Deciding unique inhabitants with sums
(work in progress)

Gabriel Scherer, Didier Rémy

INRIA – Gallium

Our ongoing work focuses on types that have a unique inhabitant—modulo program equivalence. If we were able to detect when such types appear in a program, we could perform *program inference* to releave the programmer from the obligation to write the less-interesting parts of the program. Unicity of inhabitant is a strong form of principality: inference cannot make a "wrong" guess as there is only one choice. This property can be contrasted to two different notions, inhabited types and subtyping coercions, that it respectively refines and extends:

- The usual provability notion of "having at least an inhabitant", which is the one of concern when using strongly-typed lambda-calculi to prove mathematical facts. In this setting, the dynamic semantics of terms is ignored, which makes it unsuited to *program* inference.
- The common programming notion of erasable subtyping between two types $A \leq B$: inference systems for subtyping can be seen as a restrictive type system for functions whose terms have a computational interpretation which is *always* the identity function—when it is inhabited, $A \leq B$ has a unique inhabitant. On the contrary, some types have unique inhabitants that are not the identity functions, such as `swap :` $\forall AB.\ (A * B) \to (B * A)$.

To decide uniqueness, we must be able to enumerate the *distinct* terms at a given type. As a first step, we consider the simply-typed lambda-calculus with arrows, product and sums. We are looking for a term enumeration process that is *complete*, i.e., it does not miss any computational behavior, and *canonical*, i.e., it has no duplicates. We propose an approach based on *saturation*, with encouraging results, although termination is still a conjecture.

Computational completeness Some existing proof calculi, such as *contraction-free* calculi [Dyc13], make simplifications designed to make provability decision tractable, but throw away some behaviors, loosing computational completeness. The following rule, which drops a function after its first invocation, preserves provability even in a contraction-free calculus:

$$\frac{\Gamma, A \to B \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}$$

Consider a datatype $A * (A \to (A * B))$ of infinite streams of B with internal state A. If the generating function is dropped after its first result, there is exactly one proof of $A * (A \to (A * B)) \vdash B$ (getting the first element of the stream), while there are infinitely many observably distinct programs at that type.

Focusing Focusing [And92] imposes a phase discipline on derivations by distinguishing *invertible* and *non-invertible* inference rules—invertible rules are those whose inverse is derivable. In absence of sums, focused proofs are in exact correspondance to β-short η-long proof terms; it is computationally complete. Starting from a grammar for values and neutrals:

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

we can define the set $\mathtt{Val}(\Gamma \vdash A)$ of (distinct) values of type A in the environment Γ, along with the set $\mathtt{Ne}(\Gamma \vdash A)$ of neutrals at this type, by lifting notations from terms to sets of terms:

$$\begin{array}{lll@{\quad}lll}
\mathtt{Val}(\Gamma \vdash A \to B) & := & \lambda(x\!:\!A)\,\mathtt{Val}(\Gamma, x\!:\!A \vdash B) & \mathtt{Ne}(\Gamma \vdash A_i) & \supseteq & \pi_i\,\mathtt{Ne}(\Gamma \vdash A_1 * A_2) \\
\mathtt{Val}(\Gamma \vdash A * B) & := & (\mathtt{Val}(\Gamma \vdash A),\ \mathtt{Val}(\Gamma \vdash B)) & \mathtt{Ne}(\Gamma \vdash B) & \supseteq & \mathtt{Ne}(\Gamma \vdash A \to B)\,\mathtt{Val}(\Gamma \vdash A) \\
\mathtt{Val}(\Gamma \vdash X) & := & \mathtt{Ne}(\Gamma \vdash X) \quad (X \text{ atomic}) & \mathtt{Ne}(\Gamma \vdash N) & \supseteq & \{x \mid (x\!:\!N) \in \Gamma\}
\end{array}$$

Note that `Val` is structurally recursive on the input type; it corresponds to the invertible rules. On the contrary, `Ne` corresponds to non-invertible rules, and is defined as a least fixpoint: this is where the aforementioned termination control techniques are necessary. The following property is key to showing that this enumeration is canonical:

