Semi-continuous Sized Types and Termination

Termination Checking via Type Systems

Andreas Abel

Department of Computer Science
Ludwig-Maximilians-University Munich

Computer Science Logic 2006
Szeged, Hungary
September 26, 2006

Theorem Provers for Constructive Logic

Theorem Provers built on Dependent Type Theory:

- Coq (INRIA, France)
- Epigram (Nottingham, UK)
- Agda (Chalmers, Sweden)

Their soundness is based on *termination*.

Constructive Logics

The Curry-Howard Isomorphism:

Proposition	Type
A implies B	$A \to B$
Proof	Purely Functional Program
Valid Proof	Terminating Program

Non-terminating programs lead to inconsistency:

$$f : (0 = 0) \to (0 = 1)$$
$$f(p) = f(p)$$

Type-Based Termination, Informally

Recipe

Step 1 In the type system, attach sizes to data structures.

Step 2 Using type-checking, ensure that recursive calls use only arguments with decreased size.

Step 1: Sized Binary Trees

- Let $\mathsf{BTree}^{\imath}$ denote trees of height $< \imath$.
- The empty tree has height 0, hence $\mathsf{leaf} : \mathsf{BTree}^{1}$, but also $\mathsf{leaf} : \mathsf{BTree}^{2}$, $\mathsf{leaf} : \mathsf{BTree}^{3}$, ...
- In general $\mathsf{leaf} : \mathsf{BTree}^{\imath+1}$ for all $\imath$.

$$\begin{array}{lcl} \mathsf{leaf} & : & \forall \imath.\, \mathsf{BTree}^{\imath+1} \\ \mathsf{node} & : & \forall \imath.\, \mathsf{Int} \times (\mathsf{BTree}^{\imath} \times \mathsf{BTree}^{\imath}) \to \mathsf{BTree}^{\imath+1} \end{array}$$

- BTree^{∞} contains all binary trees.
- Subtyping: $\mathsf{BTree}^{\imath} \subseteq \mathsf{BTree}^{\imath+1} \subseteq \cdots \subseteq \mathsf{BTree}^{\infty}$.

Step 2: Equality Test for Sized Binary Trees

- Code annotated with sizes:

$$\mathsf{eq} : \forall\imath.\, \mathsf{BTree}^{\imath} \to \mathsf{BTree}^{\imath} \to \mathsf{Bool}$$

$$\begin{array}{l}
\mathsf{eq}\ \mathsf{leaf}\ \mathsf{leaf} = \mathsf{true} \\
\mathsf{eq}\ \mathsf{node}(i_1,(l_1,r_1))^{\imath+1}\ \mathsf{node}(i_2,(l_2,r_2))^{\imath+1} = (i_1 == i_2)\ \ \&\& \\
\qquad \mathsf{eq}\ {l_1}^{\imath}\ {l_2}^{\imath}\ \ \&\&\ \ \mathsf{eq}\ {r_1}^{\imath}\ {r_2}^{\imath} \\
\mathsf{eq}\ _\ _ = \mathsf{false}
\end{array}$$

- Input arguments assumed to be of size $\imath + 1$.
- Recursive arguments inferred to be of size $\imath$.
- Descend in size, hence, termination.

Abstracting the Branching Type

- Generalize to F-Branching Int-labelled trees $\mathsf{Tree}^{\imath}\, F$
- Constructors:

$$\begin{array}{lcl} \mathsf{leaf} & : & \forall F \forall \imath.\, \mathsf{Tree}^{\imath+1} F \\ \mathsf{node} & : & \forall F \forall \imath.\, \mathsf{Int} \times F\,(\mathsf{Tree}^{\imath} F) \to \mathsf{Tree}^{\imath+1} F \end{array}$$

- Valid instances

binary trees	$F\,T = T \times T$
lists	$F\,T = T$
finitely branching trees	$F\,T = \mathsf{List}^{\infty}\, T$
infinitely branching trees	$F\,T = \mathsf{Nat}^{\infty} \to T$

- Invalid instance (F not monotone), e.g., $F\,T = T \to \mathsf{Bool}$

Equality of F-Branching Trees

- Generalize equality test to F-branching trees:
- Termination not inferable with untyped methods.

