

# Typed-based termination

Gilles Barthe

INRIA Sophia-Antipolis, France

- Inductive definitions are extensively used in the formalization of mathematics, programming languages, cryptographic algorithms, in reflexive tactics, etc.
- Proof assistants provide mechanisms to compute with/reason about datatype elements. In order to guarantee decidable type-checking, such mechanisms should also ensure the termination of recursive definitions
- *Types offer a practical mechanism to enforce termination, provided they are enriched with size information*
- *Size information can be inferred automatically and the results of inference are understandable*

A mechanism for enforcing termination of recursive definitions in proof assistants based on type theory should be:

- easy to understand and to use
- easy to implement and to extend
- as powerful as currently used mechanisms for structural recursive definitions

Own bias: simplicity is more important than powerfulness

- Recursors: well understood theoretically but hard to use

$$\frac{\vdash n : \mathbb{N} \quad \vdash f_0 : A \quad \vdash f_s : \mathbb{N} \rightarrow A \rightarrow A}{\vdash \mathcal{R}_{\mathbb{N}}(n, f_0, f_s) : A}$$
$$\mathcal{R}_{\mathbb{N}}(0, f_0, f_s) \rightarrow f_0$$
$$\mathcal{R}_{\mathbb{N}}((s \ n), f_0, f_s) \rightarrow f_s \ n \ \mathcal{R}_{\mathbb{N}}(n, f_0, f_s)$$

- Pattern-matching with syntactic termination criteria: convenient notation but many drawbacks
  - External to the system and complex
    - $\Rightarrow$  hard to understand why a definition is rejected
  - Language dependent
    - $\Rightarrow$  new rules are required for new language constructs, even if they are independent from recursive definitions
  - Requires the function to be fully defined
    - $\Rightarrow$  limits modular development and separate compilation

# Type-based termination: idea

- A datatype is the least fixpoint of a monotonic operator  $\hat{\cdot}$  on types
- The syntax must be able to talk about approximations of the datatype
- Structurally smaller calls are captured by requiring that the definition of  $e : \hat{\alpha} \rightarrow A$  only relies on  $f : \alpha \rightarrow A$

$$\vdash O : \mathbb{N}^{\hat{z}} \qquad \frac{\vdash n : \mathbb{N}^z}{\vdash S n : \mathbb{N}^{\hat{z}}}$$

$$\frac{\vdash n : \mathbb{N}^{\hat{z}} \quad \vdash f_0 : A \quad \vdash f_s : \mathbb{N}^z \rightarrow A}{\vdash \text{case } n \text{ of } \{O \Rightarrow f_0 \mid S \Rightarrow f_s\} : A}$$

$$\frac{f : \mathbb{N}^z \rightarrow A \vdash e : \mathbb{N}^{\hat{z}} \rightarrow A}{\vdash \text{letrec } f = e : \mathbb{N}^{\infty} \rightarrow A}$$

# $\hat{\lambda}$ (aka lambda sombrero)

- A simply typed  $\lambda$ -calculus with a mechanism for enforcing termination of recursive definitions using annotated types with “size information”
- Type-based termination is a suitable mechanism to enforce termination of recursive definitions (termination-checking by type-checking):

*G. Barthe, M.-J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-Based Termination of Recursive Definitions. MSCS, 14(1):97–141, February 2004.*

- Type-based termination can be practical, by providing a sound and complete size inference algorithm:

*G. Barthe, B. Grégoire, and F. Pastawski. Practical Inference for Type-Based Termination Proceedings of TLCA'05, LNCS, April 2005.*

- Stages  $\mathcal{S}$

$$s, r ::= \iota \mid \infty \mid \hat{s}$$

- Types  $\mathcal{T}$

$$\sigma, \tau ::= \alpha \mid \tau \rightarrow \sigma \mid d^s \vec{\tau}$$

- Expressions  $\mathcal{E}$

$$e, e' ::= x \mid \lambda x : \tau''. e \mid e e' \mid \\ c \mid \text{case}_{\tau''} e' \text{ of } \{\vec{c} \Rightarrow \vec{e}\} \mid (\text{letrec } f : \tau' = e)$$

