Well-Founded Sized Types in the Calculus of Constructions

Jorge Luis Sacchini[1]

Carnegie Mellon University

21 May 2015

[1] This publication was made possible by a JSREP grant (JSREP 4-004-1-001) from the Qatar National Research Fund (a member of The Qatar Foundation). The statements made herein are solely the responsibility of the author.

What are sized types?

- (Coinductive) types annotated with size information: $I\langle s \rangle$
- Used for ensuring termination/productivity: size information tracks size of (co)recursive calls
- Inductive types: upper bound on the size of elements

$$\mathsf{list}\langle s \rangle\ A$$

E.g. $\overbrace{[x_1, \ldots, x_n]}^{\leq s}$

- Coinductive types: lower bound on the number of elements that can be produced

$$\mathsf{stream}\langle s \rangle\ A$$

E.g. $\overbrace{[x_1, x_2, \ldots, x_n, \ldots]}^{\geq s}$

Sized Types vs. Syntactic guards

- Currently, Coq/Agda use syntactic guards to ensure termination/productivity
- Sized types provide several advantages:
  - Expressiveness: size information exported in type:

$$\mathsf{nat}\langle s \rangle \to \mathsf{nat}\langle s \rangle$$

  - Compositionality
  - Semantically intuitive

Sized Types for Termination/Productivity

Typing rule:

$$\frac{\Gamma, f : [\jmath < \imath]\, T\langle\jmath\rangle \vdash M : T\langle\imath\rangle \qquad \imath \text{ fresh}}{\Gamma \vdash (\mathsf{rec}\, f\, \imath : T\langle\imath\rangle := M) : [\imath]\, T\langle\imath\rangle}$$

Example: $T = \mathsf{nat} \to \mathsf{nat}$

$$\frac{\Gamma, f : [\jmath < \imath]\, \mathsf{nat}\langle\jmath\rangle \to \mathsf{nat}\langle\jmath\rangle \vdash M : \mathsf{nat}\langle\imath\rangle \to \mathsf{nat}\langle\imath\rangle}{\Gamma \vdash (\mathsf{rec}\, f\, \imath : \mathsf{nat}\langle\imath\rangle \to \mathsf{nat}\langle\imath\rangle := M) : [\imath]\, \mathsf{nat}\langle\imath\rangle \to \mathsf{nat}\langle\imath\rangle}$$

Example: $T = \mathsf{stream}\, A$

$$\frac{\Gamma, f : [\jmath < \imath]\, \mathsf{stream}\langle\jmath\rangle\, A \vdash M : \mathsf{stream}\langle\imath\rangle\, A}{\Gamma \vdash (\mathsf{corec}\, f\, \imath : \mathsf{stream}\langle\imath\rangle\, A := M) : [\imath]\, \mathsf{stream}\langle\imath\rangle\, A}$$

Goal

Accept corecursive definitions that use tail:

$$\mathsf{fib} = \mathsf{cons}\,0\,(\mathsf{cons}\,1\,(\mathsf{sum}\,\mathsf{fib}\,(\mathsf{tail}\,\mathsf{fib})))$$
$$\mathsf{fib} = 0, 1, 1, 2, 3, 5, \ldots$$

Reject non-productive programs like

$$\mathsf{bad} = \mathsf{cons}\,0\,(\mathsf{tail}\,\mathsf{bad})$$

Well-founded sized types

(Abel and Pientka, ICFP 2013)

- Size algebra:

$$s ::= \imath \mid \widehat{s} \mid \infty$$

- Size contexts:

$$\mathcal{K} ::= \cdot \mid \mathcal{K}, \widehat{\imath} \sqsubseteq s$$

- Constraints:

$$\mathcal{K} \vdash s \sqsubseteq r$$

$$\frac{}{\mathcal{K} \vdash s \sqsubseteq \widehat{s}} \qquad \frac{}{\mathcal{K} \vdash s \sqsubseteq \infty} \qquad \frac{\widehat{\imath} \sqsubseteq s \in \mathcal{K}}{\mathcal{K} \vdash \widehat{\imath} \sqsubseteq s}$$

$$\frac{\mathcal{K} \vdash s \sqsubseteq r' \qquad \mathcal{K} \vdash r' \sqsubseteq r}{\mathcal{K} \vdash s \sqsubseteq r} \qquad \frac{\mathcal{K} \vdash s \sqsubseteq r}{\mathcal{K} \vdash \widehat{s} \sqsubseteq \widehat{r}} \qquad \frac{\mathcal{K} \vdash \widehat{s} \sqsubseteq \widehat{r}}{\mathcal{K} \vdash s \sqsubseteq r}$$

