Sized Types for Program Generation

Caspar Popova

9/18/2025 – PLUM Reading Group

Big picture

- Compiler stress-testing: generating programs that test particular optimizations to look for bugs or performance misses

Existing work

- Orange (Nagai 2014): arithmetic optimization
- YARPGEN1 (Livinskii 2020): arithmetic
- YARPGEN2 (Livinskii 2023) : loops
- This project: recursion!

Focus: recursion optimizations

- Loopification / recursion-to-iteration / recursion elimination
- Recursion fusion
- Mutual recursion elimination
- Recursion twisting

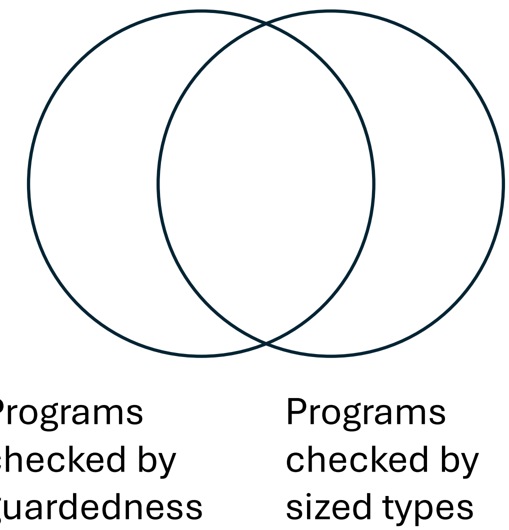
Locality Transformations for Nested Recursive Iteration Spaces – Sundararajah 2017
Control Flow Analysis for Recursion Removal – Himpe 2003

Challenge

- Programs generated for compiler testing generally need to terminate (especially differential testing)
- → how do we generate recursive, terminating programs?
- YARPGEN2's approach to terminating loops is unsatisfying, so we need another approach

Termination checking

- Guardedness predicate
- Sized types
- Well-founded relations
- Recursors/eliminators of inductive types



https://github.com/rocq-prover/rocq/wiki/CoqTerminationDiscussion

Termination checking

Let $f = \lambda x.e$ be a fixpoint where $f : d \rightarrow \theta$ and d is an inductive type …

- Guard predicate:
  - Used in Rocq & Agda
  - Condition (e can only make recursive calls to f on arguments **structurally smaller** than x) enforced **syntactically**
  - Unfold definitions, do reductions
  - **Sensitive to syntax & not compositional**
- Sized types
  - Inhabitants of inductive datatypes are given a size
  - Condition (e can only make recursive calls on that are **size smaller** than x) enforced via **types**
  - **Compositional**
  - Inspired by set-theoretic semantics

Is Sized Typing for Coq Practical? - Jonathan Chan
A Tutorial on Type-Based Termination – Gilles Barthe

Selected examples in Rocq

- Programs where sized types work better:
  - Minus/div composition
- Guardedness works better:
  - GCD (doesn’t have a single decreasing argument
- There are some programs that both fail to check without modifications
  - Ackermann

Sized types: tutorial

$$\mathcal{S} \ ::= \ \mathcal{V}_{\mathcal{S}} \ \mid \ \infty \ \mid \ \widehat{\mathcal{S}}$$

Size algebra

$$\frac{\vdash n \ : \ \mathsf{Nat}^{p}}{\vdash \mathsf{s}\, n \ : \ \mathsf{Nat}^{p+1}}$$

A different notation for successor constructor

$$\frac{}{\Gamma \vdash \mathtt{O} : \mathtt{Nat}^{\hat{s}}} \qquad \frac{\Gamma \vdash n : \mathtt{Nat}^{s}}{\Gamma \vdash \mathtt{S}\, n : \mathtt{Nat}^{\hat{s}}}$$

Natural constructors

$$\mathcal{S} ::= \mathcal{V}_{\mathcal{S}} \mid \infty \mid \widehat{\mathcal{S}}$$

Size algebra

Examples

$$\textbf{Inductive}\ \ \mathsf{Nat} := \mathsf{o} : \mathsf{Nat}^{\hat{\imath}} \mid \mathsf{s} : \mathsf{Nat}^{\imath} \to \mathsf{Nat}^{\hat{\imath}}$$

```
+    : [Nat i; Nat Inf] --> Nat Inf
-    : [Nat i; Nat Inf] --> Nat i
div  : [Nat i; Nat Inf] --> Nat i
```

$$\textbf{Inductive}\ \ \mathsf{List}\ X := \mathsf{nil} : \mathsf{List}^{\hat{\imath}}\ X \mid \mathsf{cons} : X \to \mathsf{List}^{\imath}\ X \to \mathsf{List}^{\hat{\imath}}\ X$$

```
tail : [List i X] --> X
take : [List i X; Nat Inf] --> List i X
append : [List i X; List Inf X] --> List Inf X
```

Subsize relation for subtyping

$$\mathcal{S} \ ::= \ \mathcal{V}_{\mathcal{S}} \mid \infty \mid \widehat{\mathcal{S}}$$

