Deciding unique inhabitants with sums (work in progress)

Gabriel Scherer, Didier Rémy

Gallium – INRIA
Does a given type have a unique inhabitant (modulo program equivalence)?
Setting: STLC, +, *, abstract base types.
Does a given type have a **unique** inhabitant (modulo program equivalence)?

Setting: STLC, +, *, abstract base types.

\[
(\lambda(x)\ t)\ u \rightarrow_{\beta} t[u/x] \quad (t : A \rightarrow B) =_{\eta} \lambda(x)\ t \ x \\
\pi_{i} (t_{1}, t_{2}) \rightarrow_{\beta} t_{i} \quad (t : A \ast B) =_{\eta} (\pi_{1} t, \pi_{2} t)
\]
Does a given type have a **unique** inhabitant (modulo program equivalence)?
Setting: STLC, +, ∗, abstract base types.

\[
(\lambda(x) \ t) \ u \to_{\beta} t[u/x] \quad (t : A \to B) =_{\eta} \lambda(x) \ t \ x
\]

\[
\pi_i \ (t_1, t_2) \to_{\beta} t_i \quad (t : A \ast B) =_{\eta} (\pi_1 \ t, \pi_2 \ t)
\]

\[
\delta(\sigma_i \ t, x_1.u_1, x_2.u_2) \to_{\beta} u_i[t/x_i]
\]

\[
\forall(K[A_1 + A_2] : B), \quad K[t] =_{\eta} \delta(t, x_1.K[\sigma_1 x_1], x_2.K[\sigma_2 x_2])
\]
Why?

If the type of a program hole has a unique inhabitant, we can guess it.

This could be extremely convenient for:

- over-specified program components, such as
  - highly parametric ML libraries, eg. monadic call/cc
  - dependently-typed programs (see DTP’13 talk)
- trivial program glue: I forgot the parameter order, but only one choice is typeable

Current such “tactics” are inhabitation-oriented, not unicity. Good for proving, disappointing for programming.

(related work on program/composition synthesis, Rehof et al.)
What do we already know?

Solved: Deciding inhabitation: provability in propositional intuitionistic logic. [Dyc13]

Solved: Normalizing a term, deciding equivalence between two terms. [ADHS01, BCF04, Lin07]

Instead of one or two terms, we want to work on all inhabitants at once.

Solved: in absence of abstract base types, types are finitely inhabited, and we can enumerate them [AU04].
Objectives

We want an algorithm to produce a sequence of inhabitants of $\Gamma \vdash A$ that is:

- **complete**: no program is missing (modulo $=_{\beta\eta}$)
- **canonical**: no two programs are equivalent
- **terminating**: you get the next element (or end) in finite time

Most of the work on inhabitation throws away computational completeness. Example: subsumption optimization in forward methods.
Too Simple

First idea:

- use any reasonable term enumeration procedure (with possible duplicates): focused proof search, or Herbelin’s LJT
- then use equivalence testing to remove duplicates

(You could also discard non-normalized terms; non-locality issues)
Too Simple

Problem: you may get infinitely many equivalent terms before the first different one – if any.
Too Simple

Problem: you may get infinitely many equivalent terms before the first different one – if any.

\[ f : (1 \to A + B) \vdash A + B \]

\[ f 1 \]
Too Simple

Problem: you may get infinitely many equivalent terms before the first different one – if any.

\[ f : (1 \rightarrow A + B) \vdash A + B \]

\[ f 1 \]
\[ \delta(f 1, x_1.\sigma_1 x_1, y_1.f 1) \]
Too Simple

Problem: you may get infinitely many equivalent terms before the first different one – if any.

\[ f : (1 \rightarrow A + B) \vdash A + B \]

\[ f \ 1 \]
\[ \delta(f \ 1, x_1.\sigma_1 \ x_1, y_1.f \ 1) \]
\[ \delta(f \ 1, x_1.\sigma_1 \ x_1, y_1.\delta(f \ 1, x_2.\sigma_1 \ x_2, y_2.f \ 1)) \]
Problem: you may get infinitely many equivalent terms before the first different one – if any.

\[ f : (1 \rightarrow A + B) \vdash A + B \]

\[ f \quad f 1 \quad \delta(f 1, x_1.\sigma_1 x_1, y_1.f 1) \]
\[ \delta(f 1, x_1.\sigma_1 x_1, y_1.\delta(f 1, x_2.\sigma_1 x_2, y_2.f 1)) \]
\[ \delta(f 1, x_1.\sigma_1 x_1, y_1.\delta(f 1, x_2.\sigma_1 x_2, y_2.\delta(f 1, x_3.\sigma_1 x_3, y_3.\ldots))) \]

Not terminating – unless you embed more knowledge of sum equivalence in the enumeration procedure.
Our approach: Saturation

Normalization/equivalence for sums has a non-local component: “move sum eliminations as early in the term as possible”

When enumerating terms top-down, this suggests a saturation approach: eliminate all sums as soon as possible.

