Unicity of type inhabitants; a Work in Progress

Gabriel Scherer

Gallium (INRIA Paris-Rocquencourt)

May 30, 2013
What? This talk is about a problem rather than a solution.

The question

Given a type $T$, does $T$ have a unique inhabitant? (modulo observational equivalence)

We need to fix a type system and a pure term language.

Let’s start with the simply-typed lambda-calculus (STLC) with arrows, products and sums.

Remark: (non-)relation with singleton types $\{= M\}$. 
Why? Practical motivations

A principal approach to code inference.

Informal conjecture

When programmers feel bored even before writing the code, it’s because there are no choices to be made.

Provide a feature to fill some hole (?), that fails if there are several possible choices.

```ocaml
val swap : 'a 'b 'c. ('a * 'b * 'c) -> ('a * 'c * 'b)

let swap = ?
```
Code inference example

Most general form \((\Gamma \vdash ? : \sigma)\).
Default context choice \((\emptyset)\), inferred type.
Code inference example

Most general form ($\Gamma \vdash ? : \sigma$).
Default context choice ($\emptyset$), inferred type.

Type_variant (  
  List.map (fun (name, name_loc, ctys, option, loc) ->  
      name, List.map (fun cty -> cty.ctyp_type) ctys, option)  
  cstrs
)
Code inference example

Most general form ($\Gamma \vdash ? : \sigma$).
Default context choice ($\emptyset$), inferred type.

```
Type_variant (  
    List.map (fun (name, name_loc, ctys, option, loc) ->  
        name, List.map (fun cty -> cty.ctyp_type) ctys, option)  
    cstrs
  )
```

Type_variant (  
    List.map (? (List.map (fun cty -> cty.ctyp_type))) cstrs  
)

Analysis of the typing/code. For 100 instances of  
List.map (fun ...), about 30 of them could use code inference.
Uses of code inference

Non-interactive use:
- glue between trivial parts of the program
  I forgot the argument order... but only one type-correct choice.
- more ambitious: generic boilerplate
  Is there a type whose unique inhabitant is \texttt{List.map}? (next slide)
- re-expresses other code inference feature
  type classes, implicits...

Interactive use: program-assistant tactics?

Note: we’re not using scoring/heuristics [recent C\#, Scala work].

Interaction between type and term inference. You can’t do both at once, but they can cooperate.
What’s a precise type for List.map?

\[ \forall \alpha \beta. (\alpha \rightarrow \beta) \rightarrow (List \ \alpha \rightarrow List \ \beta) \]

(? f li)
What’s a precise type for List.map?

\[ \forall \alpha \beta. (\alpha \to \beta) \to (\text{List} \ \alpha \to \text{List} \ \beta) \]  
\[ (\ ? \ f \ \text{li}) \]

\[ \forall \alpha \beta. (\alpha \rightarrow \beta) \to (\text{List} \ \alpha \rightarrow \text{List} \ \beta) \]  
\[ (\ ? \ f \circ \text{li}) \]
What’s a precise type for List.map?

\[
\forall \alpha \beta. (\alpha \to \beta) \to (\text{List } \alpha \to \text{List } \beta) \quad (? \ f \ \text{li}) \\
\forall \alpha \beta. (\alpha \leftarrow \beta) \to (\text{List } \alpha \leftarrow \text{List } \beta) \quad (? \ f \ \leftarrow \text{li}) \\
\forall \alpha \beta. (\alpha \to \beta) \to (\text{List } \alpha \to \text{List } \beta) \quad (? \ f \ \leftarrow \text{li})
\]

We are:

- using more expressive types than the host language ones
- producing purer terms
What’s a precise type for `List.map`?