Important:  $\tau''$  is not annotated,  $\tau'$  is  $\star$ -annotated

$$\text{(refl)} \frac{}{s \preccurlyeq s} \quad \text{(trans)} \frac{s \preccurlyeq r \quad r \preccurlyeq p}{s \preccurlyeq p} \quad \text{(hat)} \frac{}{s \preccurlyeq \hat{s}} \quad \text{(infnty)} \frac{}{s \preccurlyeq \infty}$$

$$\text{(refl)} \frac{}{\sigma \sqsubseteq \sigma}$$

$$\text{(data)} \frac{s \preccurlyeq r \quad \tau_i \sqsubseteq \tau'_i \quad (1 \leq i \leq \text{ar}(d))}{d^s \vec{\tau} \sqsubseteq d^r \vec{\tau}'}$$

$$\text{(func)} \frac{\tau' \sqsubseteq \tau \quad \sigma \sqsubseteq \sigma'}{\tau \rightarrow \sigma \sqsubseteq \tau' \rightarrow \sigma'}$$

$$\frac{}{\Gamma \vdash x : \sigma} \text{ if } (x : \sigma) \in \Gamma$$

$$\frac{}{\Gamma \vdash c : \text{Inst}_c^s \vec{\tau} \rightarrow d^{\widehat{s}} \vec{\tau}}$$

$$\frac{\Gamma, x : \tau \vdash e : \sigma}{\Gamma \vdash \lambda x : |\tau|. e : \tau \rightarrow \sigma}$$

$$\frac{\Gamma \vdash e : \tau \rightarrow \sigma \quad \Gamma \vdash e' : \tau}{\Gamma \vdash e e' : \sigma}$$

$$\frac{\Gamma \vdash e' : d^{\widehat{s}} \vec{\tau} \quad \Gamma \vdash e_i : \text{Inst}_{c_i}^s \vec{\tau} \rightarrow \theta \quad 1 \leq i \leq n}{\Gamma \vdash \text{case}_{d \mid \vec{\tau}} e' \text{ of } \{\vec{c} \Rightarrow \vec{e}\} : \theta}$$

$$\frac{\Gamma, f : d^{\iota} \vec{\tau} \rightarrow \theta \vdash e : d^{\widehat{\iota}} \vec{\tau} \rightarrow \theta[\iota := \widehat{\iota}] \quad \iota \text{ pos } \theta \quad \iota \text{ fresh}}{\Gamma \vdash \text{letrec } f : (d^{\iota} \vec{\tau} \rightarrow \theta)^* = e : d^s \vec{\tau} \rightarrow \theta[\iota := s]}$$

$$\frac{\Gamma \vdash e : \sigma \quad \sigma \sqsubseteq \sigma'}{\Gamma \vdash e : \sigma'}$$



$$\begin{aligned}
 \text{minus} &\equiv (\text{letrec } \text{minus}_{:\text{nat}^2 \rightarrow \text{nat} \rightarrow \text{nat}^2} = \lambda x_{:\text{nat}^2}. \lambda y_{:\text{nat}}. \\
 &\quad \text{case } x \text{ of } \{ \text{o} \Rightarrow x \\
 &\quad \quad | s \Rightarrow \lambda x'_{:\text{nat}^2}. \text{case } y \text{ of } \{ \text{o} \Rightarrow x \\
 &\quad \quad \quad | s \Rightarrow \lambda y'_{:\text{nat}}. \underbrace{\text{minus } x' y'}_{:\text{nat}^2} \} \\
 &\quad \quad \quad \} \\
 &\quad \} \\
 &): \quad \text{nat}^5 \rightarrow \text{nat} \rightarrow \text{nat}^5 \\
 \\
 (\text{letrec } \text{div}_{:\text{nat}^2 \rightarrow \text{nat} \rightarrow \text{nat}^2} = \\
 &\quad \lambda x_{:\text{nat}^2}. \lambda y_{:\text{nat}}. \text{case } x \text{ of } \{ \text{o} \Rightarrow \text{o} \\
 &\quad \quad | s \Rightarrow \lambda x'_{:\text{nat}^2}. s \underbrace{(\text{div } \underbrace{(\text{minus } x' y)}_{:\text{nat}^2}) y}_{:\text{nat}^2}} \\
 &\quad \quad \quad \} \\
 &\quad \} \\
 &): \quad \text{nat}^5 \rightarrow \text{nat} \rightarrow \text{nat}^5
 \end{aligned}$$

