Under consideration for publication in J. Functional Programming

Which simple types have a unique inhabitant?

Gabriel Scherer
Northeastern University

Didier Rémy
INRIA Paris

Abstract

Some programming language features (coercions, type-classes, implicits) rely on inferring a part of the code that is determined by its usage context. In order to better understand the theoretical underpinnings of this mechanism, we ask: when is it the case that there is a *unique* program that could have been guessed, or in other words that all possible guesses result in equivalent program fragments? Which types have a unique inhabitant?

To approach the question of unicity, we build on work in proof theory on more canonical representation of proofs. Using the proofs-as-programs correspondence, we can adapt the logical technique of focusing to obtain more canonical program representations.

In the setting of simply-typed lambda-calculus with sums, equipped with the strong $\beta\eta$-equivalence, we show that uniqueness is decidable. We present a saturating focused logic that introduces irreducible cuts on positive types "as soon as possible". Goal-directed proof search in this logic gives an effective algorithm that returns either zero, one or two distinct inhabitants for any given type.

This document is a largely extended version of the conference article Scherer and Rémy [2015]. The major changes to the presentation are the following:

- *Focusing is introduced and detailed in Section 2 (Introduction to focusing), in a way that should be self-contained and accessible to a programming-language audience.*
- *The conference presentation of focusing and related type systems used the usual types of the λ-calculus, distinguishing positive, negative and atomic types after the fact. The current presentation introduces, in Section 2.4.1 (Explicit shifts), an explicitly polarized syntax with separate grammatical categories, explicit shifts between polarities, and polarized (positive and negative) atoms. This presentation is more in line with recent expositions of focusing, and results in a more general, uniform system that is easier to generalize.*
- *Section 5 (Saturation logic for canonicity) introduces the main contribution of this work, namely the saturating focused system, in a much more didactic and detailed way than could be achieved in the conference version. Minor issues in the formalization have been corrected – we explicitly mention those.*

1 Introduction

In this article, we answer an instance of the following question: "Which types have a unique inhabitant"? In other words, for which type is there exactly one program of this type? Which logical statements have exactly one proof term?

To formally consider this question, we need to choose one specific type system, and one specific notion of equality of programs – which determines uniqueness. In this article, we work with the simply-typed λ-calculus with atoms, functions, products and sums as our type system, and we consider programs modulo $\beta\eta$-equivalence. We show that unique inhabitation is decidable in this setting; we provide and prove correct an algorithm to answer it, and suggest several applications for it. This is only a first step: simply-typed calculus *with sums* is, in some sense, the simplest system in which the question is delicate enough to be interesting. We hope that our approach can be extended to richer type systems – with polymorphism, dependent types, and substructural logics.

1.1 Why unique?

We see three different sources of justification for studying uniqueness of inhabitation: practical use of code inference, programming language design, and understanding of type theory.

In practice, if the context of a not-yet-written code fragment determines a type that is uniquely inhabited, then the programming system can automatically fill the code. This is a strongly principal form of code inference: it cannot guess wrong. Some forms of code completion and synthesis have been proposed [Perelman, Gulwani, Ball, and Grossman, 2012, Gvero, Kuncak, Kuraj, and Piskac, 2013], to be suggested interactively and approved by the programmer. Here, the strong restriction of uniqueness would make it suitable for a code elaboration pass at compile-time: it is of different nature. Of course, a strong restriction also means that it will be applicable less often. Yet we think it becomes a useful tool when combined with strongly typed, strongly specified programming disciplines and language designs – we have found in preliminary work [Scherer, 2013] potential use cases in dependently typed programming. The simply-typed lambda-calculus is very restricted compared to dependent types, or even the type systems of ML, System F, etc. used in practice in functional programming languages; but we have already found a few examples of applications (Section 8 (Evaluation)). This shows promises for future work on more expressive type systems.

For programming language design, we hope that a better understanding of the question of unicity will let us better understand, compare and extend other code inference mechanisms, keeping the question of coherence, or non-ambiguity, central to the system. Type classes or implicits have traditionally been presented [Wadler and Blott, 1989, Stuckey and Sulzmann, 2002, Oliveira, Schrijvers, Choi, Lee, Yi, and Wadler, 2014] as a mechanism for elaboration, solving a constraint or proof search problem, with coherence or non-ambiguity results proved as a second step as a property of the proposed elaboration procedure. Reformulating coherence as a unique inhabitation property, it is not anymore an operational property of the specific search/elaboration procedure used, but a semantic property of the typing environment and instance type in which search is performed. Non-

ambiguity is achieved not by fixing the search strategy, but by building the right typing environment from declared instances and potential conflict resolution policies, with a general, mechanism-agnostic procedure validating that the resulting type judgments are uniquely inhabited.

In terms of type theory, unique inhabitation is an occasion to take inspiration from the vast literature on proof inhabitation and proof search, keeping relevance in mind: all proofs of the same statement may be equally valid, but programs at a given type are distinct in important and interesting ways. We use *focusing* [Andreoli, 1992], a proof search discipline that is more canonical (enumerates less duplicates of each proof term) than simply goal-directed proof search, and its recent extension into (maximal) *multi-focusing* [Chaudhuri, Miller, and Saurin, 2008a].

1.2 Example use cases

Most types that occur in a program are, of course, not uniquely inhabited. Writing a term at a type that happens to be uniquely inhabited is a rather dull part of the programming activity, as there are no meaningful choices. While we do not hope types with inhabitants would occur all instances of boring programming tasks, we have identified two areas where they may appear:

- inferring the code of highly parametric (strongly specified) auxiliary functions
- inferring fragments of glue code in the middle of a more complex (and not uniquely determined) term

For example, if you write down the signature of `flip` $\forall\alpha\beta\gamma.(\alpha \to \beta \to \gamma) \to (\beta \to \alpha \to \gamma)$ to document your standard library, you should not have to write the code itself. The types involved can be presented equivalently as simple types, replacing prenex polymorphic variables by uninterpreted atomic types (X, Y, Z...). Our algorithm confirms that $(X \to Y \to Z) \to (Y \to X \to Z)$ is uniquely inhabited and returns the expected program – same for `curry` and `uncurry`, `const`, etc.

In the middle of a term, you may have forgotten whether the function `issue` accepts a `journal` as first argument and a `volume` as second argument, or the other way around. Suppose a language construct `?!` that infers a unique inhabitant at its expected type (and fails if there are several choices), understanding abstract types (such as `journal`) as uninterpreted atoms. You can then write `(?! journal jfp my_volume)`, and let the programming system infer the unique inhabitant of, depending on the actual argument order, either (`journal` → `volume` → `issue`) → (`journal` → `volume` → `issue`) or (`journal` → `volume` → `issue`) → (`volume` → `journal` → `issue`) – note that it would also work for `journal` × `volume` → `issue`, etc.

1.3 Aside: Parametricity?

Can we deduce unique inhabitation from the free theorem of a sufficiently parametric type? We worked out some typical examples, and our conclusion is that this is not the right approach. Although it was possible to derive uniqueness from a type's parametric interpretation, proving this implication (from the free theorem to uniqueness) requires

arbitrary reasoning steps, that is, a form of proof search. If we have to implement proof search mechanically, we may as well work with convenient syntactic objects, namely typing judgments and their derivations.

For example, the unary free theorem for the type of composition $\forall\alpha\beta\gamma.(\alpha \to \beta) \to (\beta \to \gamma) \to (\alpha \to \gamma)$ tells us that for any sets of terms $S_\alpha, S_\beta, S_\gamma$, if f and g are such that, for any $a \in S_\alpha$ we have $f\ a \in S_\beta$, and for any $b \in S_\beta$ we have $g\ b \in S_\gamma$, and if t is of the type of composition, then for any $a \in S_\alpha$ we have $t\ f\ g\ a \in S_\gamma$. The reasoning to prove unicity is as follows. Suppose we are given functions (terms) f and g. For any term a, first define $S_\alpha \stackrel{\text{def}}{=} \{a\}$. Because we wish f to map elements of S_α to S_β, define $S_\beta \stackrel{\text{def}}{=} \{f\ a\}$. Then, because we wish g to map elements of S_β to S_γ, define $S_\gamma \stackrel{\text{def}}{=} \{g\ (f\ a)\}$. We have that $t\ f\ g\ a$ is in S_γ, thus $t\ f\ g$ is uniquely determined as $\lambda a.g\ (f\ a)$.

This reasoning exactly corresponds to a (forward) proof search for the type $\alpha \to \gamma$ in the environment $\alpha, \beta, \gamma, f : \alpha \to \beta, g : \beta \to \gamma$. We know that we can always start with a λ-abstraction (formally, arrow-introduction is an invertible rule), so introduce $x : \alpha$ in the context and look for a term of type γ. This type has no head constructor, so no introduction rules are available; we shall look for an elimination (function application or pair projection). The only elimination we can perform from our context is the application $f\ x$, which gives a β. From this, the only elimination we can perform is the application $g\ (f\ x)$, which gives a γ. This has the expected goal type: our full term is $lamxg\ (f\ x)$. It is uniquely determined, as we never had a choice during term construction.

1.4 Formal definition of equivalence

We recall the syntax of the simply-typed lambda-calculus types (Figure 1), terms (Figure 2) and neutral terms. The standard typing judgment $\Gamma \vdash t : A$ is recalled in Figure 3, where Γ is a general context mapping term variables to types. The equivalence relation we consider, namely $\beta\eta$-equivalence, is defined as the least congruence satisfying the equations of Figure 4. Writing $t : A$ in an equivalence rule means that the rule only applies when the subterm t has type A – we only accept equivalences that preserve well-typedness.

Fig. 1. Types of the simply-typed calculus

$A, B, C, D ::=$		types
	$\mid X, Y, Z$	atoms
	$\mid A \to B$	function types
	$\mid A \times B$	product types
	$\mid A + B$	sum types

The equivalence rules of Figure 4 make it apparent that the η-equivalence rule for sums is more difficult to handle than the other η-rule, as it quantifies on any term context $C[\Box]$. In fact, it is only at the end of the 20th century [Ghani, 1995, Altenkirch, Dybjer, Hofmann, and Scott, 2001, Balat, Di Cosmo, and Fiore, 2004, Lindley, 2007] that decision procedures for equivalence in the lambda-calculus with sums were first proposed.

Can we reduce the question of unicity to deciding equivalence? One would think of enumerating terms at the given type, and using an equivalence test as a post-processing filter to remove duplicates: as soon as one has found two distinct terms, the type can be

Fig. 2. Terms of the lambda-calculus with sums

$$
\begin{array}{lll}
t,u,r ::= & & \text{terms} \\
\quad \mid x,y,z & & \text{variables} \\
\quad \mid \lambda x.t & & \lambda\text{-abstraction} \\
\quad \mid t\ u & & \text{application} \\
\quad \mid (t,u) & & \text{pair} \\
\quad \mid \pi_i\ t & & \text{projection } (i \in \{1,2\}) \\
\quad \mid \sigma_i\ t & & \text{sum injection } (i \in \{1,2\}) \\
\quad \mid \texttt{match}\ t\ \texttt{with} & \sigma_1\ x_1 \to u_1 \\ & \sigma_2\ x_2 \to u_2 & \text{sum elimination (case split)} \\
\\
n,m \quad := x,y,z \mid \pi_i\ n \mid n\ t & & \text{neutral terms}
\end{array}
$$

Fig. 3. Typing rules for the simply-typed lambda-calculus

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B}$$

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \pi_i\ t : A_i} \qquad \frac{\Gamma \vdash t : A_i}{\Gamma \vdash \sigma_i\ t : A_1 + A_2}$$

$$\frac{\Gamma \vdash t : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash u_1 : C \qquad \Gamma, x_2 : A_2 \vdash u_2 : C}{\Gamma \vdash \texttt{match}\ t\ \texttt{with} \mid \begin{array}{l} \sigma_1\ x_1 \to u_1 \\ \sigma_2\ x_2 \to u_2 \end{array} : C}$$

Fig. 4. $\beta\eta$-equivalence for the simply-typed lambda-calculus

$$(\lambda x.t)\ u \triangleright_\beta u[t/x] \qquad (t : A \to B) \approx_\eta \lambda x.t\ x \qquad \pi_i\ (t_1, t_2) \triangleright_\beta t_i \qquad (t : A \times B) \approx_\eta (\pi_1\ t, \pi_2\ t)$$

$$\texttt{match}\ \sigma_i\ t\ \texttt{with} \mid \begin{array}{l} \sigma_1\ x_1 \to u_1 \\ \sigma_2\ x_2 \to u_2 \end{array} \triangleright_\beta u_i[t/x_i]$$

$$\forall C[\Box], \quad C[t : A + B] \approx_\eta \texttt{match}\ t\ \texttt{with} \mid \begin{array}{l} \sigma_1\ x_1 \to C[\sigma_1\ x_1] \\ \sigma_2\ x_2 \to C[\sigma_2\ x_2] \end{array}$$

declared non-uniquely inhabited. Unfortunately, this method does not give a terminating decision procedure, as naive proof search may enumerate infinitely many equivalent proofs, taking infinite time to post-process. We need to integrate canonicity in the structure of proof search itself.

1.5 Terminology

We describe our search procedure as proof search in restricted systems of inference rule (a step of search is the bottom-up application of an inference rule to refine a partial proof).

We distinguish and discuss the following properties:

- *termination*: A search procedure is terminating if, for any input search problem, it returns a result (or fails) after a finite number of steps of search.
- *provability completeness* : A search procedure is complete for provability if, for any type that is inhabited in the unrestricted type system, it finds at least one proof term.

- *unicity completeness*: A search procedure is complete for unicity if it is complete for provability and, if there exists two proofs distinct as programs in the unrestricted calculus, then the search finds at least two proofs distinct as programs.
- *computational completeness* : A search procedure is computationally complete if, for any proof term t in the unrestricted calculus, there exists a proof in the restricted search space that is equivalent to t as a program. This implies both previous notions of completeness.
- *canonicity* : A search procedure is canonical if it has no duplicates: any two enumerated proofs are distinct as programs. Such procedures require no filtering of results after the fact. We will say that a system is *more canonical* than another if it enumerates less redundant terms, but this does not imply canonicity.

There is a tension between computational completeness and termination of the corresponding search algorithm: when termination is obtained by cutting the search space, it may remove some computational behaviors. Canonicity is not a strong requirement: we could have a terminating, unicity-complete procedure and filter duplicates after the fact, but have found no such middle-ground. This article presents a logic (Section 5) that is both *computationally complete* and *canonical* (Section 6), and can be restricted (Section 4) to obtain a *terminating* yet *unicity-complete* algorithm (Section 7).

1.6 Focusing for a less redundant proof search

Focusing [Andreoli, 1992] is a generic search discipline that can be used to decrease redundancy in the search space of proofs; it relies on the general idea that some proof steps are *invertible* (the premises are provable exactly when the conclusion is, hence performing this step during proof search can never lead you to a dead-end) while others are not. By imposing an order on the application of invertible and non-invertible proof steps, focusing restricts the number of valid proofs, but it remains complete for provability and, in fact, computationally complete (§1.5).

More precisely, a focused proof system alternates between two phases of proof search. During the *invertible phase*, rules recognized as invertible are applied as long as possible – this stops when no invertible rule can be applied anymore. During the *non-invertible phase*, non-invertible rules are applied in the following way: a formula (in the context or the goal) is chosen as the *focus*, and non-invertible rules are applied as long as possible.

For example, consider the judgment $x : X + Y \vdash X + Y$. Introducing the sum on the right by starting with a $\sigma_1\ ?$ or $\sigma_2\ ?$ would be a non-invertible proof step: we are permanently committing to a choice – which would here lead to a dead-end. On the contrary, doing a case-split on the variable x is an invertible step: it leaves all our options open. For non-focused proof search, simply using the variable $x : X + Y$ as an axiom would be a valid proof term. It is not a valid focused proof, however, as the case-split on x is a possible invertible step, and invertible rules must be performed as long as they are possible. This gives a partial proof term $\texttt{match}\ x\ \texttt{with} \mid \begin{array}{l}\sigma_1\ y \to ? \\ \sigma_2\ z \to ?\end{array}$, with two subgoals $y : X \vdash X + Y$ and $z : X \vdash X + Y$; for each of them, no invertible rule can be applied anymore, so one can only *focus* on the goal and do an injection. While the non-focused calculus had two

syntactically distinct but equivalent proofs, x and $\texttt{match}\ x\ \texttt{with} \mid \sigma_1\ y \to \sigma_1\ y \mid \sigma_2\ z \to \sigma_2\ z$, only the latter is a valid focused proof: redundancy of proof search is reduced.

We present a focused intuitionistic logic in more details in Section 2 (Introduction to focusing), and a term system, the focused λ-calculus, in Section 3 (Focused λ-calculus).

We say that a type is *positive* if building a value requires a non-invertible choice, and *negative* if using its value requires a non-invertible choice. Sums are positive, and functions and products are negative.

1.7 Limitations of focusing

In the absence of sums, focused proof terms correspond exactly to β-short η-long normal forms. In particular, focused search is *canonical* (§1.5). However focused proofs are not canonical anymore when sums are introduced. They correspond to η-long form for the strictly weaker eta-rule defined without context quantification

$$t : A + B =_{\text{weak-}\eta} \texttt{match}\ t\ \texttt{with} \mid \sigma_1\ x_1 \to \sigma_1\ x_1 \mid \sigma_2\ x_2 \to \sigma_2\ x_2$$

This can be seen for example on the judgment $z : Z_1, x : Z_1 \to X + Y \vdash X + Y$, a variant on the previous example where the sum in the context is "thunked" under a negative datatype. The expected proof is

$$\texttt{match}\ x\ z\ \texttt{with} \mid \sigma_1\ y_1 \to \sigma_1\ y_1 \mid \sigma_2\ y_2 \to \sigma_2\ y_2$$

but the focused discipline will accept infinitely many equivalent proof terms, such as

$$\texttt{match}\ x\ z\ \texttt{with} \mid \sigma_1\ y_1 \to \sigma_1\ y_1 \mid \sigma_2\ y_2 \to \texttt{match}\ x\ z\ \texttt{with} \mid \sigma_1\ y_1' \to \sigma_1\ y_1' \mid \sigma_2\ \_ \to \sigma_2\ y_2$$

The result of the application $x\ z$ can be matched upon again and again without breaking the focusing discipline.

1.8 Our idea: saturating proof search

Our idea is that instead of only deconstructing the sums that appear immediately as the top type constructor of a type in context, we shall deconstruct all the sums that can be reached from the context by applying eliminations (function application and pair projection). Each time we introduce a new hypothesis in the context, we *saturate* it by computing all neutrals of sum type that can be built using this new hypothesis. At the end of each saturation phase, all the sums that could be deduced from the context have been deconstructed, and we can move forward applying non-invertible rules on the goal. Eliminating negatives until we get a positive and matching in the result corresponds to a cut (which is not reducible, as the scrutinee is a neutral term), hence our technique can be summarized as "*Cut the positives as soon as you can*".

The idea was inspired by Sam Lindley's equivalence procedure for the lambda-calculus with sums, whose rewriting relation can be understood as moving case-splits *down* in the

derivation tree, until they get blocked by the introduction of one of the variable appearing in their scrutinee (so moving down again would break scoping) – this also corresponds to "restriction (A)" in Balat, Di Cosmo, and Fiore [2004]. In our saturating proof search in Section 5 (Saturation logic for canonicity), after introducing a new formal parameter in the context, we look for all possible new scrutinees using this parameter, and case-split on them. Of course, this is rather inefficient as most proofs will in fact not make use of the result of those case-splits, but this allows to give a common structure to all possible proofs of this judgment.

In our example $z : Z, x : Z \to X + Y \vdash X + Y$, the saturation discipline requires to cut on $x\ z$. But after this sum has been eliminated, the newly introduced variables $y_1 : X$ or $y_2 : Y$ do not allow to deduce new positives – we would need a new Z for this. Thus, saturation stops and focused search restarts, to find a unique normal form $\texttt{match}\ x\ z\ \texttt{with} \mid \begin{array}{l} \sigma_1\ y_1 \to \sigma_1\ y_1 \\ \sigma_2\ y_2 \to \sigma_2\ y_2 \end{array}$. In Section 6 (Canonicity of saturated proofs) we show that saturating proof search is *computationally complete* and *canonical* (§1.5).

1.9 Termination

The saturation process described above does not necessarily terminate. For example, consider the type of Church numerals specialized to a positive $X + Y$, that is, $X + Y \to (X + Y \to X + Y) \to X + Y$. Each time we cut on a new sum $X + Y$, we get new arguments to apply to the function $(X + Y \to X + Y)$, giving yet another sum to cut on.

In the literature on proof search for propositional logic, the usual termination argument is based on the subformula property: in a closed, fully cut-eliminated proof, the formulas that appear in subderivations of subderivations are always subformulas of the formulas of the main judgment. In particular, in a logic where judgments are of the form $S \vdash A$ where S is a finite *set* of formulas, the number of distinct judgments appearing in subderivations is finite (there is a finite number of subformulas of the main judgment, and thus finitely many possible finite sets as contexts). Finally, in a goal-directed proof search process, we can kill any recursive subgoal whose judgment already appears in the path from the root of the proof to the subgoal. There is no point in trying to complete a partial proof P_{leafward} of $S \vdash A$ as a strict subproof of a partial proof P_{rootward} of the same $S \vdash A$ (itself a subproof of the main judgment): if there is a closed subproof for P_{leafward}, we can use that subproof directly for P_{rootward}, obviating the need for proving P_{leafward} in the first place. Because the space of judgments is finite, a search process forbidding such recurring judgments always terminates.

We cannot directly apply this reasoning, for two reasons.

- Our contexts are mapping from term variables to formulas or, seen abstractly, *multi-sets* of formulas; even if the space of possible formulas is finite for the same reason as above, the space of multisets over them is still infinite.
- Erasing such multiset to sets, and cutting according to the non-recurrence criteria above, breaks *unicity completeness* (§1.5). Consider the construction of Church numerals by a judgment of the form $x : X, y : X \to X \vdash X$. One proof is just x, and all other proofs require providing an argument of type X to the function y, which

corresponds to a subgoal that is equal to our goal; they would be forbidden by the no-recurrence discipline.

We must adapt these techniques to preserve not only *provability completeness*, but also *unicity completeness* (§1.5). Our solution is to use *bounded multisets* to represent contexts and collect recursive subgoals. We store at most M variables for each given formula, for a suitably chosen M such that if there are two different programs for a given judgment $\Gamma \vdash A$, then there are also two different programs for $\lfloor \Gamma \rfloor_M \vdash A$, where $\lfloor \Gamma \rfloor_M$ is the bounded erasure keeping at most M variables at each formula.

While it seems reasonable that such a M exists, it is not intuitively clear what its value is, or whether it is a constant or depends on the judgment to prove. Could it be that a given goal A is provable in two different ways with four copies of X in the context, but uniquely inhabited if we only have three X?

In Section 4 (Counting terms and proofs) we prove that $M \stackrel{\mathsf{def}}{=} 2$ suffices. In fact, we prove a stronger result: for any $n \in \mathbb{N}$, keeping at most n copies of each formula in context suffices to find at least n distinct proofs of any goal, if they exist.

For recursive subgoals as well, we only need to remember at most 2 copies of each subgoal: if some P_{leafward} appears as the subgoal of P_{rootward} and has the same judgment, we look for a closed proof of P_{leafward}. Because it would also have been a valid proof for P_{rootward}, we have found two proofs for P_{rootward}: the one using P_{leafward} and its closed proof, and the closed proof directly. P_{leafward} itself needs not allow new recursive subgoal at the same judgment, so we can kill any subgoal that has at least two ancestors with the same judgment while preserving completeness for unicity (§1.5).

1.10 Contributions

While the logical notion of focusing is usually presented using the sequent calculus, we introduce a term syntax for a focused propositional intuitionistic logic in natural deduction style, that corresponds to (the normal-forms of a) *focused* λ-calculus. We hope that this focused λ-calculus could be useful to transfer other results from logic into the programming language community.

We show that the unique inhabitation problem for simply-typed lambda-calculus for sums is decidable, and propose an effective algorithm for it. Given a context and a type, it answers that there are zero, one, or "at least two" inhabitants, and correspondingly provides zero, one, or two distinct terms at this typing. Our algorithm relies on a novel *saturating* focused logic for intuitionistic natural deduction, with strong relations to the idea of *maximal multi-focusing* in the proof search literature [Chaudhuri, Miller, and Saurin, 2008a], that is both *computationally complete* (§1.5) and *canonical* with respect to $\beta\eta$-equivalence.

We provide an approximation result for program multiplicity of simply-typed derivations with bounded contexts. We use it to show that our *terminating* algorithm is *complete for unicity* (§1.5), but it is a general result (on the common, non-focused intuitionistic logic) that is of independent interest.

Finally, we present preliminary studies of applications for code inference. While extension to more realistic type systems is left for future work, simply-typed lambda-calculus

with atomic types already allows to encode some prenex-polymorphic types typically found in libraries of strongly-typed functional programs.

2 Introduction to focusing

2.1 Natural deduction and sequent calculus

Type systems and proof systems can both be defined by a judgment (or a family of judgments) equipped by a family of *inference rules* that define valid derivations. If you take the type system of the simply-typed calculus $\Gamma \vdash t : A$ in Figure 3 (Typing rules for the simply-typed lambda-calculus) and erase all mentions of terms and term variables, you get the *natural deduction* presentation of propositional intuitionistic logic in Figure 5 (Natural deduction proof system for propositional intuitionistic logic) – this is the usual Curry-Howard correspondence between type systems and logics.

Fig. 5. Natural deduction proof system for propositional intuitionistic logic

$$\textsc{nd-axiom}\ \frac{}{\Gamma, A \vdash A}$$

$$\textsc{nd-impl-elim}\ \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \qquad \textsc{nd-impl-intro}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

$$\textsc{nd-conj-elim}\ \frac{\Gamma \vdash A_1 \times A_2}{\Gamma \vdash A_i} \qquad \textsc{nd-conj-intro}\ \frac{\Gamma \vdash A_1 \qquad \Gamma \vdash A_2}{\Gamma \vdash A_1 \times A_2}$$

$$\textsc{nd-disj-elim}\ \frac{\Gamma \vdash A_1 + A_2 \qquad \Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma \vdash C} \qquad \textsc{nd-disj-intro}\ \frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 + A_2}$$

Note that the contexts written Γ in this new judgment $\Gamma \vdash A$ are not mappings from variables to types, but merely sets of types. In particular, when we write $\Gamma, A \vdash B$ in the premise of the function rule (or the implication-introduction rule), the type A may already belong to the set Γ: the comma in (Γ, A) denotes a *non-disjoint* union.

The rules coming from the *constructor* of a given type ($\lambda x.t$ for functions, (t_1, t_2) for pairs, $\sigma_i\, t$ for sums) are called *introduction rules* – in the right column of the figure. The rules coming from the *destructor* of a given type ($t\ u$ for functions, $\pi_i\, t$ for pairs, and $\texttt{match}\ t\ \texttt{with} \mid \sigma_1\ x_1 \to y_1 \mid \sigma_2\ x_2 \to y_2$ for sums) are called *elimination rules* – in the left column of the figure.

One may notice that the elimination rule for sums stands out, with its C formula that does not appear in other rules. Natural deduction nicely corresponds to human reasoning, but its lack of symmetry in presence of sums makes meta-theory difficult. For example, there is a well-known combinatorial argument for normalization of β-reduction in the system with functions and products (using as termination measure the set of introduction-elimination pairs ranked by complexity of the type/formula at which the reduction happens), but this

argument falls down in presence of sums and requires elaborate extensions (introducing commuting conversions) to be restored.

The *sequent calculus* presentation of propositional intuitionistic logic, given in Figure 6 (Sequent calculus proof system for propositional intuitionistic logic), is another proof system that proves the same types/formulas, but is more symmetric, making meta-theory easier, and more adapted to proof search.

Fig. 6. Sequent calculus proof system for propositional intuitionistic logic

$$\text{SEQ-AXIOM}\ \frac{}{\Gamma, A \vdash A} \qquad \text{SEQ-CUT}\ \frac{\Gamma \vdash A \qquad \Gamma, A \vdash C}{\Gamma \vdash C}$$

$$\text{SEQ-IMPL-LEFT}\ \frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C} \qquad \text{SEQ-IMPL-RIGHT}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

$$\text{SEQ-CONJ-LEFT}\ \frac{\Gamma, A_i \vdash C}{\Gamma, A_1 \times A_2 \vdash C} \qquad \text{SEQ-CONJ-RIGHT}\ \frac{\Gamma \vdash A_1 \qquad \Gamma \vdash A_2}{\Gamma \vdash A_1 \times A_2}$$

$$\text{SEQ-DISJ-LEFT}\ \frac{\Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma, A_1 + A_2 \vdash C} \qquad \text{SEQ-DISJ-RIGHT}\ \frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 + A_2}$$

In natural deduction, elimination rules tell you how to *use* a complete proof of a connective to build new proofs (rootward). The sequent calculus uses *left-introduction* rules instead, that tell you how to *consume* a hypothesis present in context to prove your goal (leafward). The existing introduction rules are unchanged in sequent calculus (it is called "right-introduction").

(We use *leafward* to mean "towards the leaves of the derivation", and *rootward* to mean "towards the root of the derivation".)

The *cut rule*

$$\text{SEQ-CUT}\ \frac{\Gamma \vdash A \qquad \Gamma, A \vdash C}{\Gamma \vdash C}$$

is necessary to obtain a simple translation of any natural deduction derivation into a sequent calculus derivation – elimination rules of natural deduction embed a form of cut rule on their type connective. For example, the following derivation shows that the elimination rule for functions is derivable: $\Gamma \vdash B$ can be deduced from $\Gamma \vdash A \to B$ and $\Gamma \vdash A$:

$$\frac{\Gamma \vdash A \to B \qquad \dfrac{\Gamma \vdash A \qquad \dfrac{}{\Gamma, B \vdash B}}{\Gamma, A \to B \vdash B}\ \text{SEQ-CUT}}{\Gamma \vdash B}\ \text{SEQ-CUT}$$

All uses of the cut rule can be eliminated – just as β-redexes in the λ-calculus, but the elimination does not correspond to β-reduction alone as it performs commuting conver-

sions. In this article we are interested with the observable identity of proofs and programs, not the dynamics of their reduction, so we shall consider *cut-free* derivations, that do not use this cut rule.

2.1.1 Non-canonicity of cut-free sequent proofs

While sequent calculus and natural deduction prove the same judgments, normal natural deduction derivations are more canonical than cut-free sequent proofs. For example, while there is a single normal natural deduction proof of the judgment $A \to B, B \to C \vdash A \to C$ (for readability, we gray out the parts of the judgments that are not involved in the current inference rule)

$$\dfrac{\dfrac{}{A \to B, B \to C, A \vdash B \to C} \qquad \dfrac{\dfrac{}{A \to B, B \to C, A \vdash A \to B} \qquad \dfrac{}{B \to C, A \vdash A}}{A \to B, B \to C, A \vdash B}}{\dfrac{A \to B, B \to C, A \vdash C}{A \to B, B \to C \vdash A \to C}}$$

there are two distinct cut-free sequent rules of the same judgment:

$$\dfrac{\dfrac{}{A \to B, B \to C, A \vdash A} \qquad \dfrac{\dfrac{}{A \to B, B \to C, A, B \vdash B} \qquad \dfrac{}{A \to B, B \to C, A, B, C \vdash C}}{A \to B, B \to C, A, B \vdash C}}{\dfrac{A \to B, B \to C, A \vdash C}{A \to B, B \to C \vdash A \to C}}$$

$$\dfrac{\dfrac{\dfrac{}{A \to B, B \to C, A \vdash A} \qquad \dfrac{}{A \to B, B \to C, A, B \vdash B}}{A \to B, B \to C, A \vdash B} \qquad \dfrac{}{A \to B, B \to C, A, B, C \vdash C}}{\dfrac{A \to B, B \to C, A \vdash C}{A \to B, B \to C \vdash A \to C}}$$

Focusing introduces restrictions on the proofs that restore a one-to-one correspondence between focused natural deduction proofs and focused sequent proofs.

2.2 Focused proofs as a subset of non-focused proofs

We introduce the *focusing discipline* as a set of conditions that make a cut-free sequent proof a valid *focused proof*. In Section 2.3 (Structural presentations of focusing), we will present different judgment structures that *structurally* enforce the focusing discipline.

2.2.1 Invertible rules

Definition 2.1 Invertible rule**.**

$$\dfrac{\mathscr{J}_1 \qquad \mathscr{J}_2 \qquad \cdots \qquad \mathscr{J}_n}{\mathscr{J}}$$

is *invertible* when the following property holds: if $\mathscr{J}$ is provable, then all of the $\mathscr{J}_1, \ldots \mathscr{J}_n$ are provable as well.

Example 2.1 (Invertible rule)**.**

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

is invertible, as witnessed by the following "inverse derivation":

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

. ◊

Example 2.2 (Non-invertible rule)**.**

$$\frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 + A_2}\ i \in \{1,2\}$$

is not invertible. For example, the judgment $A_1 + A_2 \vdash A_1 + A_2$ is provable, but none of the $A_1 + A_2 \vdash A_i$ are. ◊

Invertibility is an interesting notion for goal-directed proof search: by definition, the invertible rules are those can always be used without risk of "getting stuck". This suggests that we may study a sub-system of the proofs that always apply invertible rules whenever possible, and only try non-invertible rules once no invertible rule can be applied – focusing generalizes this idea.

On the contrary, applying non-invertible rules corresponds to making a choice: if the rule is wrongly applied, the proof attempt may fail whereas another rule would have led to a solution. In a sense, non-invertible rules are the "important" rules in a proof – and in fact, we could reconstruct a proof from only the tree of its non-invertible rules.

2.2.2 Focus

Definition 2.2 focus**.**
We define the *focus* of a non-invertible introduction rule to be the formula introduced by the rule – it is also often called the *principal formula* of the rule. To help readability, we often underline the foci in a proof:

$$\frac{\dfrac{A_j \vdash B_i}{\underline{A_1 \times A_2} \vdash B_i}}{A_1 \times A_2 \vdash \underline{B_1 + B_2}}$$

2.2.3 Positive and negative connectives

Given a proof system in sequent calculus style, a connective whose right-introduction rule is non-invertible ("important") is called *positive*, and a connective whose left-introduction rule is non-invertible is called *negative*.

In the sequent calculus proof system given in Figure 6, the implication and the conjunction are negative connectives, and the disjunction is a positive connective:

$$\frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, \underline{A \to B} \vdash C} \qquad \frac{\Gamma, A_i \vdash C}{\Gamma, \underline{A_1 \times A_2} \vdash C} \qquad \frac{\Gamma \vdash A_i}{\Gamma \vdash \underline{A_1 + A_2}}$$

It is immediate that the conjunction and disjunction rules are non-invertible: using them may remove some information from the judgment to prove. For the left-introduction of implication, non-invertibility comes from the fact that $\Gamma \vdash A$ may not be provable, when C would have been provable by using another rule.

2.2.4 *Invertibility and side-conditions*

One difficulty with the definition of invertibility given above is that it is sensitive to the way rules are presented. Consider these two different definitions of the axiom rule:

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{A \in \Gamma}{\Gamma \vdash \underline{A}}$$

The rule on the left is premise-free, so in particular it should be invertible by the definition above: its conclusion is always provable, and its premises are always provable (there are none). The rule on the right is not invertible: it may be the case that $A \notin \Gamma$, yet that the conclusion be provable by a different rule.

The problem with the rule on the left is the use of non-linear pattern-matching: we use A twice in the conclusion, and this hides an implicit side-condition. Invertibility makes perfect sense for the left- and right-introduction rules of logical connectives, which do not have such non-linear patterns: each meta-variable is used exactly once.

The same subtlety occurs in multi-succedent linear logic, where the definition of the positive right unit 1 requires the context and succedent to be empty.

$$\frac{}{\emptyset \vdash 1, \emptyset} \qquad \frac{\Gamma = \emptyset \qquad \Delta = \emptyset}{\Gamma \vdash \underline{1}, \Delta}$$

The solution to this subtlety is to always consider rules with their side-conditions (equality and emptiness checks) made explicit. The axiom rule for $\Gamma, A \vdash A$ can be reformulated in two different presentations with a simple conclusion pattern:

$$\frac{A = B}{\Gamma, \underline{A} \vdash B} \qquad \frac{A \in \Gamma}{\Gamma \vdash \underline{A}}$$

Both rules are non-invertible and equi-provable, but they do not have the same focus. For now we use the formulation on the *right*: we consider that the focus of the axiom rule is the succedent occurrence of the formula. We shall revisit this choice using finer-grained rules that make both options useful and interesting in Section 2.2.7 (Polarized atoms).

2.2.5 *The focusing phase discipline*

The focusing discipline relies on exposing a structure of consecutive *phases* of a proof, and verifying that they verify certain conditions.

Definition 2.3 phase**.**
Phases are sets of consecutive rules of the same polarity (invertible or non-invertible), defined as the maximal sets satisfying the following properties:

- two consecutive invertible rules are part of the same invertible phase
- two consecutive non-invertible rules are part of the same non-invertible phase *if* the focus of the leafward phase is a subformula occurrence of the focus of the rootward phase

When two consecutive non-invertible rules are in the same phase, we say that they have the same focus (the focused subformula of one is a subformula occurrence of the other).

