Termination of Functions that Are Passed to Their Arguments

Andreas Abel
Dept. of Comp. Sci., Chalmers

Slide 1

September 13, 2005
APPSEM II Workshop
Frauenchiemsee, Bavaria

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

Quiz: Is \texttt{eqList} terminating on all total inputs?

data \texttt{MList \ m \ a \ where}
\texttt{Nil :: MList \ m \ a}
\texttt{Cons :: a -> m (MList \ m \ a) -> MList \ m \ a}

\texttt{eqList \ eqM \ eq \ Nil \ Nil = True}

Slide 2

\texttt{eqList \ eqM \ eq \ (Cons \ a \ mas) \ (Cons \ b \ mbs)}
\texttt{= eq \ a \ b}
\texttt{&& eqM (eqList \ eqM \ eq) \ mas \ mbs}

\texttt{eqList \ eqM \ eq \ _ \ _ = False}
Answer: No!

- Counterexample:

```haskell
data Maybe a where
    Nothing :: Maybe a
    Just :: a -> Maybe a
```

```haskell
l = Cons "BLA" Nothing
eqM f _ _ = f l l
loop = eqList eqM (==) l l
```

- We see that `loop` reduces to itself.

```haskell
eqList eqM eq (Cons a mas) (Cons b mbs)
    = eq a b
    && eqM (eqList eqM eq) mas mbs
```

Quiz Reloaded: Is `eqList` now terminating on all total inputs?

```haskell
data MList m a where
    Nil :: MList m a
    Cons :: a -> m (MList m a) -> MList m a

type Eq a = a -> a -> Bool
```

```haskell
eqList :: (forall a. Eq a -> Eq (m a)) -> Eq a -> Eq (MList m a)
eqList eqM eq (Cons a mas) (Cons b mbs)
    = eq a b
    && eqM (eqList eqM eq) mas mbs
```

eqList ...
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
  - theorem proving

Termination for theorem proving

- Inductive theorem provers: e.g., Agda, Coq, Epigram, Twelf.
- Some proofs are *tree-shaped derivations*, e.g., proof that \([a, 0] = [b, 0]\).

\[
\begin{align*}
0 &= 0 \\
a &= b \\
(0 :: []) &= (0 :: []) \\
\therefore a :: (0 :: []) &= b :: (0 :: [])
\end{align*}
\]

- Some proofs are *recursive programs*, manipulating derivations.
- E.g., proof of \((l_1 = l_2) \rightarrow (l_2 = l_3) \rightarrow (l_1 = l_3)\).
- Only *terminating* programs denote valid proofs.
- E.g., program `let trans d1 d2 = trans d1 d2` has to be rejected.
Termination of Functions Over Inductive Types

- For termination, only structure of trees is interesting.
- Structure of these trees can be represented by *inductive types*.
- More inductive types:
  - lists
  - binary trees
  - natural numbers
  - tree ordinals

Sized Inductive Types

- If $T$ is an inductive type, let $T^\alpha$ denote the set of its elements with *at most $\alpha$ constructors*.
- E.g., $\text{List}^\alpha \text{Int}$ contains integer lists of length $< \alpha$.
- $\text{List}^\omega \text{Int}$ is the type of all integer lists.
- In general, $T^\infty$ denotes the full type $T$.
- Sized list constructors:

\[
\begin{align*}
\text{nil} &\in \text{List}^{\alpha+1} \text{Int} \\
\text{cons} &\in \text{Int} \rightarrow \text{List}^\alpha \text{Int} \rightarrow \text{List}^{\alpha+1} \text{Int}
\end{align*}
\]
A recursion principle from transfinite induction

- Rule for transfinite induction:

\[
\begin{align*}
P(0) & \quad P(\alpha) \rightarrow P(\alpha + 1) \quad (\forall \alpha < \lambda. P(\alpha)) \rightarrow P(\lambda) \\
P(\beta) & \quad P(\alpha)
\end{align*}
\]