“To find all elements of $\Gamma \vdash A$, find all $C_i$ such that $\Gamma \vdash C_i$, and look at trivial proofs of $\Gamma, C_1, \ldots, C_n \vdash A$."

This sounds impractical, but also pleasantly general.
Enumerating distinct terms – first without sums

Values and neutrals:

\[
\begin{align*}
\nu & ::= \lambda(x:A) \nu \quad | \quad (\nu, \nu) \quad | \quad n \\
n & ::= n \nu \quad | \quad \pi_1 n \quad | \quad \pi_2 n \quad | \quad x
\end{align*}
\]

Enumerating values and neutrals:

\[
\begin{align*}
\text{Val}(\Gamma \vdash A \to B) & := \lambda(x:A) \text{Val}(\Gamma, x:A \vdash B) \\
\text{Val}(\Gamma \vdash A \ast B) & := (\text{Val}(\Gamma \vdash A), \text{Val}(\Gamma \vdash B)) \\
\text{Val}(\Gamma \vdash X) & := \text{Ne}(\Gamma \vdash X) \quad (X \text{ atomic})
\end{align*}
\]

\[
\begin{align*}
\text{Ne}(\Gamma \vdash B) & \supset \text{Ne}(\Gamma \vdash A \to B) \text{Val}(\Gamma \vdash A) \\
\text{Ne}(\Gamma \vdash A_i) & \supset \pi_i \text{Ne}(\Gamma \vdash A_1 \ast A_2) \\
\text{Ne}(\Gamma \vdash N) & \supset \{x \mid (x:N) \in \Gamma\} \quad (N \text{ negative: } \to \ast)
\end{align*}
\]

All such elements are in \(\eta\)-long \(\beta\)-normal form: canonicity.
Termination I

Of course some sets may be infinite: watch for cycles.

\[ X, X \rightarrow X \vdash X \]

\[ \text{Val}(\Gamma \vdash X) \]

\[ \text{Val}(\Gamma \vdash X \rightarrow X) \]

\[ \text{Ne}(\Gamma \vdash X) \]

\[ := \]

[Wy04]
Val(Γ ⊢ A → B) := λ(x:A)Val(Γ, x:A ⊢ B)
Val(Γ ⊢ A ∗ B) := (Val(Γ ⊢ A), Val(Γ ⊢ B))
Val(Γ ⊢ X) := Ne(Γ ⊢ X) (X atomic)

Ne(Γ ⊢ B) ⊇ Ne(Γ ⊢ A → B) Val(Γ ⊢ A)
Ne(Γ ⊢ A_i) ⊇ π_i Ne(Γ ⊢ A_1 ∗ A_2)
Ne(Γ ⊢ N) ⊇ {x | (x:N) ∈ Γ} (N negative: → ∗)

You may have recognized the rules of a focused proof system.
Focusing suggests how to extend to sums.
\[
\begin{align*}
\text{Val}(\Gamma \vdash A \to B) &:= \lambda(x:A)\text{Val}(\Gamma, x:A \vdash B) \\
\text{Val}(\Gamma \vdash A \ast B) &:= (\text{Val}(\Gamma \vdash A), \text{Val}(\Gamma \vdash B)) \\
\text{Val}(\Gamma \vdash X) &:= \text{Ne}(\Gamma \vdash X) \quad (X \text{ atomic}) \\
\end{align*}
\]

\[
\begin{align*}
\text{Ne}(\Gamma \vdash B) \supseteq \text{Ne}(\Gamma \vdash A \to B) \text{Val}(\Gamma \vdash A) \\
\text{Ne}(\Gamma \vdash A_i) \supseteq \pi_i \text{Ne}(\Gamma \vdash A_1 \ast A_2) \\
\text{Ne}(\Gamma \vdash N) \supseteq \{x \mid (x: N) \in \Gamma\} \quad (N \text{ negative: } \to \ast)
\end{align*}
\]

You may have recognized the rules of a focused proof system. Focusing suggests how to extend to sums.

