge}'\ (y :: ys^{j})^{j+1} & = & \mathsf{if}\ \ x \leq y\ \ \mathsf{then} \\
 & & \quad x :: \mathsf{merge}\ xs^{i}\ (y :: ys)^{j+1 \leq \infty} \\
 & & \mathsf{else}\ \ y :: \mathsf{merge}'\ ys^{j}
\end{array}$$

Merge sort: definition

$$
\begin{aligned}
&\mathsf{msort} \;:\; \mathsf{List}\ \ \mathsf{Int} \to \mathsf{List}\ \ \mathsf{Int}\\[4pt]
&\mathsf{msort}\ \ [\,] \qquad\qquad = [\,]\\
&\mathsf{msort}\ \ (x :: l) \qquad = \mathsf{msort}'\ \ x\ l\\[12pt]
&\mathsf{msort}' \;:\; \quad \mathsf{Int} \to \mathsf{List}\ \mathsf{Int} \to \mathsf{List}\ \ \mathsf{Int}\\[4pt]
&\mathsf{msort}'\ x\ [\,] \qquad\ \ = [x]\\
&\mathsf{msort}'\ x\ (y :: l\ ) = \mathsf{let}\ (xs\ , ys\ ) = \mathsf{split}\ \ l\ \ \mathsf{in}\\
&\qquad\qquad\qquad\qquad\ \ \mathsf{merge}\ (\mathsf{msort}'\ \ x\ xs\ )\\
&\qquad\qquad\qquad\qquad\qquad\qquad (\mathsf{msort}'\ \ y\ ys\ )
\end{aligned}
$$

Merge sort: termination

$$
\begin{array}{lll}
\mathsf{msort} & : \mathsf{List}^{\infty}\,\mathsf{Int} \to \mathsf{List}^{\infty}\,\mathsf{Int} & \\
\mathsf{msort} & [\,] & = [\,] \\
\mathsf{msort} & (x :: l) & = \mathsf{msort}'\;\; x\;\, l
\end{array}
$$

$$
\begin{array}{lll}
\mathsf{msort}' & : \forall i.\ \mathsf{Int} \to \mathsf{List}^{i}\,\mathsf{Int} \to \mathsf{List}^{\infty}\,\mathsf{Int} & \\
\mathsf{msort}'\; x\; [\,]^{i+1} & & = [x] \\
\mathsf{msort}'\; x\; (y :: l^{i}) & & = \mathsf{let}\; (xs^{i}, ys^{i}) = \mathsf{split}\;\; l^{i}\; \mathsf{in} \\
& & \quad\; \mathsf{merge}\; (\mathsf{msort}'\;\; x\;\, xs^{i}) \\
& & \qquad\qquad\;\; (\mathsf{msort}'\;\; y\;\, ys^{i})
\end{array}
$$

Merge sort: abstract split

$$
\begin{aligned}
\mathsf{msort}'\ \mathit{split}\ x\ [\,] &= [x] \\
\mathsf{msort}'\ \mathit{split}\ x\ (y :: l\,) &= \mathsf{let}\ (xs\ , ys\ ) = \mathit{split}\ l\ \ \mathsf{in} \\
&\ \mathsf{merge}\ (\mathsf{msort}'\ \ x\ \ xs\ ) \\
&\ \phantom{\mathsf{merge}}\ (\mathsf{msort}'\ \ y\ \ ys\ )
\end{aligned}
$$

- The variable *split* can only be instantiated with non size increasing functions
- This is naturally expressed with a rank-2 size polymorphic type

Merge sort: abstract split (II)

$$\begin{aligned}
\mathsf{msort}' &: (\forall i.\, \forall A.\ \mathsf{List}^i A \to \mathsf{List}^i A \times \mathsf{List}^i A) \to \\
&\quad \forall i.\ \mathsf{Int} \to \mathsf{List}^i\, \mathsf{Int} \to \mathsf{List}^\infty\, \mathsf{Int}
\end{aligned}$$

$$\begin{aligned}
\mathsf{msort}'\ \mathit{split}\ x\ [\,]^{i+1} &= [x] \\
\mathsf{msort}'\ \mathit{split}\ x\ (y :: l^i) &= \mathsf{let}\ (xs^i, ys^i) = \mathit{split}\ l^i\ \mathsf{in} \\
&\quad\ \mathsf{merge}\ (\mathsf{msort}'\ \ x\ xs^i) \\
&\quad\ \phantom{\mathsf{merge}}\ (\mathsf{msort}'\ \ y\ ys^i)
\end{aligned}$$

- We drop the restriction of Hughes, Pareto, and Sabry and Barthe et al. that sizes should be inferable.