Termination and Polymorphism

$$\mathsf{Eq}\ T = T \to T \to \mathsf{Bool}$$

$$\mathsf{eq} : (\forall T.\mathsf{Eq}\ T \to \mathsf{Eq}\,(F\ T)) \to \forall \imath.\ \mathsf{Eq}\,(\mathsf{Tree}^{\imath} F)$$

$$\mathsf{eq}\ \mathit{eqF}\ \mathsf{leaf}\ \mathsf{leaf} = \mathsf{true}$$

$$\mathsf{eq}\ \mathit{eqF}\ \mathsf{node}(i_1, \mathit{ft}_1)\ \overbrace{\mathsf{node}(i_2, \underbrace{\mathit{ft}_2}_{F\,(\mathsf{Tree}^{\imath}\ F)})}^{\mathsf{Tree}^{\imath+1}\ F} = (i_1 == i_2)\ \ \&\&$$

$$\mathit{eqF}\quad \overbrace{(\mathsf{eq}\ \mathit{eqF})}^{\mathsf{Eq}\ T = \mathsf{Eq}\,(\mathsf{Tree}^{\imath} F)}\quad \mathit{ft}_1\ \mathit{ft}_2$$

$$\mathsf{eq}\ \_\ \_\ \_ = \mathsf{false}$$

Observe the role reversal: The recursive function $(\mathsf{eq}\ \mathit{eqF})$ becomes an argument to its own argument eqF!

Termination and Polymorphism

$$\mathsf{Eq}\ T = T \to T \to \mathsf{Bool}$$

$$\mathsf{eq} : (\forall T.\mathsf{Eq}\ T \to \mathsf{Eq}\,(F\ T)) \to \forall\imath.\,\mathsf{Eq}\,(\mathsf{Tree}^{\imath}\,F)$$

$$\mathsf{eq}\ \mathit{eqF}\ \mathsf{leaf}\ \mathsf{leaf} = \mathsf{true}$$

$$\mathsf{eq}\ \mathit{eqF}\ \mathsf{node}(i_1, \mathit{ft}_1)\ \overbrace{\mathsf{node}(i_2,\ \underbrace{\mathit{ft}_2}_{F\,(\mathsf{Tree}^{\imath}\ F)}\ )}^{\mathsf{Tree}^{\imath+1}\ F} = (i_1 == i_2)\ \ \&\&$$

$$\mathit{eqF}\quad \overbrace{(\mathsf{eq}\ \mathit{eqF})}^{\mathsf{Eq}\ T = \mathsf{Eq}\,(\mathsf{Tree}^{\imath}\,F)}\quad \mathit{ft}_1\ \mathit{ft}_2$$

$$\mathsf{eq}\ \_\ \_\ \_ = \mathsf{false}$$

Observe the role reversal: The recursive function $(\mathsf{eq}\ \mathit{eqF})$ becomes an argument to its own argument eqF!

Evaluation

No untyped formalism can handle this example:

- In the untyped setting, eq diverges, e.g., define

$$\mathit{eqF}\ \mathit{eqT}\ \mathit{ft}_1\ \mathit{ft}_2 = \mathit{eqT}\ \mathsf{node}(0, \mathit{ft}_1)\ \mathsf{node}(0, \mathit{ft}_2)$$

- and execute the function clause

$$\begin{aligned}\mathsf{eq}\ \mathit{eqF}\ \mathsf{node}(i_1, \mathit{ft}_1)\ \mathsf{node}(i_2, \mathit{ft}_2) &= \ldots \\ &\quad \mathit{eqF}\ (\mathsf{eq}\ \mathit{eqF})\ \mathit{ft}_1\ \mathit{ft}_2\end{aligned}$$

A typed formalism such as TBT uses the information that

$$\mathit{eqF} : \forall T. \mathsf{Eq}\ T \to \mathsf{Eq}\,(F\ T)$$

is polymorphic (hence, the above instance of eqF is ill-typed).