\[
\begin{align*}
\text{Ne}(\Gamma \vdash A_1 + A_2) \supseteq_{i \in \{1,2\}} \sigma_i \text{Ne}(\Gamma \vdash A_i) \\
\text{Val}(\Gamma, x:(A + B) \vdash C) := \delta(x, y.\text{Val}(\Gamma, y:A \vdash C), z.\text{Val}(\Gamma, z:B \vdash C))
\end{align*}
\]
\[ \text{Val}(\Gamma \vdash A \rightarrow B) := \lambda(x:A) \text{Val}(\Gamma, x:A \vdash B) \]
\[ \text{Val}(\Gamma \vdash A \ast B) := (\text{Val}(\Gamma \vdash A), \text{Val}(\Gamma \vdash B)) \]
\[ \text{Val}(\Gamma \vdash X) := \text{Ne}(\Gamma \vdash X) \quad (X \text{ atomic}) \]

\[ \text{Ne}(\Gamma \vdash B) \supseteq \text{Ne}(\Gamma \vdash A \rightarrow B) \text{Val}(\Gamma \vdash A) \]
\[ \text{Ne}(\Gamma \vdash A_i) \supseteq \pi_i \text{Ne}(\Gamma \vdash A_1 \ast A_2) \]
\[ \text{Ne}(\Gamma \vdash N) \supseteq \{x \mid (x:N) \in \Gamma\} \quad (N \text{ negative: } \rightarrow \ast) \]

You may have recognized the rules of a focused proof system. Focusing suggests how to extend to sums.

\[ \text{Ne}(\Gamma \vdash A_1 + A_2) \supseteq_{i \in \{1,2\}} \sigma_i \text{Ne}(\Gamma \vdash A_i) \]
\[ \text{Val}(\Gamma, x:(A + B) \vdash C) := \delta(x, y.\text{Val}(\Gamma, y:A \vdash C), z.\text{Val}(\Gamma, z:B \vdash C)) \]

This set of rules is not complete yet: 1, (1 \rightarrow X + Y) \vdash X + Y
Saturation, more precisely

We need the “all at once” counterpart of the focused rules ($P, Q$ positives)

\[
\Gamma \vdash_{noninv} Q \quad \Gamma, Q \vdash_{inv} P
\]

\[
\Gamma \vdash_{inv} P
\]

\[
\Gamma \vdash_{noninv} P
\]

(or a single multi-focusing rule)
Saturation, more precisely

We need the “all at once” counterpart of the focused rules ($P$, $Q$ positives)

\[
\frac{\Gamma \vdash_{\text{noninv}} Q \quad \Gamma, Q \vdash_{\text{inv}} P}{\Gamma \vdash_{\text{inv}} P}
\quad
\frac{\Gamma \vdash_{\text{noninv}} P}{\Gamma \vdash_{\text{inv}} P}
\]

(or a single multi-focusing rule)

\[
\text{Val}(\Gamma \vdash P) := \text{Ne}(\text{Sat}(\Gamma) \vdash P)
\quad
\text{Sat}(\Gamma) \supseteq \Gamma
\quad
\text{Sat}(\Gamma) \supseteq \bigcup_Q \text{Ne}(\Gamma \vdash Q)
\quad
\text{Sat}(\Gamma) \supseteq \text{Sat}(\text{Sat}(\Gamma))
\]
Termination II

\[ \text{Val}(\Gamma \vdash P) := \text{Ne}(\text{Sat}(\Gamma) \vdash P) \]

\[ \text{Sat}(\Gamma) \supseteq \Gamma \]
\[ \text{Sat}(\Gamma) \supseteq \bigcup_Q \text{Ne}(\Gamma \vdash Q) \]
\[ \text{Sat}(\Gamma) \supseteq \text{Sat}(\text{Sat}(\Gamma)) \]

We don’t really need to consider all \( Q \). Sub-formula property.

Finitely many \( Q \) of interest: if we erase multiplicity, saturation terminates.

\[ X, (X \rightarrow (X + Y)) \vdash \ldots \]

We could erase multiplicities to (0, 1, many). Terminates, but I suspect it over-approximates.
Conclusion

Claim: unicity is an interesting problem.

We have a clear idea of what we want.

We are complete, canonical, but termination is unclear.

Fruitful links with (multi-)focusing.

Hopefully appliable.
Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott.
Normalization by evaluation for typed lambda calculus with coproducts.

Thorsten Altenkirch and Tarmo Uustalu.
Normalization by evaluation for lambda-2.

Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore.
Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums.

Roy Dyckhoff.
Intuitionistic decision procedures since gentzen.
In Advances in Proof Theory, 2013.

Sam Lindley.
Extensional rewriting with sums.

J. B. Wells and Boris Yakobowski.
Graph-based proof counting and enumeration with applications for program fragment synthesis.