Termination Casts: A Flexible Approach to Termination with General Recursion

Aaron Stump\textsuperscript{1}  Vilhelm Sjöberg\textsuperscript{2}  Stephanie Weirich\textsuperscript{2}

\textsuperscript{1}Computer Science  
The University of Iowa
\textsuperscript{2}Computer Science  
University of Pennsylvania

U.S. National Science Foundation grants: 0702545, 0910510, 0910786
Why Dependent Types Matter

Incremental verification:

<table>
<thead>
<tr>
<th>Functional Programming</th>
<th>Dependent Types</th>
<th>Tour-de-force Verification</th>
</tr>
</thead>
</table>

standard example:

\[
[10 ; 20 ; 30 ] : \text{vec int 3}
\]

append : \(\forall (A:\text{type})(n1 \ n2 : \text{nat}).\)

\[
11 : \text{vec A n1} \rightarrow 12 : \text{vec A n2} \rightarrow \text{vec A (n1+n2)}
\]

small intellectual step from list \(A\) to vec \(A\) \(n\).

much bigger leap with other formal methods!

\(^1\)cf. [Altenkirch, McBride, McKinna 2005]
The TRELLYS Project

Goal: a new functional language with dependent types.

- Project leaders: Sheard, Stump, Weirich.
- Language: CBV, mutable state, inductive types, gen. recursion.
- Implementation plan:

  - Surface lang $\rightarrow$ **elaborate** $\rightarrow$ Core lang $\rightarrow$ **compile** $\rightarrow$ Low-level code

  - surface: type inference, automated deduction.
  - core: straightforward syntax-directed type checking.

- Currently designing the core language.
- Working group: PIs + Peyton Jones, McBride, Barras, Swierstra.
Goals for TRELLYS Core Language

Expressive programs, sound proofs.

- **Programs:**
  - dependent types.
  - \texttt{type:type}, for convenient type-level data structures.
  - general recursion.
  - liberal class of datatypes.
  - decidable type checking.

- **Proofs:**
  - logically sound fragment under Curry-Howard.
  - i.e., terminating fragment.
  - allow non-constructive reasoning.
  - reasonably simple meta-theory.
Why Support General Recursion

- One argument: some programs are truly non-terminating.
  - web servers, operating systems, interpreters.
  - should be able to write and reason about these.

- Another argument: want flexibility to ignore termination.
  - dependent types $\Rightarrow$ incrementality.
  - specify and verify what you want, ignore the rest.
  - termination may not be the critical property.

- Example: versat.
  - modern SAT solver being developed in GURU by Duckki Oe.
  - it is terminating.
  - would be very painful to prove this.
  - specification of interest is soundness.
  - need the flexibility to ignore termination.
This Talk: Termination Casts and \( \text{Terms} \)

- Study for TRELLO core language.
- Type-and-effect system for termination/possible divergence.
  - Effects \( \theta ::= \downarrow | ? \).
    
    | Judgment          | Meaning of effect \( \theta \) |
    |-------------------|-------------------------------|
    | \( \Gamma \vdash t : T \downarrow \) | \( t \) is terminating      |
    | \( \Gamma \vdash t : T ? \)    | \( t \) might not be terminating |
  - Effects have well-known connection to monads [Wadler, Thiemann 2003].
  - So may connect to [Capretta 2005].
- Types:
  
  \[
  T ::= \text{nat} | \prod^\theta x : T. T' | t = t' | \text{Terminates } t
  \]
- Equality types internalize CBV-joinability.
- Termination types internalize the \( \downarrow \) termination effect.
- Casts supported with equality types, termination types.
Two Languages for $T^{\text{eq}}$

- **Unannotated** $T^{\text{eq}}$:
  - unannotated terms $t$, as they will be evaluated.
  - for example, $\lambda x.t$.
  - non-algorithmic type-assignment system $\Gamma \vdash t : T \theta$.

- **Annotated** $T^{\text{eq}}$:
  - annotated terms $a$, types $A$.
  - for example, $\lambda^\theta x : T. t$.
  - algorithmic type computation $\Gamma \Vdash a : A \theta$.
  - erasure function $|a| = t$.