For example, we have labeled each rule of the proof below with a phase indication, using different indices for distinct phases.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{Z \vdash \underline{Z}}\,\mathrm{n}_1}{Z \vdash \underline{X+Z}}\,\mathrm{n}_1 \qquad \dfrac{\dfrac{\dfrac{}{Z \vdash \underline{X}}\,\mathrm{n}_3}{Z \vdash \underline{X+Z}}\,\mathrm{n}_3}{\underline{Y \times Z} \vdash X+Z}\,\mathrm{n}_2}{X+(Y\times Z) \vdash X+Z}\,\mathrm{i}}{\vdash X+(Y\times Z) \to X+Z}\,\mathrm{i}$$

Remark 2.1. In the literature, invertible phases are called *negative* phases, and non-invertible phases *positive* phases; this comes from one-sided presentations of linear logic judgments $\vdash \Delta$ with only succedents and no hypotheses, in which the non-invertible phases always manipulate positive connectives and invertible phases always manipulate negative connectives.

The adjectives *synchronous* and *asynchronous* are also in common usage since Andreoli [1992]. (One idea would be that asynchronous rules "have more freedom", they can be applied freely, they are the invertible rules.) ∗

Definition 2.4 Focusing conditions**.**
To be a valid focused proof, a sequent proof must respect the following conditions. In the rest of this section, we give several examples to explain and motivate those restrictions.

1. Invertible phases must be *as long as possible*: if the premise of a rule in an invertible phase matches the conclusion of an invertible rule, then it must be the conclusion of a rule in this invertible phase.
2. Non-invertible phases must be *as long as possible*: if the premise of a rule in a non-invertible phase matches a non-invertible rule of the same focus, then it must be the conclusion of a non-invertible rule in this phase.

Example 2.3 (Long invertible phases)**.** Consider the two following, equivalent proofs of $\vdash X \times Y \to X \times X$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{X \vdash \underline{X}}\,\mathrm{n}_2 \quad \dfrac{}{X \vdash \underline{X}}\,\mathrm{n}_3}{X \vdash X \times X}\,\mathrm{i}}{\underline{X \times Y} \vdash X \times X}\,\mathrm{n}_1}{\vdash X \times Y \to X \times X}\,\mathrm{i} \qquad\qquad \dfrac{\dfrac{\dfrac{}{X \times Y \vdash \underline{X}}\,\mathrm{n}_2 \quad \dfrac{\dfrac{}{X \vdash \underline{X}}\,\mathrm{n}_3}{\underline{X \times Y} \vdash X}\,\mathrm{n}_1}{X \times Y \vdash X \times X}\,\mathrm{i}}{\vdash X \times Y \to X \times X}\,\mathrm{i}$$

The first proof breaks the focusing discipline: a (non-invertible) left-introduction of the pair $X \times Y$ happens at a place where an invertible rule could have been used – the right-introduction rule for the pair $X \times X$. The second proof is a valid focused proof. ◊

This restriction allows us to reason on the polarity of connectives at the beginning of a non-invertible phase. Indeed, the invertible rules are exactly the left-introduction of positive connectives and right-introduction of a negative connective. At the start of a non-invertible phase, no invertible rule is applicable; this means that the formulas in the hypotheses are all negative or atomic, and the formula in succedent is positive or atomic.

Example 2.4 (Long non-invertible phases)**.** Consider the two following proofs of $Z_1 \times (X + Y) \vdash Z_0 + (Y + X)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{X \vdash \underline{X}}\,\mathrm{n}_4}{X \vdash \underline{Y + X}}\,\mathrm{n}_4 \quad \dfrac{\dfrac{}{Y \vdash \underline{Y}}\,\mathrm{n}_3}{Y \vdash \underline{Y + X}}\,\mathrm{n}_3}{X + Y \vdash Y + X}\,\mathrm{i}}{\dfrac{\underline{Z_1 \times (X + Y)} \vdash Y + X}{Z_1 \times (X + Y) \vdash \underline{Z_0 + (Y + X)}}\,\mathrm{n}_1}\,\mathrm{n}_2 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{X \vdash X}\,\mathrm{n}_3}{X \vdash \underline{Y + X}}\,\mathrm{n}_3}{X \vdash \underline{Z_0 + (Y + X)}}\,\mathrm{n}_3 \quad \dfrac{\dfrac{\dfrac{}{Y \vdash Y}\,\mathrm{n}_2}{Y \vdash \underline{Y + X}}\,\mathrm{n}_2}{Y \vdash \underline{Z_0 + (Y + X)}}\,\mathrm{n}_2}{X + Y \vdash Z_0 + (Y + X)}\,\mathrm{i}}{\underline{Z_1 \times (X + Y)} \vdash Z_0 + (Y + X)}\,\mathrm{n}_1$$

The first proof starts with a non-invertible phase on the focused formula $Z_0 + (X + Y)$, but then stops to perform an invertible rule. But a non-invertible rule matches the introduced subformula $Y + X$, as it is a positive on the right of the sequent; the focusing discipline is not respected. To respect the focusing discipline, one would have to introduce either X or Y, but there is not enough information in the context to know which one is provable.

In the second proof, the corresponding non-invertible rule on the goal is applied later in the proof, after the formula $X + Y$ in the context has been decomposed. It is performed in the two branches of the case analysis, with either X or Y in context, and in each branch the focused phase is complete. This proof respects the focusing discipline. ◊

An important early result about the focusing discipline is that it is complete for provability: all provable judgments have a valid focused proof.

Theorem from previous works 1 (Liang and Miller [2007])**.** *The subsystem of cut-free propositional intuitionistic sequent proofs which respect the focusing discipline is complete for provability.*

This is a strong result. It is easy to see that imposing invertible rules to be applied as easy as possible is complete – this is essentially the definition of invertibility – but imposing that the non-invertible phases be as long as possible is a much stronger restriction, and it is not at all obvious that it is complete.

2.2.6 The atomic axiom rule

Among a given class of proofs (or programs) that are equivalent to each other, some will respect the focusing discipline above and some will not. Formally, a focused subsystem is more canonical than the original, non-focused system. This selectivity is an advantage of focusing: it brings us closer to the dream land of canonical proof systems.

A common source of non-canonicity in proofs is the axiom rule:

$$\dfrac{}{\Gamma, A \vdash A}$$

For example, there are two η-equivalent proofs of $\vdash (X \to Y) \to X \to Y$:

$$\dfrac{\dfrac{}{X \to Y \vdash \underline{X \to Y}}}{\vdash \lambda x.x : (X \to Y) \to X \to Y} \qquad \dfrac{\dfrac{\dfrac{}{X \to Y, X \vdash \underline{X}} \quad \dfrac{}{X \to Y, X, Y \vdash \underline{Y}}}{X \to Y, X \vdash Y}}{\dfrac{X \to Y \vdash X \to Y}{\vdash \lambda x.\lambda y.\,\mathtt{let}\ z = x\ y\ \mathtt{in}\ z : (X \to Y) \to X \to Y}}$$

However, notice that the left proof above is not a valid focused proof. Indeed, the axiom rule is non-invertible – see Section 2.2.4 (Invertibility and side-conditions). This non-invertible rule cannot be applied while invertible rules are still applicable, and in this proof $X \to Y$ can still be (invertibly) introduced on the right.

For the same reason, the axiom rule cannot be used when the formula A starts with a positive connective, as it is then its occurrence in the context that could be invertibly introduced.

In our logic, all connectives are either positive or negative. This means that the axiom rule can only be used for formulas that do not start with a head connective, that is with atoms. It is thus exactly equivalent, under the focusing discipline, to the following *atomic axiom rule*:

$$\dfrac{}{\Gamma, X \vdash X}$$

2.2.7 *Polarized atoms*

To understand that it is non-invertible, the atomic axiom rule above can be expressed using side-conditions in two different ways:

$$\dfrac{X = A}{\Gamma, \underline{X} \vdash A} \qquad \dfrac{X \in \Gamma}{\Gamma \vdash \underline{X}}$$

The rule on the left resembles a (non-invertible) left-introduction rule, and the rule on the right resembles a (non-invertible) right-introduction rule. We could informally say that the atom X is treated as a negative connective by the left rule, and as a positive connective by the right rule.

In Section 2.2.4 (Invertibility and side-conditions) we made the arbitrary choice of using the axiom rule only when an atom is in focus on the right – we have used *negative atoms*. It is more interesting, however, to consider both options. Let us assume a given *atom polarity* function mapping any atom to a sign $\{+, -\}$. We will write X^+ when X is mapped to the positive sign, and Y^- when Y is mapped to the negative sign. We can then refine the axiom rule in two *polarized* axiom rules as follows:

$$\dfrac{X^- = A}{\Gamma, \underline{X^-} \vdash A} \qquad \dfrac{X^+ \in \Gamma}{\Gamma \vdash \underline{X^+}}$$

The left rule is associated to the *negative* polarity as it resembles a non-inversion left-introduction for a (negative) connective.

Those choices of atom polarity do not endanger the completeness theorem.

Theorem from previous works 2. *Any choice of atom polarity function preserves completeness for provability of the focused sequent-calculus.*

This formulation is not generality for the sake of generality: changing the polarity function actually enforces interesting phenomena, in particular when studying the behavior of proof search in the focused system. In particular, forcing all atoms to be negatively polarized corresponds to *backward* search, while forcing all atoms to be positively polarized corresponds to *forward* search [Chaudhuri, Pfenning, and Price, 2008b]. To understand this, let us consider the proofs of the sequent $(X \to Y), (Y \to Z), X \vdash Z$.

There are two ways to start proving this sequent. We may start from our assumption X (forward search), decide to use the implication $X \to Y$, and then we have deduced the new assumption Y. Or we may start from the goal (backward search), decide to use the implication $Y \to Z$, and then it suffices to prove Y.

$$\dfrac{\dfrac{}{X \vdash X} \quad \dfrac{?}{X \to Y, Y \to Z, X, \underline{Y} \vdash Z}}{\underline{X \to Y}, Y \to Z, X \vdash Z} \qquad \dfrac{\dfrac{?}{X \to Y, Y \to Z, X \vdash \underline{Y}} \quad \dfrac{}{Z \vdash Z}}{X \to Y, \underline{Y \to Z}, X \vdash Z}$$

In both cases this first part of the proof finishes with Y under focus, and at this point no axiom rule can be used to discharge Y, so the proof can only proceed by ending the non-invertible phase and choosing a different focus. In the left case, Y is under focus on the left; if it was a negatively polarized axiom Y^-, then the focusing discipline would not allow us to end the non-invertible phase while a negative formula is still under focus, and the proof attempt would fail. In other words, the forward-search approach can only succeed if Y is positively polarized Y^+. Conversely, the backward-search approach can only succeed with a negatively polarized Y^-.

More generally, when a non-invertible phase reaches a positive atom focused on the right (in the succedent), this atom must be in the context (have already been deduced) or the proof attempt fails. A positive atom must first be deduced from the assumptions in context, and only then proved in the goal. This is the essence of forward search; if all atoms are positive, then focused proofs are pure forward-search proofs.

Conversely, when a non-invertible phase reaches a negative atom focused on the left, (in the context), this atom must be in the succedent, so a deduction in the context can only start when it is the goal of the proof. If all atoms are negative, then focused proofs are pure backward-search (goal-directed, demand-driven) proofs.

Remark 2.2. In Section 2.1.1 (Non-canonicity of cut-free sequent proofs), we gave an example of cut-free natural deduction proof that corresponds to two distinct cut-free sequent proofs. This example relied in an essential way on the trace, in the sequent calculus proof structure, of a "backward" or "forward" search process.

In a focused sequent system with polarized atoms, only one of these two cut-free sequent proof is valid – for any choice of atom polarization. In particular, cut-free focused sequent proofs in the purely negative fragment (only negative connectives and negative atoms) correspond closely to cut-free natural deduction proofs, and this enabled Herbelin [1994] to propose a term syntax for the negative fragment of sequent calculus that is very close to the λ-calculus – although this result was not presented in terms of focusing at the time. $*$

2.3 Structural presentations of focusing

The restrictions of Section 2.2.5 (The focusing phase discipline) define a focused subsystem of the sequent calculus for propositional intuitionistic logic.

In this section, we define an isomorphic subsystem, not as a subset of the valid sequent proofs, but by giving a new proof system that enforces the invariant. We call this a "structural" presentation of focusing as it relies on the structure of specific focused inference rules.

The key idea is to separate sequent judgments $\Gamma \vdash A$ into four distinct judgments:

- $\Gamma \vdash_{\mathsf{inv}} A$ proves $\Gamma \vdash A$ by starting with an invertible phase.
- $\Gamma \vdash_{\mathsf{foc}} B$ proves $\Gamma \vdash B$ by starting with a non-invertible phase – it will choose to focus either on the left or on the right
- $\Gamma, [A] \vdash_{\mathsf{foc.l}} B$ proves $\Gamma, A \vdash B$ by focusing on A (on the left)
- $\Gamma \vdash_{\mathsf{foc.r}} [B]$ proves $\Gamma \vdash B$ by focusing on B (on the right)

The full rules are given in Figure 7. In Section 2.4.1 (Explicit shifts) we give a better, more regular system, so we do not give a name to the present system which is mostly for exposition purposes.

Fig. 7. Focused sequent calculus (without shifts)

SEQ-INV-IMPL-RIGHT
$$\frac{\Gamma, A \vdash_{\mathsf{inv}} B}{\Gamma \vdash_{\mathsf{inv}} A \to B}$$

SEQ-INV-DISJ-LEFT
$$\frac{\Gamma, A_1 \vdash_{\mathsf{inv}} B \qquad \Gamma, A_2 \vdash_{\mathsf{inv}} B}{\Gamma, A_1 + A_2 \vdash_{\mathsf{inv}} B}$$

SEQ-INV-CONJ-RIGHT
$$\frac{\Gamma \vdash_{\mathsf{inv}} B_1 \qquad \Gamma \vdash_{\mathsf{inv}} B_2}{\Gamma \vdash_{\mathsf{inv}} B_1 \times B_2}$$

SEQ-INV-FOC
$$\frac{\Gamma \text{ negative or atomic} \qquad \Gamma \vdash_{\mathsf{foc}} B \qquad B \text{ positive or atomic}}{\Gamma \vdash_{\mathsf{inv}} B}$$

SEQ-FOC-CHOOSE-RIGHT
$$\frac{\Gamma \vdash_{\mathsf{foc.r}} [B]}{\Gamma \vdash_{\mathsf{foc}} B}$$

SEQ-FOC-CHOOSE-LEFT
$$\frac{\Gamma, [A] \vdash_{\mathsf{foc.l}} B}{\Gamma, A \vdash_{\mathsf{foc}} B}$$

SEQ-FOC-DISJ-RIGHT
$$\frac{\Gamma \vdash_{\mathsf{foc.r}} [B_i]}{\Gamma \vdash_{\mathsf{foc.r}} [B_1 + B_2]}$$

SEQ-FOC-CONJ-LEFT
$$\frac{\Gamma, [A_i] \vdash_{\mathsf{foc.l}} B}{\Gamma, [A_1 \times A_2] \vdash_{\mathsf{foc.l}} B}$$

SEQ-FOC-IMPL-LEFT
$$\frac{\Gamma \vdash_{\mathsf{foc.r}} [B] \qquad \Gamma, [A] \vdash_{\mathsf{foc.l}} C}{\Gamma, [B \to A] \vdash_{\mathsf{foc.l}} C}$$

SEQ-FOC-AXIOM-LEFT
$$\frac{}{\Gamma, [X^-] \vdash_{\mathsf{foc.l}} X^-}$$

SEQ-FOC-AXIOM-RIGHT
$$\frac{}{\Gamma, X^+ \vdash_{\mathsf{foc.r}} [X^+]}$$

SEQ-FOC-INV-LEFT
$$\frac{A \text{ positive} \qquad \Gamma, A \vdash_{\mathsf{inv}} B}{\Gamma, [A] \vdash_{\mathsf{foc.l}} B}$$

SEQ-FOC-INV-RIGHT
$$\frac{B \text{ negative} \qquad \Gamma \vdash_{\mathsf{inv}} B}{\Gamma \vdash_{\mathsf{foc.r}} [B]}$$

The rules allowing to transition between judgments use explicit requirements on the polarity of formulas to enforce phases to be as long as possible. We cannot transition from the invertible judgment to the non-invertible one (ending an invertible phase) if there remain a positive on the left or an atomic on the right, that is if we could apply an invertible rule. We cannot transition from the non-invertible to the invertible judgment (ending a non-

invertible phase) if we can still apply a non-invertible introduction rule to the formula under focus – if it is positive on the right, or negative on the left.

It is possible to erase such structural proofs into sequent proofs in the restricted subsystem; the phase transition rules SEQ-INV-FOC, SEQ-INV-FOC-LEFT, SEQ-INV-FOC-RIGHT, SEQ-FOC-INV-LEFT, and SEQ-FOC-INV-RIGHT are erased in the process, but it is still a one-to-one mapping: the focusing structure, explicit in the structural presentation, is implicit in the restricted presentation: it is present in the justification of why a given proof is "valid", and a given proof is valid in a unique way.

2.4 Polarized formulas

Once a choice of polarization has been made for atoms, all formulas are either positive or negative, and can thus be split in two grammatical categories. Recent presentations of focused systems often make the transitions from one grammatical category to another explicit by placing *shifts* in the syntax.

2.4.1 Explicit shifts

We write $\langle N \rangle^+$ for a negative formula embedded into the set of positive formulas, and conversely $\langle P \rangle^-$ for a positive formula seen as a negative formula. The complete grammar of those *polarized formulas* is defined in Figure 8.

Fig. 8. Polarized propositional formulas

P, Q	$::=$		positive formulas
		$\mid X^+$	positive atom
		$\mid P + Q$	sum
		$\mid \langle N \rangle^+$	shift
N, M	$::=$		negative formulas
		$\mid X^-$	negative atom
		$\mid P \to N$	implication
		$\mid N \times M$	product
		$\mid \langle P \rangle^-$	shift
$P^{\mathsf{at}}, Q^{\mathsf{at}}$	$::= P \mid X^-$		positive or atomic
$N^{\mathsf{at}}, M^{\mathsf{at}}$	$::= N \mid X^+$		negative or atomic
Σ	$::= \emptyset \mid \Sigma, P$		positive context
Γ^{at}	$::= \emptyset \mid \Gamma^{\mathsf{at}}, N^{\mathsf{at}}$		negative or atomic context

Notice that, while the (positive) sum expects positive subformulas and (negative) product expects negative subformula, the (negative) implication expects a positive subformula on its left-hand side.

2.4.2 *Polarized judgments*

In the focused proof system of Figure 7, for the choice of focusing judgment $\Gamma \vdash_{\mathsf{foc}} A$, we know that Γ may only contain negative or atomic formulas, while A must be positive or atomic. This directly corresponds to a judgment $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}$ with polarized types.

On the other hand, for the invertible judgment $\Gamma \vdash_{\mathsf{inv}} A$, the context Γ may contain either positive or negative formulas, and similarly the goal A may be positive or negative.

For the context Γ for example, one may think of splitting it into two contexts $\Gamma;\Sigma$, with only negatives in Γ and positives in Σ. However, looking at the transition between focused judgments rather suggests using $\Gamma^{\mathsf{at}};\Sigma$, where the first judgment contains either negative types or positive atoms. Consider the two rules moving from a focused judgment to the invertible judgment – in both rules we know that Γ has only negative or atomic formulas:

SEQ-FOC-INV-LEFT
$$\frac{A \text{ positive} \qquad \Gamma, A \vdash_{\mathsf{inv}} B}{\Gamma, [A] \vdash_{\mathsf{foc.l}} B}$$

SEQ-FOC-INV-RIGHT
$$\frac{B \text{ negative} \qquad \Gamma \vdash_{\mathsf{inv}} B}{\Gamma \vdash_{\mathsf{foc.r}} [B]}$$

When moving from $\Gamma, [A] \vdash_{\mathsf{foc.l}} B$ where Γ is negative or atomic and A is positive, it is natural to use the context split $\Gamma;A$: the negative or atomic context contains hypotheses that were present before the start of the invertible phase, and the strictly positive context contains formulas that have been or will be decomposed during the invertible phase.

A similar reasoning can be applied to the goal part. When coming from the SEQ-FOC-INV-LEFT rule, we know that B is a positive or atomic formula that has already been fully decomposed by a previous invertible phase. When coming from the rule SEQ-FOC-INV-RIGHT, B is a negative formula that must be decomposed by the invertible phase. This suggests a goal of the form $N^? \mid Q^{\mathsf{at}^?}$, where $N^?$ and $Q^{\mathsf{at}^?}$ are either formulas or an empty position $\emptyset$, but exactly one of them is a formula. The N position is empty when coming from a left-focusing phase – and remains empty during the whole invertible phase – and the Q^{at} position is empty when coming from a right-focusing phase.

To summarize, we use a four-position judgment $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}}$, where N and Q^{at} are both optional but exactly one of them is a formula, with the following transition rules in and out of the focusing phase:

$$\frac{\Gamma^{\mathsf{at}};Q \vdash_{\mathsf{inv}} \emptyset \mid P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, [\langle Q \rangle^{-}] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}} \qquad \frac{\Gamma^{\mathsf{at}};\emptyset \vdash_{\mathsf{inv}} N \mid \emptyset}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [\langle N \rangle^{+}]}$$

FOCSEQ-INV-FOC
$$\frac{\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{foc}} (P^{\mathsf{at}} \mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}}; \left\langle \Gamma^{\mathsf{at}\prime} \right\rangle^{+\mathsf{at}} \vdash_{\mathsf{inv}} \langle P^{\mathsf{at}} \rangle^{-\mathsf{at}} \mid Q^{\mathsf{at}}}$$

In rule FOCSEQ-INV-FOC we use two partial shifting functions $\langle N^{\mathsf{at}} \rangle^{+\mathsf{at}}$ (respectively $\langle P^{\mathsf{at}} \rangle^{-\mathsf{at}}$) that take a negative or atomic (respectively positive or atomic) formula and turns it into a positive or atomic (respectively negative or atomic) formula.

$$\left\langle X^{+} \right\rangle^{+\mathsf{at}} \stackrel{\mathsf{def}}{=} X^{+} \qquad \langle N \rangle^{+\mathsf{at}} \stackrel{\mathsf{def}}{=} \langle N \rangle^{+}$$

$$\left\langle X^{-} \right\rangle^{-\mathsf{at}} \stackrel{\mathsf{def}}{=} X^{-} \qquad \langle P \rangle^{-\mathsf{at}} \stackrel{\mathsf{def}}{=} \langle P \rangle^{-}$$

The rule moving out of the invertible phase requires that its positive context be of the form $\langle \Gamma^{\mathsf{at}} \rangle^{+\mathsf{at}}$, that is, that it only contain negative or atomic formulas.

Similarly, we request that the goal be of the form $\langle P^{\mathsf{at}} \rangle^{-\mathsf{at}} \mid Q^{\mathsf{at}}$: if the negative formula position is not empty, then to leave the invertible phase it must be the shift of a positive or atomic formula. We know that exactly one of the formulas P^{at} and Q^{at} is defined and the other is empty, and we write $(P^{\mathsf{at}} \mid Q^{\mathsf{at}})$ to denote the formula that is defined.

The full rules of the focused sequent-calculus with polarized formulas are given in Figure 9 (Focused sequent calculus for polarized propositional intuitionistic logic).

Fig. 9. Focused sequent calculus for polarized propositional intuitionistic logic

$$\textsc{polseq-inv-impl-right}\ \frac{\Gamma^{\mathsf{at}}; \Sigma, P \vdash_{\mathsf{inv}} N \mid \emptyset}{\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} P \to N \mid \emptyset}$$

$$\textsc{polseq-inv-conj-right}\ \frac{\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N_1 \mid \emptyset \qquad \Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N_2 \mid \emptyset}{\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N_1 \times N_2 \mid \emptyset}$$

$$\textsc{polseq-inv-disj-left}\ \frac{\Gamma^{\mathsf{at}}; \Sigma, P_1 \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}} \qquad \Gamma^{\mathsf{at}}; \Sigma, P_2 \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}}; \Sigma, P_1 + P_2 \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}}}$$

$$\textsc{polseq-inv-foc}\ \frac{\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{foc}} (P^{\mathsf{at}} \mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}}; \left\langle \Gamma^{\mathsf{at}\prime} \right\rangle^{+\mathsf{at}} \vdash_{\mathsf{inv}} \langle P^{\mathsf{at}} \rangle^{-\mathsf{at}} \mid Q^{\mathsf{at}}}$$

$$\textsc{polseq-foc-choose-right}\ \frac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [P]}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P}$$

$$\textsc{polseq-foc-choose-left}\ \frac{\Gamma^{\mathsf{at}}, [N] \vdash_{\mathsf{foc.l}} Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, N \vdash_{\mathsf{foc}} Q^{\mathsf{at}}}$$

$$\textsc{polseq-foc-conj-left}\ \frac{\Gamma^{\mathsf{at}}, [N_i] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, [N_1 \times N_2] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}$$

$$\textsc{polseq-foc-impl-left}\ \frac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [Q] \qquad \Gamma^{\mathsf{at}}, [N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, [Q \to N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}$$

$$\textsc{polseq-foc-inv-left}\ \frac{\Gamma^{\mathsf{at}}; Q \vdash_{\mathsf{inv}} \mid P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, [\langle Q \rangle^{-}] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}$$

$$\textsc{polseq-foc-axiom-left}\ \frac{}{\Gamma^{\mathsf{at}}, [X^{-}] \vdash_{\mathsf{foc.l}} X^{-}}$$

$$\textsc{polseq-foc-axiom-right}\ \frac{}{\Gamma^{\mathsf{at}}, X^{+} \vdash_{\mathsf{foc.r}} [X^{+}]}$$

$$\textsc{polseq-foc-disj-right}\ \frac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [P_i]}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [P_1 + P_2]}$$

$$\textsc{polseq-foc-inv-right}\ \frac{\Gamma^{\mathsf{at}}; \emptyset \vdash_{\mathsf{inv}} N \mid}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [\langle N \rangle^{+}]}$$

Remark 2.3. While this may seem a minor syntactic difference, adding explicit shifts is in fact a radical idea, because it let us make distinction between formulas that we couldn't distinguish before: a formula can be shifted to the other polarity, and then shifted back to its original polarity, and we obtain a different formula!

There is an interesting analogy between this situation and the difference in meaning of connectives between intuitionistic and classical logic in Zeilberger [2013]. ∗

Example 2.5. The formulas $P \to Q \to N$ and $P \to \left\langle \langle Q \to N \rangle^{+} \right\rangle^{-}$ have proofs that differ in very interesting ways: a focused proof of the first formula necessarily starts by invertibly introducing both P and Q in context; but for the second formula, the invertible phase stops after introducing P in context, as the formula $\langle Q \to N \rangle^{+}$ is positive and may thus not be invertibly introduced. A focused proof may thus perform arbitrary left-focusing phases on

the context at this point, before focusing on this positive formula, unboxing its negative content, and then introducing the second function type. ◊

2.5 Defocusing

When relating non-focused proof systems with focused systems with polarized formulas, we rely on the fact that focused proofs are also valid non-focused proofs. This is self-evident when focused proofs are defined as the subset of non-focused proofs that satisfy the focusing discipline, but requires an erasure step for the structural presentations of focusing, in particular when using explicit shifts (that is, a different structure for formulas).

In Figure 10 (Polarity erasure), we define the polarity erasure operations $\lfloor P \rfloor_\pm$ and $\lfloor N \rfloor_\pm$ that return a formula without explicit shifts. It is readily extended to contexts.

Fig. 10. Polarity erasure

$$\begin{array}{lcl} \lfloor X^+ \rfloor_\pm & \stackrel{\text{def}}{=} & X \\ \lfloor P+Q \rfloor_\pm & \stackrel{\text{def}}{=} & \lfloor P \rfloor_\pm + \lfloor Q \rfloor_\pm \\ \lfloor \langle N \rangle^+ \rfloor_\pm & \stackrel{\text{def}}{=} & \lfloor N \rfloor_\pm \end{array} \qquad \begin{array}{lcl} \lfloor Y^- \rfloor_\pm & \stackrel{\text{def}}{=} & Y \\ \lfloor P \to N \rfloor_\pm & \stackrel{\text{def}}{=} & \lfloor P \rfloor_\pm \to \lfloor N \rfloor_\pm \\ \lfloor N \times M \rfloor_\pm & \stackrel{\text{def}}{=} & \lfloor N \rfloor_\pm \times \lfloor M \rfloor_\pm \\ \lfloor \langle P \rangle^- \rfloor_\pm & \stackrel{\text{def}}{=} & \lfloor P \rfloor_\pm \end{array}$$

We can then erase any proof derivation from a focused proof to a non-focused proof.

Theorem 2.1 (Polarity erasure)**.**
The following provability implications hold:

$$\begin{array}{lcl} \Gamma^{\text{at}}; \Sigma \vdash_{\text{inv}} N \mid P^{\text{at}} & \Longrightarrow & \lfloor \Gamma^{\text{at}} \rfloor_\pm, \lfloor \Sigma \rfloor_\pm \vdash \lfloor N \rfloor_\pm, \lfloor P^{\text{at}} \rfloor_\pm \\ \Gamma^{\text{at}} \vdash_{\text{foc}} P^{\text{at}} & \Longrightarrow & \lfloor \Gamma^{\text{at}} \rfloor_\pm \vdash \lfloor P^{\text{at}} \rfloor_\pm \\ \Gamma^{\text{at}}, [N] \vdash_{\text{foc.l}} P^{\text{at}} & \Longrightarrow & \lfloor \Gamma^{\text{at}} \rfloor_\pm, \lfloor N \rfloor_\pm \vdash \lfloor P^{\text{at}} \rfloor_\pm \\ \Gamma^{\text{at}} \vdash_{\text{foc.r}} [P] & \Longrightarrow & \lfloor \Gamma^{\text{at}} \rfloor_\pm \vdash \lfloor P \rfloor_\pm \end{array}$$

Proof. We use the notation $\Pi :: \mathscr{J}$ to say that Π is a proof derivation for the judgment $\mathscr{J}$; for example, $\Pi :: \Gamma \vdash_{\text{foc}} A$ means that Π is a derivation whose conclusion is $\Gamma \vdash_{\text{foc}} A$.

The proof is by direct induction, by erasing the focusing information from the proof – the proof structure is unchanged. For example:

$$\left\lfloor \frac{\Pi_P :: \Gamma^{\text{at}} \vdash_{\text{foc.r}} [P] \qquad \Pi_N :: \Gamma^{\text{at}}, [N] \vdash_{\text{foc.l}} Q^{\text{at}}}{\Gamma^{\text{at}}, [P \to N] \vdash_{\text{foc.l}} Q^{\text{at}}} \right\rfloor_\pm \stackrel{\text{def}}{=}$$

$$\frac{\lfloor \Pi_P \rfloor_\pm :: \lfloor \Gamma^{\text{at}} \rfloor_\pm \vdash \lfloor P \rfloor_\pm \qquad \lfloor \Pi_N \rfloor_\pm :: \lfloor \Gamma^{\text{at}} \rfloor_\pm, \lfloor N \rfloor_\pm \vdash \lfloor Q^{\text{at}} \rfloor_\pm}{\lfloor \Gamma^{\text{at}} \rfloor_\pm, \lfloor P \to N \rfloor_\pm \vdash \lfloor Q^{\text{at}} \rfloor_\pm}$$

Because $\lfloor P \to N \rfloor_\pm$ is equal (by definition) to $\lfloor P \rfloor_\pm \to \lfloor N \rfloor_\pm$, this is the (valid) left-introduction rule for implication in the non-focused sequent calculus.

The rules that are solely concerned with the focusing structure are erased in the process. For example:

$$\left\lfloor \frac{\Pi :: \Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}; \emptyset \vdash_{\mathsf{inv}} \emptyset \mid P^{\mathsf{at}}} \right\rfloor_{\pm} \quad \stackrel{\mathsf{def}}{=} \quad \lfloor \Pi \rfloor_{\pm} :: \lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm} \vdash \lfloor P^{\mathsf{at}} \rfloor_{\pm}$$

$$\left\lfloor \frac{\Pi :: \Gamma^{\mathsf{at}}; \emptyset \vdash_{\mathsf{inv}} N \mid \emptyset}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [\langle N \rangle^{-}]} \right\rfloor_{\pm} \quad \stackrel{\mathsf{def}}{=} \quad \lfloor \Pi \rfloor_{\pm} :: \lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm} \vdash \lfloor N \rfloor_{\pm}$$

□

In particular, this means that our structural focused system is *sound* with respect to propositional intuitionistic logic: the (defocused erasing of) formulas it proves are all provable in our reference proof system.

Furthermore, one can easily check that proofs obtained in this way remain valid focused proof – they are in the restricted subsystem defined by the focusing restrictions.

3 Focused λ-calculus

In this section we build this presentation of "focused λ-terms", which will be useful for the results of the latter sections. We could equip the focused sequent calculus with a term syntax directly, but they may seem rather exotic to readers familiar with the λ-calculus. Instead, we start by building a focused version of natural deduction, and use its natural term syntax as our notion of focused λ-calculus. Interestingly, the result is rather close to the grammar of β-normal forms familiar to users of the λ-calculus. It is mostly a refinement of the distinction, on β-normal forms, between constructors and neutral terms.

3.1 Intuitionistic natural deduction, focused

3.1.1 Invertibility of elimination rules

The notion of "invertible rule" works very well in sequent-calculus presentations of logics, but needs to be modified to fit natural deduction presentation. Consider for example:

$$\text{SEQ-IMPL-LEFT}\ \frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C} \qquad \text{ND-IMPL-ELIM}\ \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

$$\text{SEQ-DISJ-LEFT}\ \frac{\Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma, A_1 + A_2 \vdash C} \qquad \text{ND-DISJ-ELIM}\ \frac{\Gamma \vdash A_1 + A_2 \qquad \Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma \vdash C}$$

Definition 2.1 (Invertible rule) (the conclusion is invertible if and only if all premises are) works for introduction rules, but is not suited for elimination rules.

Instead, we will rely on the rootward reading of elimination rules: the elimination of implications let us deduce $\Gamma \vdash B$ from $\Gamma \vdash A \to B$ – whenever $\Gamma \vdash A$ is provable. This

suggests that we could reason about the invertibility of this rule by starting from the eliminated premise, rather than from the conclusion. We propose the following notion of invertibility for elimination rules.

Definition 3.1 Invertible elimination rule**.**
An elimination rule is invertible if, whenever its eliminated premise is provable, then its conclusion is provable if and only if it is provable using this elimination rule.

The definition captures the intuition that an invertible rule "can always be applied", but in a situation where we do not decide which rule to apply by looking at the conclusion judgment, but by looking at the eliminated premise. Let us highlight the eliminated premise in both elimination rules:

$$\frac{\boxed{\Gamma \vdash A \to B} \qquad \Gamma \vdash A}{\Gamma \vdash B} \qquad\qquad \frac{\boxed{\Gamma \vdash A_1 + A_2} \qquad \Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma \vdash C}$$

We can check that, with this definition, the elimination of implication is non-invertible and the elimination of disjunction is invertible, as expected. Invertibility fails when one of the non-eliminated premises is non-provable, while the conclusion would be – by applying another rule. For implication: if B is in the context Γ, both the eliminated premise and conclusion are provable but $\Gamma \vdash A$ may be non-provable. For disjunction: if $\Gamma \vdash C$ is provable, then we can build premises $\Gamma, A_i \vdash C$ by weakening, so the rule is applicable.

Intercalation syntax We use the following syntax from Brock-Nannestad and Schürmann [2010], itself inspired by the "intercalation calculus":

$$\text{NAT-FOC-ELIM-IMPL}\ \frac{\Gamma \Downarrow A \to B \qquad \Gamma \Uparrow A}{\Gamma \Downarrow B} \qquad\qquad \text{NAT-FOC-INTRO-DISJ}\ \frac{\Gamma \Uparrow A_i}{\Gamma \Uparrow A_1 + A_2}$$

The direction of the arrows corresponds to the direction of proof search. When we try to prove the conclusion of an introduction rule (here $\Gamma \Uparrow A_1 + A_2$) the rule says that it suffices to "look up" and prove its premise (here $\Gamma \Uparrow A_i$). When we know how to deduce the eliminated premise of an elimination rule (here $\Gamma \Downarrow A \to B$), then we can "look down" and further deduce its conclusion (here $\Gamma \Downarrow B$ – provided we can prove $\Gamma \Uparrow A$).

Structural focusing for natural deduction We give the full rules of our focused natural deduction in Figure 11 (Focused natural deduction, with explicit shifts), using explicit shifts in the style of Figure 9 (Focused sequent calculus for polarized propositional intuitionistic logic).

The involved judgments are as follows:

- $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}$, the invertible judgment, with the same structure as a sequent-calculus judgment $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}$
- $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P^{\mathsf{at}}$, the judgment starting the focusing phase (a focus has not been chosen yet), with the same structure as the sequent-calculus judgment $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P^{\mathsf{at}}$
- $\Gamma^{\mathsf{at}} \Uparrow P$, the focused introduction judgment, corresponding to the right-focusing sequent judgment $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [P]$
- $\Gamma^{\mathsf{at}} \Downarrow N$, the focused elimination judgment, corresponding to the left-focusing sequent judgment $\Gamma^{\mathsf{at}}, [N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}$.

