Termination Casts: A Flexible Approach to Termination with General Recursion

Aaron Stump[1] Vilhelm Sjöberg[2] Stephanie Weirich[2]

[1]Computer Science
The University of Iowa

[2]Computer Science
University of Pennsylvania

U.S. National Science Foundation grants: 0702545, 0910510, 0910786

Why Dependent Types Matter [1]

Incremental verification:

Functional Programming	↞ Dependent Types ↠	Tour-de-force Verification

- standard example:

```
[ 10 ; 20 ; 30 ] : vec int 3

append : Forall(A:type)(n1 n2 : nat).
         l1 : vec A n1 -> l2 : vec A n2 -> 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 $\longrightarrow$ *elaborate* $\longrightarrow$ Core lang $\longrightarrow$ *compile* $\longrightarrow$ 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.
  - `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 => 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 $\mathtt{T}^{\mathtt{eq}\downarrow}$

- Study for TRELLYS core language.
- Type-and-effect system for termination/possible divergence.
  - Effects $\theta ::= \downarrow \mid ?$.

Judgment	Meaning of effect θ
$\Gamma \vdash t : T \ \downarrow$	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:

$$T ::= \textbf{nat} \mid \Pi^{\theta} x : T . T' \mid t = t' \mid \textbf{Terminates}\ t$$

- Equality types internalize CBV-joinability.
- Termination types internalize the $\downarrow$ termination effect.
- Casts supported with equality types, termination types.

Two Languages for $\mathtt{T}^{\mathtt{eq}\downarrow}$

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

$$plus \ : \ \Pi x^{\downarrow} : \mathbf{nat}.\Pi y^{\downarrow} : \mathbf{nat}.\mathbf{nat} \ \downarrow$$

  - External: write a proof that the function is total.

$$\begin{array}{lcl} plus & : & \Pi x^{?} : \mathbf{nat}.\Pi y^{?} : \mathbf{nat}.\mathbf{nat} \ \downarrow \\ plus\_tot & : & \Pi x^{\downarrow} : \mathbf{nat}.\Pi y^{\downarrow} : \mathbf{nat}.\mathbf{Terminates}\ (plus\ x\ y) \ \downarrow \end{array}$$

- Then use **termcast** with external totality proof.

Example Use of Termination Casts

- Suppose:

$$\begin{array}{lcl} \mathit{plus} & : & \Pi x^{?} : \mathbf{nat}.\Pi y^{?} : \mathbf{nat}.\mathbf{nat}\ \downarrow \\ \mathit{plus\_tot} & : & \Pi x^{\downarrow} : \mathbf{nat}.\Pi y^{\downarrow} : \mathbf{nat}.\mathbf{Terminates}\ (\mathit{plus}\ x\ y)\ \downarrow \\ \mathit{mult\_comm} & : & \Pi x^{\downarrow} : \mathbf{nat}.\Pi y^{\downarrow} : \mathbf{nat}.(\mathit{mult}\ x\ y) = (\mathit{mult}\ y\ x)\ \downarrow \end{array}$$

- Elsewhere, suppose we want:

$$(\mathit{mult\_comm}\ (\mathit{plus}\ z\ z))$$

- As such, effect will be $?$.
- To get effect to be $\downarrow$, use a termination cast:

$$(\mathit{mult\_comm}\ \ \mathbf{termcast}\ (\mathit{plus\_tot}\ z\ z)\ (\mathit{plus}\ z\ z))$$

Typing Termination Casts

Annotated terms $a ::= \ \ldots \mid \lambda^\rho x\!: A'.a \mid \textbf{termcast}\ a\ a' \mid \textbf{terminates}\ a \mid \ \ldots$

Erasure:

$$\begin{array}{lcl} |\lambda^\rho x : A'.a| & = & \lambda x.|a| \\ |\textbf{terminates}\ a| & = & \textbf{terminates} \\ |\textbf{termcast}\ a\ a'| & = & |a'| \end{array}$$

Unannotated:

$$\frac{\Gamma, x : T' \vdash t : T\ \rho \qquad \Gamma \vdash \Pi^\rho x : T'.T}{\Gamma \vdash \lambda x . t : \Pi^\rho x : T'.T\ \downarrow}$$

$$\frac{\Gamma \vdash t : T\ \downarrow}{\Gamma \vdash \textbf{terminates} : \textbf{Terminates}\ t\ \downarrow}$$

$$\frac{\Gamma \vdash t : T\ ? \qquad \Gamma \vdash t' : \textbf{Terminates}\ t\ \downarrow}{\Gamma \vdash t : T\ \downarrow}$$

Annotated:

$$\frac{\Gamma, x : A' \Vdash a : A\ \rho \qquad \Gamma \Vdash \Pi^\rho x : A'.A}{\Gamma \Vdash \lambda^\rho x\!: A'.a : \Pi^\rho x : A'.A\ \downarrow}$$

$$\frac{\Gamma \Vdash a : A\ \downarrow}{\Gamma \Vdash \textbf{terminates}\ a : \textbf{Terminates}\ a\ \downarrow}$$

$$\frac{\Gamma \Vdash a : A\ ? \qquad \Gamma \Vdash a' : \textbf{Terminates}\ a\ \downarrow}{\Gamma \Vdash \textbf{termcast}\ a\ a' : A\ \downarrow}$$

General Recursion, Terminating Recursion

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

$$\frac{\Gamma\,,\ f\ :\ \Pi^{?} x : A'.A\,,\ x\ :\ A' \Vdash a : A\ ?}{\Gamma \Vdash \textbf{rec}\ f(x{:}A') {:}\ A = a : \Pi^{?} x : A'.A\ \downarrow}$$

$$\frac{\begin{array}{l} p \notin \textbf{fv}(|a|) \cup \textbf{fv}(|A|) \\ \Gamma\,,\ f\ :\ \Pi^{?} x : \textbf{nat}.A\,,\ x\ :\ \textbf{nat}\,, \\ p\ :\ \Pi^{\downarrow} x_1 : \textbf{nat}.\Pi^{\downarrow} p' : x\ =\ \textbf{Suc}\, x_1.\textbf{Terminates}\ \ (\, f\, x_1\,) \Vdash a : A\ \downarrow \end{array}}{\Gamma \Vdash \textbf{rec}_{\textbf{nat}}\ f(x\ p){:}\ A = a : \Pi^{\downarrow} x : \textbf{nat}.A\ \downarrow}$$

Conclusion and Future Work

- $\mathtt{T}^{\mathtt{eq}\downarrow}$ combines general recursion, sound proof system.
- Effect system, termination casts.
- Paper translates $\mathtt{T}^{\mathtt{eq}\downarrow}$ 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 => $[\![T]\!]$ deeply defined, via reducibility.
    - programming T => $[\![T]\!]$ 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).