**Rationale:**
- do meta-theory for unannotated, lift easily to annotated.
- equality defined in terms of unannotated terms.
- computationally irrelevant parts dropped by erasure.
- main example: casts.
Termination Casts

- **termcast** \( a \ a' \), where \( a \) proves \( a' \) terminates.
- Used to change the effect for \( a' \) from \( ? \) to \( \downarrow \).
- (From \( \downarrow \) to \( ? \) is built in.)
- External vs. internal termination:
  - Internal: judge the function to be total directly.
    
    \[
    \text{plus} : \Pi x \downarrow : \text{nat}.\Pi y \downarrow : \text{nat}.\text{nat} \downarrow
    \]
  - External: write a proof that the function is total.
    
    \[
    \text{plus} : \Pi x ? : \text{nat}.\Pi y ? : \text{nat}.\text{nat} \downarrow
    \]
    \[
    \text{plus}_{\text{tot}} : \Pi x \downarrow : \text{nat}.\Pi y \downarrow : \text{nat}.\text{Terminates} (\text{plus} \ x \ y) \downarrow
    \]
- Then use **termcast** with external totality proof.
Example Use of Termination Casts

- Suppose:

  \[
  \begin{align*}
  plus & : \Pi x ? : \text{nat}. \Pi y ? : \text{nat}. \text{nat} \downarrow \\
  plus_{\text{tot}} & : \Pi x \downarrow : \text{nat}. \Pi y \downarrow : \text{nat}. \text{Terminates} (plus x y) \downarrow \\
  mult_{\text{comm}} & : \Pi x \downarrow : \text{nat}. \Pi y \downarrow : \text{nat}. (mult x y) = (mult y x) \downarrow
  \end{align*}
  \]

- Elsewhere, suppose we want:

  \[
  (mult_{\text{comm}} (plus z z))
  \]

- As such, effect will be ?.

- To get effect to be \(\downarrow\), use a termination cast:

  \[
  (mult_{\text{comm}} \text{ termcast} (plus_{\text{tot}} z z) (plus z z))
  \]
Typing Termination Casts

Annotated terms $a ::= \ldots | \lambda^\rho x : A'.a | \text{termcast } a a' | \text{terminates } a | \ldots$

Erasure:

$|\lambda^\rho x : A'.a| = \lambda x . |a|$

$|\text{terminates } a| = \text{terminates}$

$|\text{termcast } a a'| = |a'|$

Unannotated:

$$
\Gamma, x : T' \vdash t : T \quad \Gamma \vdash \Pi^\rho x : T'.T
\overline{\Gamma \vdash \lambda x . t : \Pi^\rho x : T'.T \downarrow}
\quad
\Gamma \vdash t : T \downarrow
\overline{\Gamma \vdash \text{terminates} : \text{Terminates} \ t \downarrow}
\quad
\Gamma \vdash t : T \ ? \quad \Gamma \vdash t' : \text{Terminates} \ t \downarrow
\overline{\Gamma \vdash t : T \downarrow}
$$

Annotated:

$$
\Gamma, x : A' \vdash a : A \quad \Gamma \vdash \Pi^\rho x : A'.A
\overline{\Gamma \vdash \lambda^\rho x : A'.a : \Pi^\rho x : A'.A \downarrow}
\quad
\Gamma \vdash a : A \downarrow
\overline{\Gamma \vdash \text{terminates } a : \text{Terminates} \ a \downarrow}
\quad
\Gamma \vdash a : A \ ? \quad \Gamma \vdash a' : \text{Terminates} \ a \downarrow
\overline{\Gamma \vdash \text{termcast } a a' : A \downarrow}
$$
General Recursion, Terminating Recursion

Annotated terms $a ::= \ldots | \text{rec } f(x : A) : A' = a | \text{rec}_{\text{nat}} f(x \ p) : A = a | \ldots$

\[
\begin{align*}
\Gamma, f : \Pi?x:A'.A, \ x : A' & \vdash a : A ? \\
\Gamma & \vdash \text{rec } f(x : A') : A = a : \Pi?x:A'.A \downarrow \\
\end{align*}
\]

$p \not\in \text{fv}(|a|) \cup \text{fv}(|A|)$

\[
\begin{align*}
\Gamma, f : \Pi?x:\text{nat}.A, \ x : \text{nat}, \\
p : \Pi\downarrow x_1: \text{nat}.\Pi\downarrow p' : x = \text{Suc } x_1.\text{Terminates } (f \ x_1) & \vdash a : A \downarrow \\
\Gamma & \vdash \text{rec}_{\text{nat}} f(x \ p) : A = a : \Pi\downarrow x: \text{nat}.A \downarrow \\
\end{align*}
\]
Conclusion and Future Work

- $\text{T}_{eq}$ combines general recursion, sound proof system.
- Effect system, termination casts.
- Paper translates $\text{T}_{eq}$ to a theory $W'$.
- New design going forward:
  - termination casts only in surface language.
  - core language more primitive.
  - distinguish logical types from programming types, for meta-theory:
    - logical $T \Rightarrow [T]$ deeply defined, via reducibility.
    - programming $T \Rightarrow [T]$ shallowly defined, via type safety.
    - “freedom of speech”.
  - define deep $[T]$ only when needed for logical consistency.
- For more info:
  - “Equality, Quasi-Implicit Products, and Large Eliminations”
  - queuea9.wordpress.com (QA9 blog).