\[ \forall \alpha \beta. (\alpha \to \beta) \to (\text{List } \alpha \to \text{List } \beta) \quad (\text{f li}) \]
\[ \forall \alpha \beta. (\alpha \to \beta) \to (\text{List } \alpha \to \text{List } \beta) \quad (\text{f o-1i}) \]
\[ \forall \alpha \beta. (\alpha \to \beta) \to (\text{List } \alpha \to \text{List } \beta) \quad (\text{f l-1i}) \]

We are:
- using more expressive types than the host language ones
- producing purer terms

For `fold`, need to move to dependent types; decreasing gains.

\[ \forall \alpha \beta, \quad \forall (A : \star)(P : \text{List } A \to \star), \]
\[ \beta \to \]
\[ (\alpha \to \beta \to \beta) \to \]
\[ \text{List } \alpha \to \beta \]

\[ P \text{ nil } \to \]
\[ (\forall (a : A)(l : \text{List } A), P \text{ l } \to P (\text{cons } a \text{ l})) \to \]
\[ \forall (l : \text{List } A), \quad P \text{ l} \]
Why? Theoretical motivations

It’s fun: a question so simple to state must have interesting answers.

It’s an excuse to look at the proof-search research with different eyes. Look at *dynamic behavior*, rather than just yes/no inhabitation problems.
Caution required

Intuitionistic sequent calculi generally have a *contraction* rule

\[
\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \times B \vdash C}
\]

You can get rid of contraction if you preserve formulas at use site.

\[
\frac{\Gamma, A \times B, A, B \vdash C}{\Gamma, A \times B \vdash C}
\]

For sums and pairs, it is in fact not needed, but it is for arrows.

\[
\frac{\Gamma, A \rightarrow B \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \rightarrow B \vdash C}
\]

Dropping the arrow on the right is complete, but not dynamically so.
How? High-level directions

I recently started working on this. I will warmly welcome any suggestion.

Directions to explore in parallel

- Keep looking for related work.
  Diverse, hard to find, not well-connected.

- Enrich type systems to express more types with unique inhabitants.
  Substructural logics, polymorphic (parametricity), dependent types.

- Devise practical algorithms to check unicity.
  (Bulk of this talk)
Some related work

J. B. Wells and Boris Yakobowski.
Graph-based proof counting and enumeration with applications for program fragment synthesis.
In LOPSTR 2004.

Takahito Aoto.
Uniqueness of normal proofs in implicational intuitionistic logic.

Sabine Broda and Luís Damas.
On long normal inhabitants of a type.

Pierre Boureau and Sylvain Salvati.
Game semantics and uniqueness of type inhabitance in the simply-typed λ-calculus.
*Typed Lambda-Calculi and Applications*, 2011.
A few words on [Yakobowski and Wells]

Consider the graph whose nodes are sequent, and edges are valid inference rules.

When context is a set, subformula property implies finiteness.

Can be seen as a “memoization” techniques: cycles in the graph can be dropped without hurting completeness.

(Idea of the paper: from this graph structure with set-contexts, deduce information about the infinite structure of multiset-contexts.)
Facing the Decision problem: Unicity for STLC

Obvious idea: enumerate proofs, check that there is only one.

Usual problem: irrelevant permutations allowed by the proof system

\[
\begin{align*}
A, B, C, D & \vdash E \\
A, B, C \ast D & \vdash E \\
A \ast B, C \ast D & \vdash E
\end{align*}
\]

Two approaches:

- do equivalence checks after enumeration to remove duplicates (simple, not fun, not efficient in general)
- change the proof system to remove those duplicates
Mandatory step towards duplicates-free systems: Focusing
Quotient by reordering of \{\text{non,} \text{inversible}\} proof steps.