Formalization

- Sized inductive type $\mu^i X.\, A$.
- Equations and subtyping.

$$\begin{array}{lcll} \mu^{a+1} X.\, A & = & [(\mu^a X.\, A)/X]A & \\ \mu^{\infty} X.\, A & = & [(\mu^{\infty} X.\, A)/X]A & \\ \mu^a X.\, A & \leq & \mu^b X.\, A & \text{for } a \leq b \end{array}$$

- Example: lists.

$$\begin{array}{lcl} \mathsf{List}^i A & := & \mu^i X.\, 1 + A \times X \\ \\ \mathsf{nil} & : & \forall A \forall i.\, \mathsf{List}^{i+1} A \\ & := & \mathsf{inl} \langle \rangle \\ \mathsf{cons} & : & \forall A.\, A \to \forall i.\, \mathsf{List}^i A \to \mathsf{List}^{i+1} A \\ & := & \lambda a \lambda as.\, \mathsf{inr} \langle a, as \rangle \end{array}$$

Recursion

- Recursion principle (semantically):

$$\frac{\mathsf{fix}\, f \in A^{0} \qquad f \in A^{\alpha} \to A^{\alpha+1} \qquad (\mathsf{fix}\, f \in \bigcap_{\alpha<\omega} A^{\alpha}) \to \mathsf{fix}\, f \in A^{\omega}}{\forall \beta \leq \omega.\, \mathsf{fix}\, f \in A^{\beta}}$$

- Step: $\mathsf{fix}\, f \in A^{\alpha}$ implies $f\,(\mathsf{fix}\, f) = \mathsf{fix}\, f \in A^{\alpha+1}$.
- Restrict admissible types A^{α} such that
  - $\mathsf{fix}\, f \in A^{0}$ is trivial, e.g., $A^{\alpha} = (\mu^{\alpha} X.A) \to C$, ($\mu^{0} X.A$ is empty)
  - $(\bigcap_{\alpha<\lambda} A^{\alpha}) \subseteq A^{\omega}$.
- Typing rule for recursion (e.g., $A^{i} = \mathsf{List}^{i}\,\mathsf{Int} \to \mathsf{List}^{i}\,\mathsf{Int}$):

$$\frac{f : \forall i.\, A^{i} \to A^{i+1}}{\mathsf{fix}\, f : A^{a}}\; A^{i} \text{ admissible}$$

Productivity

- Productivity is dual to termination.
- A productive process should continuously turn input into output.
- Examples: editor, operating system, stream.
- Important in embedded and functional reactive programming.

Infinite structures

- 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.
- $\mathsf{Stream}^{a} A = \nu^{a} X.\, A \times X$, then

$$
\begin{array}{lcl}
(\_\,,\,\_) & : & \forall i.\, A \to \mathsf{Stream}^{i} A \to \mathsf{Stream}^{i+1} A \\
\mathsf{fst} & : & \forall i.\, \mathsf{Stream}^{i+1} A \to A \\
\mathsf{snd} & : & \forall i.\, \mathsf{Stream}^{i+1} A \to \mathsf{Stream}^{i} A
\end{array}
$$

- Subtyping: $\mathsf{Stream}^{\infty} A \leq \ldots \mathsf{Stream}^{i+1} A \leq \mathsf{Stream}^{i} A$

Corecursion example: sequence of natural numbers

- Map for streams in sugared recursion syntax:

$$\mathsf{map} : \forall X \forall Y.\, (X \to Y) \to \forall i.\, \mathsf{Stream}^i(X) \to \mathsf{Stream}^i(Y)$$
$$\mathsf{map}\ f\ (x,\ xs^i)^{i+1} = ((f\ x),\ \mathsf{map}\ \ f\ xs^i)^{i+1}$$

- Stream of natural numbers in orginal recursion syntax:

$$\mathsf{from0} : \forall i.\, \mathsf{Stream}^i(\mathsf{Int})$$
$$\mathsf{from0} = \mathsf{fix}^{\nu} \lambda nats.\, (0,\ (\mathsf{map}\ (+1)\ nats^i)^i)^{i+1}$$

Recent publications

- PhD thesis: Type Based Termination (July 2006): Treatment of higher kinded data types.
- MPC 2006: Termination of generic programs.
- CSL 2006: Characterization of admissible types.
- APLAS 2007: Mixed inductive/coinductive types.

Some related work

- Hughes, Pareto, Sabry (1996)
  Proving the correctness of reactive system using sized types.
- Barthe et al.: $\lambda\hat{}$ (2004), CIC^ (2006).
- Blanqui, Riba: Calculus of Algebraic Constructions with Size Annotations (CACSA, 2004/5); size constraints (2006).

Conclusions

- Conceptually lean way of ensuring termination.
- Well-typedness ensures termination.
  Typing derivation is termination certificate.
- Scales to higher-order functions and abstract algorithms.
- Goal: extend soundness proof to dependent types.