Lemma 1 (Canonicity of negative neutrals). *For any $n_1, n_2 \in \mathtt{Ne}(\Gamma \vdash A)$, if n_1 and n_2 are syntactically distinct, then they are distinct for contextual equivalence.*

This specification can be turned into a decision procedure; termination arguments are of two sorts. First, the *subformula property* gives a finite bound on the number of types that will be considered. Second, cycles in the equations defining `Ne` (in particular types with an infinite number of distinct inhabitants, such as Church integers $X \to (X \to X) \to X$) can be worked upon using a graph-based representation in the style of Wells and Yakobowski [WY04].

Unfortunately, focusing alone does not capture the notably difficult η-equivalence for sums. Writing $\delta(e_1,\, x.e_2,\, y.e_3)$ for $(\mathtt{match}\ e_1\ \mathtt{with}\ \mathtt{inl}\,x \to e_2 \mid \mathtt{inr}\,y \to e_3)$, the two following terms correspond to distinct, but observationally equivalent, focused proofs of $(1 \to A+B) \to A+B$: $(\lambda(f)\,\delta(f\ 1,\, x.\,\mathtt{inl}\,x,\, y.\,\mathtt{inr}\,y))$ and $(\lambda(f)\,\delta(f\ 1,\, x.\,\mathtt{inl}\,x,\, y.\,\delta(f\ 1,\, x'.\,\mathtt{inl}\,x',\, y'.\,\mathtt{inr}\,y')))$.

Saturation Coupling an enumeration of focused proofs with an equivalence checking procedure for sums [Lin07] does not work, as there may be infinitely many redundant copies. The main idea of these algorithms is to move sum elimination as high in the term as possible. We thus propose to eliminate sums as early as possible during term generation, by simutaneously eliminating all neutrals of any positive type P when the goal is itself a positive Q:

$$\mathtt{Val}(\Gamma, x\!:\!(A+B) \vdash C) := \delta(x,\, y.\mathtt{Val}(\Gamma, y\!:\!A \vdash C),\, z.\mathtt{Val}(\Gamma, z\!:\!B \vdash C))$$
$$\mathtt{Ne}(\Gamma \vdash A_1 + A_2) \supseteq_{i \in \{1,2\}} \sigma_i\ \mathtt{Ne}(\Gamma \vdash A_i)$$
$$\mathtt{Val}(\Gamma \vdash Q) := \mathtt{Ne}(\Gamma \vdash Q) \cup (\mathtt{let}\ \Delta = (\cup_P\,\mathtt{Ne}(\Gamma \vdash P))\ \mathtt{in}\ \mathtt{Val}(\Gamma, \Delta \vdash Q))$$

This saturation process is complete and canonical. Remarkably, it is strongly related to the notion of "maximal multi-focusing" [CMS08]; our proofs are also maximal in this sense.

While completeness and non-duplicability are relatively simple, termination is still a conjecture. In addition to the "inner" non-termination related to the definition of $\mathtt{Ne}(\Gamma \vdash A)$ (infinite number of distinct terms), there is now an "outer" termination problem of infinite alternance of `Val` and `Ne` layers. Previous approaches gave either completeness and termination (focusing, with duplicates) or canonicity and termination (incomplete provability calculi), while this presentation is naturally complete and canonical, but not yet proved terminating.

References

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.

[CMS08] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical sequent proofs via multi-focusing. In *IFIP TCS*, pages 383–396, 2008.

[Dyc13] Roy Dyckhoff. Intuitionistic decision procedures since gentzen. In *Advances in Proof Theory*, 2013.

[Lin07] Sam Lindley. Extensional rewriting with sums. In *TLCA*, pages 255–271, 2007.

[WY04] J. B. Wells and Boris Yakobowski. Graph-based proof counting and enumeration with applications for program fragment synthesis. In *LOPSTR*, pages 262–277, 2004.