Which simple types have a unique inhabitant?

Gabriel Scherer, Didier Rémy

Gallium – INRIA

September 1st, 2015

Code inference

Many programs are **fun** to write. Some parts can be **boring**, though.

We get bored when there is **no choice** to make.
The compiler should **guess** this code for you: code inference.

Code inference

Many programs are **fun** to write. Some parts can be **boring**, though.

We get bored when there is **no choice** to make.
The compiler should **guess** this code for you: code inference.

Some existing examples:

- Overloaded identifier disambiguation.
- Type classes, implicits.
- Proof assistants tactics.

Code inference

Many programs are **fun** to write. Some parts can be **boring**, though.

We get bored when there is **no choice** to make.
The compiler should **guess** this code for you: code inference.

Some existing examples:

- Overloaded identifier disambiguation.
- Type classes, implicits.
- Proof assistants tactics.

We should infer any code **uniquely** determined by its type.

Which types have a unique inhabitant?

Uniquely inhabited typing (Γ, A): inhabited $(\Gamma \vdash t : A)$ and

$$\Gamma \vdash t : A \ \land \ \Gamma \vdash u : A \quad \Longrightarrow \quad \Gamma \vdash t \simeq u : A$$

$(\vdash)$ in a given type system (STLC with atoms, products and **sums**)
$(\simeq)$ modulo some program equivalence (here, $\beta\eta$)

Contribution: a decision procedure (algorithm) in this setting.

Killer example

The `Monad` instance for `Exception` $A \stackrel{\text{def}}{=} A + E$ is canonical.

$$\text{return:}\quad X \quad \to \quad (X + E)$$
$$\text{bind:}\quad X + E \quad \to \quad (X \to Y + E) \quad \to \quad Y + E$$

`Functor` instance also canonical.

`Applicative` functor, two distinct choices.

$$\text{ap:}\quad (X \to Y) + E \quad \to \quad X + E \quad \to \quad Y + E$$

(Which argument to evaluate first?)

$\beta\eta$-equivalence

Type system for pure language: enforces strong normalization.

$$(\lambda x.\, t)\; u \rightarrow_\beta t[u/x] \qquad\qquad (t : A \rightarrow B) =_\eta \lambda x.\, t\; x$$

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

$$\texttt{match}\ (\mathsf{L}\ t)\ \texttt{with}\ \left|\begin{array}{l}\mathsf{L}\ x_1 \rightarrow u_1\\ \mathsf{R}\ x_2 \rightarrow u_2\end{array}\right. \rightarrow_\beta u_1[t/x_1]$$

$$(t : A + B) =_\eta \texttt{match}\ t\ \texttt{with}\ \left|\begin{array}{ll}\mathsf{L}\ x_1 \rightarrow & \mathsf{L}\ x_1\\ \mathsf{R}\ x_2 \rightarrow & \mathsf{R}\ x_2\end{array}\right.$$

$\beta\eta$-equivalence

Type system for pure language: enforces strong normalization.

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

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

$$\texttt{match}\ (\mathsf{L}\ t)\ \texttt{with}\ \left|\begin{array}{l} \mathsf{L}\ x_1 \to u_1 \\ \mathsf{R}\ x_2 \to u_2 \end{array}\right. \to_\beta\ u_1[t/x_1]$$

$$(t : A + B) =_\eta \texttt{match}\ t\ \texttt{with}\ \left|\begin{array}{ll} \mathsf{L}\ x_1 \to & \mathsf{L}\ x_1 \\ \mathsf{R}\ x_2 \to & \mathsf{R}\ x_2 \end{array}\right.$$

But:

$$(t, u) \stackrel{?}{=} \texttt{match}\ t\ \texttt{with}\ \left|\begin{array}{l} \mathsf{L}\ x_1 \to (\mathsf{L}\ x_1, u) \\ \mathsf{R}\ x_2 \to (\mathsf{R}\ x_2, u) \end{array}\right.$$

$\beta\eta$-equivalence

Type system for pure language: enforces strong normalization.

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

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

$$\texttt{match}\ (\mathsf{L}\ t)\ \texttt{with} \left| \begin{array}{l} \mathsf{L}\ x_1 \to u_1 \\ \mathsf{R}\ x_2 \to u_2 \end{array} \right. \to_\beta u_1[t/x_1]$$

$$(t : A + B) =_\eta \texttt{match}\ t\ \texttt{with} \left| \begin{array}{ll} \mathsf{L}\ x_1 \to & \mathsf{L}\ x_1 \\ \mathsf{R}\ x_2 \to & \mathsf{R}\ x_2 \end{array} \right.$$

But:

$$(t, u) \stackrel{?}{=} \texttt{match}\ t\ \texttt{with} \left| \begin{array}{l} \mathsf{L}\ x_1 \to (\mathsf{L}\ x_1, u) \\ \mathsf{R}\ x_2 \to (\mathsf{R}\ x_2, u) \end{array} \right. \qquad C[\square] \stackrel{\text{def}}{=} (\square, u)$$

$\beta\eta$-equivalence

Type system for pure language: enforces strong normalization.