Fig. 11. Focused natural deduction, with explicit shifts

$$\textsc{nat-inv-impl-intro}\ \frac{\Gamma^{\mathsf{at}};\Sigma, P \vdash_{\mathsf{inv}} N \mid \emptyset}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} P \to N \mid \emptyset}$$

$$\textsc{nat-inv-conj-intro}\ \frac{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} N_1 \mid \emptyset \qquad \Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} N_2 \mid \emptyset}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} N_1 \times N_2 \mid \emptyset}$$

$$\textsc{nat-inv-disj-elim}\ \frac{\Gamma^{\mathsf{at}};\Sigma, Q_1 \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}} \qquad \Gamma^{\mathsf{at}};\Sigma, Q_2 \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}}{\Gamma^{\mathsf{at}};\Sigma, Q_1 + Q_2 \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}}$$

$$\textsc{nat-inv-foc}\ \frac{\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{foc}} (P^{\mathsf{at}} \mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}}; \left\langle \Gamma^{\mathsf{at}\prime} \right\rangle^{+\mathsf{at}} \vdash_{\mathsf{inv}} \langle P^{\mathsf{at}} \rangle^{-\mathsf{at}} \mid Q^{\mathsf{at}}}$$

$$\textsc{nat-foc-choose-left}\ \frac{\Gamma^{\mathsf{at}} \Downarrow \langle P \rangle^{-} \qquad \Gamma^{\mathsf{at}}; P \vdash_{\mathsf{inv}} \emptyset \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}}$$

$$\textsc{nat-foc-choose-right}\ \frac{\Gamma^{\mathsf{at}} \Uparrow P}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P}$$

$$\textsc{nat-foc-sum-intro}\ \frac{\Gamma^{\mathsf{at}} \Uparrow P_i}{\Gamma^{\mathsf{at}} \Uparrow P_1 + P_2}$$

$$\textsc{nat-foc-axiom-right}\ \frac{}{\Gamma^{\mathsf{at}}, X^{+} \Uparrow X^{+}}$$

$$\textsc{nat-foc-inv-right}\ \frac{\Gamma^{\mathsf{at}};\emptyset \vdash_{\mathsf{inv}} N \mid \emptyset}{\Gamma^{\mathsf{at}} \Uparrow \langle N \rangle^{+}}$$

$$\textsc{nat-foc-impl-elim}\ \frac{\Gamma^{\mathsf{at}} \Downarrow P \to N \qquad \Gamma^{\mathsf{at}} \Uparrow P}{\Gamma^{\mathsf{at}} \Downarrow N}$$

$$\textsc{nat-foc-conj-elim}\ \frac{\Gamma^{\mathsf{at}} \Downarrow N_1 \times N_2}{\Gamma^{\mathsf{at}} \Downarrow N_i}$$

$$\textsc{nat-foc-axiom-left}\ \frac{\Gamma^{\mathsf{at}} \Downarrow X^{-}}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} X^{-}}$$

$$\textsc{nat-foc-contraction}\ \frac{}{\Gamma^{\mathsf{at}}, N \Downarrow N}$$

The mapping between the various judgments is direct, except for the focused elimination judgment whose proofs, compared to left-focusing proofs, are *turned upside down*. For example, the "release" rules that explain how the focused phase stops (on an atom or a shift) are now the rootwardmost rule of the elimination phase. Conversely, the counterpart of the sequent rule that started a left-focusing phase $(\Gamma^{\mathsf{at}}, N), [N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}$ now becomes the leaf rule concluding $\Gamma^{\mathsf{at}}, N \Downarrow N$.

Elimination or left-introduction rules for positives? While we claim that the system of Figure 11 (Focused natural deduction, with explicit shifts) is in natural deduction style, one cannot help noticing that the invertible rules are actually sequent calculus rules; in particular, we have left-introduction rules for positives, rather than elimination rules as expected. Is this system really natural deduction? We have three different angles of answer.

First, we should point out that positive eliminations do not really fit natural deduction in the first place. Even though they do have a formulation that is different from the sequent calculus one, they stand out of the rest of the system and are the source of various difficulties when studying the meta-theory of the mixed-polarity system. We are making them more sequent-like than they were before, but the worm was already in the fruit. The negative elimination rules are the usual one, and this is what makes a system distinctively natural deduction in style.

Second, in a focused system, invertible rules do not really matter, because they are automatically applied in an irrelevant order. Invertible rules could in fact be removed from

the term syntax, and reconstructed at type-checking time. Again, what really matters are the non-invertible elimination rules.

Third and finally, we tried to look for a formulation of the invertible positive rules that would be closer to the natural deduction rule, and didn't find any. In particular, it is interesting to see why the obvious idea does not work:

$$\frac{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} \langle P_1 + P_2 \rangle^- \mid \emptyset \qquad \begin{array}{r} \Gamma^{\mathsf{at}};\Sigma, P_1 \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}} \\ \Gamma^{\mathsf{at}};\Sigma, P_2 \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}} \end{array}}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}}}$$

This would be a sensible invertible rule if we could always choose it without having to make choices – choice is the privilege of non-invertible rules. It is not, because we cannot know locally side of $P_1 + P_2$ to attempt to prove in the left rule. More generally, this left premise may incur arbitrary proof search, including non-invertible rules.

One idea would be to restrict this unbounded-search premise to a more specific judgment: instead of allowing any proofs of the positives to eliminate, could we allow only "simple" proofs? Using the focused elimination judgment $\Gamma^{\mathsf{at}} \Downarrow \langle P_1 + P_2 \rangle^-$ resembles the restriction on normal natural proofs (the eliminated premise cannot be a constructor, so it should start with an elimination or axiom rule), but it is still not invertible, as the focused elimination judgment has to make choice.

There remain an even simpler notion of "being provable": hypotheses are immediately provable if they are in the assumption context. Due to the polarity invariants, we know that positives are in Σ if they are in $\Gamma^{\mathsf{at}}, \Sigma$. This suggests the following restriction of the rule above:

$$\frac{\Sigma \ni (P_1 + P_2) \qquad \begin{array}{r} \Gamma^{\mathsf{at}};\Sigma, P_1 \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}} \\ \Gamma^{\mathsf{at}};\Sigma, P_2 \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}} \end{array}}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} N \mid Q^{\mathsf{at}}}$$

This rule is exactly the sequent left-introduction rules that we use.

3.1.2 Equivalence with the focused sequent calculus

Comparing arbitrary natural deduction and sequent-calculus proofs is delicate, and in particular there is no one-to-one correspondence between cut-free proofs in either system – see Section 2.1.1. The restrictions of focusing give more structure to cut-free proofs, which allow to get a one-to-one correspondence.

Theorem 3.1 (Bijection between focused sequent calculus and focused natural deduction)**.** *There is a one-to-one correspondence between the cut-free focused sequent calculus proofs of Figure 11 (Focused natural deduction, with explicit shifts) and the cut-free focused natural deduction proofs of Figure 9 (Focused sequent calculus for polarized propositional intuitionistic logic).*

Proof. The general idea of this simple proof is that the difference between the two focused systems is a stylistic choice of direction: elimination rules in natural deduction are written "rootward", while the corresponding left-introduction rules of sequent calculus are written "leafward". To translate between the two systems, it thus suffices to reverse the direction of these parts of the proof.

For example, consider the sequent proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma^{\mathsf{at}}, [Z^-] \vdash_{\mathsf{foc.l}} Z^-}}{\Gamma^{\mathsf{at}}, [Y \times Z^-] \vdash_{\mathsf{foc.l}} Z^-}}{\Gamma^{\mathsf{at}}, [X \times (Y \times Z^-)] \vdash_{\mathsf{foc.l}} Z^-}}{\Gamma^{\mathsf{at}} \ni X \times (Y \times Z^-) \vdash_{\mathsf{foc}} Z^-}$$

It corresponds to the following natural deduction proof, which is a direct reversal:

$$\dfrac{\dfrac{\dfrac{}{\Gamma^{\mathsf{at}} \Downarrow X \times (Y \times Z^-)}}{\Gamma^{\mathsf{at}} \Downarrow Y \times Z^-}}{\Gamma^{\mathsf{at}} \Downarrow Z^-}$$

$$\Gamma^{\mathsf{at}} \ni X \times (Y \times Z^-) \vdash_{\mathsf{foc}} Z^-$$

In the general case, remark that there is a direct correspondence between:

- invertible sequent judgments $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}$ and invertible natural deduction judgments $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}$
- right-focused sequent judgments $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [P]$ and introduction-focused natural deduction judgments $\Gamma^{\mathsf{at}} \Uparrow P$

To complete our correspondence, we give a one-to-one mapping between:

- choice-of-focusing sequent judgments $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P^{\mathsf{at}}$ and focused natural deduction judgments $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P$
- *partial* left-focused and elimination-focused phases, which is the reversal we described informally; it is a correspondence between partial proof derivations of the form

$$\dfrac{\dfrac{\Gamma^{\mathsf{at}}, [N'] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}{\Pi}}{\Gamma^{\mathsf{at}}, [N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}} \qquad \longleftrightarrow \qquad \dfrac{\dfrac{\Gamma^{\mathsf{at}} \Downarrow N}{\Pi'}}{\Gamma^{\mathsf{at}} \Downarrow N'}$$

  We write $\Pi \longleftrightarrow \Pi'$ when this correspondence holds.

The correspondence on the choice-of-focusing judgments is as follows:

$$\dfrac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [P]}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P} \qquad \longleftrightarrow \qquad \dfrac{\Gamma^{\mathsf{at}} \Uparrow P}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} P}$$

$$\dfrac{\dfrac{\dfrac{\Gamma^{\mathsf{at}}, [X^-] \vdash_{\mathsf{foc.l}} X^-}{\Pi}}{\Gamma^{\mathsf{at}}, [N] \vdash_{\mathsf{foc.l}} X^-}}{\Gamma^{\mathsf{at}} \ni N \vdash_{\mathsf{foc}} X^-} \qquad \longleftrightarrow \qquad \dfrac{\dfrac{\dfrac{\Gamma^{\mathsf{at}} \Downarrow N}{\Pi'}}{\Gamma^{\mathsf{at}} \Downarrow X^-}}{\Gamma^{\mathsf{at}} \ni N \vdash_{\mathsf{foc}} X^-} \qquad \text{when } \Pi \longleftrightarrow \Pi'$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma^{\mathsf{at}}; Q \vdash_{\mathsf{inv}} \emptyset \mid P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, [\langle Q \rangle^{-}] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}}{\Pi}}{\Gamma^{\mathsf{at}}, [N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}}{\Gamma^{\mathsf{at}} \ni N \vdash_{\mathsf{foc}} P^{\mathsf{at}}} \quad \longleftrightarrow \quad \dfrac{\dfrac{\dfrac{\Gamma^{\mathsf{at}} \Downarrow N}{\Pi'}}{\Gamma^{\mathsf{at}} \Downarrow \langle Q \rangle^{-}} \qquad \Gamma^{\mathsf{at}}; Q \vdash_{\mathsf{inv}} \emptyset \mid P^{\mathsf{at}}}{\Gamma^{\mathsf{at}} \ni N \vdash_{\mathsf{foc}} P^{\mathsf{at}}}$$

$$\text{when } \Pi \longleftrightarrow \Pi'$$

The correspondence between the partial left-focused and elimination-focused phases is as follows. First, we describe the correspondence between any inference rules (that is, partial proofs of height 2):

$$\dfrac{\Gamma^{\mathsf{at}}, [N_i] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, [N_1 \times N_2] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}} \quad \longleftrightarrow \quad \dfrac{\Gamma^{\mathsf{at}} \Downarrow N \times}{\Gamma^{\mathsf{at}} \Downarrow N_i}$$

$$\dfrac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc.r}} [Q] \qquad \Gamma^{\mathsf{at}}, [N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}, [Q \to N] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}} \quad \longleftrightarrow \quad \dfrac{\Gamma^{\mathsf{at}} \Downarrow Q \to N \qquad \Gamma^{\mathsf{at}} \Uparrow Q}{\Gamma^{\mathsf{at}} \Downarrow N}$$

Then we can reverse longer proof by simply concatenating the reverses:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma^{\mathsf{at}}, [N_3] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}{\Pi_2}}{\Gamma^{\mathsf{at}}, [N_2] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}}}{\Pi_1}}{\Gamma^{\mathsf{at}}, [N_1] \vdash_{\mathsf{foc.l}} P^{\mathsf{at}}} \quad \longleftrightarrow \quad \dfrac{\dfrac{\dfrac{\Gamma^{\mathsf{at}} \Downarrow N_1}{\Pi_1'}}{\Gamma^{\mathsf{at}} \Downarrow N_2}}{\Pi_2'}}{\Gamma^{\mathsf{at}} \Downarrow N_3} \qquad \text{when} \qquad \Pi_1 \leftrightarrow \Pi_1', \quad \Pi_2' \leftrightarrow \Pi_2'$$

□

3.2 A focused term syntax: focused λ-calculus

We propose a term syntax for this focused natural deduction that is as close as reasonably possible to the λ-calculus. We would like to think of it as mostly a subset of λ-terms with minor additions.

Looking at the four judgments of our focused system, we propose the following classes of terms:

- Terms for the invertible judgments $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}$ contain a mix of constructors and destructors and have subterms of arbitrary judgments; we will simply use the class of arbitrary (cut-free) terms, with meta-variable t. We will sometimes call these *invertible terms* to insist that they come from an invertible phase.
- Terms for the focused elimination judgment $\Gamma^{\mathsf{at}} \Downarrow N$ are variables, to which a series of elimination forms (function application or pair projection) are applied. This corresponds to the usual class (in the purely negative fragment) of *neutral terms*, often written with the meta-variable n. We call them *negative* neutral terms.
- Terms for the focused introduction judgment $\Gamma^{\mathsf{at}} \Uparrow P$ are series of introduction forms, eventually followed by an invertible proof term. We call them *positive* neutral terms, and use the meta-variable p.

- The choice-of-focusing judgment $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}$ has no interesting structure of its own, but it can become either an introduction-focused or elimination-focused phase, and we use the meta-variable f where this choice occurs. We call them *focusing terms*.

The grammar is described in Figure 12 (Term grammar for the focused λ-calculus), and the corresponding typing system (using mappings from term variables to types as contexts, instead of sets) is given in Figure 13 (Typing rules for the focused λ-calculus). We call this system the *focused λ-calculus*.

Fig. 12. Term grammar for the focused λ-calculus

t,u,r	$::=$	(invertible) terms
	$\mid \lambda x.t$	λ-abstraction
	$\mid (t,u)$	pair
	$\mid \mathtt{match}\ x\ \mathtt{with}\ \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2$	variable case split
	$\mid ()$	trivial
	$\mid \mathtt{absurd}(x)$	absurd variable
	$\mid (f : P^{\mathsf{at}})$	focusing term
f,g	$::=$	focusing terms
	$\mid (n : X^-)$	negative conclusion
	$\mid \mathtt{let}\ (x : P) = n\ \mathtt{in}\ t$	positive binding
	$\mid (p : P)$	positive conclusion
n,m	$::=$	negative neutral terms
	$\mid \pi_i\ n$	projection
	$\mid n\ p$	application
	$\mid (x : N)$	negative head variable
p,q	$::=$	positive neutral terms
	$\mid \sigma_i\ p$	injection
	$\mid (x : X^+)$	positive head variable
	$\mid (t : N)$	invertible conclusion

This grammar is designed to describe well-typed terms, and we have used some typing annotations, which are not actually part of the term syntax, but describe the expected types of various subterms or variables – for the whole term to be well-typed. For example, the class p of positive neutral terms includes the whole class η of invertible terms (which itself includes f, in particular n and p), so as a grammar of untyped terms positive neutrals and invertible terms seem to be equivalent. However, because we will only allow the use of an invertible term inside a positive neutral at a negative type (and not in arbitrary positions), the two classes are very different for well-typed terms and expose interesting structure.

We could have a more explicit syntax, with term markers to indicate the various phase transitions that would remove the ambiguities, but we suspect that it would be much less pleasant to work with. In practice we will always manipulate *typed terms*, associated to their typing derivation, from which all necessary structural information can be obtained.

Remark 3.1. Our focused sequent calculus was cut-free in the literal sense of not having a cut rule. It is interesting to check that this focused natural deduction, and the focused λ-calculus, are also "cut-free" in the sense that the terms are irreducible. At first sight, this

Fig. 13. Typing rules for the focused λ-calculus

$$\textsc{FOCLC-LAM}\ \frac{\Gamma^{\mathsf{at}};\Sigma,x:P\vdash_{\mathsf{inv}} t:N\mid\emptyset}{\Gamma^{\mathsf{at}};\Sigma\vdash_{\mathsf{inv}}\lambda x.t:P\to N\mid\emptyset}\qquad \textsc{FOCLC-PAIR}\ \frac{\Gamma^{\mathsf{at}};\Sigma\vdash_{\mathsf{inv}} t_1:N_1\mid\emptyset\quad \Gamma^{\mathsf{at}};\Sigma\vdash_{\mathsf{inv}} t_2:N_2\mid\emptyset}{\Gamma^{\mathsf{at}};\Sigma\vdash_{\mathsf{inv}}(t_1,t_2):N_1\times N_2\mid\emptyset}$$

$$\textsc{FOCLC-CASE}\ \frac{\Gamma^{\mathsf{at}};\Sigma,x:Q_1\vdash_{\mathsf{inv}} t_1:N\mid P^{\mathsf{at}}\quad \Gamma^{\mathsf{at}};\Sigma,x:Q_2\vdash_{\mathsf{inv}} t_2:N\mid P^{\mathsf{at}}}{\Gamma^{\mathsf{at}};\Sigma,x:Q_1+Q_2\vdash_{\mathsf{inv}}\mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ x\to t_1\\ \sigma_2\ x\to t_2\end{array}\right.\ :N\mid P^{\mathsf{at}}}$$

$$\textsc{FOCLC-INV-FOC}\ \frac{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}\vdash_{\mathsf{foc}} f:(P^{\mathsf{at}}\mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}};\left\langle\Gamma^{\mathsf{at}\prime}\right\rangle^{+\mathsf{at}}\vdash_{\mathsf{inv}} f:\langle P^{\mathsf{at}}\rangle^{-\mathsf{at}}\mid Q^{\mathsf{at}}}$$

$$\textsc{FOCLC-CONCL-NEG}\ \frac{\Gamma^{\mathsf{at}}\vdash n\Downarrow X^-}{\Gamma^{\mathsf{at}}\vdash_{\mathsf{foc}} n:X^-}\qquad \textsc{FOCLC-LET-POS}\ \frac{\Gamma^{\mathsf{at}}\vdash n\Downarrow\langle P\rangle^-\qquad \Gamma^{\mathsf{at}};x:P\vdash_{\mathsf{inv}} t:\emptyset\mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}}\vdash_{\mathsf{foc}}\mathtt{let}\ x=n\ \mathtt{in}\ t:Q^{\mathsf{at}}}\qquad \textsc{FOCLC-VAR-NEG}\ \frac{}{\Gamma^{\mathsf{at}},x:N\vdash x\Downarrow N}$$

$$\textsc{FOCLC-VAR-POS}\ \frac{}{\Gamma^{\mathsf{at}},x:X^+\vdash x\Uparrow X^+}\qquad \textsc{FOCLC-FOC-INV}\ \frac{\Gamma^{\mathsf{at}};\emptyset\vdash_{\mathsf{inv}} t:N\mid\emptyset}{\Gamma^{\mathsf{at}}\vdash t\Uparrow\langle N\rangle^+}\qquad \textsc{FOCLC-CONCL-POS}\ \frac{\Gamma^{\mathsf{at}}\vdash p\Uparrow P}{\Gamma^{\mathsf{at}}\vdash_{\mathsf{foc}} p:P}$$

$$\textsc{FOCLC-PROJ}\ \frac{\Gamma^{\mathsf{at}}\vdash n\Downarrow N_1\times N_2}{\Gamma^{\mathsf{at}}\vdash\pi_i\ n\Downarrow N_i}\qquad \textsc{FOCLC-APP}\ \frac{\Gamma^{\mathsf{at}}\vdash n\Downarrow P\to N\qquad \Gamma^{\mathsf{at}}\vdash p\Uparrow P}{\Gamma^{\mathsf{at}}\vdash n\ p\Downarrow N}\qquad \textsc{FOCLC-INJ}\ \frac{\Gamma^{\mathsf{at}}\vdash p\Uparrow P_i}{\Gamma^{\mathsf{at}}\vdash\sigma_i\ p\Uparrow P_1+P_2}$$

seems to come from the restriction on the elimination judgment $\Gamma\vdash n\Downarrow N$, that elimination forms are only applied to neutrals and thus never create redexes. But this omits an important subtlety of the system, namely the use of a `let`-binding to represent (some) left focusing phases, $\mathtt{let}\ (x:P)=n\ \mathtt{in}\ t$.

We think that this construction should not be considered as a cut; in particular, we remark that if you substitute away all those `let`-bindings, the substituted term remains irreducible: a variable $(x:P)$ of strictly positive type will always be matched-upon by the next invertible phase, but it will always be substituted with a neutral term n so the resulting elimination will never become a redex. One can talk of this `let`-binding as an "irreducible cut". $*$

3.2.1 *Invertible commuting conversions*

We call *invertible commuting conversions*, noted by the relation $(\approx_{\mathtt{icc}})$, the equivalence relation generated by reordering two consecutive invertible rules.

Note that there are rules permuting invertible left-introduction and right-introduction rules, and rules that permute two left-introduction rules, but no rules permuting two right-introduction rules. There is no right-right permutation that would preserve typing; this is a consequence of the fact that this presentation of (focused) intuitionistic logic is single-succedent, in a multi-succedent system we could have right-right permutations.

Fig. 14. Invertible commuting conversions

$$\lambda y.\,\mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to u_1\\ \sigma_2\ z_2 \to u_2\end{array}\right. \quad \approx_{\mathtt{icc}} \quad \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to \lambda y.\,u_1\\ \sigma_2\ z_2 \to \lambda y.\,u_2\end{array}\right.$$

$$\left(t, \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to u_1\\ \sigma_2\ z_2 \to u_2\end{array}\right.\right) \quad \approx_{\mathtt{icc}} \quad \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to (t, u_1)\\ \sigma_2\ z_2 \to (t, u_2)\end{array}\right.$$

$$\left(\mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to u_1\\ \sigma_2\ z_2 \to u_2\end{array}\right., t\right) \quad \approx_{\mathtt{icc}} \quad \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to (u_1, t)\\ \sigma_2\ z_2 \to (u_2, t)\end{array}\right.$$

$$\mathtt{match}\ x'\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ y_1 \to u_1\\ \sigma_2\ y_2 \to u_2\end{array}\right.\\ \sigma_2\ z_2 \to t\end{array}\right.$$

$$\approx_{\mathtt{icc}} \quad \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ y_1 \to \mathtt{match}\ x'\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to u_1\\ \sigma_2\ z_2 \to t\end{array}\right.\\ \sigma_2\ y_2 \to \mathtt{match}\ x'\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to u_2\\ \sigma_2\ z_2 \to t\end{array}\right.\end{array}\right.$$

$$\mathtt{match}\ x'\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to t\\ \sigma_2\ z_2 \to \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ y_1 \to u_1\\ \sigma_2\ y_2 \to u_2\end{array}\right.\end{array}\right.$$

$$\approx_{\mathtt{icc}} \quad \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ y_1 \to \mathtt{match}\ x'\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to t\\ \sigma_2\ z_2 \to u_1\end{array}\right.\\ \sigma_2\ y_2 \to \mathtt{match}\ x'\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ z_1 \to t\\ \sigma_2\ z_2 \to u_2\end{array}\right.\end{array}\right.$$

3.2.2 *Defocusing into non-focused λ-terms*

We have glossed over the fact that focused λ-terms are not *quite* λ-terms as defined in Figure 2 (Terms of the lambda-calculus with sums), because they use the $\mathtt{let}\ x = t\ \mathtt{in}\ u$ form that is not formally part of the syntax.

In Figure 15 (Erasure of focusing $\lfloor t \rfloor_{\mathtt{foc}}$) we define the *erasure of focusing* operation $\lfloor - \rfloor_{\mathtt{foc}}$ that, for any focused λ-term t, gives its *erasure* as a simple λ-term $\lfloor t \rfloor_{\mathtt{foc}}$, obtained by replacing each $\mathtt{let}\ x = t\ \mathtt{in}\ u$ form by the substitution $u[t/x]$.

In Figure 10 (Polarity erasure), we established a translation from each focused sequent proof of a judgment on polarized formulas A into a non-focused sequent proof of a judgment on the corresponding depolarized formulas $\lfloor A \rfloor_{\pm}$. We can now state a similar result for focused natural deduction, strengthened with a correspondence between the proof terms themselves.

Lemma 3.2 (Type soundness of defocusing)**.**
The following implications hold:

$$\begin{array}{lcl}
\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} t : N \mid P^{\mathsf{at}} & \Longrightarrow & \lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm}, \lfloor \Sigma \rfloor_{\pm} \vdash \lfloor t \rfloor_{\mathtt{foc}} : (\lfloor N \rfloor_{\pm} \mid \lfloor P^{\mathsf{at}} \rfloor_{\pm}) \\
\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} f : P^{\mathsf{at}} & \Longrightarrow & \lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm} \vdash \lfloor f \rfloor_{\mathtt{foc}} : \lfloor P^{\mathsf{at}} \rfloor_{\pm} \\
\Gamma^{\mathsf{at}} \vdash n \Downarrow N & \Longrightarrow & \lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm} \vdash \lfloor n \rfloor_{\mathtt{foc}} : \lfloor N \rfloor_{\pm} \\
\Gamma^{\mathsf{at}} \vdash p \Uparrow P & \Longrightarrow & \lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm} \vdash \lfloor p \rfloor_{\mathtt{foc}} : \lfloor P \rfloor_{\pm}
\end{array}$$

Proof. By direct mutual induction on the premises. □

Fig. 15. Erasure of focusing $\lfloor t \rfloor_{\mathtt{foc}}$

$$
\begin{array}{lll}
\lfloor \lambda x.t \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & \lambda x.\, \lfloor t \rfloor_{\mathtt{foc}} \\
\lfloor (t_1, t_2) \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & (\lfloor t_1 \rfloor_{\mathtt{foc}}, \lfloor t_2 \rfloor_{\mathtt{foc}}) \\
\left\lfloor \mathtt{match}\ x\ \mathtt{with} \;\middle|\; \begin{array}{l} \sigma_1\ x \to u_1 \\ \sigma_2\ x \to u_2 \end{array} \right\rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & \mathtt{match}\ x\ \mathtt{with} \;\middle|\; \begin{array}{l} \sigma_1\ x \to \lfloor u_1 \rfloor_{\mathtt{foc}} \\ \sigma_2\ x \to \lfloor u_2 \rfloor_{\mathtt{foc}} \end{array} \\
\lfloor () \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & () \\
\lfloor \mathtt{absurd}(x) \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & \mathtt{absurd}(x)
\end{array}
$$

$$
\lfloor \mathtt{let}\ x = n\ \mathtt{in}\ t \rfloor_{\mathtt{foc}} \stackrel{\mathsf{def}}{=} \lfloor t \rfloor_{\mathtt{foc}} [\lfloor n \rfloor_{\mathtt{foc}} / x]
$$

$$
\begin{array}{lll}
\lfloor \pi_i\, t \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & \pi_i\, \lfloor t \rfloor_{\mathtt{foc}} \\
\lfloor n\ p \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & \lfloor n \rfloor_{\mathtt{foc}}\, \lfloor p \rfloor_{\mathtt{foc}} \\
\lfloor x \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & x \\
\lfloor \sigma_i\, t \rfloor_{\mathtt{foc}} & \stackrel{\mathsf{def}}{=} & \sigma_i\, \lfloor t \rfloor_{\mathtt{foc}}
\end{array}
$$

The following technical lemma gives a specification of defocusing translations will be useful to establish later results.

Lemma 3.3 (Composability of defocusing)**.**
Any subterm of $\lfloor t \rfloor_{\mathtt{foc}}$ is of the form

$$\lfloor u \rfloor_{\mathtt{foc}} [\lfloor n_1 \rfloor_{\mathtt{foc}} / x_1] [\lfloor n_2 \rfloor_{\mathtt{foc}} / x_2] \dots [\lfloor n_n \rfloor_{\mathtt{foc}} / x_n]$$

where u is a subterm of t, and the $\mathtt{let}\ x_i = n_i$ are the `let`-bindings in t that scope over u.

Proof. By induction on (the subterms of) t. □

3.3 Focusing completeness by big-step translation

Theorem 3.4 (Completeness of focusing)**.**
The focused λ-calculus is computationally complete: any well-typed lambda-term is $\beta\eta$-equivalent to (the `let`-substitution of) a focused λ-term.

Computational completeness could be argued to be folklore, or a direct adaptation of previous work on completeness of focusing: a careful reading of the elegant presentation of Simmons [2011] (or Laurent [2004] for linear logic) would show that its logical completeness argument in fact proves computational correctness. Without sums, it exactly corresponds to the fact that β-short η-long normal forms are computable for well-typed lambda-terms of the simply-typed calculus.

We introduce an explicit η-expanding, `let`-introducing transformation from β-normal forms to valid focused proofs for our system. Detailing this transformation also serves by building intuition for the computational completeness proof of the *saturating* focused logic in Figure 22 (Saturation translation), Section 6 (Canonicity of saturated proofs).

Proof (Computational completeness)**.** Let us recall that simply-typed lambda-calculus without fixpoints is strongly normalizing, and write $\mathsf{NF}_\beta(t)$ for the (full) β-normal form of t.

We define in Figure 16 an expansion relation $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ that turns any well-typed β-normal form $\lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm}, \lfloor \Sigma \rfloor_{\pm} \vdash t : \lfloor (N \mid Q^{\mathsf{at}}) \rfloor_{\pm}$ into a valid *focused* derivation $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} t' : N \mid Q^{\mathsf{at}}$.

Fig. 16. Translation into focused terms

$$\textsc{rew-inv-sum}\ \frac{\Gamma^{\mathsf{at}};\Sigma,x:P_1 \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(t[\sigma_1\,x/x]) \rightsquigarrow t'_1 : N \mid Q^{\mathsf{at}} \quad \Gamma^{\mathsf{at}};\Sigma,x:P_2 \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(t[\sigma_2\,x/x]) \rightsquigarrow t'_2 : N \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}};\Sigma,x:A_1+A_2 \vdash_{\mathsf{inv}} t \rightsquigarrow \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\,x \to t'_1\\ \sigma_2\,x \to t'_2\end{array}\right. : N \mid Q^{\mathsf{at}}}$$

$$\textsc{rew-inv-arrow}\ \frac{\Gamma^{\mathsf{at}};\Sigma,x:P \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(t\,x) \rightsquigarrow u' : N \mid \emptyset}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow \lambda x.u' : P \to N \mid \emptyset}$$

$$\textsc{rew-inv-prod}\ \frac{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(\pi_1\,t) \rightsquigarrow u'_1 : N_1 \mid \emptyset \qquad \Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(\pi_2\,t) \rightsquigarrow u'_2 : N_2 \mid \emptyset}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow (u'_1,u'_2) : N_1 \times N_2 \mid \emptyset}$$

$$\textsc{rew-inv-foc}\ \frac{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{foc}} t \rightsquigarrow t' : (P^{\mathsf{at}} \mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}};\left\langle \Gamma^{\mathsf{at}\prime} \right\rangle^{+\mathsf{at}} \vdash_{\mathsf{inv}} t \rightsquigarrow t' : \langle P^{\mathsf{at}} \rangle^{-\mathsf{at}} \mid Q^{\mathsf{at}}} \qquad \textsc{rew-foc-atom}\ \frac{\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow X^-}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} n \rightsquigarrow n' : X^-} \qquad \textsc{rew-foc-intro}\ \frac{\Gamma^{\mathsf{at}} \vdash t \rightsquigarrow t' \Uparrow P}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} t \rightsquigarrow t' : P}$$

$$\textsc{rew-foc-elim}\ \frac{\Gamma^{\mathsf{at}},x:P \vdash C[x] : Q^{\mathsf{at}} \qquad \Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow \langle P \rangle^- \qquad \Gamma^{\mathsf{at}};x:P \vdash_{\mathsf{inv}} C[x] \rightsquigarrow t' : \emptyset \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} C[n] \rightsquigarrow \mathtt{let}\ x = n'\ \mathtt{in}\ t' : Q^{\mathsf{at}}}$$

$$\textsc{rew-up-sum}\ \frac{\Gamma^{\mathsf{at}} \vdash t \rightsquigarrow t' \Uparrow P_i}{\Gamma^{\mathsf{at}} \vdash \sigma_i\,t \rightsquigarrow \sigma_i\,t' \Uparrow P_1+P_2} \qquad \textsc{rew-up-inv}\ \frac{\Gamma^{\mathsf{at}};\emptyset \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid \emptyset}{\Gamma^{\mathsf{at}} \vdash t \rightsquigarrow t' \Uparrow \langle N \rangle^+} \qquad \textsc{rew-up-var}\ \frac{(x:X^+) \in X^+}{\Gamma^{\mathsf{at}} \vdash x \rightsquigarrow x \Uparrow X^+}$$

$$\textsc{rew-down-var}\ \frac{(x:N) \in \Gamma^{\mathsf{at}}}{\Gamma^{\mathsf{at}} \vdash x \rightsquigarrow x \Downarrow N} \qquad \textsc{rew-down-pair}\ \frac{\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow N_1 \times N_2}{\Gamma^{\mathsf{at}} \vdash \pi_i\,n \rightsquigarrow \pi_i\,n' \Downarrow N_i}$$

$$\textsc{rew-down-arrow}\ \frac{\Gamma^{\mathsf{at}} \vdash t : P \qquad \Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow P \to N \qquad \Gamma^{\mathsf{at}} \vdash t \rightsquigarrow t' \Uparrow P}{\Gamma^{\mathsf{at}} \vdash n\,t \rightsquigarrow n'\,t' \Downarrow N}$$

We use four mutually recursive judgments, one for each judgment in the focused λ-calculus of Figure 13 (Typing rules for the focused λ-calculus): the invertible and focusing translations $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ and $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} t \rightsquigarrow t' : Q^{\mathsf{at}}$, and the negative and positive neutral translations $\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow N$ and $\Gamma^{\mathsf{at}} \vdash t \rightsquigarrow t' \Uparrow P$. For the two first judgments, the inputs are the context(s), source term, and translation type, and the output is the translated term. For the neutral judgments the translation type is an output – this reversal follows the usual bidirectional typing of normal forms.

Three distinct aspects of the translation need to be discussed:

1. Finiteness. It is not obvious that a translation derivation $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ *exists* for any $\lfloor \Gamma^{\mathsf{at}} \rfloor_\pm, \lfloor \Sigma \rfloor_\pm \vdash t : \lfloor (N \mid Q^{\mathsf{at}}) \rfloor_\pm$, because subderivations of invertible rules perform β-normalization of their source term, which may a priori make it grow without bounds. It could be the case that for certain source terms, there does not exist any finite derivation.
2. Partiality. As the rules are neither type- nor syntax-directed, it is not obvious that any input term, for example $\mathtt{match}\ t_1\ t_2\ \mathtt{with}\ \mid \sigma_1\,x_1 \to u_1 \mid \sigma_2\,x_2 \to u_2$, has a matching translation rule.

3. Non-determinism. The invertible rules are not quite typed-directed, and the REW-FOC-ELIM rule is deeply non-deterministic, as it applies for any neutral subterm of the term being translated – that is valid in the current typing environment. This non-determinism allows the translation to accept *any* valid focused derivation for an input term, reflecting the large choice space of when to apply the FOC-ELIM rule in backward focused proof search.

Totality The use of β-normalization inside subderivations precisely corresponds to the "unfocused admissibility rules" of Simmons [2011]. To control the growth of subterms in the premises of rules, we will use as a measure (or accessibility relation) the three following structures, from the less to the more important in lexicographic order:

- The (measure of the) types in the context(s) of the rewriting relation. This measure is strictly decreasing in the invertible elimination rule for sums, but increasing for the arrow introduction rule.
- The (measure of the) type of the goal of the rewriting relation. This measure is strictly decreasing in the introduction rules for arrow, products and sums, but increasing in REW-FOC-ELIM or neutral rules.
- The set of (measures of) translation judgments $\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow N$ for well-typed neutral subterms n of the translated term whose type N is of maximal mesaure.
  Note that while that complexity seems to increase in the premises of the judgment $\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow N$, this judgment should be read top-down: all the sub-neutrals of n already appear as subterms of the source t in the REW-FOC-ELIM application $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} t \rightsquigarrow ? : Q^{\mathsf{at}}$ that called $\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow ? \Downarrow N$.
  This measure is non-increasing in all non-neutral rules other than REW-FOC-ELIM, in particular the rules that require re-normalization (β-reduction or η-reduction may at best duplicate the occurrences of the neutral of maximal type, but not create new neutrals at higher types). In the sum-elimination rule, the neutral x of type $P_1 + P_2$ is shadowed by another neutral x of smaller type (P_1 or P_2). In the arrow rule, a new neutral $t\ x$ is introduced if t is already neutral, but then $t\ x : N$ is at a strictly smaller type than $t : P \to N$. In the product rule, new neutral $\pi_i\ t : N_i$ are introduced if $t : N_1 \times N_2$ is neutral, but again at strictly smaller types.
  Finally, this measure is strictly decreasing when applying REW-FOC-ELIM. Note that by typing we know that n, of shifted positive type $\langle P \rangle^-$, is not the whole term t, of positive or atomic type Q^{at} – ruling this case out is an advantage of using explicit shifts, compared to the presentation of Scherer and Rémy [2015].