Type-Based Termination, Formally

Theorem

$f = s(f) : \forall\imath.\, A(\imath)$ *is well-defined if*

1. *(bottom check)* $A(0)$ *contains all programs, e.g.,* $A(\imath) = \mathsf{BTree}^{\imath} \to C$.
2. *(descent)* $f : A(\imath)$ *implies* $s(f) : A(\imath + 1)$.
3. *(admissibility)* $\bigcap_{\alpha<\lambda} A(\alpha) \subseteq A(\lambda)$ *for all limit ordinals* $\lambda \neq 0$.

Proof.

By transfinite induction on $\imath$.

1. (base) $f : A(0)$ trivial.
2. (step) ind.hyp. $f : A(\alpha)$ implies $s(f) = f : A(\alpha + 1)$.
3. (limit) $f : \bigcap_{\alpha<\lambda} A(\alpha)$ by ind.hyp., hence $f : A(\lambda)$.

□

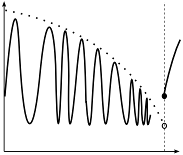
Upper Semi-Continuous Types

Definition (upper semi-continuous)

A semantical type $\mathcal{A} : \mathsf{On} \to \mathcal{P}(\mathsf{SN})$ is upper semi-continuous (*usc*) if for all limits $\lambda \neq 0$

$$\limsup_{\alpha \to \lambda} \mathcal{A}(\alpha) := \left(\bigcap_{\alpha_0 < \lambda} \bigcup_{\alpha_0 \leq \alpha < \lambda} \mathcal{A}(\alpha) \right) \subseteq \mathcal{A}(\lambda)$$

An *usc* type fulfills $\bigcap_{\alpha<\lambda} \mathcal{A}(\alpha) \subseteq \mathcal{A}(\lambda)$, hence, is admissible.

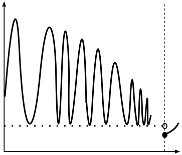


Lower Semi-Continuous Types

Definition (upper semi-continuous)

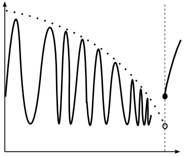
A semantical type $\mathcal{A} : \mathsf{On} \to \mathcal{P}(\mathsf{SN})$ is lower semi-continuous (*usc*) if for all limits $\lambda \neq 0$

$$\mathcal{A}(\lambda) \subseteq \liminf_{\alpha \to \lambda} \mathcal{A}(\alpha) := \bigcup_{\alpha_0 < \lambda} \bigcap_{\alpha_0 \leq \alpha < \lambda} \mathcal{A}(\alpha)$$