Size algebra

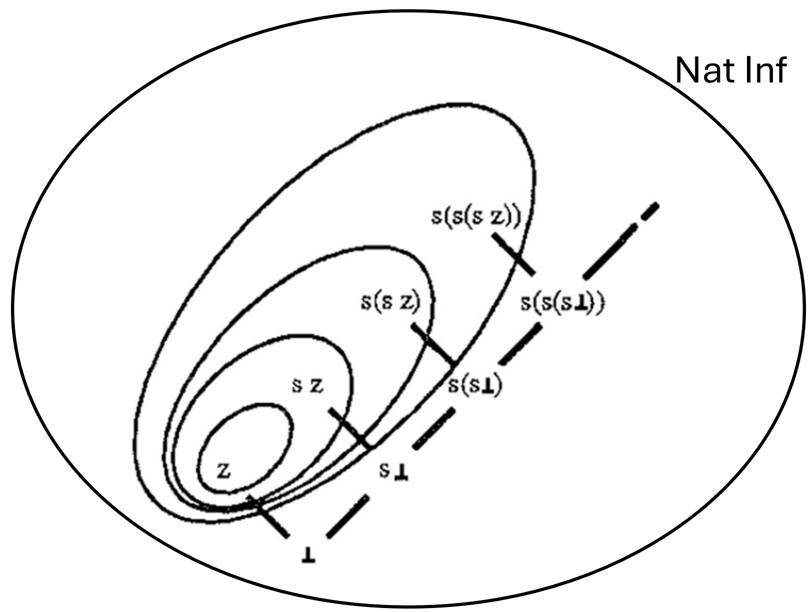
$$(refl)\ \frac{}{s \leq s} \qquad (trans)\ \frac{s \leq r \quad r \leq p}{s \leq p}$$

$$(succ)\ \frac{}{s \leq \widehat{s}} \qquad (sup)\ \frac{}{s \leq \infty}$$

Subsizing

$$\frac{s \leq r \qquad \tau \sqsubseteq \sigma}{d^s \tau \sqsubseteq d^r \sigma}$$

Subtyping rule for constructors



Set notion of sizes

Type checking: Case

$$
\begin{aligned}
\textbf{Inductive}\ \ \mathsf{Nat} \ &:= \ \mathsf{o} : \mathsf{Nat}^{\widehat{\imath}} \\
&\mid \ \mathsf{s} : \mathsf{Nat}^{\imath} \to \mathsf{Nat}^{\widehat{\imath}}
\end{aligned}
$$

Nat datatype defn

$$
\frac{\Gamma \vdash e : d^{\hat{s}} \qquad \begin{array}{c} c_k : \theta_k \to d^{\hat{i}} \\ \Gamma \vdash e_k : \theta[i := s] \to \sigma \end{array}}{\Gamma \vdash \mathsf{case}_{\sigma}\ e\ \mathsf{of}\ \{c_1 \Rightarrow e_1 \mid \ \ldots \ \mid c_n \Rightarrow e_n\} : \sigma}\ \text{CASE}
$$

Case typing rule

$$
\frac{\Gamma \vdash e : \mathsf{Nat}^{\hat{k}} \qquad \Gamma \vdash e_0 : \mathsf{Nat}^{\hat{k}} \qquad \Gamma \vdash e_s : \mathsf{Nat}^{k} \to \mathsf{Nat}^{\hat{k}}}{\Gamma \vdash \mathsf{case}_{\mathsf{Nat}^{\hat{k}}}\ x\ \mathsf{of}\ \{0 \Rightarrow e_0 \mid s \Rightarrow e_s\} : \mathsf{Nat}^{\hat{k}}}\ \text{CASE}
$$

Instantiation for natural numbers with Natk^ as σ

Type checking: Case

$$
\begin{aligned}
\textbf{Inductive}\ \ \mathsf{Nat}\ \ &:=\ \ \mathsf{o} : \mathsf{Nat}^{\hat{\imath}} \\
&\ \ \mid\ \ \mathsf{s} : \mathsf{Nat}^{\imath} \to \mathsf{Nat}^{\hat{\imath}}
\end{aligned}
$$

Nat datatype defn

$$
\frac{\dfrac{\Gamma, x : \mathsf{Nat}^{\hat{k}} \vdash x : \mathsf{Nat}^{\hat{k}} \qquad \Gamma \vdash e_0 : \mathsf{Nat}^{\hat{k}} \qquad \Gamma \vdash e_s : \mathsf{Nat}^{k} \to \mathsf{Nat}^{\hat{k}}}{\Gamma, x : \mathsf{Nat}^{\hat{k}} \vdash \mathsf{case}_{\mathsf{Nat}^{\hat{k}}}\ x\ \mathsf{of}\ \{0 \Rightarrow e_0 \mid s \Rightarrow e_s\} : \mathsf{Nat}^{\hat{k}}}\ \text{CASE}}{\Gamma \vdash \lambda x.\mathsf{case}\ \ldots : \mathsf{Nat}^{\hat{k}} \to \mathsf{Nat}^{\hat{k}}}\ \text{LAM}
$$