This three-fold measures proves termination of $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow ? : N \mid Q^{\mathsf{at}}$ seen as an algorithm: we have proved that there are no infinite derivations for the translation judgments.

Partiality The invertible translation rules are type-directed; the neutral translation rules are directed by the syntax of the neutral source term. But the focusing translation rules are neither type- nor source-directed. We have to prove that one of those three rule applies for any term – assuming that the context is negative or atomic, and the goal type positive or atomic.

The term t either starts with a constructor (introduction form), a destructor (elimination form), or it is a variable; a constructor may be neither a λ or a pair, as we assumed the type is positive or atomic. It starts with a non-empty series of sum injections, followed by a negative or atomic term, we can use REW-FOC-INTRO. Otherwise it contains (possibly after some sum injections) a positive subterm that does not start with a constructor.

If it starts with an elimination form or a variable, it may or may not be a neutral term. If it is neutral, then one of the rules REW-FOC-ATOM (if the goal is atomic) or REW-FOC-INTRO (if the goal is strictly positive) applies. If it is not neutral (in particular not a variable), it has an elimination form applied to a subterm of the form $\texttt{match}\ t\ \texttt{with} \mid \begin{array}{l}\sigma_1\ x_1 \to u_1 \\ \sigma_2\ x_2 \to u_2\end{array}$; but then (recursively) either t is a (strictly positive) neutral, or of the same form, and the rule REW-FOC-ELIM is eventually applicable.

We have proved that for any well-typed $\lfloor\Gamma^{\mathsf{at}}\rfloor_\pm, \lfloor\Sigma\rfloor_\pm \vdash t : \lfloor(N \mid Q^{\mathsf{at}})\rfloor_\pm$, there exists a translation derivation $\Gamma^{\mathsf{at}};\Gamma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ for some t'.

Non-determinism The invertible rules may be applied in any order; this means that for any t' such that $\Gamma^{\mathsf{at}};\Gamma \vdash t \rightsquigarrow t' : A$, for any $t'' =_{\mathtt{icc}} t'$ we also have $\Gamma^{\mathsf{at}};\Gamma \vdash t \rightsquigarrow t'' : A$: a non-focused term translates to a full equivalence class of commutative conversions.

The rule REW-FOC-ELIM may be applied at will (as soon as the `let`-extruded neutral n is well-typed in the current context). Applying this rule eagerly would give a valid saturated focused deduction. Not enforcing its eager application allows (but we need not formally prove it) *any* $\beta\eta$-equivalent focused proof to be a target of the translation.

Validity We prove by immediate (mutual) induction that, if $\lfloor\Gamma^{\mathsf{at}}\rfloor_\pm, \lfloor\Gamma\rfloor_\pm \vdash t : \lfloor(N \mid Q^{\mathsf{at}})\rfloor_\pm$ holds, then the focusing translations are type-preserving:

- if $\Gamma^{\mathsf{at}};\Sigma;t \vdash_{\mathsf{inv}} t' \rightsquigarrow N : Q^{\mathsf{at}} \mid$ then $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t' : N \mid Q^{\mathsf{at}}$
- if $\Gamma = \emptyset$ and $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} t \rightsquigarrow t' : Q^{\mathsf{at}}$ then $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} t' : Q^{\mathsf{at}}$
- if $\Gamma = \emptyset$ and $\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow N$ then $\Gamma^{\mathsf{at}} \vdash n' \Downarrow N$
- if $\Gamma = \emptyset$ and $\Gamma^{\mathsf{at}} \vdash t \rightsquigarrow t' \Uparrow P$ then $\Gamma^{\mathsf{at}} \vdash t' \Uparrow P$

Soundness Finally, we prove that the translation preserves $\beta\eta$-equivalence. If $\lfloor\Gamma^{\mathsf{at}}\rfloor_\pm, \lfloor\Sigma\rfloor_\pm \vdash t : \lfloor(N \mid Q^{\mathsf{at}})\rfloor_\pm$ and $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$, then $t \approx_{\beta\eta} t'$, that is, $t \approx_{\beta\eta} \lfloor t'\rfloor_{\mathtt{foc}}$.

As for validity, this is proved by mutual induction on all judgments. The interesting cases are the invertible rules and the focusing elimination rule; all other cases are discarded by immediate induction.

The invertible rules correspond to an η-expansion step. For REW-INV-PROD, we have that $t \approx_\eta (\pi_1\ t, \pi_2\ t)$, and can thus deduce by induction hypothesis that $t \approx_{\beta\eta} (u'_1, u'_2)$. For REW-INV-ARROW, we have that $t \approx_\eta \lambda x.t$, and can thus deduce by induction hypothesis that

$t \approx_{\beta\eta} \lambda x.t'$. For REW-INV-SUM, let us write t as $C[x]$ with $x \notin C$, we have that

$$\begin{array}{lll} t & = & C[x : A + B] \\ & \approx_{\eta} & \texttt{match}\ x\ \texttt{with} \mid \begin{array}{l} \sigma_1\ x \to C[\sigma_1\ x] \\ \sigma_2\ x \to C[\sigma_2\ x] \end{array} \\ & = & \texttt{match}\ x\ \texttt{with} \mid \begin{array}{l} \sigma_1\ x \to t[\sigma_1\ x/x] \\ \sigma_2\ x \to t[\sigma_2\ x/x] \end{array} \\ & \approx_{\beta\eta} & \texttt{match}\ x\ \texttt{with} \mid \begin{array}{l} \sigma_1\ x \to t'_1 \\ \sigma_2\ x \to t'_2 \end{array} \qquad \text{(by induction hypothesis)} \end{array}$$

In the case of the rule REW-FOC-ELIM, the fundamental transformation is the `let`-binding that preserves $\beta\eta$-equivalence.

$$\begin{array}{lll} t & = & t[x/n][n/x] \\ & \approx_{\beta\eta} & \texttt{let}\ x = n\ \texttt{in}\ t[x/n] \\ & \approx_{\beta\eta} & \texttt{let}\ x = n'\ \texttt{in}\ t' \qquad \text{(by induction hypothesis)} \end{array}$$

Conclusion We have proved computational completeness of the focused logic: for any $\lfloor \Gamma^{\mathsf{at}} \rfloor_{\pm}, \lfloor \Gamma \rfloor_{\pm} \vdash t : \lfloor (N \mid Q^{\mathsf{at}}) \rfloor_{\pm}$, there exists some $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} t' : N \mid Q^{\mathsf{at}}$, such that $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_{\beta}(t) \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$, with $t \approx_{\beta\eta} \lfloor t' \rfloor_{\mathtt{foc}}$. □

4 Counting terms and proofs

In Section 2.1 (Natural deduction and sequent calculus) we presented a correspondence between well-typed terms in the simply-typed lambda-calculus, with (typing) derivations for the judgment $\Gamma \vdash t : A$, and natural-deduction proofs of propositional intuitionistic logic, written as (logic) derivations for judgments of the form $\Gamma \vdash A$. This correspondence is not one-to-one. In typing judgments $\Gamma \vdash t : A$, the context Γ is a mapping from free variables to their type. In logic derivations, the context Γ is a set of hypotheses; there is no notion of variable, and at most one hypothesis of each type in the set. This means, for example, that the following logic derivation

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}}{A \vdash A \to A}}{\emptyset \vdash A \to A \to A}$$

corresponds to two *distinct* programs, namely $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$. We say that those programs have the same *shape*, in the sense that the erasure of their typing derivation gives the same logic derivation – and they are the only programs of this shape.

Despite, or because, not being one-to-one, this correspondence is very helpful to answer questions about type systems. For example, the question of whether, in a given typing environment Γ, the type A is inhabited, can be answered by looking instead for a valid logic derivation of $\lfloor \Gamma \rfloor \vdash A$, where $\lfloor \Gamma \rfloor$ denotes the erasure of the mapping Γ into a set of hypotheses. In Section 1.9 (Termination) we have argued that only a finite number of different types need to be considered to find a valid proof (this is the case for propositional

logic because of the *subformula property*). As a consequence, there are finitely many set-of-hypothesis Δ, and the search space of sequents $\Delta \vdash B$ to consider during proof search is finite. This property is key to the termination of proof search algorithms for propositional logic. Note that it would not work if we searched typing derivations $\Gamma \vdash t : A$ directly: even if there are finitely many types of interest, the set of mappings from variables to such types is infinite.

In the present artile, we are interested in a different problem. Instead of knowing whether there *exists* a term t such that $\Gamma \vdash t : A$, we want to know whether this term is *unique* – modulo a given notion of program equivalence. Intuitively, this can be formulated as a search problem where search does not stop at the first candidate, but tries to find whether a second one (that is nonequivalent as a program) exists. In this setting, the technique of searching for logic derivations $\lfloor \Gamma \rfloor \vdash A$ instead is not enough, because a unique logic derivation may correspond to several distinct programs of this shape: summarizing typing environments as set-of-hypotheses loses information about (non)-unicity, it is not complete for unicity.

To better preserve this information, one could keep track of the number of times a hypothesis has been added to the context, representing contexts as *multisets* of hypotheses; given a logic derivation annotated with such counts in the context, we can precisely compute the number of programs of this shape. However, even for a finite number of types/formulas, the space of such multisets is infinite; this breaks termination arguments. A natural idea is then to *approximate* multisets by labeling hypotheses with 0 (not available in the context), 1 (added exactly once), or $\bar{2}$ (available two times *or more*); this two-or-more approximation has three possible states, and there are thus finitely many contexts annotated in this way.

The question we answer in this section is the following: is the two-or-more approximation correct? By correct, we mean that if the *precise* number of times a given hypothesis is available varies, but remains in the same approximation class, then the total number of programs of this shape may vary, but will itself remain in the same approximation class. A possible counter-example would be a logic derivation $\Delta \vdash B$ such that, if a given hypothesis $A \in \Delta$ is present exactly twice in the context (or has two free variables of this type), there is one possible program of this shape, but having three copies of this hypothesis would lead to several distinct programs.

Is this approximation correct? We found it surprisingly difficult to have an intuition on this question (guessing what the answer should be), and discussions with colleagues indicate that there is no obvious guess – people have contradictory intuitions on this. We show (Corollary 4.6 (Two-or-more approximation)) that this approximation is in fact correct.

4.1 Terms, types and derivations

We will manipulate several different systems of inference rules and discuss the relations between them: the type system, the logic, and inference systems annotated with counts (precise and approximated). To work uniformly over those various judgments, we will redefine their context structure as a mapping from types to some set. A set of hypothesis is now seen as a mapping from types to booleans, a multiset is a mapping to natural number,

and typing judgment is a mapping from types to sets of free variables (we inverse the usual association order).

In this section, we shall write $\mathbb{T}$ for the set of formulas or types defined in Figure 1 (Types of the simply-typed calculus). Besides the set of types $\mathbb{T}$, we will write $\mathbb{V}$ for the set of term variables $x, y, \ldots$, $\mathbb{B}$ for the set of booleans $\{1,0\}$, $\mathbb{N}$ for the (non-negative) natural numbers, and $\bar{2}$ for the set $\{0,1,\bar{2}\}$ used by the two-or-more approximation – note the bar on $\bar{2}$ to indicate the extra element $\bar{2}$ and avoid confusion with other notations for the booleans.

We write $E \to F$ for the set of functions from the set E to the set F, and $\mathsf{cardinal}(E)$ for the cardinal of the set E.

To make our discussion of *shapes* (of propositional judgments) precise and notationally convenient, we give a syntax for them in Figure 17 (Syntax of propositional shapes), instead of manipulating derivation trees directly. A shape is a variable-less proof-term; we will manipulate *explicitly typed* shapes, where variables have been replaced with their typing information.

Fig. 17. Syntax of propositional shapes

$$
\begin{array}{lll}
S,T & := & & \text{typed shapes} \\
& | & A,B,C & \text{axioms} \\
& | & \lambda A.S & \lambda\text{-abstraction} \\
& | & S\ T & \text{application} \\
& | & (S,T) & \text{pair} \\
& | & \pi_i\ S & \text{projection} \\
& | & \sigma_i\ S & \text{sum injection} \\
& | & \texttt{match}\ S\ \texttt{with} \left| \begin{array}{l} \sigma_1\ A_1 \to T_1 \\ \sigma_2\ A_2 \to T_2 \end{array} \right. & \text{sum destruction}
\end{array}
$$

Shapes correspond to logic derivations, that is, proof term without variables. Instead of a variable $x : A$, we just use the shape A. Similarly, the term $\lambda x.t$, where the bound variable x has type A, becomes the shape $\lambda A.S$, where S is the shape of t.

There is an immediate mapping from valid derivations of the usual logic judgment $\Gamma \vdash A$ into shapes, which suggests reformulating the judgment as $S :: \Gamma \vdash A$. Valid judgments are then in direct one-to-one mapping with their valid derivations – a principle all our different judgments will satisfy. A gramatically correct shape S may be invalid, that is, not correspond to any valid logic derivation $S :: \Gamma \vdash A$ – for example $\pi_1\ (\lambda A.B)$ is an invalid shape. We will only consider valid shapes, classified by the provability judgment $\Gamma \vdash A$, in the rest of this document.

We will manipulate the following judgments, each annotated with a propositional shape S:

- the provability judgment $S :: \Gamma \vdash A$, where the context Γ is in $\mathbb{T} \to \mathbb{B}$ – isomorphic to sets of types;
- the typing judgment $S :: E \vdash t : A$, where the context E is in $\mathbb{T} \to \mathscr{P}(\mathbb{V})$ – isomorphic to mappings from term variables to types;
- various counting judgments of the form $S :: \Phi \vdash_K A : a$ for a set K, where Φ is in $\mathbb{T} \to K$ – mapping from types to a multiplicity in K – and a, in K, represents the output count of the derivation.

The context annotations of all those judgments each have a (commutative) monoid structure $((+_M), 0_M)$ of a binary operation and its unit/neutral element: $((\vee), 0)$ for $\mathbb{B}$ and $((\cup), \emptyset)$ for $\mathscr{P}(\mathbb{V})$. Our counting sets K will even have the stronger algebraic structure of a *semiring*, we detail this in Section 4.2 (Counting terms in semirings). This is used to define common notations as follows.

The binary operation of the monoid can be lifted to whole context, and we will write Γ, Δ for the addition of contexts: $(\Gamma, \Delta)(A) = \Gamma(A) +_M \Delta(A)$. We will also routinely specify a context as a *partial* mapping from types to annotations, for example the singleton mapping $[A \mapsto a]$ (for some a in the codomain of the mapping); by this, we mean that the value for any other element of the domain is the neutral element 0_M. In particular, the notation Γ, A on sets of hypotheses corresponds to the addition $\Gamma, [A \mapsto 1]$ in $\mathbb{T} \to \mathbb{B}$, and the notation $\Gamma, x : A$ on mapping from variables to types corresponds to the addition $\Gamma, [A \mapsto \{x\}]$ in $\mathbb{T} \to \mathscr{P}(\mathbb{V})$.

Finally, for any function $f : E \to F$, we will write $\lfloor \_ \rfloor_f : \mathbb{T} \to E \to \mathbb{T} \to F$ the pointwise lifting of f on contexts: $\lfloor \Phi \rfloor_f(A) \stackrel{\text{def}}{=} f(\Phi(A))$. In particular, $\lfloor \_ \rfloor_{\neq \emptyset}$ erases typing environments $\mathbb{T} \to \mathscr{P}(\mathbb{V})$ into logic contexts $\mathbb{T} \to \mathbb{B}$, $\lfloor \_ \rfloor_{\neq 0}$ erases multiplicity-annotated contexts $\mathbb{T} \to \mathbb{N}$ into logic context $\mathbb{T} \to \mathbb{B}$, and $\lfloor \_ \rfloor_{\mathsf{cardinal}()}$ erases typing environments $\mathbb{T} \to \mathscr{P}(\mathbb{V})$ into multiplicity-annotated contexts $\mathbb{T} \to \mathbb{N}$.

The logic and typing judgments are defined in Figure 18 (Shaped provability judgment) and Figure 19 (Shaped typing judgment). In logic derivations we will simply write A for the singleton mapping $[A \mapsto 1]$. In typing derivations, we write $x : A$ for the singleton mapping $[A \mapsto \{x\}]$. Similarly, the variable freshness condition $x \notin E$ means $(\forall A \in \mathbb{T}, x \notin E(A))$.

Fig. 18. Shaped provability judgment

$$\frac{\Gamma(A) = 1}{A :: \Gamma \vdash A}$$

$$\frac{S :: \Gamma, A \vdash B}{\lambda A.S :: \Gamma \vdash A \to B} \qquad \frac{S :: \Gamma \vdash A \to B \qquad T :: \Gamma \vdash A}{S\,T :: \Gamma \vdash B}$$

$$\frac{S :: \Gamma \vdash A \qquad T :: \Gamma \vdash B}{(S,T) :: \Gamma \vdash A \times B} \qquad \frac{S :: \Gamma \vdash A_1 \times A_2}{\pi_i\,S :: \Gamma \vdash A_i}$$

$$\frac{S :: \Gamma \vdash A_i}{\sigma_i\,S :: \Gamma \vdash A_1 + A_2} \qquad \frac{S :: \Gamma \vdash A + B \qquad T_1 :: \Gamma, A_1 \vdash C \qquad T_2 :: \Gamma, A_2 \vdash C}{\texttt{match}\ S\ \texttt{with} \mid \sigma_1\,A_1 \to T_1 \mid \sigma_2\,A_2 \to T_2 \;\; :: \Gamma \vdash C}$$

Note that while changing the logic judgment from $\Gamma \vdash A$ to $S :: \Gamma \vdash A$ has the clear notational benefit of making valid judgments equivalent to derivations, this argument does not apply to changing the typing judgment from $E \vdash t : A$ to $S :: E \vdash t : A$, as the valid judgments $E \vdash t : A$ are already in one-to-one correspondence with their derivations; S adds some extra redundancy and could be computed from the triple (E, t, A) (or directly from t if we had used *explicitly typed* λ-terms). The benefit of $S :: E \vdash t : A$ is to let us talk very simply of the logical shape of a program, without having to define an additional erasure function from typing derivation to logical derivations: the set of programs of shape

Fig. 19. Shaped typing judgment

$$\frac{x \in E(A)}{A :: E \vdash x : A}$$

$$\frac{x \notin E \qquad S :: E, x : A \vdash t : B}{\lambda A.S :: E \vdash \lambda x.t : A \to B} \qquad \frac{S :: E \vdash t : A \to B \qquad T :: E \vdash u : A}{S\,T :: E \vdash t\,u : B}$$

$$\frac{S :: E \vdash t : A \qquad T :: E \vdash u : B}{(S,T) :: E \vdash (t,u) : A \times B} \qquad \frac{S :: E \vdash t : A_1 \times A_2}{\pi_i\,S :: E \vdash \pi_i\,t : A_i}$$

$$\frac{S :: E \vdash t : A_i}{\sigma_i\,S :: E \vdash \sigma_i\,t : A_1 + A_2}$$

$$\frac{S :: E \vdash t : A + B \quad x \notin E, y \notin E \quad T_1 :: E, x_1 : A_1 \vdash u_1 : C \quad T_2 :: E, x_2 : A_2 \vdash u_2 : C}{\texttt{match}\ S\ \texttt{with} \mid \begin{array}{l}\sigma_1\,A_1 \to T_1 \\ \sigma_2\,A_2 \to T_2\end{array} :: E \vdash \texttt{match}\ t\ \texttt{with} \mid \begin{array}{l}\sigma_1\,x_1 \to u_1 \\ \sigma_2\,x_2 \to u_1\end{array} : C}$$

S and type A in the environment E is simply defined as:

$$\{t \mid S :: E \vdash t : A\}$$

4.2 Counting terms in semirings

We are trying to connect two distinct ways of "counting" things about a logic derivation $S :: \Gamma \vdash A$. One is precise, it counts the number of distinct programs of shape S, and the other is the two-or-more approximation.

We generalize those two ways of counting as instances of a generic counting scheme that works for any *semiring* $(K, 0_K, 1_K, +_K, \times_K)$. A semiring is defined as a two-operation structure where $(0_K, +_K)$ and $(1_K, \times_K)$ are monoids, $(+_K)$ commutes and distributes over $(\times_K)$ (which may or may not commute), 0_K is a zero/absorbing element for $(\times_K)$, but $(+_K)$ and $(\times_K)$ need not have inverses[1]

The usual semiring is $(\mathbb{N}, 0, 1, +, *)$, and it will give the precise counting scheme. The 2-or-more semiring, which we will call $\bar{2}$, will correspond to the approximated scheme:

- its support is $\bar{2} = \{0, 1, \bar{2}\}$; 0_K is 0, 1_K is 1
- we define the addition by $1 +_K 1 = \bar{2}$ and $\bar{2} +_K 1 = \bar{2} +_K \bar{2} = \bar{2}$.
- we define the (commutative) multiplication by $\bar{2} \times_K \bar{2} = \bar{2}$.

Definition 4.1 Semiring notations**.**
Addition and multiplication can be lifted pointwise from K to $\mathbb{T} \to K$: for any $A \in \mathbb{T}$ we define $(\Phi +_K \Psi)(A) \stackrel{\text{def}}{=} \Phi(A) +_K \Psi(A)$ and $(\Phi \times_K \Psi)(A) \stackrel{\text{def}}{=} \Phi(A) \times_K \Psi(A)$.

Finally, we define a morphism from the semiring $\mathbb{N}$ to the semiring $\bar{2}$. Recall that $\varphi : K \to K'$ is a semiring morphism if $\varphi(0_K) = 0_{K'}$, $\varphi(1_K) = 1_{K'}$, $\varphi(a +_K b) = \varphi(a) +_{K'} \varphi(b)$ and $\varphi(a \times_K b) = \varphi(a) \times_{K'} \varphi(b)$.

[1] For a *ring* $(K, 0_K, 1_K, +_K, \times_K)$, $(+_K)$ must be invertible, so $\mathbb{Z}$ is a ring while $\mathbb{N}$ is only a semiring.

Definition 4.2 The 2-or-more morphism $\varphi_{\bar{2}}$.
We define $\varphi_{\bar{2}} : \mathbb{N} \to \bar{2}$ as follows:

$$\begin{cases} \varphi_{\bar{2}}(0) = 0 \\ \varphi_{\bar{2}}(1) = 1 \\ \varphi_{\bar{2}}(n) = \bar{2} & \text{if } n \geq 2 \end{cases}$$

$\varphi_{\bar{2}}$ is a semiring morphism.

Note that $(\mathbb{B}, 0, 1, \vee, \wedge)$ is also a semiring. For any semiring K, the function $(\_ \neq 0_K) : K \to \mathbb{B}$ (which we may also write $(\neq 0)$) is a semiring morphism.

4.2.1 Semiring-annotated derivations

Given a semiring K, we now define derivations $S :: \Phi \vdash_K A : a$ where Φ is a set of types labeled with counts in K (that is, an element of the product $\mathbb{T} \to K$ for some set Γ), and a is itself in K.

We construct those inference rules such that, when K is instantiated with the semiring of natural numbers $\mathbb{N}$, they really count the different programs of the same shape. For example, consider a logic derivation $S :: \Gamma \vdash B$ starting with a function elimination rule

$$\frac{S_1 :: \Gamma \vdash A \to B \qquad S_2 :: \Gamma \vdash A}{S_1\ S_2 :: \Gamma \vdash B}$$

A program of this shape is of the form $t\ u$, at type B; it can be obtained by pairing any possible program t (of shape S_1) at type $A \to B$ with any possible program u at type A (of shape S_2), so the number of possible applications is the product of the number of possible functions and possible arguments. Formally, we have that, for any typing environment E, writing $\mathsf{cardinal}(S)$ for the cardinal of the set S:

$$\{t_0 \mid S_1\ S_2 :: E \vdash B\} = \left\{ (t\ u) \mid \begin{array}{l} S_1 :: E \vdash t : A \to B, \\ S_2 :: E \vdash u : A \end{array} \right\} \qquad \mathsf{cardinal}(\{t_0 \mid S_1\ S_2 :: E \vdash B\}) = \mathsf{cardinal}(\{t \mid S_1 :: E \vdash t : A \to B\}) \times \mathsf{cardinal}(\{u \mid S_2 :: E \vdash u : A\})$$

This suggests the following semiring-annotated inference rule:

$$\frac{S_1 :: \Phi \vdash_K A \to B : a_1 \qquad S_2 :: \Phi \vdash_K B : a_2}{S_1\ S_2 :: \Phi \vdash_K B : a_1 \times_K a_2}$$

The other rules are constructed in the same way, and the full inference system is given in Figure 20 (Shaped counting judgment). We write $A : a$ for the singleton mapping $[A \mapsto a]$.

The identity rule says that if we have a different program variables of type A in our context, then using the variable rule of our typing judgment we can form a different programs. In particular, if A is absent from the context Φ, we have $A :: \Phi \vdash A : 0$. In the function-introduction rule, the number of programs of the form $\lambda x.t : A \to B$ is the number of programs $t : B$ in a context enriched with one extra variable of type A. The most complex rule is the sum elimination rule: the number of case-eliminations $(\texttt{match}\ t\ \texttt{with}\ \mid \sigma_1\ x_1 \to u_1 \mid \sigma_2\ x_2 \to u_2) : C$ is the product of the number of possible scrutinees $t : A + B$ and cases $u_1 : C$ and $u_2 : C$, with u_1 and u_2 built from one extra formal variable of type A or B accordingly.

Fig. 20. Shaped counting judgment

$$\frac{}{A :: \Phi \vdash_K A : \Phi(A)}$$

$$\frac{S :: \Phi, A : 1 \vdash_K B : a}{\lambda A.S :: \Phi \vdash_K A \to B : a} \qquad \frac{S_1 :: \Phi \vdash_K A \to B : a_1 \qquad S_2 :: \Phi \vdash_K A : a_2}{S_1\ S_2 :: \Phi \vdash_K B : a_1 \times a_2}$$

$$\frac{S_1 :: \Phi \vdash_K A : a_1 \qquad S_2 :: \Phi \vdash_K B : a_2}{(S_1, S_2) :: \Phi \vdash_K A \times B : a_1 \times a_2} \qquad \frac{S :: \Phi \vdash_K A_1 \times A_2 : a}{\pi_i\ S :: \Phi \vdash_K A_i : a}$$

$$\frac{S :: \Phi \vdash_K A_i : a}{\sigma_i\ S :: \Phi \vdash_K A_1 + A_2 : a}$$

$$\frac{S :: \Phi \vdash_K A_1 + A_2 : a_1 \qquad T_1 :: \Phi, A_1 : 1 \vdash_K C : a_2 \qquad T_2 :: \Phi, A_2 : 1 \vdash_K C : a_3}{\texttt{match}\ S\ \texttt{with} \mid \begin{array}{l} \sigma_1\ A_1 \to T_1 \\ \sigma_2\ A_2 \to T_2 \end{array} :: \Phi \vdash_K C : a_1 \times a_2 \times a_3}$$

We now precisely formulate the fact that the system $\vdash_{\mathbb{N}}$ really counts the number of programs of a given shape. Recall that $\lfloor \_ \rfloor_{\mathsf{cardinal}()} : (\mathbb{T} \to \mathscr{P}(\mathbb{V})) \to (\mathbb{T} \to \mathbb{N})$ erases a typing environment into a multiplicity-annotated context.

Lemma 4.1 (Cardinality count)**.**
For any typing environment $E \in \mathbb{T} \to \mathscr{P}(\mathbb{V})$, shape S and type A, the following is derivable:

$$S :: \lfloor E \rfloor_{\mathsf{cardinal}()} \vdash_{\mathbb{N}} A : \mathsf{cardinal}(\{t \mid S :: E \vdash t : A\})$$

Proof. By induction on the shape S, using the following equalities (obtained by inversion of the shape-directed typing judgment):

$$\{t_0 \mid A :: E \vdash t_0 : A\} = \{x \in E(A)\}$$

$$\{t_0 \mid \lambda A.S :: E \vdash t_0 : A \to B\} = \{\lambda x.t \mid S :: E, x : A \vdash t : A\}$$

$$\{t_0 \mid S\ T :: E \vdash t_0 : B\} = \left\{ t\ u \mid \begin{array}{l} S :: E \vdash t : A \to B \\ T :: E \vdash u : A \end{array} \right\}$$

$$\{t_0 \mid (S, T) :: E \vdash t_0 : A\} = \left\{ (t, u) \mid \begin{array}{l} S :: E \vdash t : A \\ T :: E \vdash u : B \end{array} \right\}$$

$$\{t_0 \mid \pi_i\ S :: E \vdash t_0 : A\} = \{\pi_i\ t \mid S :: E \vdash t : A\}$$

$$\{t_0 \mid \sigma_i\ S :: E \vdash t_0 : A\} = \{\sigma_i\ t \mid S :: E \vdash t : A\}$$

$$\{t_0 \mid \texttt{match}\ S\ \texttt{with} \mid \begin{array}{l} \sigma_1\ A_1 \to T_1 \\ \sigma_2\ A_2 \to T_2 \end{array} :: E \vdash t_0 : C\}$$

$$= \left\{ \texttt{match}\ t\ \texttt{with} \mid \begin{array}{l} \sigma_1\ x_1 \to u_1 \\ \sigma_2\ x_2 \to u_2 \end{array} \mid \begin{array}{l} S :: E \vdash t : A_1 + A_2 \\ T_1 :: E, x_1 : A_1 \vdash u_1 : C \\ T_2 :: E, x_2 : A_2 \vdash u_2 : C \end{array} \right\}$$

□

While the inference system $\vdash_{\mathbb{N}}$ corresponds to counting programs of a given shape (we formally claim and prove it below), other semirings indeed correspond to counting schemes of interest. The system $\vdash_{\bar{2}}$ corresponds to the "two-or-more" approximation, as can be exemplified by the following derivations:

$$\dfrac{\dfrac{\dfrac{}{(A) :: A : \bar{2} \vdash_{\bar{2}} A : \bar{2}}}{(\lambda A.A) :: A : 1 \vdash_{\bar{2}} A \to A : \bar{2}}}{(\lambda A.\lambda A.A) :: \emptyset \vdash_{\bar{2}} A \to A \to A : \bar{2}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{(A) :: A : \bar{2} \vdash_{\bar{2}} A : \bar{2}}}{(\lambda A.A) :: A : \bar{2} \vdash_{\bar{2}} A \to A : \bar{2}}}{(\lambda A.\lambda A.A) :: A : 1 \vdash_{\bar{2}} A \to A \to A : \bar{2}}}{(\lambda A.\lambda A.\lambda A.A) :: \emptyset \vdash_{\bar{2}} A \to A \to A \to A : \bar{2}}$$

When adding the hypothesis in the context in the context, its count goes from 0 to 1 – we have $\emptyset(A) = 0$ by definition. When adding it the second time, its count goes from 1 to $\bar{2}$. But on the third addition on the right, the count remains $\bar{2}$, as in the semiring $\bar{2}$ we have $\bar{2} + 1 = \bar{2}$.

The $\vdash_{\mathbb{B}}$ system intuitively corresponds to a system where the two possible counts are "zero" and "one-or-more", that is, it only counts inhabitation. There is a precise correspondence between this system and the logic derivation we formulated: derivations of the form $S :: \Gamma \vdash A : 1$ are in one-to-one correspondence with valid logic derivations $S :: \Gamma \vdash A$, and derivations $S :: \Gamma \vdash A : 0$ correspond to *invalid* logic derivations, where the shape S is valid but the context Γ lacks some hypothesis used in S. In particular, $\emptyset \vdash A : 0$ is always provable by immediate application of the variable rule.

Lemma 4.2 (Provability count)**.**
There is a one-to-one correspondence between logic derivations of $S :: \Gamma \vdash A$ and $\mathbb{B}$-counting derivations of $S :: \Gamma \vdash_{\mathbb{B}} A : 1$.

Proof. Immediate by induction on the shape S. □

4.2.2 Semiring morphisms determine correct approximations

The key reason why the two-or-more approximation is correct is that the mapping from $\mathbb{N}$ to $\bar{2}$ is a semiring morphism and, as such, preserves the annotation structure of counting derivations.

Theorem 4.3 (Morphism of derivations)**.**
If $\varphi : K \to K'$ is a semiring morphism and $S :: \Phi \vdash A : a$ holds, then $S :: \lfloor \Phi \rfloor_{\varphi} \vdash A : \varphi(a)$ also holds.

Proof. By induction on S.

$$A :: \Phi \vdash_K A : \Phi(A) \qquad \Rightarrow \qquad A :: \lfloor \Phi \rfloor_{\varphi} \vdash_{K'} A : \varphi(\Phi(A))$$

$$\dfrac{S :: \Phi, A : 1_K \vdash_K B : a}{\lambda A.S :: \Phi \vdash_K A \to B : a} \qquad \Rightarrow \qquad \dfrac{S :: \lfloor \Phi \rfloor_{\varphi}, A : 1'_K \vdash_{K'} B : \varphi(a)}{\lambda A.S :: \lfloor \Phi \rfloor_{\varphi} \vdash_{K'} A \to B : \varphi(a)}$$

To use our induction hypothesis, we needed the fact that $\lfloor\Phi\rfloor_\varphi, A : 1'_K$ is equal to $\lfloor\Phi, A : 1_K\rfloor_\varphi$; this comes from the fact that φ is a semiring morphism: $\varphi(1_K) = \varphi(1'_K)$ and $\varphi(a +_K b) = \varphi(a) +'_K \varphi(b)$, thus $\lfloor\Phi,\Psi\rfloor_\varphi = \lfloor\Phi\rfloor_\varphi, \lfloor\Psi\rfloor_\varphi$.

$$\frac{S_1 :: \Phi \vdash_K A \to B : a_1 \qquad S_2 :: \Phi \vdash_K A : a_2}{S_1\ S_2 :: \Phi \vdash_K B : a_1 \times a_2}$$

$$\Rightarrow \qquad \frac{S_1 :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A \to B : \varphi(a_1) \qquad S_2 :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A : \varphi(a_2)}{S_1\ S_2 :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} B : \varphi(a_1) \times \varphi(a_2)}$$

To conclude we then use the fact that $\varphi(a_1) \times \varphi(a_2) = \varphi(a_1 \times a_2)$.

$$\frac{S_1 :: \Phi \vdash_K A : a_1 \qquad S_2 :: \Phi \vdash_K B : a_2}{(S_1, S_2) :: \Phi \vdash_K A \times B : a_1 \times a_2}$$

$$\Rightarrow \qquad \frac{S_1 :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A : \varphi(a_1) \qquad S_2 :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} B : \varphi(a_2)}{(S_1, S_2) :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A \times B : \varphi(a_1) \times \varphi(a_2)}$$

$$\frac{S :: \Phi \vdash_K A_1 \times A_2 : a}{\pi_i\ S :: \Phi \vdash_K A_i : a} \quad \Rightarrow \quad \frac{S :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A_1 \times A_2 : \varphi(a)}{\pi_i\ S :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A_i : \varphi(a)}$$

$$\frac{S :: \Phi \vdash_K A_i : a}{\sigma_i\ S :: \Phi \vdash_K A_1 + A_2 : a} \quad \Rightarrow \quad \frac{S :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A_i : \varphi(a)}{\sigma_i\ S :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A_1 + A_2 : \varphi(a)}$$

$$\frac{S :: \Phi \vdash_K A + B : a_1 \qquad T_1 :: \Phi, A_1 : 1_K \vdash_K C : a_2 \qquad T_2 :: \Phi, A_2 : 1_K \vdash_K C : a_3}{\texttt{match}\ S\ \texttt{with} \mid \begin{array}{l}\sigma_1\ A_1 \to T_1 \\ \sigma_2\ A_2 \to T_2\end{array} \quad :: \Phi \vdash_K C : a_1 \times a_2 \times a_3}$$

$$\Rightarrow$$

$$\frac{S :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} A + B : \varphi(a_1) \quad T_1 :: \lfloor\Phi\rfloor_\varphi, A_1 : 1'_K \vdash_{K'} C : \varphi(a_2) \quad T_2 :: \lfloor\Phi\rfloor_\varphi, A_2 : 1'_K \vdash_{K'} C : \varphi(a_3)}{\texttt{match}\ S\ \texttt{with} \mid \begin{array}{l}\sigma_1\ A_1 \to T_1 \\ \sigma_2\ A_2 \to T_2\end{array} \quad :: \lfloor\Phi\rfloor_\varphi \vdash_{K'} C : \varphi(a_1) \times \varphi(a_2) \times \varphi(a_3)}$$

□

From there, it remains to point out that the right-hand side count is uniquely determined by the context multiplicity.

Lemma 4.4 (Determinism)**.**
If $S :: \Phi \vdash_K A : a$ *and* $S :: \Phi \vdash_K A : b$ *then* $a = b$.