Continuous Types

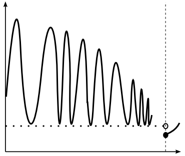
usc



$\limsup_{\alpha\to\lambda} \mathcal{A}(\alpha) \subseteq \mathcal{A}(\lambda)$

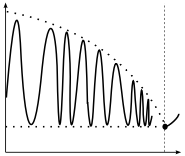
+

lsc



$\mathcal{A}(\lambda) \subseteq \liminf_{\alpha\to\lambda} \mathcal{A}(\alpha)$

continuous



$\lim_{\alpha\to\lambda} \mathcal{A}(\alpha) = \mathcal{A}(\lambda)$

Closure Properties of Semi-Continuity

usc	condition	*lsc*	condition
$\mathcal{A}$ *usc*	$\mathcal{A}$ monotone	$\mathcal{A}$ *lsc*	$\mathcal{A}$ antitone
$\mathcal{A} + \mathcal{B}$ *usc*	$\mathcal{A}$, $\mathcal{B}$ *usc*	$\mathcal{A} + \mathcal{B}$ *lsc*	$\mathcal{A}$, $\mathcal{B}$ *lsc*
$\mathcal{A} \times \mathcal{B}$ *usc*	$\mathcal{A}$, $\mathcal{B}$ *usc*	$\mathcal{A} \times \mathcal{B}$ *lsc*	$\mathcal{A}$, $\mathcal{B}$ *lsc*
$\mathcal{A} \to \mathcal{B}$ *usc*	$\mathcal{A}$ *lsc*, $\mathcal{B}$ *usc*	—	
$\nu\mathcal{F}$ *usc*	$\mathcal{F}$ *usc*	$\mu\mathcal{F}$ *lsc*	$\mathcal{F}$ *lsc*

Why Upper Semi-Continuity is Vital

Let $\mathsf{pred} : \forall\imath.\, \mathsf{Nat}^{\imath+1} \to \mathsf{Nat}^{\imath}$ such that $\mathsf{pred}\,0$ raises an exception. Define

$$f : \forall\imath.\, \overbrace{(\mathsf{Nat}^{\infty} \to \mathsf{Nat}^{\imath}) \to X}^{A(\imath)}$$

$$f(g : \mathsf{Nat}^{\infty} \to \mathsf{Nat}^{\imath+1}) = f\ ((\mathsf{pred} \circ g \circ \mathsf{succ}) : \mathsf{Nat}^{\infty} \to \mathsf{Nat}^{\imath})$$

Now $f(\mathsf{id})$ loops.

The definition passes the bottom check and the descent criterion, but $A(\imath)$ is neither *usc* nor admissible.

Related Work

Expressivity	Xi	Par	Ama	Gim	Fra	**A**	Bar	Bla	Buch
term. measures	+	−	−	−	−	−	−	+	o
dep. types	o	−	−	+	−	−	+	+	−
polymorphism	+	o	−	+	−	**+**	+	+	−
infinite branch.	−	−	−	+	+	**+**	+	−	+
semi-cont.	−	ω	−	−	−	**+**	−	−	−
productivity	−	+	+	+	+	+	+	−	+
Features									
symbolic exec.	−	−	+	+	+	+	+	+	+
soundness	V	D	SN	−	SN	SN	o	SN	D
ordinals	$< \omega$	$\leq \omega$	On	−	Ω	Ω_ω	−	$< \omega$	$\leq \omega$
equi-rec.	−	+	−	−	−	+	−	−	−
size inference	−	+	−	−	−	−	+	−	−

Conclusions

- Termination checking can be integrated into type checking
- Especially powerful in combination with polymorphism
- Type-Based Termination is a modern technology, still under active development

Future Work

- Extend to dependent types
- Investigate semi-continuity for dependent types
- Find intuitive explanations for non-admissibility of types
- Integrate into a theorem prover

Acknowledgements

Technical discussions on my thesis

Klaus Aehlig Thorsten Altenkirch
Ikegami Daisuke Martin Hofmann
John Hughes Ralph Matthes Tarmo Uustalu

Financial support

Graduiertenkolleg Logik in der Informatik
Project CoVer (Swedish Foundation for Strategic Research)
TYPES
APPSEM II

Formalizing Sized Types

- Capture the structure of data types, forget about constructor names.
- Types are build from the primitives $+$ (disjoint sum), $\times$ (cartesian product), $\to$ (function space).
- Sized types $\mu^{\imath} F$ are recursive types obtained by iterating a type transformer F:

$$
\begin{array}{lcl}
\mu^{0} F & = & \emptyset \\
\mu^{\alpha+1} F & = & F\,(\mu^{\alpha}\, F) \\
\mu^{\lambda} F & = & \bigcup_{\alpha<\lambda} \mu^{\alpha} F
\end{array}
$$

- E.g., type constructor for binary trees:
  $\mathsf{BTreeF}\ X = 1 + \mathsf{Int} \times (X \times X)$
- Sized binary trees: $\mathsf{BTree}^{\imath} = \mu^{\imath}\, \mathsf{BTreeF}$.