$\text{conc} : \text{List} (\text{List } \tau) \rightarrow \text{List } \tau$

$\text{map} : (\tau \rightarrow \sigma) \rightarrow \text{List}^s \tau \rightarrow \text{List}^s \sigma \equiv \lambda f:_{\tau \rightarrow \sigma}.$

(letrec  $\text{map}_{:\text{List}^s \tau \rightarrow \text{List}^s \sigma} = \lambda x:_{\text{List}^{\hat{s}} \tau}.$

case  $x$  of {nil  $\Rightarrow$  nil

| cons  $\Rightarrow \lambda z:_{\tau}. \lambda x':_{\text{List}^s \tau}. \text{cons } \underbrace{(f \ z)}_{:\sigma} \ \underbrace{(\text{map } x')}_{:\text{List}^s \sigma}}_{:\text{List}^{\hat{s}} \sigma}$ )

$\text{flatten} : \text{Tree}^s \tau \rightarrow \text{List } \tau \equiv$

(letrec  $\text{flatten}_{:\text{Tree}^s \tau \rightarrow \text{List } \tau} = \lambda x:_{\text{Tree}^{\hat{s}} \tau}.$

cases  $x$  of {branch  $\Rightarrow$

$\lambda z:_{\tau}. \lambda x':_{\text{List} (\text{Tree}^s \tau)}. \text{cons } z \ \underbrace{(\text{conc } (\text{map } \text{flatten } x'))}_{:\text{List} (\text{List } \tau)}}_{:\text{List } \tau}$ )

- Subject reduction

$$\Gamma \vdash e : \sigma \quad \wedge \quad e \rightarrow e' \quad \Rightarrow \quad \Gamma \vdash e' : \sigma$$

- Strong normalization

$$\Gamma \vdash e : \sigma \quad \Rightarrow \quad e \in \text{SN}$$

- Strict extension of  $\lambda_{\mathcal{G}}$  with syntactic guard condition

Consider a *stage valuation*  $\pi : \mathcal{V}_S \rightarrow \Omega + 1$ .

$$\begin{aligned} \llbracket \iota \rrbracket_\pi &= \pi(\iota) \text{ if } \iota \in \mathcal{V}_S \\ \llbracket \infty \rrbracket_\pi &= \Omega \\ \llbracket \hat{s} \rrbracket_\pi &= \begin{cases} \llbracket s \rrbracket_\pi + 1 & \text{if } \llbracket s \rrbracket_\pi < \Omega \\ \llbracket s \rrbracket_\pi & \text{if } \llbracket s \rrbracket_\pi = \Omega \end{cases} \end{aligned}$$

# Model construction – Types

Consider a *type valuation*  $\xi : \mathcal{V}_{\mathcal{T}} \rightarrow \text{SAT}$ .