Proof. Immediate by induction on derivations. Note that the fact that the judgments are indexed by the same shape S is essential here. □

Corollary 4.5 (Relation under morphism)**.**
If $\varphi : K \to K'$ *is a semiring morphism and* $\lfloor\Phi_1\rfloor_\varphi = \lfloor\Phi_2\rfloor_\varphi$*, then* $S :: \Phi_1 \vdash_K A : a_1$ *and* $S :: \Phi_2 \vdash_K A : a_2$ *imply* $\varphi(a_1) = \varphi(a_2)$

Proof. By Theorem 4.3 (Morphism of derivations), we have $S :: \lfloor \Phi_1 \rfloor_\varphi \vdash_{K'} A : \varphi(a_1)$ and $S :: \lfloor \Phi_2 \rfloor_\varphi \vdash_{K'} A : \varphi(a_2)$. If $\lfloor \Phi_1 \rfloor_\varphi = \lfloor \Phi_2 \rfloor_\varphi$ we can conclude by Lemma 4.4 (Determinism) that $\varphi(a_1) = \varphi(a_2)$. □

Corollary 4.6 (Two-or-more approximation)**.**
The 2-or-more approximation is correct to decide unicity of inhabitants of a given shape S. If $\lfloor E_1 \rfloor_{\varphi_{\bar{2}} \cdot \mathsf{cardinal}()} = \lfloor E_2 \rfloor_{\varphi_{\bar{2}} \cdot \mathsf{cardinal}()}$, then

$$\varphi_{\bar{2}}(\mathsf{cardinal}(\{t \mid S :: E_1 \vdash t : A\})) = \varphi_{\bar{2}}(\mathsf{cardinal}(\{t \mid S :: E_2 \vdash t : A\}))$$

Proof. By Lemma 4.1 (Cardinality count), counting the inhabitants corresponds to the system $\vdash_{\mathbb{N}}$, so we have

$$S :: \lfloor E_1 \rfloor_{\mathsf{cardinal}()} \vdash_{\mathbb{N}} A : \mathsf{cardinal}(\{t \mid S :: E_1 \vdash t : A\})$$

$$S :: \lfloor E_2 \rfloor_{\mathsf{cardinal}()} \vdash_{\mathbb{N}} A : \mathsf{cardinal}(\{t \mid S :: E_2 \vdash t : A\})$$

The result then directly comes from the previous corollary, given that $\varphi_{\bar{2}}$ is a semiring morphism. □

4.2.3 n-or-more logics

The result can be extended to any "n-or-more" approximation scheme given by the semiring $\bar{n}$ and semiring morphism $\varphi_{\bar{n}} : \mathbb{N} \to \bar{n}$ defined as follows (assuming $n \geqslant 1$):

$$\bar{n} \stackrel{\text{def}}{=} \{0, 1, \ldots, n-1, n\} \qquad 0_{\bar{n}} \stackrel{\text{def}}{=} 0 \qquad 1_{\bar{n}} \stackrel{\text{def}}{=} 1$$

$$(a +_{\bar{n}} b) \stackrel{\text{def}}{=} \min(a +_{\mathbb{N}} b, n) \qquad (a \times_{\bar{n}} b) \stackrel{\text{def}}{=} \min(a \times_{\mathbb{N}} b, n)$$

To check that $\varphi_{\bar{n}}$ is indeed a morphism, one needs to remark that having either $a \geqslant n$ or $b \geqslant n$ implies $(a +_{\mathbb{N}} b) \geqslant n$ and, if a and b are non-null, $(a *_{\mathbb{N}} b) \geqslant n$.

5 Saturation logic for canonicity

Equipped with the understanding of program equivalence acquired through our study of focusing (Section 3), it is now time to go back to our original question: which types have a unique inhabitant? In this section, we provide a decision algorithm for this question in the context of simply-typed lambda-calculus with products and unit types, sums and empty types.

With the technical ideas that we have built throughout this document, the idea can be described in a concise way: we define a variant of focusing for intuitionistic natural deduction that is canonical and has a structural presentation which makes goal-directed proof search possible in this subsystem. The key idea is to use *saturation* in non-invertible phases, that is a complete forward search for left focused phases, until reaching a saturated state (all deducible strict positives have been deduced), then doing right focus and continuing with goal-directed (backward) search. We also need a precise notion of saturation to ensure both completeness and termination.

In Section 5.2 (A saturating focused type system), we will present the typing rule of our *saturated* focused type system, which can be understood as a variant of multi-focused

λ-calculus. This system serves as a declarative *specification* of saturation, but it does not suffice to obtain an algorithm as its goal-directed proof search process is not always terminating. It Section 7 (Unique inhabitation algorithm) we introduce an algorithmic restriction of the system in which proof search is terminating and gives a deduction procedure for unicity – and prove its correctness.

5.1 Introduction to saturation for unique inhabitation

The rules of program equivalence for the full, pure simply-typed λ-calculus were given in Figure 4 ($\beta\eta$-equivalence for the simply-typed lambda-calculus). Of particular interest is the distinction between the *weak eta*-rule ($\approx_{\texttt{weak}\,\eta}$) and the *strong* η-rule ($\approx_\eta$) for sums

$$(t : A_1 + A_2) \;\triangleright_{\texttt{weak}\,\eta}\; \texttt{match}\; t\; \texttt{with} \;\Big|\; \begin{array}{l} \sigma_1\, y_1 \to \sigma_1\, y_1 \\ \sigma_2\, y_2 \to \sigma_2\, y_2 \end{array}$$

$$\forall C[x], \quad C[t : A_1 + A_2] \;\triangleright_\eta\; \texttt{match}\; t\; \texttt{with} \;\Big|\; \begin{array}{l} \sigma_1\, y_1 \to C[\sigma_1\, y_1] \\ \sigma_2\, y_2 \to C[\sigma_2\, y_2] \end{array}$$

5.1.1 Non-canonicity of simple focusing: splitting points

Simple focusing, as described in Section 3 (Focused λ-calculus), classifies terms that are also called β-short η-long normal forms, but they in fact correspond to *weak* β-short $\texttt{weak}\,\eta$-long normal forms. In the purely negative fragment (no sums and empty types), the weak and strong η-rules coincide, so focusing captures the right notion of normal form and is a canonical system.

Focusing fails to be canonical when positives are added; consider for example the following goal:

$$x : Z_1^+, f : Z_1^+ \to X^+ + X^+, g : X^+ \to Y^- \vdash ? : Z_0^+ + (Y^- \times Y^-)$$

The three following programs are equivalent, yet are syntactically distinct valid focused terms (normal forms). Note that there are other possible ways to write a well-typed β-short $\texttt{weak}\,\eta$-long normal form at this type – but they are all equivalent.

$$\sigma_1 \left(\begin{array}{l} \texttt{let } y = f\ x \texttt{ in match } y \texttt{ with} \left| \begin{array}{l} \sigma_1\ z_1 \to g\ z_1 \\ \sigma_2\ z_2 \to g\ z_2 \end{array} \right. \\ , \\ \texttt{let } y = f\ x \texttt{ in match } y \texttt{ with} \left| \begin{array}{l} \sigma_1\ z_1 \to g\ z_1 \\ \sigma_2\ z_2 \to g\ z_2 \end{array} \right. \end{array} \right)$$

$$\texttt{let } y = f\ x \texttt{ in match } y \texttt{ with} \left| \begin{array}{l} \sigma_1\ z_1 \to \sigma_1\ (g\ z_1, g\ z_1) \\ \sigma_2\ z_2 \to \sigma_1\ (g\ z_2, g\ z_2) \end{array} \right.$$

$$\texttt{let } y = f\ x \texttt{ in match } y \texttt{ with} \left| \begin{array}{l} \sigma_1\ z_1 \to \sigma_1\ (g\ z_1, g\ z_1) \\ \sigma_2\ z_2 \to \texttt{let } y' = f\ x \texttt{ in } \left(\texttt{match } y' \texttt{ with} \left| \begin{array}{l} \sigma_1\ z' \to (g\ z_2, g\ z'_1) \\ \sigma_2\ z' \to (g\ z_2, g\ z'_2) \end{array} \right. \right) \end{array} \right.$$

We can prove that these three terms are $\beta\eta$-equivalent, but an informal explanation also helps following these examples.

The first two terms perform the same splitting (binding then pattern-matching) of $f\ x$, but one does it once before building the pair, and the other does it separately in each branch of the pair. Because we assume that f is a pure function[2], it must returns the same thing in each element of the pair, and the final results are thus identical. To prove that the two terms are identical, it suffices to extrude the binding $\texttt{let } y = f\ x \texttt{ in } ?$ from the pair elements in the second case, and extrude the pattern-matching as well. (In terms of focusing, we are suggesting to permute two independent non-invertible phases, the let binding and the sum injection; pair construction and variable case-split are implicitly moved around as well, being the invertible phases that systematically follow each non-invertible phase.)

The third term is slightly different, as instead of performing two splits in two parallel branches (as the first term), it performs two splits in sequence, with the second split being in scope of the (right branch of) the first split. The reasoning to informally justify the equivalence with the second term is that, at the time when y' (that is $f\ x$) is matched over, we *already know* the value of $f\ x$: if this branch has been taken, it is because $f\ x$ is equal to $\sigma_2\ z_2$ for some z_2 that is currently in scope. We can thus replace y' with $\sigma_2\ z_2$ in the nested pattern-matching. Performing a β-reduction step then gives exactly the second term.

Remark 5.1. Note that this example of non-canonicity of the focusing discipline would break if we replaced the context hypothesis $f : Z_1^+ \to X^+ + X^+$ by a mere sum $x_0 : X^+ + X^+$. Indeed, the focusing discipline would recognize it as a positive in context, to be split before the start of the first non-invertible phase, and this would give a unique focused derivation.

By wrapping this positive under a (negative) function type, we make it out of reach from the simple focusing discipline. In a system with explicit shifts (here we assumed minimal shifts), see Section 2.4.1 (Explicit shifts), we could also simply put the sum under a double-shift delay. ∗

[2] Note that non-termination plus lazy pairs would already allow to observe a difference between those two terms.

5.1.2 Canonicity for term equivalence: extrusion

These examples allow to understand where non-canonicity comes from. We have (focused) terms that are syntactically distinct but semantically equivalent. They differ by the place, and the number of times, on which a particular subterm (here $g\ ()$) of sum type is bound and matched over. We need to quotient over this source of difference, by imposing a unique place at which those subterms should be bound and matched, that can be decided during goal-directed proof search.

Definition 5.1 Splitting**.**
Splitting a (sub)term is pattern-matching over it, possibly after having bound it to a variable name. We call *splitting point* the place where the term is bound and pattern-matched.

In the work on deciding equivalence of λ-terms with sums, the solution is to move each subterm of sum type as high/early as possible in the term, to split them there – and merge equal subterms that end up being split at the same place. This is clearly visible in the rewriting-based work of Ghani [1995] and Lindley [2007], but it is also perceptible in the normalization-by-evaluation work [Balat, Di Cosmo, and Fiore, 2004, Altenkirch, Dybjer, Hofmann, and Scott, 2001]. For example, Balat, Di Cosmo, and Fiore [2004] define a notion of quasi-normal form for terms with sums, with a side-condition (Condition (B), page 5) says that a split term must become ill-typed if we move it before the latest series of variable bindings (in fact, the latest invertible phase). This is a way to guarantee that subterms of sum type are split as early as possible in the term.

Those procedures proceed by moving subterms (invertible and non-invertible phases) around, so in particular they rely on the presence of one initial term to normalize, or two initial terms to compare: it makes sense to search, for example, for all neutral subterms n of the initial term that are valid at some possible splitting point (the start of a non-invertible phase) and extrude them.

5.1.3 Canonicity for term enumeration: saturation

On the contrary, the problem of unique inhabitation requires enumerating proof terms out of the blue, without starting from a pre-existing proof term to transform. When reaching a potential splitting point (the start of a non-invertible phase) during term enumeration (goal-directed proof search), there are no subterms to collect and extrude, only recursive sub-goals that have not yet be filled. This crucial difference leads us to taking a quite different (yet related) approach.

Another way to see the situation of term normalization or equivalence is that the initial term serves as an oracle to answer the following question: “which terms should we split now, that will be *useful* to the rest of the proof?”. Useful sub-terms are those that it is necessary to bind now to build a term equivalent (computationally) to the initial term. We can also see them as an over-approximation of a set of terms that we must split to find a proof at all; we use the initial term as a base of “hints” (its subterms) to find a proof of the desired judgment – a proof with the particular property of being equivalent to the initial term we started from.

To move from term normalization or comparison to term enumeration, our idea is to drop the usefulness criterion. We cannot know in advance, at this stage of the proof search,

without having searched for the sub-goals, which terms of positive types will actually be used by the proof(s) that we will find, but we can split *all of them*. Then we start again enumerating terms of the desired type, in a context extended with (the decomposition of) all those freshly split sums. Some splits will prove useful to build all terms of our enumeration, some will only be used by some of those distinct terms, and some will not be used at all. This is the idea of *saturation*.

What exactly do we mean by "all terms of positive types"? We are only interested in the terms of positive type whose value is unknown, because they come from the (unknown) formal variables in the typing context of the search. Those are the neutral terms n, m that are obtained by taking a variable x of the context, and applying pair projections $\pi_i\ n$ or function applications $n\ p$ on it until we reach a result of sum type. One can think of a neutral term $n : A_1 + A_2$ as a specific "observation" of the richly-typed value of its head variable x; saturation, which splits all those neutral terms, is the process of learning everything we can learn from our context by these observations, before continuing the proof search.

Saturation should come before any committing choice. If we delay these observations, and first perform a non-invertible introduction step, we can get in a dead search branch, because we do not have enough information at hand to know which choice to make (consider again the proofs of $f : () \to X + Y \vdash ? : Y + X$). This justifies performing saturation "as early as possible", or at least before making any mistake, that is before the start of each non-invertible (right) introduction phase.

In the rest of this section, we will see

- A structural presentation of a focused *saturating* type system, which encapsulates this idea of saturation as a typing rule.
- A simple mechanism to avoid splitting the same neutral of positive type during two successive saturation phases, to preserve canonicity; Section 5.2 (A saturating focused type system).
- Various methods to avoid saturating on infinitely many distinct neutrals, or repeating saturation infinitely long before reaching a stable state, to preserve termination; Section 7 (Unique inhabitation algorithm).

5.1.4 An example of saturation

Let us consider our previous example showing that focusing alone is not canonical:

$$x : Z_1^+, f : Z_1^+ \to \left\langle X^+ + X^+ \right\rangle^-, g : X^+ \to Y^- \vdash ? : Z_0^+ + \left\langle Y^- \times Y^- \right\rangle^+$$

The context Γ^{at} is negative or atomic, and the goal is positive. In our focused logic, we would start by looking for all n of positive type such that $\Gamma^{\mathsf{at}} \vdash n \Downarrow P$. There is exactly one such $(n : P)$ in this context, it is $(f\ x : X^+ + X^+)$. Saturation would thus start with the following phase:

$$\texttt{let}\ y = f\ x\ \texttt{in}\ ?$$

and the following invertible phase would be

$$\texttt{match}\ y\ \texttt{with} \mid \begin{array}{l} \sigma_1\ y \to ? \\ \sigma_2\ y \to ? \end{array}$$

leaving us with two subgoals, each with a context of the form

$$x : Z_1^+, f : Z_1^+ \to \langle X^+ + X^+ \rangle^-, g : X^+ \to Y^-, y : X^+$$

As the two goals are identical, we will focus here on one of them, the other proceeds in the exact same way.

At this point, a new focusing phase begins, looking for all negative neutrals with a positive type. But the addition of a X^+ in the context did not give us any way to deduce a new neutral: we can still build $f\ x$, but we have already saturated over it. At this point, saturation stops, and our algorithm tries all possible (non-invertible) rules to prove our goal.

In our case the goal is a strict positive (rather than a negative atom) so we look for all possible positive neutrals p at this type. Proof search will thus attempt to use a term of the form $\sigma_1\ ?$, and prove the remaining goal Z^+, and also to use a term of the form $\sigma_2\ ?$ and prove the remaining goal $\langle Y^- \times Y^- \rangle^+$. In the first case Z_0^+, search fails immediately: a strictly positive neutral at this type is a variable, and there is none in the context. In the second case, $\langle Y^- \times Y^- \rangle^+$, the focused introduction phase stops at the shift, and a new invertible phase starts.

The invertible phase for $Y^- \times Y^-$ creates two identical goals, so we can focus on any of them, trying to prove Y^-. A new saturation phase start, but there is still no new negative neutral of positive type in sight. The search algorithm then tries to prove the goal, and because we have a negative atom it looks for a negative neutral at this type. All negative neutrals of type Y^- in this context are of the form $g\ ?$, with a subgoal of type X^+, to be filled by a positive neutral; there is exactly one positive neutral at this type, namely y; because there is only one choice, we know that this goal has a unique inhabitant.

This leaves us with a unique program of this type, namely

$$\texttt{let}\ y = f\ x\ \texttt{in}\ \texttt{match}\ y\ \texttt{with} \left| \begin{array}{l} \sigma_1\ y \to (g\ y, g\ y) \\ \sigma_2\ y \to (g\ y, g\ y) \end{array} \right.$$

Positive variant We could also consider a variant of this goal with a different choice of atom polarities – there are other possible choices but this one is interesting.

$$x : Z_1^+, f : Z_1^+ \to \langle X^+ + X^+ \rangle^-, g : X^+ \to \langle Y^+ \rangle^- \vdash ? : Z^+ + \left\langle \langle Y^+ \rangle^- \times \langle Y^+ \rangle^- \right\rangle^+$$

As before, the first saturation step has exactly one neutral to introduce, $\texttt{let}\ y = f\ x\ \texttt{in}\ ?$, with $y : X^+ + X^+$. But, after the following invertible phase $\texttt{match}\ y\ \texttt{with}\ \mid \sigma_1\ y \to ? \mid \sigma_2\ y \to ?$, saturated proof search differs from the previous one as a new positive becomes provable, $(g\ y : Y^+)$. This judgment is still uniquely inhabited, but with a different saturated proof term:

$$\texttt{let}\ y = f\ x\ \texttt{in}\ \texttt{match}\ y\ \texttt{with} \left| \begin{array}{l} \sigma_1\ y \to \texttt{let}\ z = g\ y\ \texttt{in}\ (z, z) \\ \sigma_2\ y \to \texttt{let}\ z = g\ y\ \texttt{in}\ (z, z) \end{array} \right.$$

5.2 A saturating focused type system

In Figure 21 (Cut-free saturating focused type system (in natural deduction style)) we give the full typing rules for our *saturating* focused λ-calculus. They share many similarities

with the focused λ-calculus of Section 3 (Focused λ-calculus), with several changes that we will describe in detail. The calculus is described by four mutually recursive judgments, whose role we will detail in this section.

- The invertible judgment $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t : N \mid Q^{\mathsf{at}}$, which is very close to the invertible judgment $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t : N \mid Q^{\mathsf{at}}$ of the focused λ-calculus.
- The saturating judgment $\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} f : Q^{\mathsf{at}}$ is where most of the novelty lies, in particular the SAT rule that enforces saturating. It is inspired by the "choice of focusing" judgment $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} f : Q^{\mathsf{at}}$ of the simple focused λ-calculus, but behaves in a different way.
- The focused introduction and elimination judgments $\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} p \Uparrow P$ and $\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} n \Downarrow N$, which are identical to the corresponding judgments of the focused λ-calculus.

In addition, the type system is parametrized by a family of selection functions $\mathtt{Select}_{\Gamma^{\mathsf{at}}}(\_)$; for any negative or atomic context Γ^{at} and positive or atomic goal type P^{at}, it takes as input a (potentially infinite) set of neutrals of positive type (n, P) and returns a finite subset of its input. This parameter represents choices that can be made by an algorithm derived from this logic.

5.2.1 *Invertible phase*

Our approach to decide unique inhabitation is to design a generic term enumeration procedure that only enumerates distinct terms (no duplicates) and enumerate distinct terms lazily. Given such a procedure, it suffices to enumerate at most two term to decide unicity.

How can we enumerate all distinct values of type $(A_1 \times A_2)$? Well, we know from the η-equivalence of products $(t : A_1 \times A_2) = (\pi_1\, t, \pi_2\, t)$ that any term of type $A_1 \times A_2$ is equivalent to some pair (t, u) of some $t : A_1$ and $u : B_2$, and it thus suffices to enumerate all distinct values of A_1, of A_2, and take their (lazily enumerated) cartesian product. Similarly, to enumerate all distinct values of type $(A \to B)$, it suffices to enumerate B in an environment extended with a formal variable $x : A$, and return $\lambda x.\, t$ for each distinct t in B.

The invertible judgment $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t : N \mid Q^{\mathsf{at}}$ corresponds to this enumeration reasoning. In term of focusing, we say that the λ-introduction rule is "invertible", which means here that we can always assume terms of function types are built using it, without losing any generality. Same things for product – and unit, obviously.

A novelty of the focusing-based point of view is that this "without loss of generality" reasoning not only applies to terms with an invertible *constructor* (the negative types), but also terms that can be *destructed* without any loss of generality (the positive types). If we have a variable of sum type in the context, any possible well-typed term can be rewritten to begin with a case-split on this variable.

SINV-CASE

$$\frac{\begin{array}{c}\Gamma^{\mathsf{at}};\Sigma, x : P_1 \vdash_{\mathsf{sinv}} t_1 : N \mid Q^{\mathsf{at}} \\ \Gamma^{\mathsf{at}};\Sigma, x : P_2 \vdash_{\mathsf{sinv}} t_2 : N \mid Q^{\mathsf{at}}\end{array}}{\Gamma^{\mathsf{at}};\Sigma, x : P_1 + P_2 \vdash_{\mathsf{sinv}} \mathtt{match}\ x\ \mathtt{with} \mid \begin{array}{l}\sigma_1\, x \to t_1 \\ \sigma_2\, x \to t_2\end{array} : N \mid Q^{\mathsf{at}}}$$

When reading this rule, one should first read the rule without the terms, or with the terms replaced by not-yet-filled holes, and think of the goal-directed search process: whenever

Fig. 21. Cut-free saturating focused type system (in natural deduction style)

$$\textsc{SINV-LAM}\ \frac{\Gamma^{\mathsf{at}};\Sigma,x:P\vdash_{\mathit{sinv}} t:N\mid\emptyset}{\Gamma^{\mathsf{at}};\Sigma\vdash_{\mathit{sinv}}\lambda x.t:P\to N\mid\emptyset}\qquad \textsc{SINV-PAIR}\ \frac{\Gamma^{\mathsf{at}};\Sigma\vdash_{\mathit{sinv}} t_1:N_1\mid\emptyset\quad \Gamma^{\mathsf{at}};\Sigma\vdash_{\mathit{sinv}} t_2:N_2\mid\emptyset}{\Gamma^{\mathsf{at}};\Sigma\vdash_{\mathit{sinv}}(t_1,t_2):N_1\times N_2\mid\emptyset}$$

$$\textsc{SINV-CASE}\ \frac{\Gamma^{\mathsf{at}};\Sigma,x:P_1\vdash_{\mathit{sinv}} t_1:N\mid Q^{\mathsf{at}}\quad \Gamma^{\mathsf{at}};\Sigma,x:P_2\vdash_{\mathit{sinv}} t_2:N\mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}};\Sigma,x:P_1+P_2\vdash_{\mathit{sinv}}\mathtt{match}\ x\ \mathtt{with}\ \mid \sigma_1\ x\to t_1\mid \sigma_2\ x\to t_2\ :N\mid Q^{\mathsf{at}}}$$

$$\textsc{SINV-SAT}\ \frac{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime}\vdash_{\mathsf{sat}} f:(P^{\mathsf{at}}\mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}};\left\langle\Gamma^{\mathsf{at}\prime}\right\rangle^{+\mathsf{at}}\vdash_{\mathit{sinv}} f:\left\langle P^{\mathsf{at}}\right\rangle^{-\mathsf{at}}\mid Q^{\mathsf{at}}}$$

$$\textsc{SAT}\ \frac{(\bar n,\bar P)\stackrel{\mathsf{def}}{=}\mathtt{Select}_{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}}(\{(n,P)\mid(\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}\vdash_{\mathsf{s}} n\Downarrow\langle P\rangle^-)\wedge\exists x\in\Gamma^{\mathsf{at}\prime},x\in n\})\quad \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime};\bar x:\bar P\vdash_{\mathit{sinv}} t:\emptyset\mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime}\vdash_{\mathsf{sat}}\mathtt{let}\ \bar x=\bar n\ \mathtt{in}\ t:Q^{\mathsf{at}}}$$

$$\textsc{SAT-UP}\ \frac{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} p\Uparrow P}{\Gamma^{\mathsf{at}};\emptyset\vdash_{\mathsf{sat}} p:P}\qquad \textsc{SAT-DOWN}\ \frac{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} n\Downarrow X^-}{\Gamma^{\mathsf{at}};\emptyset\vdash_{\mathsf{sat}} n:X^-}$$

$$\textsc{SAT-UP-SINV}\ \frac{\Gamma^{\mathsf{at}};\emptyset\vdash_{\mathit{sinv}} t:N\mid\emptyset}{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} t\Uparrow\langle N\rangle^+}\qquad \textsc{SAT-UP-ATOM}\ \frac{}{\Gamma^{\mathsf{at}},x:X^+\vdash_{\mathsf{s}} x\Uparrow X^+}\qquad \textsc{SAT-DOWN-VAR}\ \frac{}{\Gamma^{\mathsf{at}},x:N\vdash_{\mathsf{s}} x\Downarrow N}$$

$$\textsc{SAT-DOWN-PROJ}\ \frac{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} n\Downarrow N_1\times N_2}{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}}\pi_i\ n\Downarrow N_i}\qquad \textsc{SAT-DOWN-APP}\ \frac{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} n\Downarrow P\to N\qquad \Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} p\Uparrow P}{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} n\ p\Downarrow N}\qquad \textsc{SAT-UP-INJ}\ \frac{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}} p\Uparrow P_i}{\Gamma^{\mathsf{at}}\vdash_{\mathsf{s}}\sigma_i\ p\Uparrow P_1+P_2}$$

we want to enumerate all terms at this typing judgment, it suffices to enumerate the possible terms t_1 and t_2 in the premises, and for each pair of such terms (in the cartesian product of the enumeration) return the term $(\mathtt{match}\ x\ \mathtt{with}\ \mid \sigma_1\ x\to t_1\mid \sigma_2\ x\to t_2)$. In other words, all distinct terms are (equivalent to a term) of the shape $(\mathtt{match}\ x\ \mathtt{with}\ \mid \sigma_1\ x\to ?_1\mid \sigma_2\ x\to ?_2)$, with the holes $?_i$ filled as per the premise judgments.

Remark 5.2. This arguably distinguishes focusing from other approaches such as bidirectional type-checking, which are essentially identical on the purely negative fragment. Focusing is justified in a general enough setting to easily extend to sum types. It predicts that some type-directed transformations should be guided by the typing context, rather than the goal type. ∗

The negative types are those whose construction (introduction) rule is invertible, and the positive types are those whose destruction (elimination) rule is invertible. This means that while the goal is negative, or while there remains a negative in the context, an invertible rule can be applied. The structure of our judgments forces us to apply these invertible rules as long as possible; we only leave the invertible judgment in the transition rule SINV-SAT,

which is only available when the context has only negative or atomic formulas, and the goal is positive or atomic:

$$\text{SINV-SAT}\quad \frac{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} f : (P^{\mathsf{at}} \mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}};\left\langle \Gamma^{\mathsf{at}\prime} \right\rangle^{+\mathsf{at}} \vdash_{\mathit{sinv}} t : \left\langle P^{\mathsf{at}} \right\rangle^{-\mathsf{at}} \mid Q^{\mathsf{at}}}$$

More precisely, the polarity constraint is enforced by the fact that the function $\langle \_ \rangle^{+\mathsf{at}}$ takes a negative formula, and returns a positive formula (by shifting) or a negative atom (atoms are preserved); so the judgment context is in the image of this function only if all its formulas are shifted positive formulas or negative atoms. Same thing for $\langle \_ \rangle^{-\mathsf{at}}$ in the goal.

On the goal side, let us recall that a convention of the $(A \mid B)$ notation is that exactly one of the sides is empty, and the other is a formula. The invertible judgment maintains two different formula positions,

Finally, let us comment on the role of the two contexts Γ^{at} and $\Gamma^{\mathsf{at}\prime}$ appearing in this rule, and in general Γ^{at} (a context of negative or atomic formulas) and the second context Σ (a context of positive formulas). Γ^{at} never evolves when applying rules of the invertible phase: it is the "old" context, in which the invertible phase started, unchanged. On the contrary, Σ is the context of formulas that are added to the context during the phase (by introducing a λ-abstraction, or by decomposing a formula already in Σ). It contains the "new" formulas that were unknown at the beginning of the invertible phase.

5.2.2 *Saturation phase – a first look*

The saturation phase only starts where all possible invertible rules have been applied. Any rule we can apply now is *non-invertible*: it requires making a choice, and it may be the wrong choice – going to a dead end.

There are two kinds of non-invertible rules: the ones that try to use variables from the context (for example choosing to call a function from the context, which may fail if we can't build a value of the argument's type), and the ones that try to construct values at the goal type (if the goal is a sum $A_1 + A_2$, it would be an injection constructor $\sigma_i \_$, representing the choice to either build a A_1 or a A_2). In the (asymmetric) intuitionistic logic, using the context is better choice, as failure there does not require backtracking (at worst we do not manage to call the function, and we continue the proof with something else); thus, we try to deduce everything we can from the context first, and do a choice on the goal type only later – this is saturation, done by the SAT rule.

$$\text{SAT}\quad \frac{\begin{array}{c}(\bar{n},\bar{P}) \stackrel{\mathsf{def}}{=} \mathtt{Select}_{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}}(\{(n,P) \mid (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{s}} n \Downarrow \langle P \rangle^{-}) \wedge \exists x \in \Gamma^{\mathsf{at}\prime}, x \in n\}) \\ \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime};\bar{x} : \bar{P} \vdash_{\mathit{sinv}} t : \emptyset \mid Q^{\mathsf{at}}\end{array}}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t : Q^{\mathsf{at}}}$$

The SAT rule is the central and most complex rule of our saturated calculus. We do not know how to explain it in one go – the current definition evolved by refinement. Instead of trying to dissect it now, we will use a two-step approach: first describe informally what it does, *assume* that it does it correctly to understand the rest of the rules and the big picture

of how the whole type system works, and then go back to its definition once the general mechanics is in place.

What the SAT rule does is the following: it looks for all the way that a positive formula can be *deduced* from the context, that is, proved by a neutral term $n : \langle P \rangle^-$. It adds all these deductions to the current context, and goes to the invertible judgments again – where these positive formulas are decomposed by the invertible rules, before starting another step of saturation. Note that the goal type is not changed by saturation, it is still positive or atomic, and is thus not decomposed by the following invertible phase. Only the types just deduced by saturation change during inversion.

With this description, it looks like the saturation process would never stop. This is where the separation, in the invertible judgment, between the "old" context and the "new" context come in. Eventually, it will become the case that all positive formulas deducible from the context have been deduced, and the next saturation phase will not split on any new formula. The invertible phase will start, but stop immediately after (no positive formula from the context to decompose), and call the saturation judgment again with $\Gamma^{at\prime}$ being the empty set $\emptyset$. When the "new" context is empty, we know that saturation has reached a stable state, and we allow saturation to stop: instead of the SAT rule, the proof may continue with either SAT-UP or SAT-DOWN, that escape the saturation judgment by finally trying to construct a term/proof of the goal type. At this point, we have done all possible deductions from the context, so we can make arbitrary choices (in fact, try all those choices), as there is nothing more to learn to help us making those choices.

The rules SAT-UP and SAT-DOWN do not overlap, only one of them is usable depending on the goal type. If it is a positive formula, we try to prove by a series of introduction rules (in terms of focusing, this is a right focusing phase). If it is a negative atom, we try to prove it by a series of elimination rules (in terms of focusing, this is a left focusing phase that ends on a negative atom).

$$\textsc{SAT-UP}\ \frac{\Gamma^{at} \vdash_s p \Uparrow P}{\Gamma^{at}; \emptyset \vdash_{sat} p : \langle P \rangle^-} \qquad \textsc{SAT-DOWN}\ \frac{\Gamma^{at} \vdash_s n \Downarrow X^-}{\Gamma^{at}; \emptyset \vdash_{sat} n : X^-}$$

5.2.3 *Focused introduction and elimination phases*

The judgment $\Gamma^{at} \vdash_s p \Uparrow P$, entered from the SAT-UP rule, tries to prove a positive formula by a series of introduction rules, by building a term out of value constructors. At each step of this judgment we need to make a non-invertible choice; to enumerate all possible proofs, we just backtrace on each of those choices. When we reach a (shifted) negative formula $\langle N \rangle^+$ in the rule SAT-UP-SINV, there are no non-invertible constructors to apply anymore, so we revert to the invertible judgment.

$$\textsc{SAT-UP-INJ}\ \frac{\Gamma^{at} \vdash_s p \Uparrow P_i}{\Gamma^{at} \vdash_s \sigma_i\ p \Uparrow P_1 + P_2} \qquad \textsc{SAT-UP-SINV}\ \frac{\Gamma^{at}; \emptyset \vdash_{sinv} t : N \mid \emptyset}{\Gamma^{at} \vdash_s t \Uparrow \langle N \rangle^+}$$

The judgment $\Gamma^{at} \vdash_s n \Downarrow N$, entered from the SAT-DOWN rule, describes a series of elimination steps (function application or pair projection) applied to a head variable taken in the context. Unlike all other judgments of natural deduction or sequent calculus, the

rules of this judgment should be read from leaf to root. A proof start from a variable chosen from the context, in the rule SAT-DOWN-VAR, that is of negative type, and applies a series of non-invertible elimination rules, passing an argument (if the negative type is a function) or projecting one component (if the negative type is a product).

$$\text{SAT-DOWN-VAR}\ \frac{}{\Gamma^{\mathsf{at}}, x : N \vdash_{\mathsf{s}} x \Downarrow N} \qquad \text{SAT-DOWN-APP}\ \frac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} n \Downarrow P \to N \qquad \Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} p \Uparrow P}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} n\ p \Downarrow N} \qquad \text{SAT-DOWN-PROJ}\ \frac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} n \Downarrow N_1 \times N_2}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} \pi_i\ n \Downarrow N_i}$$

Notice that the input type of function is a positive type, and that we look for an argument as a positive neutral term p by typing it with the non-invertible introduction judgment $\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} p \Uparrow P$. Some proof systems are "less focused", in that they allow function arguments to start with a more general invertible phase.

The ending rule of the introduction judgment $\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} p \Uparrow P$ enforces the fact that an introduction phase ends only when the formula becomes negative (or, in the SAT-UP-ATOM rule, when we reach a positive axiom). The elimination judgment goes in the other direction, so it is the "caller" of this judgment (the rule who has the elimination judgment as a premise) that decides when it can end. In the SAT-DOWN rule, we only consider elimination phases that end on a negative atom, and in the SAT rule we only consider elimination phases that end of a (shifted) positive formula.

$$\text{SAT}\ \frac{(\bar{n}, \bar{P}) \stackrel{\mathsf{def}}{=} \mathtt{Select}_{\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime}}(\{(n,P) \mid (\boxed{\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{s}} n \Downarrow \langle P \rangle^{-}}) \wedge \exists x \in \Gamma^{\mathsf{at}\prime}, x \in n\}) \qquad \Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime}; \bar{x} : \bar{P} \vdash_{\mathit{sinv}} t : \emptyset \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t : Q^{\mathsf{at}}}$$

$$\text{SAT-DOWN}\ \frac{\Gamma^{\mathsf{at}} \vdash_{\mathsf{s}} n \Downarrow X^{-}}{\Gamma^{\mathsf{at}}; \emptyset \vdash_{\mathsf{sat}} n : X^{-}}$$

5.2.4 The saturation rule – a deeper look

A naive attempt at defining the SAT rule would look as follows:

$$\text{SAT-1}\ \frac{(\bar{n}, \bar{P}) \stackrel{\mathsf{def}}{=} \{(n,P) \mid (\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{s}} n \Downarrow \langle P \rangle^{-})\} \qquad \Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime}; \bar{x} : \bar{P} \vdash_{\mathit{sinv}} t : \emptyset \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t : Q^{\mathsf{at}}}$$

This definition looks for all ways to deduce (by a neutral proof term) a positive from the current context, adds it to the context, and continues with an invertible phase that will decompose those positives. It has two independent problems:

1. A single neutral term n will be introduced many times, by all saturation steps where it is typable – by monotonicity, subsequent saturation steps will introduce all the proofs of the previous iteration steps, plus some more. This breaks canonicity, which relies on the fact that each possible neutral (each possible observation of the formal context) is given a *unique* name. Consider for example the judgment

$$x_0 : Z_1^+, x : Z_1^+ \to \langle X^+ \rangle^{-}; \emptyset \vdash_{\mathit{sinv}} ? : \emptyset \mid X^+$$

The first saturation phase will deduce X^+ by introducing the proof $y_1 \stackrel{\text{def}}{=} x\,x_0$ of type X^-. It is followed by an invertible phase that will stop immediately, as there is no connective to decompose in the context or the goal. Then a new saturation phase starts; because there is no provision in SAT-1 against performing the same deduction again, the term could introduce $y_2 \stackrel{\text{def}}{=} x\,x_0$ of type X^+. This could go on indefinitely, but forgetting about the termination aspect for a moment, we have a canonicity problem: it now appear that there are two distinct ways to build the goal X^+, using either y_1 or y_2 – formal variables in the context are considered distinct.
We need a way to remember which neutrals have been introduced in previous saturation step, not to re-introduce them again; not doing so would break canonicity of the proof system, and thus soundness of the unicity-deciding algorithm.