- Recursive programs via fixed-point combinator \( \text{fix } f = f (\text{fix } f) \).
- Instance \( P(\alpha) = (\text{fix } f \in A^\alpha) \):
- Use transfinite induction to define a recursive program:

\[
\begin{align*}
\text{fix } f \in A^0 & \quad f \in A^\alpha \rightarrow A^{\alpha + 1} \\
& \quad (\forall \alpha < \lambda. \text{fix } f \in A^\alpha) \rightarrow \text{fix } f \in A^\lambda
\end{align*}
\]

Handling base and limit case

- Recursion principle:

\[
\begin{align*}
\text{fix } f \in A^0 & \quad f \in A^\alpha \rightarrow A^{\alpha + 1} \\
& \quad (\text{fix } f \in \bigcap_{\alpha < \lambda} A^\alpha) \rightarrow \text{fix } f \in A^\lambda
\end{align*}
\]

- Restrict admissible types \( A^\alpha \) such that
  - \( \text{fix } f \in A^0 \) is trivial, e.g., \( A^\alpha = T^\alpha \rightarrow C \),
  - \( (\bigcap_{\alpha < \lambda} A^\alpha) \subseteq A^\lambda \).
- Specialized rule

\[
\begin{align*}
(\forall \alpha. f \in A^\alpha \rightarrow A^{\alpha + 1}) & \quad \text{fix } f \in A^\beta \rightarrow A^\alpha \text{ admissible}
\end{align*}
\]
Type-Based Termination

• When termination checking a function clause

\[ f : A^\infty \]
\[ f \, p_1 \ldots \, p_n = t(f), \]

• assume \( f \) to be of type \( A^\alpha \) on the right hand side,
• assume \( f \) of type \( A^{\alpha+1} \) on the left hand side,
• check well-typedness.
• For details and soundness, see draft of my thesis.

http://www.tcs.ifi.lmu.de/~abel/diss/

Sized Monadic Lists

In context \( \alpha : \text{ord}, M : \ast \rightarrow \ast, A : \ast \) we have

\[
\begin{align*}
\text{MList}^\alpha M A & : \ast \\
\text{nil} & : \text{MList}^{\alpha+1} M A \\
\text{cons} & : A \rightarrow M (\text{MList}^\alpha M A) \rightarrow \text{MList}^{\alpha+1} M A
\end{align*}
\]

Slide 11

Slide 12
Solving the Quiz

- With $\text{Eq } A = A \rightarrow A \rightarrow \text{Bool}$ we can type monadic list equality as follows:

$$\text{eqMList : } \forall M. \ (\forall A. \text{Eq } A \rightarrow \text{Eq } (M A)) \rightarrow \forall A. \text{Eq } A \rightarrow \text{Eq } (M\text{List}^\infty M A)$$

$$\text{Eq } (M\text{List}^\infty M A) \text{MList}^\infty M A
\frac{\text{eqMList } eqM eq (\text{cons } a \text{ mas}) (\text{cons } b \text{ mbs}) = eq a b \text{ and}}{\text{Eq } M (M\text{List}^\infty M A)}\]$$

- A bit surprisingly, the quiz can be answered affirmatively.

Related works on type-based termination

- Hughes, Pareto, Sabry (POPL 1996)  
  *Proving the correctness of reactive systems using sized types*

- Amadio and Coupet-Grimal (FoSSaCS 1998)  
  *Analysis of a guard condition in type theory*

- Xi (LICS 2001),  
  *Prg. termination verification with dep. types*

- Chin, Khoo (HOSC 2001),  
  *Calculating sized types*

- Barthe, Frade, Giménez, Pinto, Uustalu (MSCS 2004)  
  *Type-based termination of recursive definitions*

- Blanqui (RTA 2004),  
  *A type-based termination criterion for dependently-typed higher-order rewrite systems*

- Barthe et. al. (TLCA 2005): Inferring sized types

- Buchholz (2003),  
  *Recursion on non-wellfounded trees*
Acknowledgements

- Technical discussions on my thesis:
  - Klaus Aehlig
  - Thorsten Altenkirch
  - Martin Hofmann
  - John Hughes
  - Ralph Matthes
  - Thomas Streicher
  - Tarmo Uustalu

- Stipends
  - GKLI
  - CoVer

- Colleagues at Munich and Chalmers for support