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.

$$(\lambda(x)\, t)\ u \to_\beta t[u/x] \qquad (t : A \to B) =_\eta \lambda(x)\, t\ x$$

$$\pi_i\ (t_1, t_2) \to_\beta t_i \qquad (t : A * B) =_\eta (\pi_1\ t, \pi_2\ t)$$

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

$$(\lambda(x)\, t)\ u \to_\beta t[u/x] \qquad\qquad (t : A \to B) =_\eta \lambda(x)\, t\ x$$

$$\pi_i\ (t_1, t_2) \to_\beta t_i \qquad\qquad (t : A * B) =_\eta (\pi_1\ t, \pi_2\ t)$$

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

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

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.

$$f : (1 \to A + B) \vdash A + B$$

$f\ 1$
$\delta(f\ 1,\ x_1.\sigma_1\ x_1,\ y_1.f\ 1)$

Too Simple

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

$$f : (1 \to A + B) \vdash A + B$$

$f\ 1$
$\delta(f\ 1,\ x_1.\sigma_1\ x_1,\ y_1.f\ 1)$
$\delta(f\ 1,\ x_1.\sigma_1\ x_1,\ y_1.\delta(f\ 1,\ x_2.\sigma_1\ x_2,\ y_2.f\ 1))$

Too Simple

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

$$f : (1 \to A + B) \vdash A + B$$

$f\ 1$
$\delta(f\ 1,\ x_1.\sigma_1\ x_1,\ y_1.f\ 1)$
$\delta(f\ 1,\ x_1.\sigma_1\ x_1,\ y_1.\delta(f\ 1,\ x_2.\sigma_1\ x_2,\ y_2.f\ 1))$
$\delta(f\ 1,\ x_1.\sigma_1\ x_1,\ y_1.\delta(f\ 1,\ x_2.\sigma_1\ x_2,\ y_2.\delta(f\ 1,\ x_3.\sigma_1\ x_3,\ y_3.\dots)))$

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:

$$
\begin{array}{lclclclcl}
v & ::= & \lambda(x{:}A)\, v & \mid & (v, v) & & & \mid & n \\
n & ::= & n\ v & \mid & \pi_1\ n & \mid & \pi_2\ n & \mid & x
\end{array}
$$

Enumerating values and neutrals:

$$
\begin{array}{lcl}
\texttt{Val}(\Gamma \vdash A \to B) & := & \lambda(x{:}A)\, \texttt{Val}(\Gamma, x{:}A \vdash B) \\
\texttt{Val}(\Gamma \vdash A * B) & := & (\texttt{Val}(\Gamma \vdash A),\ \texttt{Val}(\Gamma \vdash B)) \\
\texttt{Val}(\Gamma \vdash X) & := & \texttt{Ne}(\Gamma \vdash X) \qquad (X \text{ atomic})
\end{array}
$$

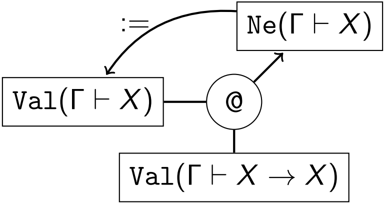
$$
\begin{array}{lcl}
\texttt{Ne}(\Gamma \vdash B) & \supseteq & \texttt{Ne}(\Gamma \vdash A \to B)\, \texttt{Val}(\Gamma \vdash A) \\
\texttt{Ne}(\Gamma \vdash A_i) & \supseteq & \pi_i\, \texttt{Ne}(\Gamma \vdash A_1 * A_2) \\
\texttt{Ne}(\Gamma \vdash N) & \supseteq & \{x \mid (x{:}N) \in \Gamma\} \qquad (N \text{ negative: } \to\ *)
\end{array}
$$

All such elements are in η-long β-normal form: canonicity.

Termination I

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

$$X, X \to X \vdash X$$



[WY04]

$$
\begin{array}{lcl}
\mathtt{Val}(\Gamma \vdash A \to B) & := & \lambda(x\!:\!A)\,\mathtt{Val}(\Gamma, x\!:\!A \vdash B) \\
\mathtt{Val}(\Gamma \vdash A * B) & := & (\mathtt{Val}(\Gamma \vdash A),\ \mathtt{Val}(\Gamma \vdash B)) \\
\mathtt{Val}(\Gamma \vdash X) & := & \mathtt{Ne}(\Gamma \vdash X) \qquad (X \text{ atomic})
\end{array}
$$

$$
\begin{array}{lcl}
\mathtt{Ne}(\Gamma \vdash B) & \supseteq & \mathtt{Ne}(\Gamma \vdash A \to B)\,\mathtt{Val}(\Gamma \vdash A) \\
\mathtt{Ne}(\Gamma \vdash A_i) & \supseteq & \pi_i\,\mathtt{Ne}(\Gamma \vdash A_1 * A_2) \\
\mathtt{Ne}(\Gamma \vdash N) & \supseteq & \{x \mid (x\!:\!N) \in \Gamma\} \qquad (N \text{ negative: } \to *)
\end{array}
$$