2. The present definition introduces, at each saturation steps, *all* the neutrals of positive types. Even without taking the previously introduced ones into account, there may be too much new neutrals, leading saturated proof search into an infinite loop. There are two different sources of non-termination:

   - A single saturation step may, with this definition, introduce infinitely many positives. Consider for example a variable in context $x : \mathbb{N} \to P$, where $\mathbb{N}$ is a type of natural numbers, defined as $\mathbb{N} \stackrel{\text{def}}{=} (X^- \to X^-) \to X^- \to X^-$ for example, and P is some positive type. With such a variable x in the context $\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime}$, the set

   $$\{(n, P) \mid (\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{s}} n \Downarrow \langle P \rangle^-)\}$$

   is infinite (it contains $x\,0$, $x\,1$, $x\,2$, etc., with the usual definition of natural constants)). Even if we extended our syntax to accommodate infinitely-wide let-bindings $\texttt{let}\ \bar{x} = \bar{n}\ \texttt{in}\ \_$, the following invertible phase would have to deconstruct infinitely many copies of the type P in context, so there would be no finite proof (term) in this type system for any goal with $x : \mathbb{N} \to P$ in context.
   - Even if each saturation step is finite, saturation may keep going on indefinitely if each step introduces a new variable to use. Consider for example that for some "stream state" type X^+ we have in the typing environment a state value $x_0 : X^+$ and a "next" function $y : X^+ \to X^+ + Y^+$ that returns the next state if it exists, or a value of some type Y^+ if there is no next state – we reached the end of the stream. The first saturation step can use x_0 to deduce a new value $y\,x_0$ of positive type $X^+ + Y^+$; the invertible phase will pattern-match on this new value, and in the left branch we will have a new variable $x_1 : X^+$ in context. The second saturation phase can deduce a new value $y\,x_1$ of positive type, and the second invertible phase will decompose it and (in the left branch) bind a new variable $x_2 : X^+$ in context. This saturation process can continue indefinitely, even though each saturation step only introduces finitely many positives. This corresponds to the incremental construction of an infinite term spine, matching over an unbounded

stream:

$$\texttt{let } x_1 = y\, x_0 \texttt{ in } \left|\begin{array}{l} \texttt{match } x_1 \texttt{ with} \\ \left|\begin{array}{l} \sigma_1\, x_1 \to \texttt{let } x_2 = y\, x_1 \texttt{ in } \left|\begin{array}{l} \texttt{match } x_2 \texttt{ with} \\ \left|\begin{array}{l} \sigma_1\, x_2 \to \texttt{let } x_3 = y\, x_2 \texttt{ in } \ldots \\ \sigma_2\, x_2 \to \ldots \end{array}\right. \end{array}\right. \\ \sigma_2\, x_1 \to \ldots \end{array}\right.$$

In particular, the "new context" $\Gamma^{\mathsf{at}'}$ will always contain at least one new variable x_n of type X^+; it will never be empty, and the rules exiting the saturation cycle, SAT-UP and SAT-DOWN, will never be applicable. No matter what the goal type is (as long as it is positive, that is there is at least one saturation step), a system using the rule SAT-1 would have no (finite) proof term as soon as those "state" and "next" variables are in context.

Those are not canonicity issues (we are not enumerating duplicates), but termination and completeness issues. If some judgments that should be provable have no finite proofs, it means that our system is incomplete (even for provability), and also that proof search and enumeration will not terminate. To prevent this, we must somehow allow the logic to "drop" some new variables produced by saturation (when it is correct to do so), so that no single saturation step binds infinitely many variables, and so that repeated saturation steps eventually reach a stable state with an empty "new" context. This is done by keeping at most two variables of each type, using the Corollary 4.6 (Two-or-more approximation) of Section 4 (Counting terms and proofs).

Avoiding redundant splits An idea to solve the first problem (not splitting on the same neutral terms in several saturation processes) is to simply index all judgments will the set of all neutrals split so far, and to remove those neutrals from any following saturation step. This is, in fact, not necessary, thanks to our structural separation of the context between an "old" context Γ^{at} and a "new" context $\Gamma^{\mathsf{at}'}$. The new context contains exactly the variables that were split by the last invertible phase, and the old context the older ones, that were already available during the previous saturation step.

There is thus a very simple characterization of which neutrals n were already split in a previous saturation step, and should not be split again. They are the neutrals that are already typable in the old context Γ^{at}, or conversely the neutrals that do not use any variable from the new context $\Gamma^{\mathsf{at}'}$. This is the simplification that justifies keeping the static separation between the old and new context in the invertible rules.

An improved (but still unsatisfying) reformulation of the preliminary SAT-1 rule, that avoids redundant splits, is as follows:

SAT-2

$$\frac{\begin{array}{c}(\bar{n},\bar{P}) \stackrel{\text{def}}{=} \{(n,P) \mid (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}'} \vdash_{\mathsf{s}} n \Downarrow \langle P\rangle^-) \text{ and } (\exists x \in \Gamma^{\mathsf{at}'}, x \in n)\} \\ \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}'};\bar{x} : \bar{P} \vdash_{\mathsf{sinv}} t : \emptyset \mid Q^{\mathsf{at}}\end{array}}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}'} \vdash_{\mathsf{sat}} \texttt{let } \bar{x} = \bar{n} \texttt{ in } t : Q^{\mathsf{at}}}$$

This new rule forces us to introduce only (and all) the terms that are "new", in the sense that they use the new context $\Gamma^{\mathsf{at}'}$ – this is checked by the condition $(\exists x \in \Gamma^{\mathsf{at}'}, x \in t)$.

Finite saturation proofs As we have seen with a few examples, some contexts have saturation processes that split infinitely many new neutrals, either during a single step or through infinitely many steps never reaching a fixpoint. This is not surprising or wrong: some types are inhabited by infinitely many distinct programs. However, while we expect that enumerating all those programs would require infinitely many steps, we would like to be able to have finite proofs for each of those programs, which our current saturation rules does not allow.

To have finite proofs even during an infinite saturation process, it suffices to allow some proofs to use only *a subset* of the split subterms. Instead of SAT-2, consider the following rule, which only replaces the $(\stackrel{\mathsf{def}}{=})$ in the first premise by a $(\subseteq)$:

$$\textsc{sat-2-sub}\quad \frac{(\bar{n},\bar{P}) \subseteq \{(n,P) \mid (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{s}} n \Downarrow \langle P\rangle^{-}) \text{ and } (\exists x \in \Gamma^{\mathsf{at}\prime}, x \in n)\} \quad \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime};\bar{x} : \bar{P} \vdash_{\mathit{sinv}} t : \emptyset \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t : Q^{\mathsf{at}}}$$

It may seem that this definition of the saturation rule allow the goal-directed proof enumeration process to stop the saturation earlier than it should (in particular if we select $\bar{n} \stackrel{\mathsf{def}}{=} \emptyset$, then saturation stops) and thus make the search incomplete. But as the enumeration process is looking for all possible proof terms of the judgment, it may consider all possible subsets, and thus not miss a single term; note that each finite term uses only a finite subset of the split neutrals, so we can always assume $\bar{n}$ finite.

Unfortunately, this weaker condition also causes a loss of canonicity: two proof terms may be essentially the same, but differ by the fact that one saturates on a few additional neutrals – otherwise unused.

Canonicity by deterministic restriction This gets us to the final version of our rule:

$$\textsc{sat}\quad \frac{(\bar{n},\bar{P}) \stackrel{\mathsf{def}}{=} \mathtt{Select}_{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}}(\{(n,P) \mid (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{s}} n \Downarrow \langle P\rangle^{-}) \wedge \exists x \in \Gamma^{\mathsf{at}\prime}, x \in n\}) \quad \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime};\bar{x} : \bar{P} \vdash_{\mathit{sinv}} t : \emptyset \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t : Q^{\mathsf{at}}}$$

In this version, the choice of which subset of neutrals to saturate on is fixed once and for all by the saturation function $\mathtt{Select}_{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}}(\_)$. For a given choice of saturation function, all proof search processes for a given judgment will select the same set of neutrals. This avoids the previous canonicity issue: two terms cannot differ merely by the choice of which neutrals to saturate over.

Comparison with the previous approach of Scherer and Rémy [2015] In the previous presentation of Scherer and Rémy [2015], we did not use a fixed saturation-selection function; instead, the saturation rule had one extra requirement that all the neutrals introduced by saturation where "useful" in some sense.

$$\text{SAT}\ \frac{(\bar{n},\bar{P}) \subseteq \{(n,P) \mid (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{s}} n \Downarrow \langle P\rangle^{-}) \wedge \exists x \in \Gamma^{\mathsf{at}\prime}, x \in n\} \quad \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime};\bar{x}:\bar{P} \vdash_{\mathit{sinv}} t : \emptyset \mid Q^{\mathsf{at}} \quad \forall x \in \bar{x}, t\ \mathsf{uses}\ x}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t : Q^{\mathsf{at}}}$$

The $(t\ \mathsf{uses}\ x)$ judgment, which we have not defined here, corresponds to the fact that the introduced variable x is used after the first invertible phase.

This condition did not affect proof search, as it is expressed on the proof term t that is only known after the search for this recursive subgoal has taken place. The set of useful neutrals was obtained by filtering the saturating neutrals after the fact. This mean that each possible outcome of the proof search (the term t) was uniquely associated with a "minimal" saturating set, avoiding any canonicity issue.

However, this side-condition creates a difficulty when we try to combine the canonicity result for this logic with the "two-or-more" restriction of Section 4 (Counting terms and proofs) to obtain a system that both is complete for unicity and has terminating proof search. Indeed, at this point we need to argue that if two distinct derivations of the same shape exist, then two derivations also exist when the contexts have been restricted to two distinct variables.

The proof of this result in Section 4 (Counting terms and proofs) relies on the assumption that, in a proof term of a given shape, all variables at a given type in a context can be used when the goal is of this type – after restricting our contexts to have at most two variables of the same type, we replace bound variable occurrences by one of those two variables, and the shape is unchanged. This assumption becomes invalid in a system where using a variable or another has consequences on the validity of the whole term, as is the case in the presence of this usage rule: if your saturation phase introduces three variables x_1, x_2, x_3 of the same type, it is only valid to keep them in the term of all three are used later. Rewriting all occurrences of x_3 to become either x_2 or x_1 (to obtain a derivation using at most two distinct variables of each type) breaks this condition, unless you also remove the corresponding `let`-binding. In other words, it would be possible to adapt the proof of Section 4 (Counting terms and proofs) to this setting, by adding an extra normalization step, but it adds complexity to the result and is not worth it. Our use of the selection function nicely side-steps this issue, by preserving the validity of any variable-variable replacement.

5.3 The roles of forward and backward search in a saturated logic

Focusing is a fruitful theoretical tool to propose a more logical understanding of proof search strategies – see for example Chaudhuri, Pfenning, and Price [2008b], Chaudhuri [2010], Farooque, Graham-Lengrand, and Mahboubi [2013]. This flexibility is built out of two components whose interaction can be subtle. On one hand, the way formulas are polarized prevents or enforce certain shapes of proof terms, for example forward- or backward-chaining, as we detailed in Section 2.2.7 (Polarized atoms). On the other hand, there are several distinct strategies for proof search, notably the rather natural judgment-directed or goal-directed backward search, and the inverse method, a form of saturation-based forward search. The strength of focusing is to move a lot of the sophistication from the search strategy into the logic itself: a lot of subtle operational ideas on good proof strategies can

be obtained by using one of those two simple strategies with a subtle logic or polarisation of formulas.

To prove a judgment of the form $\Delta \vdash A$, the natural intuition for goal-directed search procedure is to look at A and search for all possible ways to introduce its head connective. A focused system has a richer behavior, in that it will also decompose the positives of Δ, but this reliance on the context remains "superficial" in the sense that only the first positive layer of those formulas will be peeled of by the invertible phase. The "real" work happens at the end of the invertible phase, where choices must be made, and typically various attempts will be made, with a backtracking discipline to roll back the wrong choices, for example the right introduction on a sum that happened too early.

On the contrary, on a judgment of the form $\Delta \vdash A$, an inverse method will, in rough terms, look as the subformulas of Δ, A as the "search space" of facts to prove. It will try to build proofs in a leafward-rootward fashion, from elementary deduction in this search space to more elaborated facts, until maybe a deduction implying the original goal $\Delta \vdash A$: happens.

It is interesting to consider the operational search behavior of our saturated logic when using a simple judgment-directed backward search implementation.

Goal-directed proof search in our saturated logic starts in a state where all of the context is "new", it has not been saturated over: $\emptyset; \Delta \vdash A$. During the invertible phase, it behaves like others goal-directed procedures, and extract a negative or atomic context Γ^{at} of "new" formulas, and a refined goal Q^{at}, and start the saturation phase $\emptyset; \Gamma^{\mathsf{at}} \vdash_{\mathsf{sat}} ? : Q^{\mathsf{at}}$.

The saturation phase does not behave like a goal-directed backward procedure, it is a phase of forward search. However, there is an important difference with the inverse method or other approaches that are "full" forward search: the "search space" of the saturation is not the complete goal $\Gamma^{\mathsf{at}} \vdash Q^{\mathsf{at}}$, it is only Γ^{at}. We are not trying to discover arbitrary facts that will help us in eventually proving our goal Q^{at}, we are restricting the set of deductions to subformulas of the context Γ^{at}. So it is a forward search phase, but it is "localized" by the use of only a part of the judgment.

After this local saturation phase ends, goal-directed search starts over with non-invertible steps attempting to prove the goal formula, and the corresponding backtracking behavior of backward search. The right rules that happen during this right focusing phase will change the goal formula to a negative subformula of Q^{at}. This creates opportunities for the following invertible to move parts of the goal into the context, expanding the "horizon" of the following saturation phases.

To summarize, there is an alternation of backward and forward search phases. The forward search is bounded by the context, while the backward search is directed by the goal formula, and transmits new hypothesis to the context, expanding the reach of the subsequent forward phases.

Interestingly, this mixture of backward and forward search exists in some seemingly unrelated work on logic programming, in particular in Lollimon López, Pfenning, Polakow, and Watkins [2005]; we give a detailed comparison in Section 9.1.2 (Lollimon: backward and forward search together).

6 Canonicity of saturated proofs

6.1 Big-step saturating translation

To prove the main theorems on saturating focused logic, we describe how to convert a focused λ-term into a valid saturated proof derivation. This can be done either as a small-step rewrite process, or as a big-step transformation. The small-step rewrite would be very similar to the *preemptive rewriting* relation of Scherer [2015a]; we will here use a big-step transformation, as in Scherer and Rémy [2015], by defining in Figure 22 a type-preserving translation judgments of the form $\Gamma;\Sigma \vdash_{\mathsf{sinv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$, which turns a focused term t into a valid *saturating* focused term t'.

Fig. 22. Saturation translation

$$\text{REW-SINV-LAM}\quad \frac{\Gamma^{\mathsf{at}};\Sigma,x:P \vdash_{\mathsf{sinv}} t \rightsquigarrow t' : N \mid}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} \lambda x.t \rightsquigarrow \lambda x.t' : P \to A \mid \emptyset}$$

$$\text{REW-SINV-PAIR}\quad \frac{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t_1 \rightsquigarrow t'_1 : N_1 \mid \emptyset \qquad \Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t_2 \rightsquigarrow t'_2 : N_2 \mid \emptyset}{\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} (t_1,t_2) \rightsquigarrow (t'_1,t'_2) : N_1 \times N_2 \mid \emptyset}$$

$$\text{REW-SINV-CASE}\quad \frac{\begin{array}{c}\Gamma^{\mathsf{at}};\Gamma,x:P_1 \vdash_{\mathsf{sinv}} t_1 \rightsquigarrow t'_1 : N \mid Q^{\mathsf{at}} \\ \Gamma^{\mathsf{at}};\Gamma,x:P_2 \vdash_{\mathsf{sinv}} t_2 \rightsquigarrow t'_2 : N \mid Q^{\mathsf{at}}\end{array}}{\Gamma^{\mathsf{at}};\Gamma,x:P_1+P_2 \vdash_{\mathsf{sinv}} \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ x \to t_1 \\ \sigma_2\ x \to t_2\end{array}\right. \rightsquigarrow \mathtt{match}\ x\ \mathtt{with}\ \left|\begin{array}{l}\sigma_1\ x \to t'_1 \\ \sigma_2\ x \to t'_2\end{array}\right. : N \mid Q^{\mathsf{at}}}$$

$$\text{REW-SINV-SAT}\quad \frac{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} f \rightsquigarrow f' : (P^{\mathsf{at}} \mid Q^{\mathsf{at}})}{\Gamma^{\mathsf{at}};\left\langle \Gamma^{\mathsf{at}\prime} \right\rangle^{+\mathsf{at}} \vdash_{\mathsf{sinv}} f \rightsquigarrow f' : \left\langle P^{\mathsf{at}} \right\rangle^{-\mathsf{at}} \mid Q^{\mathsf{at}}}$$

$$\text{REW-SAT-INTRO}\quad \frac{\Gamma^{\mathsf{at}} \vdash p \rightsquigarrow p' \Uparrow P}{\Gamma^{\mathsf{at}};\emptyset \vdash_{\mathsf{sat}} p \rightsquigarrow p' : P} \qquad \text{REW-SAT-ATOM}\quad \frac{\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow X}{\Gamma^{\mathsf{at}};\emptyset \vdash_{\mathsf{sat}} n \rightsquigarrow n' : X}$$

$$\text{REW-SAT}\quad \frac{\begin{array}{c}(\bar{n},\bar{P}) \stackrel{\mathsf{def}}{=} \mathtt{Select}_{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}}(\{(n,P) \mid (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash n \Downarrow P)\}) \\ \forall n \in t,\ (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash n \Downarrow P) \implies n \in \bar{n} \\ \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime};\bar{x}:\bar{P} \vdash_{\mathsf{sinv}} t[\bar{x}/\bar{n}] \rightsquigarrow t' : Q^{\mathsf{at}} \mid\end{array}}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} t \rightsquigarrow \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t' : Q^{\mathsf{at}}}$$

$$\text{REW-SINTRO-SUM}\quad \frac{\Gamma^{\mathsf{at}} \vdash p \rightsquigarrow p' \Uparrow A_i}{\Gamma^{\mathsf{at}} \vdash \sigma_i\ p \rightsquigarrow \sigma_i\ p' \Uparrow A_1+A_2} \qquad \text{REW-SINTRO-END}\quad \frac{\Gamma^{\mathsf{at}};\emptyset \vdash_{\mathsf{sinv}} t \rightsquigarrow t' : N \mid}{\Gamma^{\mathsf{at}} \vdash t \rightsquigarrow t' \Uparrow \langle N \rangle^{+}} \qquad \text{REW-SINTRO-AXIOM}\quad \frac{(x : X^{+}) \in \Gamma^{\mathsf{at}}}{\Gamma^{\mathsf{at}} \vdash x \rightsquigarrow x \Uparrow X^{+}}$$

$$\text{REW-SELIM-PAIR}\quad \frac{\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow A_1 \times A_2}{\Gamma^{\mathsf{at}} \vdash \pi_i\ n \rightsquigarrow \pi_i\ n' \Downarrow A_i} \qquad \text{REW-SELIM-ARR}\quad \frac{\Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow P \to N \qquad \Gamma^{\mathsf{at}} \vdash p \rightsquigarrow p' \Uparrow P}{\Gamma^{\mathsf{at}} \vdash n\ p \rightsquigarrow n'\ p' \Downarrow N} \qquad \text{REW-SELIM-START}\quad \frac{(x : N) \in \Gamma^{\mathsf{at}}}{\Gamma^{\mathsf{at}} \vdash x \rightsquigarrow x \Downarrow N}$$

$$(\mathtt{let}\ x = n\ \mathtt{in}\ t)[y/n] \stackrel{\mathsf{def}}{=} t[y/x][y/n]$$

Backward search for saturated proofs corresponds to enumerating the canonical inhabitants of a given type. Our translation can be seen as a restriction of this proof search

process, searching inside the $\beta\eta$-equivalence class of t. Because saturating proof terms are canonical (to be shown), the restricted search is deterministic – modulo invertible commuting conversions.

Compared to the focusing translation of Figure 16 used to prove completeness of focusing with respect to the non-focused λ-calculus in Section 3.3 (Focusing completeness by big-step translation), this rewriting is simpler as it starts from an already-focused proof whose overall structure is not modified. The only real change is moving from the left-focusing rule REW-FOC-ELIM to the saturating rule REW-SAT. Instead of allowing to cut on any neutral subterm, we enforce a maximal cut on exactly all the neutrals of t that can be typed in the current environment. Because we know that "old" neutrals have already been cut and replaced with free variables earlier in the translation, this is fact respects the saturation condition.

Compared to the focusing translation, the termination of this translation is immedate induction: thanks to the focused structure of the input, every recursive call happens on a strictly smaller term. In the REW-SAT rule, the recusive call is on $t[\bar{x}/\bar{n}]$, which is not be strictly smaller if the $\bar{n}$ are variables, which can happen for $x : \langle P \rangle^-$. But this case is only possible when x is in the "new" context, as this neutral uses no other variable that could be in the new context; and this variable gets replaced by a variable in the post-saturation new context at the strictly smaller type P, so it can only happen finitely many times.

Assumptions on the selection function The REW-SAT rule makes an interesting assumption on the selection function:

$$\text{REW-SAT}\quad \frac{\begin{array}{c}(\bar{n},\bar{P}) \stackrel{\text{def}}{=} \mathtt{Select}_{\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime}}(\{(n,P) \mid (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash n \Downarrow P)\}) \\ \forall n \in t,\ (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime} \vdash n \Downarrow P) \quad\Longrightarrow\quad n \in \bar{n} \\ \Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime};\bar{x} : \bar{P} \vdash_{\mathit{sinv}} t[\bar{x}/\bar{n}] \rightsquigarrow t' : Q^{\mathsf{at}} \mid \end{array}}{\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} t \rightsquigarrow \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t' : Q^{\mathsf{at}}}$$

This rule can only be applied if all the neutrals of the translated n that are typeable in the present context happen to be part of the neutrals selected for saturation. This is a requirement that most selection functions will *not* meet: for any choice of selection functions there are many t such that no valid derivation of the form $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ exist for any t'.

However, for any t we can construct some – and in fact many – valuation functions for which such a $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ exists for some t'. If we start from an arbitrary selection function satisfying SELECT-SPECIF, we can build another selection function that meets this requirement by simply adding all the neutral subterms that happen during this translation. As we are only adding new neutrals, the resulting selection still satisfies SELECT-SPECIF. Any finite derivation of the translation judgment will only add finitely many new neutrals this way, which means that the returned selection function still returns finite sets of neutrals for each context.

We say that a selection function is *adequate* for some term $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t : N \mid Q^{\mathsf{at}}$ if it does select all neutrals of t, in the sense that there exists a derivation $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ for some t'. Note that different adequate selection functions will result in different translations

t'. In general we will implictly assume that the selection function is adequate for the terms considered.

Lemma 6.1 (Translation soundness)**.**
If $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t : N \mid Q^{\mathsf{at}}$ *and* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ *then* $t \approx_{\beta\eta} t'$.

Proof. By immediate induction. □

Lemma 6.2 (Translation validity)**.**
Suppose that $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t : N \mid Q^{\mathsf{at}}$ *holds in the focused logic, and that* t *has no "old" neutral: for no* $n \in t$ *do we have* $\Gamma^{\mathsf{at}} \vdash n \Downarrow \langle P \rangle^{-}$. *Then,* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$ *implies that* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t' : N \mid Q^{\mathsf{at}}$ *in the saturated focusing logic.*

Proof. The restriction on "old" neutrals is necessary because the REW-SAT rule would not know what to do on such old neutrals – it assumes that they were all substituted away for fresh variable in previous inference steps.

With this additional invariant the proof goes by immediate induction. In the REW-SAT rule, this invariant tells us that the bindings satisfy the freshness condition of the SAT rule of saturated logic, and because we select *all* such fresh bindings we preserve the property that the extended context $\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}'}$ has no old neutrals either. □

Lemma 6.3 (Translation determinism)**.**
If the selection function is adequate for $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t : N \mid Q^{\mathsf{at}}$, *then there exists a* unique t' *such that* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$.

Proof. By immediate induction. □

Note that the indeterminacy of invertible step ordering is still present in saturating focused logic: a *non-focused* term t may have several saturated translations that only equal upto commuting conversions ($\approx_{\mathtt{icc}}$). However, there is no more variability than in the focused proof of the non-saturating focused logic; because we translate from those, we can respect the ordering choices that are made, and the *translation* is thus fully deterministic.

Theorem 6.4 (Computational completeness of saturating focused logic)**.**
If we have $\emptyset;\Sigma \vdash_{\mathsf{inv}} t : N \mid Q^{\mathsf{at}}$ *in the non-saturating focused logic, then for an adequate saturation function and some* $t' \approx_{\beta\eta} t$ *we have* $\emptyset;\Sigma \vdash_{\mathsf{sinv}} t' : N \mid Q^{\mathsf{at}}$ *in the saturating focused logic.*

Proof. This is an immediate corollary of the previous results. For an adequate selection function, there is a unique t' such that $\emptyset;\Sigma \vdash_{\mathsf{sinv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$. By Lemma 6.2 (Translation validity) we have that $\emptyset;\Sigma \vdash_{\mathsf{sinv}} t' : N \mid Q^{\mathsf{at}}$ in the saturating focused calculus – the condition that there be no old neutrals is trivially true for the empty context $\emptyset$. Finally, by Lemma 6.1 (Translation soundness) we have that $\lfloor t \rfloor_{\mathtt{foc}} \approx_{\beta\eta} \lfloor u \rfloor_{\mathtt{foc}}$. □

Lemma 6.5 (Determinacy of saturated translation)**.**
For any u_1, u_2, *if we have* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow u_1 : N \mid Q^{\mathsf{at}}$ *and* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow u_2 : N \mid Q^{\mathsf{at}}$ *then we have* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} u_1 \rightsquigarrow r_1 : N \mid Q^{\mathsf{at}}$ *and* $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{sinv}} u_2 \rightsquigarrow r_2 : N \mid Q^{\mathsf{at}}$ *with* $r_1 \approx_{\mathtt{icc}} r_2$.

Proof sketch. There are only two sources of non-determinism in the focused translation:

- an arbitrary choice of the order in which to apply the invertible rules
- a neutral `let`-extrusion may happen at any point between the first scope where it is well-defined to the lowest common ancestors of all uses of the neutral in the term.

The first source of non-determinism gives $(\approx_{\mathtt{icc}})$-equivalent derivations. The second disappears when doing the saturating translation, which enforces a unique placement of `let`-extrusions at the first scope where the strictly positive neutrals are well-defined.

As a result, two focused translations of the same term may differ in both aspect, but their saturated translations differ at most by $(\approx_{\mathtt{icc}})$. □

6.2 Normalization and canonicity

Definition 6.1 Normalization by saturation**.**
For a well-typed (non-focused) λ-term $\lfloor\Gamma^{\mathsf{at}}\rfloor_{\pm}, \lfloor\Sigma\rfloor_{\pm} \vdash t : \lfloor(N \mid Q^{\mathsf{at}})\rfloor_{\pm}$, we write $\mathsf{NF}_{\mathsf{sat}}(t)$ for any saturated term t'' such that

$$\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_{\beta}(t) \rightsquigarrow t' : N \mid Q^{\mathsf{at}} \qquad \Gamma^{\mathsf{at}};\Sigma \vdash_{\mathit{sinv}} t' \rightsquigarrow t'' : N \mid Q^{\mathsf{at}}$$

Note that all possible t'' are equal modulo $(\approx_{\mathtt{icc}})$, by Lemma 6.5 (Determinacy of saturated translation).

Lemma 6.6 (Saturation congruence)**.**
For any context $C[\Box]$ and term t we have

$$\mathsf{NF}_{\mathsf{sat}}(C[t]) \approx_{\mathtt{icc}} \mathsf{NF}_{\mathsf{sat}}(C[\mathsf{NF}_{\mathsf{sat}}(t)])$$

Proof. We reason by induction on $C[\Box]$. Without loss of generality we will assume $C[\Box]$ atomic. It is either a redex-forming context

$$\Box\, u \qquad \pi_k\, \Box \qquad \mathtt{match}\ \Box\ \mathtt{with} \mid \begin{array}{l}\sigma_1\ x \to u_1\\ \sigma_2\ x \to u_2\end{array}$$

or a non-redex forming context

$$u\, \Box \qquad \sigma_i\, \Box \qquad (u, \Box) \qquad (\Box, u)$$

$$\mathtt{match}\ u\ \mathtt{with} \mid \begin{array}{l}\sigma_1\ x \to \Box\\ \sigma_2\ x \to u_2\end{array} \qquad \mathtt{match}\ u\ \mathtt{with} \mid \begin{array}{l}\sigma_1\ x \to u_1\\ \sigma_2\ x \to \Box\end{array}$$

If it is a non-context-forming redex, then we have $\mathsf{NF}_{\beta}(C[t]) = C[\mathsf{NF}_{\beta}(t)]$. The focused and saturated translations then work over $C[\mathsf{NF}_{\beta}(t)]]$ just as they work with $\mathsf{NF}_{\beta}(t)$, possibly adding bindings before $C[\Box]$ instead of directly on the (translations of) $\mathsf{NF}_{\beta}(t)$. The results are in the $(\approx_{\mathtt{icc}})$ relation.

The interesting case is when $C[\Box]$ is a redex-forming context: a reduction may overlap the frontier between $C[\Box]$ and the plugged term. In that case, we will reason on the saturated normal form $\mathsf{NF}_{\mathsf{sat}}(t)$. Thanks to the strongly restricted structure of focused and saturated normal form, we have precise control over the possible reductions.

Application case $C[\Box] \stackrel{\mathsf{def}}{=} \Box\, u$. We prove that there exist t' such that $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : P \to N \mid \emptyset$, and a r such that both $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t\ u \rightsquigarrow r : N \mid \emptyset$ and $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t'\ u \rightsquigarrow r : N \mid \emptyset$ hold. This implies the desired result – after translation of r into a saturated term. The proof proceeds by induction on the derivation $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t\ u \rightsquigarrow r : N \mid \emptyset$ (we know that all possible such translations have finite derivations).

To make the proof easier to follow, we introduce the notation $\mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Sigma \vdash t)$ to denote a focused translation t' of $\mathsf{NF}_{\beta}(t)$ (that is, $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow t' : N \mid Q^{\mathsf{at}}$, where N, Q^{at} are

uniquely defined by $\Gamma^{\mathsf{at}};\Sigma \vdash_{\mathsf{inv}} t' : N \mid Q^{\mathsf{at}}$)). This notation should be used with care because it is not well-determined: there are many such possible translations. Statements using the notation should be interpreted existentially: $P(\mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Sigma \vdash t))$ means that there exists a translation t' of t such that $P(t')$ holds. The current goal (whose statement took the full previous paragraph) can be rephrased as follows:

$$\mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Sigma \vdash t\ u) = \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Sigma \vdash \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Sigma \vdash t)\ u)$$

We will simply write $\mathsf{NF}_{\mathsf{foc}}(t)$ when the typing environment of the translation is clear from the context.

If Σ contains a sum type, it is of the form $(\Sigma', x : C_1 + C_2)$ and we can get by induction hypothesis that

$$\mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Sigma', x : C_i \vdash t\ u) = \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Sigma', x : C_i \vdash \mathsf{NF}_{\mathsf{foc}}(t)\ u)$$

for i in $\{1,2\}$, from which we can conclude with

$$\begin{array}{lll} & \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Gamma', x : C_1 + C_2 \vdash t\ u) & \\ = & \texttt{match}\ x\ \texttt{with} & \begin{array}{l} \sigma_1\ x \to \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Gamma', x : C_1 \vdash t\ u) \\ \sigma_2\ x \to \ldots C_2 \ldots \end{array} \\ = & \texttt{match}\ x\ \texttt{with} & \begin{array}{l} \sigma_1\ x \to \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Gamma', x : C_1 \vdash \mathsf{NF}_{\mathsf{foc}}(t)\ u) \\ \sigma_2\ x \to \ldots C_2 \ldots \end{array} \\ = & \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Gamma', x : C_1 + C_2 \vdash \mathsf{NF}_{\mathsf{foc}}(t)\ u) & \end{array}$$

Otherwise Σ is of the form $\langle \Gamma^{\mathsf{at}'} \rangle^{+\mathsf{at}}$.

Any focused translation of t at type $N \to P$ is thus necessarily of the form $\lambda x.\, \mathsf{NF}_{\mathsf{foc}}(t\ x)$. In particular, any $\mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t)\ u)$, that is, any $\mathsf{NF}_{\mathsf{foc}}((\lambda x.\, \mathsf{NF}_{\mathsf{foc}}(t\ x))\ u)$, is equal by stability of the translation to β-reduction to a term of the form $\mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t\ x)[u/x])$. On the other hand, $\mathsf{NF}_{\mathsf{foc}}(t\ u)$ can be of several different forms.

Note that $t\ u$ is translated at the same type as $t\ x$. In particular, if this is a negative type, they both begin with a suitable η-expansion (of a product or function type); in the product case for example, we have $\mathsf{NF}_{\mathsf{foc}}(t\ u) = (\mathsf{NF}_{\mathsf{foc}}(\pi_1\ (t\ u)), \mathsf{NF}_{\mathsf{foc}}(\pi_2\ (t\ u)))$, and similarly $\mathsf{NF}_{\mathsf{foc}}(t\ x) = (\mathsf{NF}_{\mathsf{foc}}(\pi_1\ (t\ x)), \mathsf{NF}_{\mathsf{foc}}(\pi_2\ (t\ x)))$: we can then conclude by induction hypothesis on those smaller pairs of terms $\pi_i\ (t\ u)$ and $\pi_i\ (t\ x)$ for i in $\{1,2\}$. We can thus assume that $t\ u$ is of positive or atomic type, and will reason by case analysis on the β-normal form of t.

If $\mathsf{NF}_\beta(t)$ is of the form $\lambda x.t'$ for some t', then $\mathsf{NF}_{\mathsf{foc}}(t\ u)$ is equal to $\mathsf{NF}_{\mathsf{foc}}((\lambda x.t')\ u)$, that is, $\mathsf{NF}_{\mathsf{foc}}(t'[u/x])$. Finally, we have $\mathsf{NF}_{\mathsf{foc}}(t\ x) = \mathsf{NF}_{\mathsf{foc}}((\lambda x.t')\ x) = \mathsf{NF}_{\mathsf{foc}}(t')$, which let us conclude from our assertion that $\mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t)\ u)$ is equal to $\mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t\ x)[u/x])$.

If $\mathsf{NF}_\beta(t)$ contains a strictly positive neutral subterm $n : P$ (this is in particular always the case when it is of the form $\texttt{match}\ t'\ \texttt{with}\ \ldots$, we can `let`-extrude it to get

$$\mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}'} \vdash t) \qquad = \texttt{let}\ x = \mathsf{NF}_{\mathsf{foc}}(n)\ \texttt{in}\ \mathsf{NF}_{\mathsf{foc}}(\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}'};x : P \vdash t[x/n])$$

But then $\mathsf{NF}_{\mathsf{foc}}(n) : P$ is also a strictly positive neutral subterm of $(\texttt{let}\ x = \mathsf{NF}_{\mathsf{foc}}(n)\ \texttt{in}\ \ldots)$, so we have

$$\begin{array}{rl} & \mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t)\ u) \\ = & \mathsf{NF}_{\mathsf{foc}}((\texttt{let}\ x = \mathsf{NF}_{\mathsf{foc}}(n)\ \texttt{in}\ \mathsf{NF}_{\mathsf{foc}}(t[x/n]))\ u) \\ = & \texttt{let}\ x = \mathsf{NF}_{\mathsf{foc}}(n)\ \texttt{in}\ \mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t[x/n])\ u[x/n]) \\ = & \texttt{let}\ x = \mathsf{NF}_{\mathsf{foc}}(n)\ \texttt{in}\ \mathsf{NF}_{\mathsf{foc}}((t\ u)[x/n]) \\ = & \mathsf{NF}_{\mathsf{foc}}(t\ u) \end{array}$$

Finally, if $\mathsf{NF}_{\beta}(t)$ contains no strictly positive neutral subterm, the rule REW-UP-ARROW applies: $\mathsf{NF}_{\mathsf{foc}}(t\ u)$ is of the form $n\ \mathsf{NF}_{\mathsf{foc}}(u)$, where $n \stackrel{\mathsf{def}}{=} \mathsf{NF}_{\mathsf{foc}}(t)$. In this case we also have $\mathsf{NF}_{\mathsf{foc}}(t\ x) = n\ x$, and thus

$$\begin{array}{rl} & \mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t)x\ u) \\ = & \mathsf{NF}_{\mathsf{foc}}(\mathsf{NF}_{\mathsf{foc}}(t\ x)[u/x]) \\ = & \mathsf{NF}_{\mathsf{foc}}(n\ u) \\ = & \mathsf{NF}_{\mathsf{foc}}(t\ u) \end{array}$$

Projection case $C[\Box] \stackrel{\mathsf{def}}{=} \pi_i\ \Box$ This case is proved in the same way as the application case: after some sum eliminations, the translation of t is an η-expansion of the product, which is related to the translations $\mathsf{NF}_{\mathsf{foc}}(\pi_i\ t)$, which either reduce the product or build a neutral term $\pi_i\ n$ after introducing some `let`-bindings.