\[
\begin{align*}
\Gamma; \Delta, A \vdash B & \quad \quad \Gamma; \Delta, A, B \vdash C & \quad \quad \Gamma; \Delta, A \vdash C \quad \quad \Gamma; \Delta, B \vdash C \\
\Gamma; \Delta \vdash A \to B & \quad \quad \Gamma; \Delta, A \times B \vdash C & \quad \quad \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, X; \Delta \vdash C & \quad \quad \Gamma \vdash [P] & \quad \quad \Gamma, [N] \vdash X \quad \quad \Gamma, [N] \vdash P \\
\Gamma; \Delta, X \vdash C & \quad \quad \Gamma; \emptyset \vdash P & \quad \quad \Gamma, N; \emptyset \vdash X \quad \quad \Gamma; P \vdash Q \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash [A] & \quad \quad \Gamma \vdash [B] \\
\Gamma \vdash [A \times B] & \quad \quad \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash [A_i] & \quad \quad \Gamma; \emptyset \vdash N \\
\Gamma \vdash [A_1 + A_2] & \quad \quad \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, [N] \vdash A \to B & \quad \quad \Gamma \vdash [A] & \quad \quad \Gamma, [N] \vdash B \\
\Gamma, [X] \vdash X & \quad \quad \\
\end{align*}
\]

Focused proofs correspond to $\beta$-normal, $\eta$-long terms. Good!
Shortcomings of Focusing

Too many proofs of $(X \rightarrow Y + Z) \rightarrow X \rightarrow X$.

fun f x -> ?
Shortcomings of Focusing

Too many proofs of \((X \rightarrow Y + Z) \rightarrow X \rightarrow X\).

fun f x -> ?

fun f x -> x

fun f x -> match f x with
    | L y -> ?
    | R z -> ?

Remark: \((Y + Z) \rightarrow X \rightarrow X\) would be fine.
Shortcomings of Focusing

Too many proofs of \((X \rightarrow Y + Z) \rightarrow X \rightarrow X\).

\[
\begin{align*}
\text{fun } f \ x \rightarrow \ ? & \quad \rightarrow \ x \\
\text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with} & \quad \text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with} \\
| \ L \ y \rightarrow \ ? & \quad | \ L \ y \rightarrow \ x \\
| \ R \ z \rightarrow \ ? & \quad | \ R \ z \rightarrow \ x \\
\end{align*}
\]

\[
\begin{align*}
\text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with} & \quad \text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with} \\
| \ L \ y \rightarrow (\text{match } f \ x \ \text{with} & \quad | \ L \ y \rightarrow \ x \\
| \ L \ y' \rightarrow \ ? & \quad | \ R \ z \rightarrow (\text{match } f \ x \ \text{with} \\
| \ R \ z \rightarrow \ ?) & \quad | \ L \ y \rightarrow \ ? \\
| \ R \ z \rightarrow \ x & \quad | \ R \ z' \rightarrow \ ?) \\
\end{align*}
\]

\[
\begin{align*}
\text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with} & \quad \text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with} \\
| \ L \ y \rightarrow (\text{match } f \ x \ \text{with} & \quad | \ L \ y \rightarrow \ x \\
| \ L \ y' \rightarrow \ ? & \quad | \ R \ z \rightarrow (\text{match } f \ x \ \text{with} \\
| \ R \ z \rightarrow \ ?) & \quad | \ L \ y \rightarrow \ ? \\
| \ R \ z \rightarrow \ x & \quad | \ R \ z' \rightarrow \ ?) \\
\end{align*}
\]
Shortcomings of Focusing

Too many proofs of \((X \rightarrow Y + Z) \rightarrow X \rightarrow X\).