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

$$
\begin{array}{lcl}
\texttt{Val}(\Gamma \vdash A \to B) & := & \lambda(x : A)\, \texttt{Val}(\Gamma, x : A \vdash B) \\
\texttt{Val}(\Gamma \vdash A * B) & := & (\texttt{Val}(\Gamma \vdash A),\ \texttt{Val}(\Gamma \vdash B)) \\
\texttt{Val}(\Gamma \vdash X) & := & \texttt{Ne}(\Gamma \vdash X) \qquad (X \text{ atomic})
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{Ne}(\Gamma \vdash B) & \supseteq & \texttt{Ne}(\Gamma \vdash A \to B)\, \texttt{Val}(\Gamma \vdash A) \\
\texttt{Ne}(\Gamma \vdash A_i) & \supseteq & \pi_i\, \texttt{Ne}(\Gamma \vdash A_1 * A_2) \\
\texttt{Ne}(\Gamma \vdash N) & \supseteq & \{x \mid (x : N) \in \Gamma\} \qquad (N \text{ negative: } \to *)
\end{array}
$$

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

$$\texttt{Ne}(\Gamma \vdash A_1 + A_2) \supseteq_{i \in \{1,2\}} \sigma_i\, \texttt{Ne}(\Gamma \vdash A_i)$$

$$\texttt{Val}(\Gamma, x : (A + B) \vdash C) := \delta(x,\ y.\texttt{Val}(\Gamma, y : A \vdash C),\ z.\texttt{Val}(\Gamma, z : B \vdash C))$$

$$\begin{array}{lcl} \texttt{Val}(\Gamma \vdash A \to B) & := & \lambda(x\!:\!A)\,\texttt{Val}(\Gamma, x\!:\!A \vdash B) \\ \texttt{Val}(\Gamma \vdash A * B) & := & (\texttt{Val}(\Gamma \vdash A),\ \texttt{Val}(\Gamma \vdash B)) \\ \texttt{Val}(\Gamma \vdash X) & := & \texttt{Ne}(\Gamma \vdash X) \qquad (X \text{ atomic}) \end{array}$$

$$\begin{array}{lcl} \texttt{Ne}(\Gamma \vdash B) & \supseteq & \texttt{Ne}(\Gamma \vdash A \to B)\,\texttt{Val}(\Gamma \vdash A) \\ \texttt{Ne}(\Gamma \vdash A_i) & \supseteq & \pi_i\,\texttt{Ne}(\Gamma \vdash A_1 * A_2) \\ \texttt{Ne}(\Gamma \vdash N) & \supseteq & \{x \mid (x\!:\!N) \in \Gamma\} \qquad (N \text{ negative: } \to *) \end{array}$$

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

$$\texttt{Ne}(\Gamma \vdash A_1 + A_2) \supseteq_{i \in \{1,2\}} \sigma_i\,\texttt{Ne}(\Gamma \vdash A_i)$$

$$\texttt{Val}(\Gamma, x\!:\!(A + B) \vdash C) := \delta(x,\ y.\texttt{Val}(\Gamma, y\!:\!A \vdash C),\ z.\texttt{Val}(\Gamma, z\!:\!B \vdash C))$$

This set of rules is not complete yet: $1, (1 \to X + Y) \vdash X + Y$

Saturation, more precisely

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

$$\frac{\Gamma \vdash_{noninv} Q \qquad \Gamma, Q \vdash_{inv} P}{\Gamma \vdash_{inv} P} \qquad\qquad \frac{\Gamma \vdash_{noninv} P}{\Gamma \vdash_{inv} P}$$

(or a single multi-focusing rule)

Saturation, more precisely

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

$$\frac{\Gamma \vdash_{noninv} Q \qquad \Gamma, Q \vdash_{inv} P}{\Gamma \vdash_{inv} P} \qquad\qquad \frac{\Gamma \vdash_{noninv} P}{\Gamma \vdash_{inv} P}$$

(or a single multi-focusing rule)

$$\mathtt{Val}(\Gamma \vdash P) := \mathtt{Ne}(\mathtt{Sat}(\Gamma) \vdash P) \qquad \begin{array}{rcl} \mathtt{Sat}(\Gamma) & \supseteq & \Gamma \\ \mathtt{Sat}(\Gamma) & \supseteq & \bigcup_Q \mathtt{Ne}(\Gamma \vdash Q) \\ \mathtt{Sat}(\Gamma) & \supseteq & \mathtt{Sat}(\mathtt{Sat}(\Gamma)) \end{array}$$

Termination II

$$\texttt{Val}(\Gamma \vdash P) := \texttt{Ne}(\texttt{Sat}(\Gamma) \vdash P)$$

$$\begin{array}{rcl} \texttt{Sat}(\Gamma) & \supseteq & \Gamma \\ \texttt{Sat}(\Gamma) & \supseteq & \bigcup_Q \texttt{Ne}(\Gamma \vdash Q) \\ \texttt{Sat}(\Gamma) & \supseteq & \texttt{Sat}(\texttt{Sat}(\Gamma)) \end{array}$$

We don't really need to consider **all** Q. Sub-formula property.

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

$$X, (X \to (X + Y)) \vdash \dots$$

We could erase multiplicities to $(0, 1, \text{many})$.
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.
In *LICS*, pages 303–310, 2001.

Thorsten Altenkirch and Tarmo Uustalu.
Normalization by evaluation for lambda^{-2}.
In *FLOPS*, pages 260–275, 2004.

Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore.
Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums.
In *POPL*, pages 64–76, 2004.

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

Sam Lindley.
Extensional rewriting with sums.
In *TLCA*, pages 255–271, 2007.

J. B. Wells and Boris Yakobowski.
Graph-based proof counting and enumeration with applications for program fragment synthesis.
In *LOPSTR*, pages 262–277, 2004.