Sum elimination case Reusing the notations of the application case, show that

$$\mathsf{NF}_{\mathsf{foc}}(\texttt{match}\ t\ \texttt{with} \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2) = \mathsf{NF}_{\mathsf{foc}}(\texttt{match}\ \mathsf{NF}_{\mathsf{foc}}(t)\ \texttt{with} \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2)$$

In the case of the function application or pair projection, the congruence proof uses the fact that the translation of t (of function or product type) necessarily starts with a λ-abstraction or pair construction – in fact, we follow the incremental construction of the first invertible phase, in particular we start by eliminating sums from the context.

In the case of the sum elimination, we must follow the translation into focused form further: we know the first invertible phase of $\mathsf{NF}_{\mathsf{foc}}(t)$ may only have sum-eliminations (pair or function introductions would be ill-typed as t has a sum type $A + B$).

As in the application case, we can then extrude neutrals from t, and the extrusion can be mirrored in both $\mathsf{NF}_{\mathsf{foc}}(\texttt{match}\ t\ \texttt{with}\ \ldots)$ and $\mathsf{NF}_{\mathsf{foc}}(\texttt{match}\ \mathsf{NF}_{\mathsf{foc}}(t)\ \texttt{with}\ \ldots)$. Finally, we reason by case analysis on $\mathsf{NF}_{\beta}(t)$.

If $\mathsf{NF}_{\beta}(t)$ is of the form $\sigma_i\ t'$, then we have

$$\begin{array}{rl} & \mathsf{NF}_{\mathsf{foc}}(\texttt{match}\ \mathsf{NF}_{\mathsf{foc}}(t)\ \texttt{with} \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2) \\ = & \mathsf{NF}_{\mathsf{foc}}(\texttt{match}\ \sigma_i\ \mathsf{NF}_{\mathsf{foc}}(t')\ \texttt{with} \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2) \\ = & \mathsf{NF}_{\mathsf{foc}}(u_i[\mathsf{NF}_{\mathsf{foc}}(t')/x]) \end{array}$$

and

$$\begin{aligned}
& \mathsf{NF}_{\mathsf{foc}}(\mathtt{match}\ t\ \mathtt{with} \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2\ ) \\
= \ & \mathsf{NF}_{\mathsf{foc}}(\mathtt{match}\ \mathsf{NF}_\beta(t)\ \mathtt{with} \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2\ ) \\
= \ & \mathsf{NF}_{\mathsf{foc}}(\mathtt{match}\ \sigma_i\ t'\ \mathtt{with} \mid \sigma_1\ x \to u_1 \mid \sigma_2\ x \to u_2\ ) \\
= \ & \mathsf{NF}_{\mathsf{foc}}(u_i[t'/x])
\end{aligned}$$

What is left to prove is that $\mathsf{NF}_{\mathsf{foc}}(u_i[\mathsf{NF}_{\mathsf{foc}}(t')/x]) = \mathsf{NF}_{\mathsf{foc}}(u_i[t'/x])$ but that is equivalent (by stability of the focusing translation by β-reduction) to $\mathsf{NF}_{\mathsf{foc}}((\lambda x.\, u_i)\ \mathsf{NF}_{\mathsf{foc}}(t')) = \mathsf{NF}_{\mathsf{foc}}((\lambda x.\, u_i)\ t')$, which is exactly the application case proved previously.

This is in fact the only possible case: when all strictly positive neutrals have been extruded, then $\mathsf{NF}_\beta(t)$ is necessarily an injection $\sigma_i\ t'$ (already handled) or a variable x (this corresponds to the case where t itself reduces to a strictly positive neutral), but this variable would be in the context and of strictly positive type, so this case is already handled as well. □

Theorem 6.7 (Canonicity of saturating focused logic)**.**
If we have $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathit{sinv}} t : N \mid Q^{\mathsf{at}}$ *and* $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathit{sinv}} u : N \mid Q^{\mathsf{at}}$ *in saturating focused logic with* $t \not\approx_{\mathtt{icc}} u$*, then* $t \not\approx_{\beta\eta} u$*.*

Proof. By contrapositive: if $t \approx_{\beta\eta} u$ (that is, if $\lfloor t \rfloor_{\mathtt{foc}} \approx_{\beta\eta} \lfloor u \rfloor_{\mathtt{foc}}$) then $t \approx_{\mathtt{icc}} u$.

The difficulty to prove this statement is that $\beta\eta$-equivalence does not preserve the structure of saturated proofs: an equivalence proof may go through intermediate steps that are neither saturated nor focused or in β-normal form.

We will thus go through an intermediate relation, which we will write $(\approx_{\mathtt{sat}})$, defined as follows on arbitrary well-typed lambda-terms:

$$\frac{\begin{array}{rr} \emptyset; \Sigma \vdash_{\mathsf{inv}} t : N \mid Q^{\mathsf{at}} & \emptyset; \Sigma \vdash_{\mathsf{inv}} u : N \mid Q^{\mathsf{at}} \\ \emptyset; \Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(t) \rightsquigarrow t' : N \mid Q^{\mathsf{at}} & \emptyset; \Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(u) \rightsquigarrow u' : N \mid Q^{\mathsf{at}} \\ \emptyset; \Sigma \vdash_{\mathit{sinv}} t' \rightsquigarrow t'' : N \mid Q^{\mathsf{at}} & \emptyset; \Sigma \vdash_{\mathit{sinv}} u' \rightsquigarrow u'' : N \mid Q^{\mathsf{at}} \\ \multicolumn{2}{c}{t'' \approx_{\mathtt{icc}} u''} \end{array}}{\Sigma \vdash t \approx_{\mathtt{sat}} u : N \mid Q^{\mathsf{at}}}$$

It follows from the previous results that if $t \approx_{\mathtt{sat}} u$, then $t \approx_{\beta\eta} u$. We will now prove the converse inclusion: if $t \approx_{\beta\eta} u$ (and they have the same type), then $t \approx_{\mathtt{sat}} u$ holds. In the particular case of terms that happen to be (let-expansions of) valid saturated focused derivations, this will tell us in particular that $t \approx_{\mathtt{icc}} u$ holds – the desired result.

The computational content of this canonicity proof is an equivalence algorithm: $(\approx_{\mathtt{sat}})$ is a decidable way to check for $\beta\eta$-equality, by normalizing terms to their saturated (or maximally multi-focused) structure.

β-reductions It is immediate that $(\approx_\beta)$ is included in $(\approx_{\mathtt{sat}})$. Indeed, if $t \approx_\beta u$ then $\mathsf{NF}_\beta(t) = \mathsf{NF}_\beta(u)$ and $t \approx_{\mathtt{sat}} u$ is trivially satisfied.

Negative η-expansions We can prove that if $t \approx_\eta u$ through one of the equations

$$(t : A \to B) \approx_\eta \lambda x.\, t\ x \qquad\qquad (t : A \times B) \approx_\eta (\pi_1\ t, \pi_2\ t)$$

then both t and u are rewritten in the same focused proof r. We have both $\emptyset;\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow r : N \mid \emptyset$ and $\emptyset;\Sigma \vdash_{\mathsf{inv}} u \rightsquigarrow r : N \mid \emptyset$, and thus $t \approx_{\mathtt{sat}} u$. Indeed we have:

$$\frac{\emptyset;\Sigma, x : P \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(t\ x) \rightsquigarrow r : N \mid \emptyset}{\emptyset;\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow \lambda x.\, r : P \to N \mid \emptyset}$$

$$\frac{\mathsf{NF}_\beta((\lambda x.\, t\ x)\ x) = \mathsf{NF}_\beta(t\ x) \qquad \emptyset;\Sigma, x : P \vdash_{\mathsf{inv}} \mathsf{NF}_\beta((\lambda x.\, t\ x)\ x) \rightsquigarrow r : N \mid \emptyset}{\emptyset;\Sigma \vdash_{\mathsf{inv}} \lambda x.\, t\ x \rightsquigarrow \lambda x.\, r : P \to N \mid \emptyset}$$

and

$$\frac{\forall i \in \{1,2\}, \quad \emptyset;\Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(\pi_i\ t) \rightsquigarrow r_i : N_i \mid \emptyset}{\emptyset;\Sigma \vdash_{\mathsf{inv}} t \rightsquigarrow (r_1, r_2) : (N_1, N_2) \mid \emptyset}$$

$$\frac{\pi_i\ (\pi_1\ t, \pi_2\ t) = t \qquad \forall i \in \{1,2\}, \quad \emptyset;\Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(\pi_i\ (\pi_1\ t, \pi_2\ t)) \rightsquigarrow r_i : N_i \mid \emptyset}{\emptyset;\Sigma \vdash_{\mathsf{inv}} (\pi_1\ t, \pi_2\ t) \rightsquigarrow (r_1, r_2) : N_1 \times N_2 \mid \emptyset}$$

Positive η-expansion: sum type The interesting case is the positive η-expansion

$$\forall C[\Box : \lfloor P_1 \rfloor_\pm + \lfloor P_2 \rfloor_\pm], \quad C[t] \approx_\eta \mathtt{match}\ t\ \mathtt{with} \mid \begin{array}{l} \sigma_1\ x \to C[\sigma_1\ x] \\ \sigma_2\ x \to C[\sigma_2\ x] \end{array}$$

We do a case analysis on the (weak head) β-normal form of t. If it is an injection of the form $\sigma_i\ t'$, then the equation becomes true by a simple β-reduction:

$$\mathtt{match}\ \sigma_i\ t'\ \mathtt{with} \mid \begin{array}{l} \sigma_1\ x \to C[\sigma_1\ x] \\ \sigma_2\ x \to C[\sigma_2\ x] \end{array} \quad \rightsquigarrow_\beta C[\sigma_i\ t']$$

Otherwise the β-normal form of t is a term of sum type that does not start with an injection. In particular, $\mathsf{NF}_\beta(t)$ is not reduced when reducing the whole term $C[t]$ (only possibly duplicated): for some multi-hole context $C'[x]$ we have $\mathsf{NF}_\beta(C[t]) = C'[\mathsf{NF}_\beta(t)]$ and

$$\mathsf{NF}_\beta(\mathtt{match}\ t\ \mathtt{with} \mid \begin{array}{l} \sigma_1\ x \to C[\sigma_1\ x] \\ \sigma_2\ x \to C[\sigma_2\ x] \end{array}) \quad = \quad \mathtt{match}\ \mathsf{NF}_\beta(t)\ \mathtt{with} \mid \begin{array}{l} \sigma_1\ x \to C'[\sigma_1\ x] \\ \sigma_2\ x \to C'[\sigma_2\ x] \end{array}$$

Without loss of generality, we can assume that $\mathsf{NF}_\beta(t)$ is a neutral term. Indeed, if it is not, it starts with a (possibly empty) series of non-invertible elimination forms, applied to a positive elimination – which is itself either a neutral or of this form. It eventually contains a neutral strict subterm of strictly positive type valid in the current scope. The focused translation can then cut on this strictly positive neutral. If it is a sum type, the translation splits on it, and replace occurrences of this neutral with either $\sigma_1\ z$ or $\sigma_2\ z$ for some fresh z. This can be done on both terms equated by the η-equivalence for sums, and returns (two pairs of) η-equivalent terms with one less possible neutral strict subterm.

Let $n \stackrel{\mathsf{def}}{=} \mathsf{NF}_\beta(t)$. It remains to show that the translations of $C'[n]$ is equal modulo $(\approx_{\mathtt{icc}})$ to the translation of $\mathtt{match}\ n\ \mathtt{with} \mid \begin{array}{l} \sigma_1\ x \to C'[\sigma_1\ x] \\ \sigma_2\ x \to C'[\sigma_2\ x] \end{array}$. In fact, we show that they translate

to the same focused proof:

$$\dfrac{\dfrac{\Gamma^{\mathsf{at}} \vdash n : P_1 + P_2 \qquad \Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow P_1 + P_2 \qquad \Gamma^{\mathsf{at}}; x : P_1 \vdash_{\mathsf{inv}} C'[\sigma_1\ x] \rightsquigarrow r_1 : \emptyset \mid Q^{\mathsf{at}} \qquad \Gamma^{\mathsf{at}}; x : P_2 \vdash_{\mathsf{inv}} C'[\sigma_2\ x] \rightsquigarrow r_2 : \emptyset \mid Q^{\mathsf{at}}}{\Gamma^{\mathsf{at}}; x : P_1 + P_2 \vdash_{\mathsf{inv}} C'[x] \rightsquigarrow \mathtt{match}\ x\ \mathtt{with} \mid \begin{matrix}\sigma_1\ x \to r_1 \\ \sigma_2\ x \to r_2\end{matrix} : \emptyset \mid Q^{\mathsf{at}}}}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} C'[n] \rightsquigarrow \mathtt{let}\ x = n\ \mathtt{in}\ \mathtt{match}\ x\ \mathtt{with} \mid \begin{matrix}\sigma_1\ x \to r_1 \\ \sigma_2\ x \to r_2\end{matrix} : Q^{\mathsf{at}}}$$

$$\dfrac{\dfrac{\begin{matrix}\Gamma^{\mathsf{at}} \vdash n : P_1 + P_2 \\ \Gamma^{\mathsf{at}} \vdash n \rightsquigarrow n' \Downarrow P_1 + P_2 \qquad \mathsf{NF}_\beta(\mathtt{match}\ \sigma_i\ x\ \mathtt{with} \mid \begin{matrix}\sigma_1\ x \to C'[\sigma_1\ x] \\ \sigma_2\ x \to C'[\sigma_2\ x]\end{matrix}) = C'[\sigma_i\ x] \\ \Gamma^{\mathsf{at}}; x : P_1 \vdash_{\mathsf{inv}} C'[\sigma_1\ x] \rightsquigarrow r_1 : \emptyset \mid Q^{\mathsf{at}} \qquad \Gamma^{\mathsf{at}}; x : P_2 \vdash_{\mathsf{inv}} C'[\sigma_2\ x] \rightsquigarrow r_2 : \emptyset \mid Q^{\mathsf{at}}\end{matrix}}{\Gamma^{\mathsf{at}}; x : P_1 + P_2 \vdash_{\mathsf{inv}} \mathtt{match}\ x\ \mathtt{with} \mid \begin{matrix}\sigma_1\ x \to C'[\sigma_1\ x] \\ \sigma_2\ x \to C'[\sigma_2\ x]\end{matrix} \rightsquigarrow \mathtt{match}\ x\ \mathtt{with} \mid \begin{matrix}\sigma_1\ x \to r_1 \\ \sigma_2\ x \to r_2\end{matrix} : \emptyset \mid Q^{\mathsf{at}}}}{\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} \mathtt{match}\ n\ \mathtt{with} \mid \begin{matrix}\sigma_1\ x \to C'[\sigma_1\ x] \\ \sigma_2\ x \to C'[\sigma_2\ x]\end{matrix} \rightsquigarrow \mathtt{let}\ x = n\ \mathtt{in}\ \mathtt{match}\ x\ \mathtt{with} \mid \begin{matrix}\sigma_1\ x \to r_1 \\ \sigma_2\ x \to r_2\end{matrix} : Q^{\mathsf{at}}}$$

Transitivity Given $t \approx_{\mathtt{sat}} u$ and $u \approx_{\mathtt{sat}} r$, do we have $t \approx_{\mathtt{sat}} r$? In the general case we have

$$\dfrac{\begin{matrix}\emptyset; \Sigma \vdash_{\mathsf{inv}} t : A \mid \emptyset \qquad \emptyset; \Sigma \vdash_{\mathsf{inv}} u : A \mid \emptyset \\ \emptyset; \Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(t) \rightsquigarrow t' : A \mid \emptyset \qquad \emptyset; \Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(u) \rightsquigarrow u'_1 : A \mid \emptyset \\ \emptyset; \Sigma \vdash_{\mathsf{sinv}} t' \rightsquigarrow t'' : A \mid \emptyset \qquad \emptyset; \Sigma \vdash_{\mathsf{sinv}} u'_1 \rightsquigarrow u''_1 : A \mid \emptyset \\ t'' \approx_{\mathtt{icc}} u''_1\end{matrix}}{\Sigma \vdash t \approx_{\mathtt{sat}} u : A}$$

$$\dfrac{\begin{matrix}\emptyset; \Sigma \vdash_{\mathsf{inv}} u : A \mid \emptyset \qquad \emptyset; \Sigma \vdash_{\mathsf{inv}} r : A \mid \emptyset \\ \emptyset; \Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(u) \rightsquigarrow u'_2 : A \mid \emptyset \qquad \emptyset; \Sigma \vdash_{\mathsf{inv}} \mathsf{NF}_\beta(r) \rightsquigarrow r' : A \mid \emptyset \\ \emptyset; \Sigma \vdash_{\mathsf{sinv}} u'_2 \rightsquigarrow u''_2 : A \mid \emptyset \qquad \emptyset; \Sigma \vdash_{\mathsf{sinv}} r' \rightsquigarrow r'' : A \mid \emptyset \\ u''_2 \approx_{\mathtt{icc}} r''\end{matrix}}{\Sigma \vdash u \approx_{\mathtt{sat}} r : A}$$

By Lemma 6.5 (Determinacy of saturated translation) we have that $u''_1 \approx_{\mathtt{icc}} u''_2$. Then, by transitivity of $(\approx_{\mathtt{icc}})$:

$$t'' \approx_{\mathtt{icc}} u''_1 \approx_{\mathtt{icc}} u''_2 \approx_{\mathtt{icc}} r''$$

Congruence If $\Sigma \vdash t_1 \approx_{\mathtt{sat}} t_2 : A$, do we have that $C[t_1] \approx_{\mathtt{sat}} C[t_2]$ for any term context C?

This is an immediate application of Lemma 6.6 (Saturation congruence): it tells us that $\mathsf{NF}_{\mathsf{sat}}(C[t_1]) \approx_{\mathtt{icc}} \mathsf{NF}_{\mathsf{sat}}(C[\mathsf{NF}_{\mathsf{sat}}(t_1)])$ and $\mathsf{NF}_{\mathsf{sat}}(C[t_1]) \approx_{\mathtt{icc}} \mathsf{NF}_{\mathsf{sat}}(C[\mathsf{NF}_{\mathsf{sat}}(t_2)])$. So, by transitivity of $(\approx_{\mathtt{icc}})$ we only have to prove $\mathsf{NF}_{\mathsf{sat}}(C[\mathsf{NF}_{\mathsf{sat}}(t_1)]) \approx_{\mathtt{icc}} \mathsf{NF}_{\mathsf{sat}}(C[\mathsf{NF}_{\mathsf{sat}}(t_1)])$, which is a consequence of our assumption $\mathsf{NF}_{\mathsf{sat}}(t_1) \approx_{\mathtt{icc}} \mathsf{NF}_{\mathsf{sat}}(t_2)$ and congruence of $(\approx_{\mathtt{icc}})$. □

7 Unique inhabitation algorithm

The saturating focused logic corresponds to a computationally complete presentation of the structure of canonical proofs we are interested in. From this presentation it is extremely easy to derive a terminating search algorithm complete for unicity – we moved from a whiteboard description of the saturating rules to a working implementation of the algorithm usable on actual examples in exactly one day of work. The implementation [Scherer, 2015b] is around 700 lines of readable OCaml code.

In Section 1.9 (Termination), we justified the decidability of inhabitation for propositional logic. Decidability results for quantifier-free logics are easily obtained by constructing a search space, for the proofs of a given judgment, that is both complete for provability (it contains a proof it he judgment is at all provable) and finite. Three key observations were used to exhibit this finite search space:

1. Cut-free proofs in propositional logic have the subformula property, which bounds the formula appearing in the proof the finite set of sub-formulas of the root judgment.
2. The contexts of the logic are *sets* of formulas, and in particular the set of contexts over the finite set of formulas is finite. Thus, the set of possible judgments is finite.
3. We can restrict ourselves to the subset of proof where, along any path of the proof tree, all judgments occurs at most once – and all provable formulas remain provable under that restriction. This sub-system of *recurrence-free* proofs is thus complete for provability, and is finite – as the set of possible judgments is finite.

In the present section, we would like to justify our implementation by proposing a similarly finite subsystem of our saturation logic, which enjoys canonical proofs. The goal is to be able to decide whether a type is uniquely inhabited by exploring this subsystem, so it should be unicity complete.

The subformula property is preserved in saturated proof terms, which are cut-free proofs with additional structure. But the two other restrictions above are too brutal for our needs. They preserve completeness for provability, but they lose many computational behaviors, they break computational completeness and even unicity completeness. We refine them into two restrictions that give us finiteness (and, in particular, break computational completeness for types with infinitely many distinct inhabitants) but preserve unicity completeness, and in fact let us enumerate at least n different inhabitants if they exist.

1. To detect non-unicity, it suffices to keep at most *two* variables of each type in the context. This suggest a definition of contexts as 2-bounded multisets of formulas, which give a finite context space over a finite space of formulas. The fact that this restriction is unicity complete was proved in Section 4 (Counting terms and proofs).
2. Similarly, we restrict ourselves to the subset of proofs where, along any path of the proof tree, all judgments occur at most two times. This relaxation of the *recurrence-free* criterion suffices to recover completeness for unicity, as we shall prove in this section.

7.1 Implementing search

7.1.1 Implementation overview

The central idea to cut the search space while remaining complete for unicity is the *two-or-more* approximation. We use a *plurality* monad `Plur`, defined in set-theoretic terms as $\texttt{Plur}(S) \stackrel{\text{def}}{=} 1 + S + S \times S$, representing zero, one or "at least two" distinct elements of the set S. Each typing judgment is reformulated into a search function which takes as input the context(s) of the judgment and its goal, and returns a plurality of proof terms – we search not for *one* proof term, but for (a bounded set of) *all* proof terms. Reversing the usual mapping from variables to types, the contexts map types to pluralities of formal variables – just as we did in Section 4 (Counting terms and proofs).

In the search algorithm, the SINV-END rule does not merely pass its new context Γ' to the saturation rules, but it also *trims* it by applying the two-or-more rule: if the old context Γ already has two variables of a given formula N, drop all variables for N from Γ'; if it already has one variable, retain at most one variable in Γ'. This amounts to defining a selection function $\texttt{Select}_{\Gamma,\Gamma'}(\_)$ for use in the SAT rule. This trimming respects the selection requirement SELECT-SPECIF, as it always keep at least one proof of each formula provable in either Γ or Γ'. Proving that it is complete for unicity was the topic of Section 4 (Counting terms and proofs).

To effectively implement the saturation rules, a useful tool is an *obligation search* function (called `select_oblis` in our prototype) which takes a selection predicate on positive or atomic formulas P^{at}, and searches for (a plurality of) each negative formula N from the context that might be the starting point of an elimination judgment of the form $\Gamma \vdash n \Downarrow P^{\mathsf{at}}$, for a P^{at} accepted by the selection predicate. For example, if we want to prove X and there is a formula $Y \to Z \times X$, this formula will be part of the search results – although we do not know yet if we will be able to prove Y. For each such P^{at}, it returns a *proof obligation*, that is either a valid derivation of $\Gamma \vdash n \Downarrow P^{\mathsf{at}}$, or a *request*, giving some formula Q and expecting a derivation of $\Gamma \vdash ? \Uparrow Q^{\mathsf{at}}$ before returning another proof obligation for P^{at}.

The rule SAT-ATOM $(\Gamma; \emptyset \vdash_{\mathsf{sat}} ? : X^-)$ uses this obligation search function to search for all negatives that could potentially be eliminated into a X^-, and feeding (pluralities of) answers to the returned proof obligations (by recursively searching for introduction judgments) to obtain (pluralities of) elimination proofs of X^-.

The rule SAT uses the selection function to find the negatives that could be eliminated in any strictly positive formula and tries to fullfill (pluralities of) proof obligations. This returns a binding context (with a plurality of neutrals for each positive formula), which is filtered a posteriori to keep only the "new" bindings – that use the new context. The new binding are all added to the search environment, and saturating search is called recursively. It returns a plurality of proof terms; each of them results in a proof derivation (where the saturating set is trimmed to retain only the bindings useful to that particular proof term).

Finally, to ensure termination while remaining complete for unicity, we do not search for proofs where a given subgoal occurs strictly more than twice along a given search path. This is easily implemented by threading an extra "memory" argument through each recursive call, which counts the number of identical subgoals below a recursive call and kills the search (by returning the "zero" element of the plurality monad) at two. Note

that this does not correspond to memoization in the usual sense, as information is only propagated along a recursive search branch, and never shared between several branches.

This fully describes the algorithm, which is easily derived from the logic. It is effective, and our implementation answers instantly on all the (small) types of polymorphic functions we tried. But it is not designed for efficiency, and in particular saturation duplicates a lot of work (re-computing old values before throwing them away).

We can give a presentation of the algorithm as a system of inference rules that is terminating and deterministic. Using the two-or-more counting approximation result of Section 4 (Counting terms and proofs), we can prove the correctness of this presentation.

7.1.2 A formal description of the algorithm

In Figure 23 (Saturation algorithm) we present a complete set of inference rules that captures the behavior of our search algorithm.

Data structures The judgments uses several kinds data-structures.

- 2-*sets* $S, T \ldots$, are sets restricted to having at most two (distinct) elements; we use $\{\ldots\}_2$ to build a 2-set, and $(\cup_2)$ for union of two-sets (keeping at most two elements in the resulting union). We use the usual notation $x \in S$ for 2-set membership. To emphasize the distinction, we will sometimes write $\{\ldots\}_\infty$ for the usual, unbounded sets. Remark that 2-sets correspond to the "plurality monad" of Section 7.1.1 (Implementation overview): a monad is more convenient to use in an implementation, but for inference rules we use the set-comprehension notation.
- 2-*mappings* are mappings from a set of keys to 2-sets. In particular, Γ^{at} denotes a 2-mapping from negative or atomic types to 2-sets of formal variables. We use the application syntax $\Gamma^{\mathsf{at}}(N^{\mathsf{at}})$ for accessing the 2-set bound to a specific key, $N^{\mathsf{at}} \mapsto S$ for the singleton mapping from one variable to one 2-set, and $(\oplus)$ for the union of 2-mappings, which applies $(\cup_2)$ pointwise:

  $$(\Gamma^{\mathsf{at}} \oplus \Gamma^{\mathsf{at}\prime})(N^{\mathsf{at}}) \stackrel{\mathsf{def}}{=} \Gamma^{\mathsf{at}}(N^{\mathsf{at}}) \cup_2 \Gamma^{\mathsf{at}\prime}(N^{\mathsf{at}})$$

  Finally, we write $\emptyset$ for the mapping that maps any key to the empty 2-set.
- *multisets* M are mappings from elements to a natural number count. The "memories" of subgoal ancestors are such mappings (where the keys are "judgments" of the form $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}$), and our rules will guarantee that the value of any key is at most 2. We use the application syntax $M(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}})$ to access the count of any element, and $(+)$ for pointwise addition of multisets:

  $$(M + M')(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}) \stackrel{\mathsf{def}}{=} M(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}) + M'(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}})$$

- (ordered) *lists* Σ of strictly positive formulas.

Finally, we use a *substraction operation* $(-_2)$ between 2-mappings, that can be defined from the 2-*set restriction* operation $S \setminus_2 n$ (where n is a natural number in $\{0,1,2\}$). Recall

that $\mathsf{cardinal}(S)$ is the cardinal of the set (or 2-set) S.

$$(\Gamma^{\mathsf{at}\prime} -_2 \Gamma^{\mathsf{at}})(N^{\mathsf{at}}) \stackrel{\mathsf{def}}{=} \Gamma^{\mathsf{at}\prime}(N^{\mathsf{at}}) \setminus_2 \mathsf{cardinal}(\Gamma^{\mathsf{at}}(N^{\mathsf{at}}))$$

$$S \setminus_2 0 \stackrel{\mathsf{def}}{=} S \qquad \emptyset \setminus_2 1 \stackrel{\mathsf{def}}{=} \emptyset \qquad \{a, \dots\}_2 \setminus_2 1 \stackrel{\mathsf{def}}{=} \{a\}_2 \qquad S \setminus_2 2 \stackrel{\mathsf{def}}{=} \emptyset$$

Note that $\{a,b\} \setminus_2 1$ is not uniquely defined: it could be either a or b, the choice does not matter. The defining property of $S \setminus_2 n$ is that it is a minimal 2-set S' such as $S' \cup_2 T = S$ for some set T.

Judgments The algorithm is presented as a system of judgment-directed (that is, directed by the types in the goal and the context(s)) inference rules. It uses the following five judgment forms:

- invertible judgments $M \mathbin{@} \Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime}; \Sigma \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : N \mid Q^{\mathsf{at}}$
- saturation judgments $M \mathbin{@} \Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime} \vdash^{\mathsf{alg}}_{\mathsf{sat}} S : Q^{\mathsf{at}}$
- post-saturation judgments $M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}}_{\mathsf{post}} S : Q^{\mathsf{at}}$
- introduction judgments $M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Uparrow P$
- elimination judgments $M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Downarrow N$

All algorithmic jugments respect the same conventions:

- M is a *memory* (remembering ancestors judgments for termination), a multiset of judgments of the form $\Gamma \vdash A$
- $\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}\prime}$ are 2-mappings from negative or atomic types to 2-sets of formal variables (we will call those "contexts")
- Σ is an ordered list of pairs $x : P$ of formal variables and positive types
- S is a 2-set of proof terms of the saturating focused logic

The S position is the output position of each judgment (the algorithm returns a 2-set of distinct proof terms); all other positions are input positions; any judgment has exactly one applicable rule, determined by the value of its input positions.

Sets of terms We extend the term construction operations to 2-sets of terms:

$$\begin{array}{lcl}
\lambda x.S & \stackrel{\mathsf{def}}{=} & \{\lambda x.t \mid t \in S\}_2 \\
S\ T & \stackrel{\mathsf{def}}{=} & \{t\ u \mid t \in S, u \in T\}_2 \\
(S,T) & \stackrel{\mathsf{def}}{=} & \{(t,u) \mid t \in S, u\ in T\}_2 \\
\pi_i\ S & \stackrel{\mathsf{def}}{=} & \{\pi_i\ t \mid t \in S\}_2 \\
\sigma_i\ S & \stackrel{\mathsf{def}}{=} & \{\sigma_i\ t \mid t \in S\}_2 \\
\mathtt{match}\ x\ \mathtt{with} \left| \begin{array}{l} \sigma_1\ x \to S_1 \\ \sigma_2\ x \to S_2 \end{array} \right. & \stackrel{\mathsf{def}}{=} & \{\mathtt{match}\ x\ \mathtt{with} \left| \begin{array}{l} \sigma_1\ x \to t_1 \\ \sigma_2\ x \to t_2 \end{array} \right. \mid t_i \in S_i\}_2
\end{array}$$

Invertible rules The invertible focused rules $\Gamma^{\mathsf{at}}; \Sigma \vdash_{\mathsf{inv}} ? : N \mid Q^{\mathsf{at}}$ exhibit "don't care" non-determinism in the sense that their order of application is irrelevant and captured by invertible commuting conversions (see Section 3.2.1). In the algorithmic judgment, we enforce a specific order through the two following restrictions.

Fig. 23. Saturation algorithm

ALG-SINV-SUM

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};x : P_1,\Sigma \vdash^{\mathsf{alg}}_{\mathsf{inv}} S_1 : N \mid Q^{\mathsf{at}} \qquad M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};x : P_2,\Sigma \vdash^{\mathsf{alg}}_{\mathsf{inv}} S_2 : N \mid Q^{\mathsf{at}}}{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};x : P_1 + P_2,\Sigma \vdash^{\mathsf{alg}}_{\mathsf{inv}} \mathtt{match}\ x\ \mathtt{with} \mid \begin{array}{l}\sigma_1\ x \to S_1 \\ \sigma_2\ x \to S_2\end{array} : N \mid Q^{\mathsf{at}}}$$

ALG-SINV-PROD

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{inv}} S_1 : N_1 \mid \emptyset \qquad M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{inv}} S_2 : N_2 \mid \emptyset}{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{inv}} (S_1,S_2) : N_1 \times N_2 \mid \emptyset}$$

ALG-SINV-ARR

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};x : P \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : N \mid \emptyset}{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{inv}} \lambda x.S : P \to N \mid \emptyset}$$

ALG-SINV-RELEASE

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \oplus (N^{\mathsf{at}\prime} \mapsto \{x\}_2);\Sigma \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : N \mid Q^{\mathsf{at}}}{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};x : \left\langle N^{\mathsf{at}\prime} \right\rangle^{+\mathsf{at}},\Sigma \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : N \mid Q^{\mathsf{at}}}$$

ALG-SINV-END

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}};(\Gamma^{\mathsf{at}\prime} -_2 \Gamma^{\mathsf{at}}) \vdash^{\mathsf{alg}}_{\mathsf{sat}} S : Q^{\mathsf{at}}}{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : \emptyset \mid Q^{\mathsf{at}}}$$

ALG-SAT-KILL

$$\frac{M(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}) = 2}{M \mathbin{@} \Gamma^{\mathsf{at}};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{sat}} \emptyset : Q^{\mathsf{at}}}$$

ALG-SAT-POST

$$\frac{M(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}) < 2 \qquad M \oplus_2 (\Gamma \vdash P) \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}}_{\mathsf{post}} S : Q^{\mathsf{at}}}{M \mathbin{@} \Gamma^{\mathsf{at}};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{sat}} S : Q^{\mathsf{at}}}$$

ALG-POST-INTRO

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Uparrow P}{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}}_{\mathsf{post}} S : P}$$

ALG-POST-ATOM

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Downarrow X^-}{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}}_{\mathsf{post}} S : X^-}$$

ALG-SAT

$$\frac{\begin{array}{c}\Gamma' \neq \emptyset \\ \forall (P \mid P \text{ subformula } (\Gamma^{\mathsf{at}},\Gamma^{\mathsf{at}\prime})), \quad S_P \stackrel{\mathsf{def}}{=} \bigcup\nolimits_2 \{S_{\mathsf{ne}} \mid M \mathbin{@} \Gamma,\Gamma' \vdash^{\mathsf{alg}} S_{\mathsf{ne}} \Downarrow P\} \\ B \stackrel{\mathsf{def}}{=} \bigoplus_P \{P \mapsto \{x_n\}_2 \mid n \in S_P\} \\ M \mathbin{@} \Gamma,\Gamma';\emptyset;B \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : \emptyset \mid Q^{\mathsf{at}} \\ S' \stackrel{\mathsf{def}}{=} \left\{ \mathtt{let}\ \bar{x} = \bar{n}\ \mathtt{in}\ t \;\middle|\; \begin{array}{l} t \in S, \\ (\bar{x},\bar{n}) \stackrel{\mathsf{def}}{=} \{(x_n,n) \mid \exists P,\ x_n \in B(P)\}_\infty \end{array} \right\}_2\end{array}}{M \mathbin{@} \Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}\prime} \vdash^{\mathsf{alg}}_{\mathsf{sat}} S' : Q^{\mathsf{at}}}$$

ALG-SINTRO-SUM

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S_1 \Uparrow P_1 \qquad M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S_2 \Uparrow P_2}{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} (\sigma_1\ S_1) \cup_2 (\sigma_2\ S_2) \Uparrow P_1 + P_2}$$

ALG-SINTRO-VAR

$$\frac{S \stackrel{\mathsf{def}}{=} \{x \mid (x : X^+) \in \Gamma^{\mathsf{at}}\}_2}{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Uparrow X^+}$$

ALG-SINTRO-END

$$\frac{M \mathbin{@} \Gamma^{\mathsf{at}};\emptyset;\emptyset \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : N \mid \emptyset}{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Uparrow \langle N \rangle^-}$$

ALG-SELIM

$$\frac{\begin{array}{c}N \text{ subformula } \Gamma^{\mathsf{at}} \\ S_{\mathsf{var}} \stackrel{\mathsf{def}}{=} \Gamma^{\mathsf{at}}(N) \\ S_{\mathsf{proj}} \stackrel{\mathsf{def}}{=} \bigcup\nolimits_2 \{\pi_i\ S \mid M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Downarrow M_1 \times M_2, M_i = N\} \\ S_{\mathsf{app}} \stackrel{\mathsf{def}}{=} \bigcup\nolimits_2 \{S\ T \mid M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Downarrow P \to N, M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} T \Uparrow P\}\end{array}}{M \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S_{\mathsf{var}} \cup_2 S_{\mathsf{proj}} \cup_2 S_{\mathsf{app}} \Downarrow N}$$

First, the negative or atomic formulas that are shifted in the positive context Σ are moved incrementally to a temporary context $\Gamma^{\mathsf{at}'}$. By using an ordered list for the positive context, we fix the order in which positives are deconstructed. When the head of the ordered list has been fully deconstructed (it is negative or atomic), the new rule ALG-SINV-RELEASE moves it into $\Gamma^{\mathsf{at}'}$.

Second, the invertible right-introduction rules are restricted to judgments whose ordered context Σ is empty. This enforces that left-introductions are always applied fully before any right-introduction. Note that we could arbitrarily decide to enforce the opposite order by un-restricting right-introduction rules, and requiring that left-introduction (and releases) only happen when the succedent is positive or atomic.

