An Extension of Martin-Löf Type Theory with Sized Types

Andreas Abel[1] and Théo Winterhalter[2]

[1] Department of Computer Science and Eng., Gothenburg University, Sweden
abela@chalmers.se
[2] École Normale Supérieure de Cachan, France
theo.winterhalter@ens-cachan.fr

Abstract

We present a dependent type theory for which termination checking is entirely type-based, through the use of sized types. Sizes are absent from terms to ensure they are irrelevant for computation and reasoning. The novelty of our approach is the combination of first class size quantification $\forall\, i \to T$ and dependent types, justified by a predicative semantics.

Proof assistants based on dependent types, such as the very successful implementation Coq [9], rely on a termination checker to validate proofs by induction, which are represented as purely functional, recursive programs. The prevailing structural termination checkers have the main drawback that they lack compositionality, i. e., one cannot abstract out arbitrary parts of a program without leaving the termination checker clueless.

Type-based termination – suggested for functional programming [8] and dependent type theory [6, 3] already two decades ago – inherits the compositionality of polymorphic type systems for termination checking. Sized types allow to encode size-change behavior of functions in their types, and this refined type signature can be used for termination checking later without referring to the code of these functions. This way, the termination checker does not need to inline code and break abstraction barriers, and it works well with abstract and modularized developments. Following experiments with the prototype MiniAgda [1], the proof assistant Agda [2] is the first practical system utilizing sized types for termination checking.

However, in the presence of dependent types, size witnesses in terms may get in the way of equality, preventing the user from proving expected properties about their programs. Consider the function gscale on sized natural numbers, which is a generalization of multiplication and division. For instance, gscale (subtract 1) (add 2) implements scaling by $\frac{3}{2}$. It can be implemented in Agda using first-class size polymorphism for argument f. Note that $\uparrow$ is the size successor. The function scale1, where both f and g are the identity, should behave as the identity function.

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{Nat} : (i : \mathsf{Size}) \to \mathsf{Set}\ \mathsf{where} \\
\quad \mathsf{zero} \ : \ \forall\, i \qquad\qquad\quad \to \mathsf{Nat}\ (\uparrow i) \\
\quad \mathsf{suc} \ \ : \ \forall\, i\ (n : \mathsf{Nat}\ i) \ \to \mathsf{Nat}\ (\uparrow i) \\
\\
\mathsf{gscale} : (\forall\ i \to \mathsf{Nat}\ i \to \mathsf{Nat}\ i) \to (\mathsf{Nat}\ \infty \to \mathsf{Nat}\ \infty) \to \forall\ i \to \mathsf{Nat}\ i \to \mathsf{Nat}\ \infty \\
\mathsf{gscale}\ f\ g\ .(\uparrow j)\ (\mathsf{zero}\ j) \quad\ = \mathsf{zero}\ \infty \\
\mathsf{gscale}\ f\ g\ .(\uparrow j)\ (\mathsf{suc}\ j\ n) \ = g\ (\mathsf{suc}\ \infty\ (\mathsf{gscale}\ f\ g\ j\ (f\ j\ n))) \\
\\
\mathsf{scale1} = \mathsf{gscale}\ (\lambda\ \_\ \ x \to x)\ (\lambda\ x \to x)
\end{array}
$$

However, in a proof of scale1 $i\ n\ \equiv\ n$ by induction on n we get stuck in both cases, since the size arguments on the constructors zero and suc are not unifiable ($\infty \neq j$). In the following proof skeleton, we only show the goal and its reduced form for both cases.

$$
\begin{array}{lcl}
\mathsf{scale1\text{-}id} : \forall\, i\, (n : \mathsf{Nat}\ i) \to\ \mathsf{scale1}\ i\ n \equiv n & & \\
\mathsf{scale1\text{-}id}\ .(\uparrow j)\ (\mathsf{zero}\ j) & = & \mathsf{goal}\ (\mathsf{scale1}\ (\uparrow j)\ (\mathsf{zero}\ j) \equiv \mathsf{zero}\ j) \\
 & & \mathsf{goal}\ (\mathsf{zero}\ \infty \equiv \mathsf{zero}\ j) \\
\mathsf{scale1\text{-}id}\ .(\uparrow j)\ (\mathsf{suc}\ j\ n) & = & \mathsf{goal}\ (\mathsf{scale1}\ (\uparrow j)\ (\mathsf{suc}\ j\ n) \equiv \mathsf{suc}\ j\ n) \\
 & & \mathsf{goal}\ (\mathsf{suc}\ \infty\ (\mathsf{scale1}\ j\ n) \equiv \mathsf{suc}\ j\ n)
\end{array}
$$

We aspire a language where sizes are irrelevant for computation and definitional equality. The purpose of this work is to exhibit a calculus with size annotations omitted at the term level and only present on the type level, handling first-class quantification by subtyping.

This work consists of the definition of such a system along with a bidirectional algorithm for type checking and an algorithmic equality directed by size-erased types. The language we present extends Martin-Löf Type Theory by sized natural numbers, case distinction, and size-based recursion over natural numbers. In our effort to make sizes irrelevant, size abstraction and application is silent in terms which means there are no size arguments in terms that could get in the way of equality. Unlike previous works on sized dependent types [5, 4, 7, 10], we permit arbitrary rank (not only ML-style) size quantification $\forall\, i \to T$ in types.

We provide algorithms for evaluation and conversion checking. The use of sized-erased terms in algorithmic equality allows it to be syntax-directed for inferable terms.

We introduce a logical relation indexed by environments of ordinals to deduce normalization and subject reduction and eventually soundness of conversion checking. We then continue to prove its completeness and termination. Finally, we are able to derive a bidirectional type checker, assuming we have an algorithm to guess sizes verifying linear constraints.

References

[1] Andreas Abel. MiniAgda: Integrating sized and dependent types. In *Partiality and Recursion (PAR 2010)*, volume 43 of *EPTCS*, pages 14–28, 2010.

[2] AgdaTeam. The Agda Wiki, 2015.

[3] Roberto M. Amadio and Solange Coupet-Grimal. Analysis of a guard condition in type theory (extended abstract). In *FoSSaCS'98*, volume 1378 of *LNCS*, pages 48–62. Springer, 1998.

[4] Gilles Barthe, Benjamin Grégoire, and Fernando Pastawski. CIC^: Type-based termination of recursive definitions in the Calculus of Inductive Constructions. In *LPAR'06*, volume 4246 of *LNCS*, pages 257–271. Springer, 2006.

[5] Frédéric Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. In *RTA'04*, volume 3091 of *LNCS*, pages 24–39. Springer, 2004.

[6] Eduardo Giménez. Structural recursive definitions in type theory. In *ICALP'98*, volume 1443 of *LNCS*, pages 397–408. Springer, 1998.

[7] Benjamin Grégoire and Jorge Luis Sacchini. On strong normalization of the calculus of constructions with type-based termination. In *LPAR'10*, volume 6397 of *LNCS*, pages 333–347. Springer, 2010.

[8] John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *POPL'96*, pages 410–423. ACM, 1996.

[9] INRIA. *The Coq Proof Assistant Reference Manual*. INRIA, version 8.5 edition, 2016.

[10] Jorge Luis Sacchini. Type-based productivity of stream definitions in the calculus of constructions. In *LICS'13*, pages 233–242. IEEE CS Press, 2013.