\[
\text{fun } f \ x \rightarrow \ ? \quad \text{fun } f \ x \rightarrow \ ?
\]

\[
\text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with}
= \quad \text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with}
| \ L \ y \ \rightarrow \ ? \quad | \ L \ y \ \rightarrow \ x
| \ R \ z \ \rightarrow \ ? \quad | \ R \ z \ \rightarrow \ x
\]

\[
\text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with}
= \quad \text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with}
| \ L \ y \ \rightarrow \ (\text{match } f \ x \ \text{with}
| | \ L \ y' \ \rightarrow \ ?
| | \ R \ z \ \rightarrow \ ?)
| \ R \ z \ \rightarrow \ x
| \ R \ z' \ \rightarrow \ ?)
\]

\[
\text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with}
= \quad \text{fun } f \ x \rightarrow \ \text{match } f \ x \ \text{with}
| \ L \ y \ \rightarrow \ (\text{match } f \ x \ \text{with}
| | \ L \ y' \ \rightarrow \ ? \ | \ R \ z \ \rightarrow \ ?)
| \ R \ z \ \rightarrow \ (\text{match } f \ x \ \text{with}
| | \ L \ y \ \rightarrow \ ? \ | \ R \ z' \ \rightarrow \ ?)
\]

Remark: \((Y + Z) \rightarrow X \rightarrow X\) would be fine.
η-equivalence for sum types

Weak, local equivalence:
\[
\begin{align*}
e &= \text{match } e \text{ with} \\
| \text{L } y &\rightarrow \text{L } y \\
| \text{R } z &\rightarrow \text{R } z
\end{align*}
\]

In particular:
\[
\begin{align*}
t &= \text{match } e \text{ with} \\
| \text{L } y &\rightarrow t \\
| \text{R } y &\rightarrow t
\end{align*}
\]

and...

Full, non-local, categorical equivalence

\[
\begin{align*}
C[e] &= \text{match } e \text{ with} \\
| \text{L } y &\rightarrow C[\text{L } y] \\
| \text{R } z &\rightarrow C[\text{R } z]
\end{align*}
\]
match e with
| L y -> C1[y][match e with M1]
| R z -> C2[z][match e with M2]

\[ \text{=} \text{(strong } \eta\text{-sum)} \]
match e with
  | L y -> C1[y] [match e with M1]
  | R z -> C2[z] [match e with M2]

= (strong \(\eta\)-sum)

match e with
  | L y0 ->
    (match L y0 with
      | L y -> C1[y] [match L y0 with M1]
      | R z -> C2[z] [match L y0 with M2])
  | R z0 ->
    (match R z0 with
      | L y -> C1[y] [match R z0 with M1]
      | R z -> C2[z] [match R z0 with M2])

= (\(\beta\)-sum)
match e with
  | L y -> C1[y][match e with M1]
  | R z -> C2[z][match e with M2]

= (strong $\eta$-sum)

match e with
  | L y0 ->
    (match L y0 with
     | L y -> C1[y][match L y0 with M1]
     | R z -> C2[z][match L y0 with M2])
  | R z0 ->
    (match R z0 with
     | L y -> C1[y][match R z0 with M1]
     | R z -> C2[z][match R z0 with M2])

= ($\beta$-sum)

match e with
  | L y0 -> C1[y0][match L y0 with M1]
  | R z0 -> C2[z0][match R z0 with M2]
Checking strong $\eta$-equivalence for sums

[Balat and Di Cosmo, 2004]; [Lindley, 2005]
General idea: move sum destructions as early as possible, then remove duplicates.

fun f g ...
  match ... with
    ...
    fun x y ...
      match ... with
        ...
        fun g z ...
          match f x with ...
Remark

$(\to)$ and $(\oplus)$ are enemies in intuitionistic logic. Both can be introduced reversibly, but not both at the same time.

\[
\Gamma, A \vdash B \\
\Gamma \vdash A \to B
\]

\[
\Gamma \vdash A_i \\
\Gamma \vdash A_1 + A_2
\]

\[
\Gamma, A \vdash B \\
\Gamma \vdash (A \to B), \Delta
\]

\[
\Gamma \vdash A_1, A_2, \Delta \\
\Gamma \vdash (A_1 + A_2), \Delta
\]

(Remark in remark: intuitionistic focusing makes arbitrary choices. Related to various translations into linear logic [Chaudhuri and Miller].)
Goal: integrate sum equivalence into proof search.