Inductive Types

Defined as in CIC (constructor types, strictly positive, ...)

$$\begin{aligned}&\mathsf{Inductive}\ \mathsf{nat} : \mathsf{Type} :=\\ &\quad \mid \mathsf{O} : \mathsf{nat}\\ &\quad \mid \mathsf{S} : \mathsf{nat} \to \mathsf{nat}\end{aligned}$$

Typing judgment: $\mathcal{K}; \Gamma \vdash t : T$

$$\frac{\mathcal{K}; \Gamma \vdash t : \mathsf{nat}\langle r \rangle \qquad \mathcal{K} \vdash \widehat{r} \sqsubseteq s}{\mathcal{K}; \Gamma \vdash \mathsf{S}\,[r]\,t : \mathsf{nat}\langle s \rangle}$$

Intuitively: $\mathsf{nat}\langle s \rangle \approx \sum_{i<s} \mathsf{nat}\langle i \rangle$.

Elimination of inductive types

Typing judgment: $\mathcal{K}, \Gamma \vdash t : T$

$$\frac{\mathcal{K}, \widehat{\kappa} \sqsubseteq s; \Gamma \vdash u_1 : U \qquad \mathcal{K}; \Gamma \vdash t : \mathsf{nat}\langle s \rangle \qquad \mathcal{K}, \widehat{\kappa} \sqsubseteq s; \Gamma, x{:}\mathsf{nat}\langle \kappa \rangle \vdash u_2 : U \qquad \kappa \notin \mathsf{SV}(u_i)}{\mathcal{K}; \Gamma \vdash (\mathsf{case}\ t\ \mathsf{of}\ \mathsf{O}\ [\kappa] \Rightarrow u_1; \mathsf{S}\ [\kappa]\ x \Rightarrow u_2) : U}$$

Example: addition

$$\begin{array}{l} \mathsf{rec}\ \mathsf{plus}\ \imath(x : \mathsf{nat}\langle \imath \rangle)(y : \mathsf{nat}) : \mathsf{nat} := \\ \quad \mathsf{case}\ x\ \mathsf{of} \\ \quad \mid \mathsf{O}\ [\kappa] \Rightarrow y \\ \quad \mid \mathsf{S}\ [\kappa]\ x \Rightarrow \mathsf{S}(\mathsf{plus}[\kappa]\ x\ y) \qquad\qquad (\widehat{\kappa} \sqsubseteq \imath) \end{array}$$

Example

```
log2 O           = O
log2 (S O)       = O
log2 (S (S n)) = S (log2 (S (half n)))
```

$\mathsf{half} : [\imath]\,\mathsf{nat}\langle\imath\rangle \to \mathsf{nat}\langle\imath\rangle$

$\mathsf{rec}\,\mathsf{log2}\,\imath(x : \mathsf{nat}\langle\imath\rangle) : \mathsf{nat}\langle\imath\rangle :=$

$\quad \mathsf{case}\,x\,\mathsf{of}$

$\quad\quad \mid \mathsf{O}\,[\kappa] \Rightarrow \mathsf{O}\,[\kappa]$

$\quad\quad \mid \mathsf{S}\,[\kappa]\,x \Rightarrow \mathsf{case}\,x\,\mathsf{of}$ $\qquad (\widehat{\kappa} \sqsubseteq \imath)$

$\quad\quad\quad\quad \mid \mathsf{O}\,[\jmath] \Rightarrow \mathsf{O}\,[\jmath]$

$\quad\quad\quad\quad \mid \mathsf{S}\,[\jmath]\,x' \Rightarrow \mathsf{S}\,[\kappa]\,(\mathsf{log2}[\kappa]\,(\mathsf{S}[\jmath]\,(\mathsf{half}[\jmath]\,x'))) \quad (\widehat{\jmath} \sqsubseteq \kappa, \widehat{\kappa} \sqsubseteq \imath)$

(Co)inductive Types

$$\begin{array}{l}\mathsf{CoInductive\ stream}\,(A : \mathsf{Type}) : \mathsf{Type} := \\ \quad \mid \mathsf{cons} : A \times \mathsf{stream}\,A \to \mathsf{stream}\,A\end{array}$$

$$\frac{\mathcal{K}, \widehat{\imath} \sqsubseteq s; \Gamma \vdash t : A \qquad \mathcal{K}, \widehat{\imath} \sqsubseteq s; \Gamma \vdash u : \mathsf{stream}\langle \imath \rangle\, A \qquad \imath \notin \mathsf{SV}(u)}{\mathcal{K}; \Gamma \vdash \mathsf{cons}(\imath.\langle t, u \rangle) : \mathsf{stream}\langle s \rangle\, A}$$