$$\begin{aligned} \llbracket \alpha \rrbracket_{\pi, \xi} &= \xi(\alpha) \text{ if } \alpha \in \mathcal{V}_{\mathcal{T}} \\ \llbracket \tau \rightarrow \sigma \rrbracket_{\pi, \xi} &= \llbracket \tau \rrbracket_{\pi, \xi} \rightarrow \llbracket \sigma \rrbracket_{\pi, \xi} \\ \llbracket d^s \vec{\tau} \rrbracket_{\pi, \xi} &= D(\llbracket \vec{\tau} \rrbracket_{\pi, \xi}, \llbracket s \rrbracket_{\pi}) \end{aligned}$$

where  $D(\vec{X}, x)$  is defined by induction on  $x$  by

$$\begin{aligned} D(\vec{X}, 0) &= \ulcorner \emptyset \urcorner \\ D(\vec{X}, y + 1) &= \lceil c_1 \llbracket \vec{\sigma}_1 \rrbracket_{\pi, \xi(\delta := D(\vec{X}, y), \vec{\alpha} := \vec{X})} \\ &\quad \cup \dots \cup \\ &\quad c_n \llbracket \vec{\sigma}_n \rrbracket_{\pi, \xi(\delta := D(\vec{X}, y), \vec{\alpha} := \vec{X})} \rceil \\ &\quad \text{where } D(c_i) = (\delta, \vec{\alpha}, \vec{\sigma}_i) \\ D(\vec{X}, x) &= \bigcup_{y < x} D(\vec{X}, y) \text{ if } x \text{ is a limit ordinal} \end{aligned}$$

- Term valuations are substitutions  $\rho$ .
- $(\pi, \xi, \rho)$  be a valuation.
  - 1  $(\pi, \xi, \rho)$  satisfies a context  $\Gamma$ , written  $(\pi, \xi, \rho) \models \Gamma$ , if  $\rho(x) \in \llbracket \tau \rrbracket_{\pi, \xi}$  for each  $(x : \tau) \in \Gamma$ .
  - 2  $(\pi, \xi, \rho)$  satisfies a typing judgment  $\Gamma \vdash e : \sigma$ , if

$$(\pi, \xi, \rho) \models \Gamma \Rightarrow \llbracket e \rrbracket_{\rho} \in \llbracket \sigma \rrbracket_{\pi, \xi}$$

- A typing judgment  $\Gamma \vdash e : \sigma$  is *valid*, written  $\Gamma \models e : \sigma$ , if every valuation satisfies it.

## Soundness

$$\Gamma \vdash e : \sigma \Rightarrow \Gamma \models e : \sigma$$

- Given a context  $\Gamma$  and an expression  $e$ , compute if it exists the (most general) type  $\tau$  s.t.  $\Gamma \vdash e : \tau$  is derivable in  $\hat{\lambda}$ , and return fail otherwise
- Pitfall 1: lack of unconstrained principal type

$$\lambda f : \mathbb{N} \rightarrow \mathbb{N}. \lambda x : \mathbb{N}. f (f x)$$

cannot have a principal type without using constraints

- Pitfall 2: need for  $\star$ -annotated types

$$\text{letrec } f : \mathbb{N} \rightarrow \mathbb{N} = \lambda x : \mathbb{N}. O$$

has types  $\mathbb{N}^z \rightarrow \mathbb{N}^z$  and  $\mathbb{N}^z \rightarrow \mathbb{N}^{\hat{J}}$ .

# Inference algorithm: expected properties

- Input:  $\Gamma$  and  $e$
- Output:  $\tau$  and constraint system  $C$

such that:

**Soundness:** for every stage substitution  $\rho \models C$ , we have  
 $\rho\Gamma \vdash e : \rho\tau$ .

**Completeness:** for every stage substitution  $\rho'$  s.t.  $\rho'\Gamma \vdash e : \tau'$ ,  
there exists a stage substitution s.t.  $\rho \models C$  and  
 $\rho\Gamma = \rho'\Gamma$  and  $\rho\tau \sqsubseteq \tau'$

where:

- A constraint system is a set of stage inequalities  $s_1 \preceq s_2$   
 $\Rightarrow$  no disjunction
- A stage substitution  $\rho$  satisfies  $\rho$ , written  $\rho \models C$ , if  $\rho s_1 \preceq \rho s_2$   
for every inequality in  $C$ .