Our idea: Context saturation

Each time we introduce new things in the context, do *all possible* *destructions* that involve them *and* might get used in a proof term.
Saturation example

With saturation,

\[
\text{fun } f \ x \rightarrow \\
\text{match } f \ x \text{ with} \\
\quad | \ L \ y \rightarrow (\text{match } f \ x \text{ with } L \ y' \rightarrow \ ? \ | \ R \ z \rightarrow \ ?) \\
\quad | \ R \ z \rightarrow (\text{match } f \ x \text{ with } L \ y \rightarrow \ ? \ | \ R \ z \rightarrow \ ?)
\]

is ruled out. But for:

\[
\text{fun } f \ x \rightarrow \text{match } f \ x \text{ with} \\
\quad | \ L \ y \rightarrow x \\
\quad | \ R \ z \rightarrow x
\]

it depends.

It would be ruled out as well if our proof search was sophisticated enough to notice that neither \( Y \) nor \( Z \) can help prove \( X \).
Saturation Facts

Conjecture: a search calculus enforcing saturation solves the sum equivalence problem.

Danger: without clever ideas for checking “potential usefulness” of destructs, this method is impractical.

Hope: this approach allows to solve not only the -sum problem, but generalizes nicely to other constructors with tricky equalities.

Embarassing detail: no other example known, so generalization of little value; suggestions appreciated.
But: saturation is not obvious

A saturating calculus surprisingly hard to define.

Nave idea: at the end of each reversible phase (or incrementally during them), saturate the context. Focusing phases will only run with saturated contexts.

Context saturation operation sat(Γ)?

sat(Γ; A → B) = sat(Γ, A → B; B) when Γ ⊢ A.
Problem when A of the form B → C: re-saturation needed (recursively).

Termination? Practicality? We need something clever here.
Exploring the theorem proving countryside

Saturation seems costly in general, but sometimes it is required to solve inhabitation.

\[(X \rightarrow Y + Z) \rightarrow X \rightarrow Z + Y\]

Let’s look at the automated theorem proving literature. Hopefully their techniques/optimizations have helpful semantic content.

Most research centered on classical logic – easy shortcuts due to arrow/sum permutation. But:

- The inverse method has been adapted to linear [Chaudhuri], intuitionistic logic. Sequent-saturation technique – may help for context saturation?
- Connection-based, or Matrix-based calculi; horribly complicated, but probably helpful to avoid redundant work.
Presentation of the Inverse Method

Based on a termination argument that we can reuse for saturation: the subformula property.
Subformulas of \((X \rightarrow Y + Z) \rightarrow X \rightarrow X\)
- (positively) \(X; (X \rightarrow X); ((X \rightarrow Y + Z) \rightarrow X \rightarrow X)\)
- (negatively) \((Y + Z); (X \rightarrow Y + Z); X\)

Some rules:

\[
\begin{align*}
\frac{X \text{ atom}}{X \vdash X} & \quad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} & \quad \frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \ast B} \\
\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} & \quad \frac{\Gamma \vdash B \quad A \notin \Gamma}{\Gamma \vdash A \rightarrow B}
\end{align*}
\]
Inverse Method: Pros and Cons

Note: already encoded some neededness information.

Can be refined with
- polarization
- focusing (derived constructors)

Has been used in practice to refute provability (Imogen, [McLaughlin and Pfenning, 2008]), so is practically able to perform saturation.

But: unclear how its inherent sharing/subsumption preserves the dynamic semantics of proof-terms.
Going further

Current idea: perform an inverse method to forward-explore the sequent space, then go backward to collect maximized proof.

Going on in parallel:

- “path calculi” are optimizations techniques on top of the inverse method that allow to further prune the search space [Degtyarev and Voronkov, 2001] and may help even further on “neededness” question.
- understand and integrate ideas from connection-based calculi [Galmiche and Méry, recent]
Thanks.

Any questions?