Intuitively: $\mathsf{stream}\langle \imath \rangle\, A \approx \prod_{\imath < s} \mathsf{stream}\langle \imath \rangle\, A$

Elimination of coinductive types

Size application:

$$\frac{\mathcal{K}; \Gamma \vdash t : \mathsf{stream}\langle s \rangle\, A \qquad \mathcal{K} \vdash \widehat{r} \sqsubseteq s}{\mathcal{K}; \Gamma \vdash t\,\{r\} : A \times \mathsf{stream}\langle r \rangle\, A}$$

Case analysis:

$$\frac{\mathcal{K}; \Gamma \vdash t : \mathsf{stream}\langle s \rangle\, A \qquad \mathcal{K} \vdash \widehat{r} \sqsubseteq s \qquad \mathcal{K}; \Gamma, x{:}A, y{:}\mathsf{stream}\langle r \rangle\, A \vdash u : U}{\mathcal{K}; \Gamma \vdash (\mathsf{case}\, t\,\{r\}\, \mathsf{of}\, \mathsf{cons}\, x\, y \Rightarrow u) : U}$$

(Co)inductive Types

Example:

$$\mathsf{sum} : [\imath]\,\mathsf{stream}\langle\imath\rangle\,\mathsf{nat} \to \mathsf{stream}\langle\imath\rangle\,\mathsf{nat} \to \mathsf{stream}\langle\imath\rangle\,\mathsf{nat}$$

$$\mathsf{tail}[r] : \mathsf{stream}\langle s\rangle\,\mathsf{nat} \to \mathsf{stream}\langle r\rangle\,\mathsf{nat} \qquad (\text{if } \widehat{r} \sqsubseteq s)$$

$$\mathsf{corec}\,\mathsf{fib}\,\imath : \mathsf{stream}\langle\imath\rangle\,\mathsf{nat} :=$$
$$\mathsf{cons}(\jmath.\langle 0, \mathsf{cons}(\kappa.\langle 1, \mathsf{sum}[\kappa]\ \ \mathsf{fib}[\kappa]\ \ (\mathsf{tail}[\kappa]\ \ \mathsf{fib}[\jmath])\rangle)\rangle) \quad (\widehat{\kappa} \sqsubseteq \jmath, \widehat{\jmath} \sqsubseteq \imath)$$

Evaluation

- Unfolding of (co)recursive definitions needs to be restricted:

$$(\mathsf{rec}\, f := t) \to t[\mathsf{rec}\, f := t/f]$$

- How do we evaluate (co)recursive definitions?
- Strongly normalizing on open terms (decidability of conversion)

Evaluation

Idea 1: use Coq evaluation rules

- Recursive definitions are unfolded when the recursive argument (given by `struct`) is in constructor form:

$$(\mathrm{rec}\, f := t)\, \vec{u}\, (C\, \vec{a}) \to t[\mathrm{rec}\, f := t/f]\, \vec{u}\, (C\, \vec{a})$$

- Counterexample:

$$\begin{array}{l}
\mathrm{rec}\, \mathsf{log2}\, (x : \mathsf{nat}) : \mathsf{nat} := \\
\quad \mathrm{case}\, x\, \mathrm{of} \\
\quad \mid \mathsf{O} \Rightarrow \mathsf{O} \\
\quad \mid \mathsf{S}\, x \Rightarrow \mathrm{case}\, x\, \mathrm{of} \\
\qquad\qquad \mid \mathsf{O} \Rightarrow \mathsf{O} \\
\qquad\qquad \mid \mathsf{S}\, x' \Rightarrow \mathsf{S}(\mathsf{log2}(\mathsf{S}\, (\mathsf{half}\, x')))
\end{array}$$

Evaluation

Idea 1: use Coq evaluation rules

- Corecursive definitions are unfolded inside case analysis:

$$\mathsf{case}\,(\mathsf{corec}\,f := t)\,\vec{u}\,\mathsf{of}\ldots \to \mathsf{case}\,t[\mathsf{corec}\,f := t/f]\,\vec{u}\,\mathsf{of}\ldots$$

- Counterexample:

$$\begin{aligned}\mathsf{tail}\,\mathsf{fib} &\to \mathsf{cons}\,0\,(\mathsf{cons}\,1\,(\mathsf{sum}\,\mathsf{fib}\,(\mathsf{tail}\,\mathsf{fib})))\\ &\to \mathsf{cons}\,0\,(\mathsf{cons}\,1\,(\mathsf{sum}\,\mathsf{fib}\,(\mathsf{cons}\,1\,(\mathsf{sum}\,\mathsf{fib}\,(\mathsf{tail}\,\mathsf{fib})))))\\ &\to \ldots\end{aligned}$$