- For most constructs inference rules propagate constraints

$$\text{Infer}(V, \Gamma, e_1 e_2) = V_2, C_1 \cup C_2, \tau$$

where

$$\begin{aligned}(V_1, C_1, \tau_2 \rightarrow \tau) &:= \text{Infer}(V, \Gamma, e_1) \\ (V_2, C_2) &:= \text{Check}(V_1, \Gamma, e_2, \tau_2)\end{aligned}$$

- For letrec the inference rule performs some checks and possibly produces a set of constraints

$$\text{Infer}(V, \Gamma, \text{letrec } f : d^* \vec{\tau} \rightarrow \theta = e') = V_{e'}, C_f, d^{\alpha \vec{\tau}} \rightarrow \bar{\theta}$$

where

$$\begin{aligned}(V_1, V^*, d^{\alpha \vec{\tau}} \rightarrow \bar{\theta}) &:= \text{annotrec}(V, d^* \vec{\tau} \rightarrow \theta) \\ \hat{\theta} &:= \bar{\theta}[\alpha_i := \hat{\alpha}_i]_{\alpha_i \in V^*} \\ (V_{e'}, C_{e'}) &:= \text{Check}(V_1, \Gamma; f : d^{\alpha \vec{\tau}} \rightarrow \bar{\theta}, e', d^{\hat{\alpha} \vec{\tau}} \rightarrow \hat{\theta}) \\ C_f &:= \text{RecCheck}(\alpha, V^*, V_1 \setminus V^*, C_{e'} \cup \bar{\theta} \sqsubseteq \hat{\theta})\end{aligned}$$

# Classification of stage variables

If we take  $\iota$  to be a base stage, we can classify stage variables as:

- $S_\iota \stackrel{\text{def}}{=} \text{those } \mathbf{known} \text{ to evaluate to } \iota \text{ with hats.}$
- $S_{\iota \preceq} \stackrel{\text{def}}{=} \text{those } \mathbf{known} \text{ to evaluate to } \iota \text{ with hats or } \infty.$
- $S_{\neg\iota} \stackrel{\text{def}}{=} \text{those } \mathbf{known} \text{ not to evaluate to } \iota \text{ with hats.}$
- $S_\infty \stackrel{\text{def}}{=} \text{those } \mathbf{known} \text{ to evaluate to } \infty.$

RecCheck computes the above sets using rules (below) and checks  $S_\iota \cap S_\infty = \emptyset$  and positivity given as subtyping constraint.

- $S_\iota$  and  $S_{\neg\iota}$  downwards closed.
- If  $\iota \in S_\iota$  and  $\iota \leq j$  then  $j \in S_{\iota \preceq}$ .
- $S_{\iota \preceq} \cap S_{\neg\iota} \subseteq S_\infty$ .
- If  $\hat{\alpha} \preceq \alpha$  then  $s \in S_\infty$ .

# Simplifying constraints

- Inferred types are “minimal” and readable, but they may be overly verbose
- Simple heuristics can be used to simplify types, e.g.

$$\textit{add} : \mathbb{N}^l \rightarrow \mathbb{N}^j \rightarrow \mathbb{N}^\infty$$

is inferred, but simplified to

$$\textit{add} : \mathbb{N}^\infty \rightarrow \mathbb{N}^\infty \rightarrow \mathbb{N}^\infty$$

- It should be possible to define a notion of canonical type/constraint pair, and define an algorithm that reduces each type/constraint pair to canonical form

- Type-based termination is a suitable mechanism to enforce termination of recursive definitions
- Worked out an extension to dependent types  $CC$ , together with a prototype implementation
- Future work: (hopefully) integrate type-based termination in a proof assistant, (possibly) richer size algebras, support for more measures
- Related work: Mendler; Giménez; Amadio and Coupet; Abel; Blanqui; Xi; Grobauer; Hughes, Pareto and Sabry; Chin and Khoo.