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

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

$$\texttt{match}\; (\mathsf{L}\; t)\; \texttt{with} \left|\begin{array}{l} \mathsf{L}\; x_1 \to u_1 \\ \mathsf{R}\; x_2 \to u_2 \end{array}\right. \to_\beta u_1[t/x_1]$$

$$\forall C[\square : A + B],$$
$$C[t : A + B] =_\eta \texttt{match}\; t\; \texttt{with} \left|\begin{array}{l} \mathsf{L}\; x_1 \to C[\mathsf{L}\; x_1] \\ \mathsf{R}\; x_2 \to C[\mathsf{R}\; x_2] \end{array}\right.$$

$\beta\eta$-equivalence

Type system for pure language: enforces strong normalization.

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

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

$$\texttt{match}\ (\mathsf{L}\ t)\ \texttt{with} \left|\begin{array}{l} \mathsf{L}\ x_1 \to u_1 \\ \mathsf{R}\ x_2 \to u_2 \end{array}\right. \to_\beta\ u_1[t/x_1]$$

$$\forall C[\square : A + B],$$
$$C[t : A + B] =_\eta \texttt{match}\ t\ \texttt{with} \left|\begin{array}{l} \mathsf{L}\ x_1 \to C[\mathsf{L}\ x_1] \\ \mathsf{R}\ x_2 \to C[\mathsf{R}\ x_2] \end{array}\right.$$

Equivalence algorithm decidable (Neil Ghani, 1995). Unicity?

Unicity

A search process, enumerating distinct normal forms.

We know about **proof** search.

We know about **program** equivalence.

Unicity

A search process, enumerating distinct normal forms.

We know about **proof** search.

We know about **program** equivalence.

Is there a proof or type system that characterizes distinct programs?
No duplicates.

the Graal of program equivalence

A type system for "normal forms" ($\Gamma \vdash_{\mathsf{nf}} v : A$) that is

canonical: syntactically distinct $\Rightarrow$ semantically distinct

complete: each STLC program is equivalent to a typable normal form

Unicity test by goal-directed search in this system.

$$\Gamma \vdash_{\mathsf{nf}} ? : A$$

Contribution: this, for simply-typed λ-calculus with sums.

β-short (weak)-η-long does not cut it

$f : (X \rightarrow Y + Y), x : X \vdash ? : X$

β-short (weak)-η-long does not cut it

$f : (X \to Y + Y), x : X \vdash ? : X$

x

β-short (weak)-η-long does not cut it

$f : (X \to Y + Y), x : X \vdash ? : X$

x

$$\texttt{match}\ f\ x\ \texttt{with} \left|\begin{array}{l} \mathrm{L}\ y_1 \to x \\ \mathrm{R}\ y_2 \to x \end{array}\right.$$

β-short (weak)-η-long does not cut it

$f : (X \to Y + Y), x : X \vdash ? : X$

x

$\texttt{match } f\ x \texttt{ with} \left|\begin{array}{l} \mathsf{L}\ y_1 \to x \\ \mathsf{R}\ y_2 \to x \end{array}\right.$

$\texttt{match } f\ x \texttt{ with} \left|\begin{array}{l} \mathsf{L}\ y_1 \to \texttt{match } f\ x \texttt{ with} \left|\begin{array}{l} \mathsf{L}\ z_1 \to x \\ \mathsf{R}\ z_2 \to x \end{array}\right. \\ \mathsf{R}\ y_2 \to x \end{array}\right.$

$\ldots$

β-short (weak)-η-long does not cut it

$f : (X \to Y + Y), x : X \vdash ? : X$

x

$\texttt{match}\ f\ x\ \texttt{with} \left| \begin{array}{l} \mathsf{L}\ y_1 \to x \\ \mathsf{R}\ y_2 \to x \end{array} \right.$

$\texttt{match}\ f\ x\ \texttt{with} \left| \begin{array}{l} \mathsf{L}\ y_1 \to \texttt{match}\ f\ x\ \texttt{with} \left| \begin{array}{l} \mathsf{L}\ z_1 \to x \\ \mathsf{R}\ z_2 \to x \end{array} \right. \\ \mathsf{R}\ y_2 \to x \end{array} \right.$

$\dots$

In general: equivalent programs may differ by matching on the same subterm at different places.