Add LAM to bind x

Cheat sheet

Reduction rule for fixpoints:

$$(\text{letrec}_\tau\ f = e) \quad \to \quad e[f := (\text{letrec}_\tau\ f = e)]$$

Type checking: Rec

$$\frac{\Gamma, f : d^{i} \to \theta \vdash e : d^{\hat{i}} \to \theta[i := \hat{i}]}{\Gamma \vdash (\mathsf{letrec}_{d \to \theta}\ f = e) : d^{s} \to \theta[i := s]}\ \text{REC}$$

Rec typing rule

$$\dfrac{\dfrac{\dfrac{\Gamma, x : \mathsf{Nat}^{\hat{k}} \vdash x : \mathsf{Nat}^{\hat{k}} \qquad \Gamma \vdash e_0 : \mathsf{Nat}^{\hat{k}} \qquad \Gamma \vdash e_s : \mathsf{Nat}^{k} \to \mathsf{Nat}^{\hat{k}}}{\Gamma, x : \mathsf{Nat}^{\hat{k}} \vdash \mathsf{case}_{\mathsf{Nat}^{\hat{k}}}\ x \ \mathsf{of}\ \{0 \Rightarrow e_0 \mid s \Rightarrow e_s\} : \mathsf{Nat}^{\hat{k}}}\ \text{CASE}}{\Gamma, \boxed{f : \mathsf{Nat}^{k} \to \mathsf{Nat}^{k}} \vdash \lambda x.\mathsf{case}\ \ldots : \mathsf{Nat}^{\hat{k}} \to \mathsf{Nat}^{\hat{k}}}\ \text{LAM}}{\Gamma \vdash (\mathsf{letrec}_{\mathsf{Nat} \to \mathsf{Nat}}\ f = \lambda x.\mathsf{case}\ \ldots) : \mathsf{Nat}^{s} \to \mathsf{Nat}^{s}}\ \text{REC}$$

Extending example with REC

$\boxed{\theta = \mathsf{Nat}^{k}}$

Type checking: Recursive application

$$
\dfrac{
\dfrac{
\Gamma, x : \mathsf{Nat}^{\hat{k}} \vdash x : \mathsf{Nat}^{\hat{k}}
\qquad
\Gamma \vdash e_0 : \mathsf{Nat}^{\hat{k}}
\qquad
\boxed{\dfrac{\Gamma, f : \mathsf{Nat}^{k} \to \mathsf{Nat}^{k}, x' : \mathsf{Nat}^{k} \vdash (f x') : \mathsf{Nat}^{k}}{\Gamma \vdash \lambda x'.(f x') : \mathsf{Nat}^{k} \to \mathsf{Nat}^{\hat{k}}}\ \text{LAM}}
}{
\Gamma, x : \mathsf{Nat}^{\hat{k}} \vdash \mathsf{case}_{\mathsf{Nat}^{\hat{k}}}\ x\ \mathsf{of}\ \{0 \Rightarrow e_0 \mid s \Rightarrow \boxed{e_s}\} : \mathsf{Nat}^{\hat{k}}
}\ \text{CASE}
}{
\dfrac{
\Gamma, f : \mathsf{Nat}^{k} \to \mathsf{Nat}^{k} \vdash \lambda x.\mathsf{case}\ \ldots : \mathsf{Nat}^{\hat{k}} \to \mathsf{Nat}^{\hat{k}}
}{
\Gamma \vdash (\mathsf{letrec}_{\mathsf{Nat} \to \mathsf{Nat}}\ f = \lambda x.\mathsf{case}\ \ldots) : \mathsf{Nat}^{s} \to \mathsf{Nat}^{s}
}\ \text{R}
}\ \text{LAM}
$$

Type production: rec

$$\frac{\Gamma, f : d^i \to \theta \vdash e : d^{\hat{i}} \to \theta[i := \hat{i}]}{\Gamma \vdash (\mathsf{letrec}_{d \to \theta}\ f = e) : d^s \to \theta[i := s]}\ \text{REC}$$

Rec typing rule

$$\frac{\Gamma, f : d^i \to \theta \vdash \Box : d^{\hat{i}} \to \theta[i := \hat{i}] \rightsquigarrow e}{\Gamma \vdash \Box : \forall i. d^i \to \theta \rightsquigarrow (\mathsf{letrec}_{d \to \theta}\ f = e)}\ \text{REC}$$

Rec production rule

Program generation with sized types

- Adapt sized typing rules into production rules to generate terminating recursive programs to test compiler optimizations

RQs & evaluation

1. How many recursive calls/steps are taken before base case?
2. Is the new generator more effective at finding **particular** (recursion related) bugs in compilers?

Comments & questions ☺