- Furthermore: it does not satisfy SR

Evaluation

Idea 2: reduce only on known sizes

$$(\mathrm{rec}\, f := t)[\infty] \to t[\mathrm{rec}\, f := t/f]$$

- Satisfies SR, SN
- Sizes get in the way of proofs (cumbersome)
- Size inference is hopeless

Evaluation

Idea 3: guarded unfolding

- Every (co)recursive call is guarded by a size:

$$\mathsf{rec}\,\mathsf{f}\,\imath : \ldots := \ldots \mathsf{case}\,t\,\mathsf{of}$$
$$\mid \mathsf{C}\,[\kappa]\,\vec{x} \Rightarrow \ldots \mathsf{f}[\kappa] \ldots \qquad \widehat{\kappa} \sqsubseteq \imath$$

- Example:

$$\mathsf{rec}\,\mathsf{plus}\,\imath(x : \mathsf{nat}\langle\imath\rangle)(y : \mathsf{nat}) : \mathsf{nat} :=$$
$$\mathsf{case}\,x\,\mathsf{of}$$
$$\mid \mathsf{O}\,[\kappa] \Rightarrow y$$
$$\mid \mathsf{S}\,[\kappa]\,x \Rightarrow \mathsf{S}(\mathsf{plus}[\kappa]\,x\,y)$$

Evaluation

Idea 3: guarded unfolding

- Every (co)recursive call is guarded by a size:

$$\begin{array}{l} \mathsf{corec}\ \mathsf{f}\ \imath : \ldots := \ldots \\ \quad \mathsf{cons}(\kappa.(\ldots \mathsf{f}[\kappa] \ldots) \qquad \widehat{\kappa} \sqsubseteq \imath \end{array}$$

- Example:

$$\begin{array}{l} \mathsf{corec}\ \mathsf{fib}\ \imath : \mathsf{stream}\langle \imath \rangle\ \mathsf{nat} := \\ \quad \mathsf{cons}(\jmath.\langle 0, \mathsf{cons}(\kappa.\langle 1, \mathsf{sum}[\kappa]\ \mathsf{fib}[\kappa]\ (\mathsf{tail}[\kappa]\ \mathsf{fib}[\jmath])\rangle)\rangle) \end{array}$$

Evaluation

Idea 3: guarded unfolding

- Introduce blocked (co)recursive function:

$$t ::= \ldots \mid \uparrow \mathsf{rec}\, f\, \imath\, \Delta : T := t$$

- Unfolding blocks (co)recursive calls:

$$(\mathsf{rec}\, f := t) \to t[\uparrow \mathsf{rec}\, f := t/f]$$

- Example:

$$\begin{aligned} \mathsf{plus} \to \lambda(x\, y : \mathsf{nat}).\mathsf{case}\, x\, \mathsf{of} \\ \mid \mathsf{O}\, [\kappa] \Rightarrow y \\ \mid \mathsf{S}\, [\kappa]\, x \Rightarrow \mathsf{S}\, ((\uparrow \mathsf{rec}\, \mathsf{plus}\, \ldots)[\kappa]\, x\, y) \end{aligned}$$

Reducing case analysis unblocks recursive calls

Evaluation

Idea 3: guarded unfolding
Unfolding blocks (co)recursive calls:

$$(\mathsf{corec}\, f := t) \to t[\uparrow \mathsf{corec}\, f := t/f]$$

Example:

$$\mathsf{fib} \to \mathsf{cons}(\jmath.\langle 0, \mathsf{cons}(\kappa.\langle 1, \mathsf{sum}(\uparrow(\mathsf{corec}\, \mathsf{fib}\ldots)[\kappa] \\ (\mathsf{tail}\,(\uparrow(\mathsf{corec}\, \mathsf{fib}\ldots)[\jmath])\rangle)\rangle)$$

Size application unblocks corecursive calls

Current state

- Metatheory developed for ∞-reduction
  - SR (standard proof)
  - SN (based on Λ-set model)
- Metatheory partially developed for guarded reduction
  - SR (standard proof)
- Size-inference implemented using guarded reduction (slow)
- More details at
  `http://www.qatar.cmu.edu/~sacchini/well-founded/`

Future work

- Develop (better) alternative reduction rules (copatterns?)
- More efficient constraint solving algorithms
- Complete metatheory (SN for guarded reduction)
- Extend to generic well-founded predicates

http://www.qatar.cmu.edu/~sacchini/well-founded/

Thank you!