Normalization by Evaluation for Sized Dependent Types

Andreas Abel[1], Andrea Vezzosi[1], and Théo Winterhalter[2]

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

Abstract

Sized types have been developed to make termination checking more perspicuous, more powerful, and more modular by integrating termination into type checking. In dependently-typed proof assistants where proofs by induction are just recursive functional programs, the termination checker is an integral component of the trusted core, as validity of proofs depends on termination. However, a rigorous integration of full-fledged sized types into dependent type theory is lacking so far. Such an integration is non-trivial, as explicit sizes in proof terms might get in the way of equality checking, making terms appear distinct that should have the same semantics. In this work, we integrate dependent types and sized types with higher-rank size polymorphism, which is essential for generic programming and abstraction. We introduce a size quantifier $\forall$ which lets us ignore sizes in terms for equality checking, alongside with a second quantifier Π for abstracting over sizes that do affect the semantics of types and terms. Judgmental equality is decided by an adaptation of normalization-by-evaluation.

Agda [5] features first-class size polymorphism [1] in contrast to theoretical accounts of sized dependent types [6, 7, 10] who typically just have prenex (ML-style) size quantification. Consequently, Agda's internal language contains size expressions in terms wherever a size quantifier is instantiated. However, these size expressions, which are not unique due to subtyping, can get in the way of reasoning about sizeful programs. Consider the type of sized natural numbers.

```
data Nat : Size → Set where
  zero : ∀ i → Nat (i + 1)
  suc  : ∀ i → Nat i → Nat (i + 1)
```

We define subtraction $x \dot{-} y$ on natural numbers, sometimes called the monus function, which computes $\max(0, x - y)$. It is defined by induction on the size j of the second argument y, while the output is bounded by size i of the first argument x. (The input-output relation of monus is needed for a natural implementation of Euclidean divison.)

```
monus : ∀ i → Nat i → ∀ j → Nat j → Nat i
monus i       x       .(j + 1) (zero j)   = x
monus .(i + 1) (zero i)  .(j + 1) (suc j y) = zero i
monus .(i + 1) (suc i x) .(j + 1) (suc j y) = monus i x j y
```

We wish to prove that subtracting x from itself yields 0, by induction on x. The case $x = 0$ should be trivial, as $x \dot{-} 0 = x$ by definition, hence, $0 \dot{-} 0 = 0$. A simple proof by reflexivity should suffice. However, the goal shows a mismatch between size ∞ and size i coming from the computation of monus $(i + 1)$ (zero i) $(i + 1)$ (zero i).

```
monus-diag : ∀ i → (x : Nat i) → zero ∞ ≡ monus i x i x
monus-diag .(i + 1) (zero i)  = {! zero ∞ ≡ zero i !} -- goal
monus-diag .(i + 1) (suc i x) = monus-diag i x
```

The proof could be completed by an application of reflexivity if Agda ignored sizes where they act as *type argument*, i. e., in constructors and term-level function applications, but not in types where they act as regular argument, e. g., in $\mathsf{Nat}\,i$.

The problem is solved by distinguishing relevant (Π) from irrelevant ($\forall$) size quantification. The relevant quantifier is the usual dependent function space over sizes. In particular, the congruence rule for size application requires matching size arguments:

$$\frac{\Gamma \vdash t = t' : \Pi i.\, T \qquad \Gamma \vdash a : \mathsf{Size}}{\Gamma \vdash t\,a = t'\,a : T[a/i]}$$

Typically, the relevant quantifier is used in types of types, for instance, in its non-dependent form, in $\mathsf{Nat} : \mathsf{Size} \to \mathsf{Set}$. In contrast, the irrelevant size quantifier is used in types of programs and ignores sizes in size applications. The rules for application, while Church-style, *de facto* implement Curry-style quantification:

$$\frac{\Gamma \vdash t = t' : \forall i.\, T \qquad \Gamma^{\oplus} \vdash a, a', b : \mathsf{Size}}{\Gamma \vdash t\,a = t'\,a' : T[b/i]} \qquad \frac{\Gamma \vdash t : \forall i.\, T \qquad \Gamma^{\oplus} \vdash a, b : \mathsf{Size}}{\Gamma \vdash t\,a : T[b/i]}$$

Further, the size arguments are scoped in the *resurrected* [9] context $\Gamma^{\oplus}$ and, thus, are allowed to mention irrelevant size variables. Those are introduced by irrelevant size abstraction and marked by the $\div$-symbol in the context. In contrast, the quantified size variable may occur relevantly in the *type*.

$$\frac{\Gamma, i \div \mathsf{Size} \vdash t : T}{\Gamma \vdash \lambda i.\, t : \forall i.\, T} \qquad \frac{\Gamma, i : \mathsf{Size} \vdash T : \mathsf{Set}}{\Gamma \vdash \forall i.\, T : \mathsf{Set}}$$

The lack of type unicity arising from the size application rule has prevented us from adopting the usual incremental algorithm for equality checking [8, 3]. However, we have succeeded to employ normalization by evaluation [2] for deciding judgmental equality [4].

References

[1] Andreas Abel. Type-based termination, inflationary fixed-points, and mixed inductive-coinductive types. In *FICS 2012*, volume 77 of *EPTCS*, pages 1–11, 2012. Invited talk.

[2] Andreas Abel, Thierry Coquand, and Peter Dybjer. Normalization by evaluation for Martin-Löf Type Theory with typed equality judgements. In *LICS'07*, pages 3–12. IEEE CS Press, 2007.

[3] Andreas Abel and Gabriel Scherer. On irrelevance and algorithmic equality in predicative type theory. *LMCS*, 8(1:29):1–36, 2012. TYPES'10 special issue.

[4] Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. Normalization by evaluation for sized dependent types. Accepted for presentation at ICFP, 2017.

[5] AgdaTeam. The Agda Wiki, 2017.

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

[7] 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.

[8] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. *ACM TOCL*, 6(1):61–101, 2005.

[9] Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *LICS'01*, pages 221–230. IEEE CS Press, 2001.

[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.