Need to quotient over that.

Intuition

Enforce sum elimination **as early as possible**.

During goal-directed search, we don't know yet which sums will be useful.
(Type system: maximally-early introduction is a non-local criterion)

Cannot enforce early elimination of all useful subterms.

Intuition

Enforce sum elimination **as early as possible**.

During goal-directed search, we don't know yet which sums will be useful.
(Type system: maximally-early introduction is a non-local criterion)

Cannot enforce early elimination of all useful subterms.

Just eliminate **all possible sums** : saturation.

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{? : X}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{? : X}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in } ? : X}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in } ? : X}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in match } z \texttt{ with } \left|\begin{array}{l} \mathsf{L}\ {y_1}^{Y} \to ? : X \\ \mathsf{R}\ {y_2}^{Y} \to ? : X \end{array}\right.}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in match } z \texttt{ with } \left|\begin{array}{l} \mathsf{L}\ y_1{}^{Y} \to ? : X \\ \mathsf{R}\ y_2{}^{Y} \to ? : X \end{array}\right.}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in match } z \texttt{ with } \left| \begin{array}{l} \mathsf{L}\ {y_1}^{Y} \to ? : X \\ \mathsf{R}\ {y_2}^{Y} \to ? : X \end{array} \right.}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in match } z \texttt{ with } \left|\begin{array}{l} \mathsf{L}\ {y_1}^{Y} \to x \\ \mathsf{R}\ {y_2}^{Y} \to ? : X \end{array}\right.}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in match } z \texttt{ with } \left|\begin{array}{l} \mathsf{L}\ y_1{}^{Y} \to x \\ \mathsf{R}\ y_2{}^{Y} \to x \end{array}\right.}$$

Demo time

Implementation available:
https://gitlab.com/gasche/unique-inhabitant

$$f : (X \to Y + Y), x : X$$

$$\vdash$$

$$\boxed{\texttt{let } z^{Y+Y} = f\ x \texttt{ in match } z \texttt{ with } \left|\begin{array}{l} \mathsf{L}\ {y_1}^{Y} \to x \\ \mathsf{R}\ {y_2}^{Y} \to x \end{array}\right.}$$

Final result: zero, one or two (distinct) terms.

Saturation

We alternate goal-directed (backward) search and (forward) saturation.

Saturation of Γ: compute **all possible** neutral terms $\Gamma \vdash n : A + B$ and deconstruct (some of) them.

Saturation

We alternate goal-directed (backward) search and (forward) saturation.

Saturation of Γ: compute **all possible** neutral terms $\Gamma \vdash n : A + B$ and deconstruct (some of) them.

freshness condition: neutrals typeable in a strictly smaller Γ are old, don't deconstruct them again

$\Rightarrow$ canonicity

Saturation

We alternate goal-directed (backward) search and (forward) saturation.

Saturation of Γ: compute **all possible** neutral terms $\Gamma \vdash n : A + B$ and deconstruct (some of) them.

freshness condition: neutrals typeable in a strictly smaller Γ are old, don't deconstruct them again

$\Rightarrow$ canonicity

subformula property: the sums $(A + B)$ that appear in Γ suffice

two-or-more property: at most two different neutrals of each type suffice

$\Rightarrow$ termination

Conclusion

We build upon proof theory and logic programming – **focusing** (bidirectional typing, better).

Contribution: a focused **saturating** proof/type system, canonical and computationally complete for STLC with sums.
$\Rightarrow$ **decidability** of unique inhabitation
$\Rightarrow$ new insights on program equivalence (empty type?)

In the paper: other practical examples, detailed related work.

Conclusion

We build upon proof theory and logic programming – **focusing** (bidirectional typing, better).

Contribution: a focused **saturating** proof/type system, canonical and computationally complete for STLC with sums.
$\Rightarrow$ **decidability** of unique inhabitation
$\Rightarrow$ new insights on program equivalence (empty type?)

In the paper: other practical examples, detailed related work.

Future work: extend to polymorphism and... dependent types.

Conclusion

We build upon proof theory and logic programming – **focusing** (bidirectional typing, better).

Contribution: a focused **saturating** proof/type system, canonical and computationally complete for STLC with sums.
$\Rightarrow$ **decidability** of unique inhabitation
$\Rightarrow$ new insights on program equivalence (empty type?)

In the paper: other practical examples, detailed related work.

Future work: extend to polymorphism and... dependent types.

Thanks. Any question?