After the decomposition of Σ is finished, the final invertible rule ALG-SINV-END uses 2-mapping substractions $\Gamma^{\mathsf{at}} -_2 \Gamma^{\mathsf{at}'}$ to trim the new context $\Gamma^{\mathsf{at}'}$ before handing it to the saturation rules: for any given formula N^{at}, all bindings for N^{at} are removed from $\Gamma^{\mathsf{at}'}$ if there are already two in Γ^{at}, and at most one binding is kept if there is already one in Γ^{at}. Morally, the reason why it is *correct* to trim (that is, it does not endanger unicity completeness is that the next rules in bottom-up search will only use the merged context $\Gamma^{\mathsf{at}} \cup_2 \Gamma^{\mathsf{at}'}$ (which is preserved by trimming by construction of $(-_2)$), or saturate with bindings from $\Gamma^{\mathsf{at}'}$. Any strictly positive that can be deduced by using one of the variables present in $\Gamma^{\mathsf{at}'}$ but removed from $\Gamma^{\mathsf{at}} \cup_2 \Gamma^{\mathsf{at}'}$ has already been deduced from Γ^{at}. It is *useful* to trim in this rule (we could trim much more often) because subsequent saturated rules will test the new context $\Gamma^{\mathsf{at}'} -_2 \Gamma^{\mathsf{at}}$ for emptyness, so it is interesting to minimize it. In any case, we need to trim in at least one place in order for typing judgments not to grow unboundedly.

Saturation rules If the (trimmed) new context is empty, we test whether the judgment of the current subgoal has already occurred twice among its ancestors; in this case, the rule ALG-SAT-KILL terminates the search process by returning the empty 2-set of proof terms. In the other case, the number of occurrences of this judgment is incremented in the rule ALG-SAT-POST, and one of the (transparent) "post-saturation" rules ALG-POST-INTRO or ALG-POST-ATOM are applied.

This is the only place where the memory M is accessed and updated. The reason why this suffices is any given phase (invertible phase, or phase of non-invertible eliminations and introductions) is only of finite length, and either terminates or is followed by a saturation phase; because contexts grow monotonously in a finite space (of 2-mappings rather than arbitrary contexts), the trimming of rule ALG-SINV-END returns the empty context after a finite number of steps: an infinite search path would need to go through ALG-SAT-POST infinitely many times, and this suffices to prove termination.

The most important and complex rule is ALG-SAT, which proceeds in four steps. First, we compute the 2-set S_P of all ways to deduce any strict positive P from the context – for any P we need not remember more than two ways. We know that we need only look for P that are deducible by elimination from the context $\Gamma^{\mathsf{at}}, \Gamma^{\mathsf{at}'}$ – the finite set of subformulas is a good enough approximation. Because we retain at least one neutral of each newly provable positive P, this algorithm corresponds to a selection function that satisfies SELECT-SPECIF.

Second, we build a context B binding a new formal variable x_n for each elimination neutral n – it is crucial for canonicity that all n are new and semantically distinct from each

other at this point, otherwise duplicate bindings would be introduced. Third, we compute the 2-set S of all possible (invertible) proofs of the goal under this saturation context B, and add the `let`-bindings to those proof terms in the final returned 2-set.

Non-invertible introduction and elimination rules The introduction rule ALG-SINTRO-SUM collects solutions using either left or right introductions, and unites them in the result 2-set. Similarly, all elimination rules are merged in one single rule ALG-SELIM, which corresponds to all ways to deduce a given formula N: directly from the context, by projection of a pair, or application of a function. The search space for this sequent is finite, as goal types grow strictly at each type, and we can kill search for any type that does not appear as a subformula of the context.

(The inference-rule presentation differs from our OCaml implementation at this point. The implementation is more effective, it uses continuation-passing style to attempt to provide function arguments only for the applications we know are found in context and may lead to the desired result. Such higher-order structure is hard to render in an inference rule, so we approximated it with a more declarative presentation here. This is the only such simplification.)

7.2 Correctness

Lemma 7.1 (Termination)**.**
The algorithmic inference system only admits finite derivations.

Proof. We show that each inference rule is of finite degree (it has a finite number of premises), and that there exists no infinite path of inference rules – concluding with König's Lemma.

Degree finiteness The rules that could be of infinite degree are ALG-SAT (which quantifies over all positives P) and ALG-SELIM (which quantifies over arbitrarily many elimination derivations). But both rules have been restricted through the subformula property to only quantify on finitely many formulas (ALG-SAT) or possible elimination schemes (ALG-SELIM).

Infinite paths lead to absurdity We first assert that any given phase (invertible, saturation, introductions/eliminations) may only be of finite length. Indeed, invertible rules have either the context or the goal decreasing structurally. Saturation rules are either ALG-SAT if $\Gamma^{\mathsf{at}'} \neq \emptyset$, which is immediately followed by elimination and invertible rules, or ALG-SAT-KILL or ALG-SAT-POST if $\Gamma^{\mathsf{at}'} = \emptyset$, in which case the derivation either terminates or continues with a non-invertible introduction or elimination. Introductions have the goal decreasing structurally, and eliminations have the goal *increasing* structurally, and can only form valid derivations if it remains a subformula of the context Γ^{at}.

Given that any phase is finite, any infinite path will necessarily have an infinite number of phase alternation. By looking at the graph of phase transitions (invertible goes to saturating which goes to introductions or eliminations, which go to invertible), we see that each phase will occur infinitely many times along an infinite path. In particular, an infinite path would

have infinitely many invertible and saturation phases; the only transition between them is the rule ALG-SINV-END which must occur infinitely many times in the path.

Now, because the rules grow the context monotonically, an infinite path must eventually reach a maximal stable context Γ^{at}, that never grows again along the path. In particular, for infinitely many ALG-SINV-END we have Γ^{at} maximal and thus $\Gamma^{at'} -_2 \Gamma^{at} = \emptyset$ – if the trimming was not empty, $\Gamma^{at'}$ would grow strictly after the next saturation phase, while we assumed it was maximal.

This means that either ALG-SAT-KILL or ALG-SAT-POST incurs infinitely many times along the infinite path. Those rules check the memory count of the current (context, goal) pair $\Gamma^{at} \vdash_{foc} Q^{at}$. Because of the subformula property (formulas occurring in subderivations are subformulas of the root judgment concluding the complete proof), there can be only finitely many different $\Gamma^{at} \vdash_{foc} Q^{at}$ pair (Γ^{at} is a 2-mapping which grows monotonically).

An infinite path would thus necessarily have infinitely many steps ALG-SAT-KILL or ALG-SAT-POST with the same (context, goal) pair. This is impossible, as a given pair can only go at most twice through ALG-SAT-POST, and going through ALG-SAT-KILL terminates the path. There is no infinite path. □

Lemma 7.2 (Totality and Determinism)**.**
For any algorithmic judgment there is exactly one applicable rule.

Proof. Immediate by construction of the rules. Invertible rules $M \mathbin{@} \Gamma^{at}; \Gamma^{at'}; \Sigma \vdash^{alg}_{inv} S : N \mid Q^{at}$ are directed by the shape of the context Σ and the goal N. Saturation rules $M \mathbin{@} \Gamma^{at}; \Gamma^{at'} \vdash^{alg}_{sat} S : Q^{at}$ are directed by the new context $\Gamma^{at'}$. If $\Gamma^{at'} = \emptyset$, the memory $M(\Gamma^{at} \vdash_{foc} Q^{at})$ decides whether to kill or post-saturate, in which case the shape of the goal (either strict positive or atomic) directs the post-saturation rule. Finally, non-invertible introductions $M \mathbin{@} \Gamma^{at} \vdash^{alg} S \Uparrow P$ are directed by the goal P, and there is exactly one non-invertible elimination rule. □

Remark 7.1. The choice we made to restrict the ordering of invertible rules is not necessary – we merely wanted to demonstrate an example of such restrictions, and reflect the OCaml implementation. We could keep the same indeterminacy as in previous systems; totality would be preserved (all judgments have one applicable rule), but determinism dropped. There could be several S such that $M \mathbin{@} \Gamma^{at}; \Gamma^{at'}; \Sigma \vdash^{alg}_{inv} S : A \mid$, which would correspond to (2-set restrictions of) sets of terms equal upto invertible commuting conversion. ∗

Lemma 7.3 (Soundness)**.**
For any algorithmic judgment returning a 2-set S, any element $t \in S$ is a valid proof term of the corresponding saturating judgment.

Proof sketch. By induction, this is immediate for all rules except ALG-SAT. This rule is designed to fit the requirements of the saturated logic SAT rule. ⊔

Definition 7.1 Recurrent ancestors**.**
Consider a complete algorithmic derivation of a judgment with empty initial memory $\emptyset$. Given any subderivation $\Pi_{leafward}$, we call *recurrent ancestor* any other subderivation $\Pi_{rootward}$ that is on the path between $\Pi_{leafward}$ and the root (it has $\Pi_{leafward}$ as a strict subderivation) and whose derived judgment is identical to the one of $\Pi_{leafward}$ except for the memory M and the output set S.

Lemma 7.4 (Correct Memory)**.**
In a complete algorithmic derivation whose conclusion's memory is M, each subderivation of the form $M' \mathbin{@} \Gamma^{\mathsf{at}};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{sat}} S : Q^{\mathsf{at}}$ has a number of recurrent ancestors equal to

$$M'(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}) - M(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}})$$

Proof. This is immediately proved by reasoning on the path from the start of the complete derivation to the subderivation. By construction of the algorithmic judgment, each judgment of the form $M' \mathbin{@} \Gamma^{\mathsf{at}'};\emptyset \vdash^{\mathsf{alg}}_{\mathsf{sat}} S' : Q^{\mathsf{at}}$ is proved by either the rule ALG-SAT-KILL, which terminates the path with the invariant maintained, or the rule ALG-SAT-POST, which continues the path with the invariant preserved by incrementing the count in memory. □

Lemma 7.5 (Recurrence Decrementation)**.**
If a saturated logic derivation contains $n+2$ occurrences of the same judgment along a given path, then there is a valid saturated logic derivation with $n+1$ occurrences of this judgment.

Proof. If t is the proof term with $n+2$ occurrences of the same judgment along a given path, let u_1 be the subterm corresponding to the very last occurrence of the judgment, and u_2 the last-but-one. The term $t[u_1/u_2]$ is a valid proof term (of the same result as t), with only $n+1$ occurrences of this same judgment. □

Note that this transformation changes the computational meaning of the term – it must be used with care, as it could break unicity completeness.

Theorem 7.6 (Provability completeness)**.**
If a memory M contains multiplicities of either 0 or 1 (never 2 or more), then any algorithmic judgment with memory M is complete for unicity: if the corresponding saturating judgment is inhabited, then the algorithmic judgment returns an inhabited 2-set.

Proof. If the saturating judgment $\Gamma^{\mathsf{at}};\Gamma^{\mathsf{at}'} \vdash_{\mathsf{sat}} t : Q^{\mathsf{at}}$ holds for a given t, we can assume without loss of generality that t contains no two recurring occurrences of the same judgment along any path – indeed, it suffices to repeatedly apply Lemma 7.5 (Recurrence Decrementation) to obtain such a t with no recurring judgment.

The proof of our result goes by induction on (the saturated derivation of) this no-recurrence t, mirroring each inference step into an algorithmic inference rule returning an inhabited set. Consider the following saturated rule for example:

$$\frac{\Gamma^{\mathsf{at}} \vdash u \Uparrow P_1}{\Gamma^{\mathsf{at}} \vdash \sigma_1\ u \Uparrow P_1 + P_2}$$

We can build the corresponding algorithmic rule

$$\frac{M' \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S_1 \Uparrow P_1 \qquad M' \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S_2 \Uparrow P_2}{M' \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} \sigma_1\ S_1 \cup_2 \sigma_2\ S_2 \Uparrow P_1 + P_2}$$

By induction hypothesis we have that S_1 is inhabited; from it we deduce that $\sigma_1\ S_1$ is inhabited, and thus $\sigma_1\ S_1 \cup_2 \sigma_2\ S_2$ is inhabited.

It would be tempting to claim that the resulting set is inhabited *by* t. That, in our example above, u inhabits S_1 and thus $t = \sigma_1\ u$ inhabits $\sigma_1\ S_1 \cup_2 \sigma_2\ S_2$. This stronger statement is

incorrect, however, as the union of 2-sets may drop some inhabitants if it already has found two distinct terms.

The first difficulty in the induction are with judgments of the form $\Gamma^{\mathsf{at}}; \emptyset \vdash_{\mathsf{sat}} u : Q^{\mathsf{at}}$: to build an inhabited result set, we need to use the rule ALG-SAT-POST and thus check that $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}$ does not occur twice in the current memory M'. By Lemma 7.4 (Correct Memory), we know that $M'(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}})$ is the sum of the number of recurrent ancestors and of $M(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}})$. By definition of t (as a term with no repeated judgment), we know that $\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}}$ did not already occur in t itself – the count of recurrent ancestors is 0. By hypothesis on M we know that $M(\Gamma^{\mathsf{at}} \vdash_{\mathsf{foc}} Q^{\mathsf{at}})$ is at most 1, so the sum cannot be 2 or more.

The second and last subtlety happens at the SINV-END rule for $\Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime} \vdash_{\mathit{sinv}} f : \emptyset \mid Q^{\mathsf{at}}$. We read saturated derivation of the premise $\Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} f : Q^{\mathsf{at}}$, but build an algorithmic derivation in the trimmed context $M \mathbin{@} \Gamma^{\mathsf{at}}; (\Gamma^{\mathsf{at}\prime} -_2 \Gamma^{\mathsf{at}}) \vdash^{\mathsf{alg}}_{\mathsf{sat}} S : Q^{\mathsf{at}}$. It is not necessarily the case that f is well-defined in this restricted context. But that is not an issue for inhabitation: the only variables removed from $\Gamma^{\mathsf{at}\prime}$ are those for which at least one variable of the same type appears in Γ^{at}. We can thus replace each use of a trimmed variable by another variable of the same type in Γ^{at}, and get a valid derivation of the exact same size. □

Theorem 7.7 (Unicity completeness)**.**
If a memory M contains multiplicities of 0 only, then any algorithmic judgment with memory M is complete for unicity: if the corresponding saturating judgment has two distinct inhabitants, then the algorithmic judgment returns a 2-set of two distinct elements.

Proof. Consider a pair of distinct inhabitants $t \neq u$ of a given judgment. Without loss of generality, we can assume that t has no judgment ocurring twice or more. (We cannot also assume that u has no judgment occurring twice, as the recurrence reduction of a general u may be equal to t.)

Without loss of generality, we will also assume that t and u use a consistent ordering for invertible rules (for example the one presented in the algorithmic judgment); this assumption can be made because reordering inference steps gives a term in the $(\approx_{\mathtt{icc}})$ equivalence class, that is thus $\beta\eta$-equivalent to the starting term.

Finally, to justify the SINV-END rule we need to invoke the "two or more" result of Section 4 (Counting terms and proofs), as we detail here. Without loss of generality we assume that t and u never use more than two variables of any given type (additional variables are weakened as soon as they are introduced). If t and u have distinct shapes (they are in disjoint equivalent classes of terms that erase to the same logic derivation), we immediately know that the disequality $t \neq u$ is preserved. If they have the same shape, we need to invoke Corollary 4.6 (Two-or-more approximation) to know that we can pick two distinct terms in this restricted space.

We then prove our result by parallel induction on t and u: the saturated judgment is inhabited by at least two distinct inhabitants. As long as their subterms start with the same syntactic construction, we keep inducing in parallel. Their head constructor may only differ in a non-invertible introduction or elimination rule (we assumed that invertible steps were

performed in the same order), for example we may have

$$\frac{\Gamma^{\mathsf{at}} \vdash p \Uparrow P_1}{\Gamma^{\mathsf{at}} \vdash \sigma_1\ p \Uparrow P_1 + P_2} \qquad \frac{\Gamma^{\mathsf{at}} \vdash q \Uparrow P_2}{\Gamma^{\mathsf{at}} \vdash \sigma_2\ q \Uparrow P_1 + P_2}$$

We then invoke Theorem 7.6 (Provability completeness) on p and q: we can build corresponding derivations $M' \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} S \Uparrow A$ and $M' \mathbin{@} \Gamma^{\mathsf{at}} \vdash^{\mathsf{alg}} T \Uparrow B$ where S and T are inhabited, and thus $\sigma_1\ S \cup_2 \sigma_2\ T$ is inhabited by at least two distinct terms. The memory hypothesis of the provability theorem is fulfilled: because we know that there are no repetitions in t, and that we iterated in parallel on the structures of t and u, we know that each judgment was seen at most once during the parallel induction. As we assumed our starting memory was all 0, the memory M' at the point where t and u differ is thus, by Lemma 7.4 (Correct Memory), of at most 1 for any judgment.

There is one difficulty during the parallel induction, which is the SINV-END case. We read a saturated derivations of premise $\Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} t : Q^{\mathsf{at}}$ and $\Gamma^{\mathsf{at}}; \Gamma^{\mathsf{at}\prime} \vdash_{\mathsf{sat}} u : Q^{\mathsf{at}}$, but build an algorithmic derivation in the trimmed context $\mathscr{M} \mathbin{@} \Gamma^{\mathsf{at}}; (\Gamma^{\mathsf{at}\prime} -_2 \Gamma^{\mathsf{at}}) \vdash^{\mathsf{alg}}_{\mathsf{sat}} S : Q^{\mathsf{at}}$. This is why we restricted t and u to not use more than two different variables of each type, so that they remain well-typed under this restriction. □

Theorem 7.8.
Our unicity-deciding algorithm is terminating and unicity complete.

Proof. Our unicity-deciding algorithm takes a judgment $\lfloor \Sigma \rfloor_{\pm} \vdash \lfloor (N \mid X^{+}) \rfloor_{\pm}$ and returns the 2-set S uniquely determined by a complete algorithmic derivation of the judgment $\emptyset \mathbin{@} \emptyset; \emptyset; \Gamma \vdash^{\mathsf{alg}}_{\mathsf{inv}} S : N \mid X^{+}$ – whose memory is empty. There always exists exactly one derivation by Lemma 7.2 (Totality and Determinism), and it is finite by Lemma 7.1 (Termination). Our algorithm can compute the next rule to apply in finite time, and all derivations are finite, so the algorithm is terminating. This root judgment has an empty memory, hence it is complete for unicity by Theorem 7.7 (Unicity completeness). □

7.3 Optimizations

The search space restrictions described above are those necessary for *termination*. Many extra optimizations are possible, that can be adapted from the proof search literature – with some care to avoid losing completness for unicity. For example, there is no need to cut on a positive if its atoms do not appear in negative positions (nested to the left of an odd number of times) in the rest of the goal. We did not develop such optimizations, except for two low-hanging fruits we describe below.

Eager redundancy elimination Whenever we consider selecting a proof obligation to prove a strict positive during the saturation phase, we can look at the negatives that will be obtained by cutting it. If all those atoms are already present at least twice in the context, this positive is *redundant* and there is no need to cut on it. Dually, before starting a saturation phase, we can look at whether it is already possible to get two distinct neutral proofs of the goal from the current context. In this case it is not necessary to saturate at all.

This optimization is interesting because it significantly reduces the redundancy implied by only filtering of old terms after computing all of them. Indeed, we intuitively expect that most types present in the context are in fact present twice (being unique tends to be the

exception rather than the rule in programming situations), and thus would not need to be saturated again. Redundancy of saturation still happens, but only on the "frontier formulas" that are present exactly once.

Subsumption by memoization One of the techniques necessary to make the inverse method competitive is *subsumption* [McLaughlin and Pfenning, 2008]: when a new judgment is derived by forward search, it is only added to the set of known results if it is not subsumed by a more general judgment (same goal, smaller context) already known.

In our setting, being careful not to break computational completeness, this rule becomes the following. We use (monotonic) mutable state to grow a memoization table of each proved subgoal, indexed by the right-hand side formula. Before proving a new subgoal, we look for all already-computed subgoals of the same right-hand side formula. If one exists with exactly the same context, we return its result. But we also return eagerly if there exists a *larger* context (for inclusion) that returned zero result, or a *smaller* context that returned two-or-more results.

Interestingly, we found out – by experimenting with our implementation – that this optimization would be unsound in presence of the empty type 0. Its equational theory tells us that in an inconsistent context (0 is provable), all proofs are equal. Thus a type may have two inhabitants in a given context, but a larger context that is inconsistent (let us prove 0) will have a unique inhabitant, breaking monotonicity.

8 Evaluation

In this section, we give some practical examples of code inference scenarios that our current algorithm can solve, and some that it cannot – because the simply-typed theory is too restrictive.

The key to our application is to translate a type using prenex-polymorphism into a simple type using atoms in stead of type variables – this is semantically correct given that bound type variables in System F are handled exactly as simply-typed atoms. The approach, of course, is only a very first step and quickly shows it limits. For example, we cannot work with polymorphic types in the environment (ML programs typically do this, for example when typing a parametrized module, or type-checking under a type-class constraint with polymorphic methods), or first-class polymorphism in function arguments. We also do not handle higher-kinded types – even pure constructors.

All the examples mentioned in this section are available as tests in our prototype implementation [Scherer, 2015b].

8.1 Inferring polymorphic library functions

The Haskell standard library contains a fair number of polymorphic functions with unique types. The following examples have been checked to be uniquely defined by their types:

$$\texttt{fst} : \forall\alpha\beta.\ \alpha \times \beta \to \alpha \qquad \texttt{curry} : \forall\alpha\beta\gamma.\ (\alpha \times \beta \to \gamma) \to \alpha \to \beta \to \gamma$$
$$\texttt{uncurry} : \forall\alpha\beta\gamma.\ (\alpha \to \beta \to \gamma) \to \alpha \times \beta \to \gamma$$
$$\texttt{either} : \forall\alpha\beta\gamma.(\alpha \to \gamma) \to (\beta \to \gamma) \to \alpha + \beta \to \gamma$$

When the API gets more complicated, both types and terms become harder to read and uniqueness of inhabitation gets much less obvious. Consider the following operators chosen arbitrarily in the lens [Kmett, 2012] library.

```
(<.) :: Indexable i p => (Indexed i s t -> r)
       -> ((a -> b) -> s -> t) -> p a b -> r
(<.>) :: Indexable (i, j) p => (Indexed i s t -> r)
        -> (Indexed j a b -> s -> t) -> p a b -> r
(%@~) :: AnIndexedSetter i s t a b
        -> (i -> a -> b) -> s -> t
non :: Eq a => a -> Iso' (Maybe a) a
```

The type and type-class definitions involved in this library usually contain first-class polymorphism, but the documentation [Kmett, 2013] provides equivalent "simple types" to help user understanding. We translated the definitions of `Indexed`, `Indexable` and `Iso` using those simple types. We can then check that the first three operators are unique inhabitants; `non` is not.

8.2 Inferring module implementations or type-class instances

The `Arrow` type-class is defined as follows:

```
class Arrow (a : * -> * -> * ) where
  arr :: (b -> c) -> a b c
  first :: a b c -> a (b, d) (c, d)
  second :: a b c -> a (d, b) (d, c)
  (***) :: a b c -> a b' c' -> a (b, b') (c, c')
  (&&&) :: a b c -> a b c' -> a b (c, c')
```

It is self-evident that the arrow type $(\rightarrow)$ is an instance of this class, and *no code should have to be written* to justify this: our prototype is able to infer that all those required methods are uniquely determined when the type constructor `a` is instantiated with an arrow type. This also extends to subsequent type-classes, such as `ArrowChoice`.

As most of the difficulty in inferring unique inhabitants lies in sums, we study the "exception monad", that is, for a fixed type X, the functor $\alpha \mapsto X + \alpha$. Our implementation determines that its `Functor` and `Monad` instances are uniquely determined, but that its `Applicative` instance is not.

Indeed, the type of the `Applicative` method `ap` specializes to the following: $\forall \alpha\beta.\, X + (\alpha \rightarrow \beta) \rightarrow X + \alpha \rightarrow X + \beta$. If both the first and the second arguments are in the error case X, there is a non-unique choice of which error to return in the result.

This is in fact a general result on applicative functors for types that are also monads: there are two distinct ways to prove that a monad is also an applicative functor.

```
ap :: Monad m => m (a -> b) -> m a -> m b
ap mf ma = do                ap mf ma = do
  f <- mf                      a <- ma
  a <- ma                      f <- mf
  return (f a)                 return (f a)
```

Note that the type of `bind` for the exception monad, namely $\forall\alpha\beta.\ X+\alpha \to (\alpha \to X+\beta) \to X+\beta$, has a sum type thunked under a negative type. It is one typical example of a type which cannot be proved unique by the focusing discipline alone, and which is correctly recognized unique by our algorithm.

8.3 Artificial examples

Our prototype will correctly detect that

$$\forall\alpha\beta.\ \alpha \to (\alpha \to \beta+\beta) \to \beta$$

is uniquely inhabited. This type is an example of uniquely inhabited type that is not "negatively non-duplicated", as the type β has several occurrences in negative position (nested to the left of an odd number of arrows); negative non-duplication is a sufficient criterion used in previous work on unique inhabitation [Aoto and Ono, 1994] that does not scale to sums.

A more interesting example is the continuation monad. If we define with a monomorphic return type

$$\texttt{Cont}\ \gamma\ \alpha \qquad \stackrel{\text{def}}{=} \qquad (\alpha \to \gamma) \to \gamma$$

then the `bind` operation on an arbitrary monad $\texttt{Cont}\ A$ is not uniquely inhabited. In fact, the identity at this type, $\texttt{Cont}\ A \to \texttt{Cont}\ A$, is already not uniquely inhabited.

In an extension of our prototype with unit and empty type, however, we could check that if we use 0 as the return type, then both $\texttt{Cont}\ 0 \to \texttt{Cont}\ 0$ and the `bind` operation on $\texttt{Cont}\ 0$ are uniquely inhabited.

This example highlights the interest of properly handling the empty type. The equational theory is very different from a fixed atom X^+ with variable of this type in the environment. We conjecture that a similar result would be obtained with a definition of continuations using a polymorphic return type, but handling polymorphism comes at a higher cost in complexity.

8.4 Non-applications

Here are two related ideas we wanted to try, but that do not fit in the simply-typed lambda-calculus; the uniqueness algorithm must be extended to richer type systems to handle such applications.

We can check that specific instances of a given type-class are canonically defined, but it would be nice to show as well that some of the operators defined on *any* instance are uniquely defined from the type-class methods – although one would expect this to often fail in practice if the uniqueness checker doesn't understand the equational laws required of valid instances. Unfortunately, this would require uniqueness check with polymorphic types in context (for the polymorphic methods).

Another idea is to verify the coherence property of a set of declared instances by translating instance declarations into terms, and checking uniqueness of the required instance types. In particular, one can model the inheritance of one class upon another using a pair type ($\texttt{Comp}\ \alpha$ as a pair of a value of type $\texttt{Eq}\ \alpha$ and `Comp`-specific methods); and the system

can then check that when an instance of `Eq` X and `Comp` X are declared, building `Eq` X directly or projecting it from `Comp` X correspond to $\beta\eta$-equivalent elaboration witnesses. Unfortunately, all but the most simplistic examples require parametrized types and polymorphic values in the environment to be faithfully modelled.

8.5 On impure host programs

The type system in which program search is performed does not need to exactly coincide with the ambiant type system of the host programming language, for which the code-inference feature is proposed – forcing the same type-system would kill any use from a language with non-termination as an effect. Besides doing term search in a pure, terminating fragment of the host language, one could also refine search with type annotations in a richer type system, for example using dependent types or substructural logic – as long as the found inhabitants can be erased back to host types.

However, this raises the delicate question of, among the unique $\beta\eta$-equivalence class of programs, which candidate to select to be actually injected into the host language. For example, the ordering or repetition of function calls can be observed in a host language passing impure function as arguments, and η-expansion of functions can delay effects. Even in a pure language, η-expanding sums and products may make the code less efficient by re-allocating data. There is a design space here that we have not explored.

9 Related and Future Work

9.1 Previous work on unique inhabitation

The problem of unique inhabitation for the simply-typed lambda-calculus (without sums) has been formulated by Mints [1981], with early results by Babaev and Soloviev [1982], and later results by Aoto and Ono [1994], Aoto [1999] and Broda and Damas [2005].

These works have obtained several different *sufficient conditions* for a given type to be uniquely inhabited. While these cannot be used as an algorithm to decide unique inhabitation for any type, they reveal fascinating connections between unique inhabitation and proof or term structures. Some sufficient criteria are formulated on the types/formulas themselves, other on terms (a type is uniquely inhabited if it is inhabited by a term of a given structure).

A simple criterion on types given in Aoto and Ono [1994] is that "negatively non-duplicated formulas", that is formulas where each atom occurs at most once in negative position (nested to the left of an odd number of arrows), have at most one inhabitant. This was extended by Broda and Damas [2005] to a notion of "deterministic" formulas, defined using a specialized representation for simply-typed proofs named "proof trees".

Aoto [1999] proposed a criterion based on terms: a type is uniquely inhabited if it "provable without non-prime contraction", that is if it has at least *one* inhabitant (not necessarily cut-free) whose only variables with multiple uses are of atomic type. Recently, Bourreau and Salvati [2011] used game semantics to give an alternative presentation of Aoto's results, and a syntactic characterization of *all* inhabitants of negatively non-duplicated formulas.

Those sufficient conditions suggest deep relations between the static and dynamics semantics of restricted fragments of the lambda-calculus – it is not a coincidence that contraction at non-atomic types is also problematic in definitions of proof equivalence coming from categorial logic [Dosen, 2003]. However, they give little in the way of a decision procedure for all types – conversely, our decision procedure does not by itself reveal the structure of the types for which it finds unicity.

An indirectly related work is the work on retractions in simple types (A is a retract of B if B can be surjectively mapped into A by a λ-term). Indeed, in a type system with a unit type 1, a given type A is uniquely inhabited if and only if it is a retract of 1. Stirling [2013] proposes an algorithm, inspired by dialogue games, for deciding retraction in the lambda-calculus with arrows and products; but we do not know if this algorithm could be generalized to handle sums. If we remove sums, focusing already provides an algorithm for unique inhabitation.

9.1.1 Counting inhabitants

Broda and Damas [2005] remark that normal inhabitants of simple types can be described by a context-free structure. This suggests, as done in Zaoinc [1995], counting terms by solving a set of polynomial equations. Further references to such "grammatical" approaches to lambda-term enumeration and counting can be found in Dowek and Jiang [2011].

Of particular interest to us was the recent work of Wells and Yakobowski [2004]. It is similar to our work both in terms of expected application (program fragment synthesis) and methods, as it uses (a variant of) the focused calculus LJT [Herbelin, 1994] to perform proof search. It has sums (disjunctions), but because it only relies on focusing for canonicity it only implements the *weak* notion of η-equivalence for sums – it is not canonical, as discussed in Section 5.1.1 (Non-canonicity of simple focusing: splitting points), it counts an infinite number of inhabitants in presence of a sum thunked under a negative. Their technique to ensure termination of enumeration is very elegant. Over the graph of all possible proof steps in the type system (using multisets as contexts: an infinite search space), they superimpose the graph of all possible non-cyclic proof steps in the logic (using sets as contexts: a finite search space). Termination is obtained, in some sense, by traversing the two in lockstep. We took inspiration from this idea to obtain our termination technique: our bounded multisets can be seen as a generalization of their use of set-contexts.

9.1.2 Non-classical theorem proving and more canonical systems

Automated theorem proving has motivated fundamental research on more canonical representations of proofs: by reducing the number of redundant representations that are equivalent as programs, one can reduce the search space – although that does not necessarily improve speed, if the finer representation requires more book-keeping. Most of this work was done first for (first-order) classical logic; efforts porting them to other logics (linear, intuitionistic, modal) are of particular interest, as they often reveal the general idea behind particular techniques, and are sometimes an occasion to reformulate these techniques in terms closer to type theory.

An important line of work studies connection-based, or matrix-based, proof methods. They have been adapted to non-classical logics as soon as Wallen [1987]. It is possible to present connection-based search "uniformly" for many distinct logics [Otten and Kreitz, 1996], changing only one logic-specific check to be performed a posteriori on connections (axiom rules) of proof candidates. In an intuitionistic setting, that would be a comparison on indices of Kripke Worlds; it is strongly related to *labeled logics* [Galmiche and Méry, 2013]. On the other hand, matrix-based methods rely on guessing the number of duplications of a formula (contractions) that will be used in a particular proof, and this technique seems difficult to extend to second-order polymorphism – by picking a presentation closer to the original logic, namely focused proofs, we hope for an easier extension.

Some contraction-free calculi have been developed with automated theorem proving for intuitionistic logic in mind. A presentation is given in Dyckhoff [1992] – the idea itself appeared as early as Vorob'ev [1958]. The idea is that sums and (positive) products do not need to be deconstructed twice, and thus need not be contracted on the left. For functions, it is actually sufficient for provability to implicitly duplicate the arrow in the argument case of its elimination form ($A \to B$ may have to be used again to build the argument A), and to forget it after the result of application (B) is obtained. More advanced systems typically do case-distinctions on the argument type A to refine this idea, see Dyckhoff [2013] for a recent survey. Unfortunately, such techniques to reduce the search space break computational completeness: they completely remove some programmatic behaviors. Consider the type $\texttt{Stream}(A,B) \stackrel{\text{def}}{=} A \times (A \to A \times B)$ of infinite streams of state A and elements B: with this restriction, the next-element function can be applied at most once, hence $\texttt{Stream}(X,Y) \to Y$ is uniquely inhabited in those contraction-free calculi. (With focusing, only negatives are contracted, and only when picking a focus.)

Focusing was introduced for linear logic [Andreoli, 1992], but is adaptable to many other logics. For a reference on focusing for intuitionistic logic, see Liang and Miller [2007]. Our programs are lambda-terms, so we use a natural deduction presentation (instead of the more common sequent-calculus presentation) of focused logic, closely inspired by the work of Brock-Nannestad and Schürmann [2010] on intuitionistic linear logic.

Some of the most promising work on automated theorem proving for intuitionistic logic comes from applying the so-called "Inverse Method" (see Degtyarev and Voronkov [2001] for a classical presentation) to focused logics. The inverse method was ported to linear logic in Chaudhuri and Pfenning [2005], and turned into an efficient implementation of proof search for intuitionistic logic in McLaughlin and Pfenning [2008]. It is a "forward" method: to prove a given judgment, start with the instances of axiom rules for all atoms in the judgment, then build all possible valid proofs until the desired judgment is reached – the subformula property, bounding the search space, ensures completeness for propositional logic. Focusing allows important optimization of the method, notably through the idea of "synthetic connectives": invertible or non-invertible phases have to be applied all in one go, and thus form macro-steps that speed up saturation.

In comparison, our own search process alternates forward and backward-search. At a large scale we do a backward-directed proof search, but each non-invertible phase performs saturation, that is a complete forward-search for positives. Note that the search space of those saturation phases is not the subformula space of the main judgment to prove,

but the (smaller) subformula space of the current subgoal's context. When saturation is complete, backward goal-directed search restarts, and the invertible phase may grow the context, incrementally widening the search space. (The forward-directed aspects of our system could be made richer by adding positive products and positively-biased atoms; this is not our main point of interest here. Our coarse choice has the good property that, in the absence of sum types in the main judgment, our algorithm immediately degrades to simple, standard focused backward search.)

Maximal multi-focusing An important result for canonical proof structures is *maximal multi-focusing* [Miller and Saurin, 2007, Chaudhuri, Miller, and Saurin, 2008a]. Multi-focusing refines focusing by introducing the ability to focus on several formulas at once, in parallel, and suggests that, among formulas equivalent modulo valid permutations of inference rules, the "more parallel" ones are more canonical. Indeed, *maximal* multi-focused proofs turn out to be equivalent to existing more-canonical proof structures such as linear proof nets [Chaudhuri, Miller, and Saurin, 2008a] and classical expansion proofs [Chaudhuri, Hetzl, and Miller, 2012].

In Scherer [2015a] we proposed a multi-focused natural deduction and a λ-calculus interpretation for it, whose maximal multi-focused terms are canonical for $\mathsf{\Lambda C}{\to},{\times},{+}$. Saturating focused proofs are almost maximal muli-focused proofs in this sense. The difference is that multi-focusing allow to focus on both variables in the context and the goal in the same time, while our right-focusing rule SAT-INTRO can only be applied sequentially after SAT (which does multi-left-focusing). To recover the exact structure of maximal multi-focusing, one would need to allow SAT to also focus on the right, and use it only when the right choices do not depend on the outcome on saturation of the left (the foci of the same set must be independent), that is when none of the bound variables are used (typically to saturate further) before the start of the next invertible phase. This is a rather artificial restriction from a backward-search perspective. Maximal multi-focusing is more elegant, declarative in this respect, but is less suited to proof search.

Lollimon: backward and forward search together We described in Section 5.3 (The roles of forward and backward search in a saturated logic) the way our saturated proof search mixes backward and forward search. It is interesting to compare it to Lollimon, a system presented in López, Pfenning, Polakow, and Watkins [2005] which similarly mixes backward and forward search.

Lollimon is part of the research on logic programming that understands the execution of logic program as given by the operational behavior of proof search in a well-chosen logic – typically with uniform proofs or focusing. Cut-elimination is not the only way to give an operational semantics to proof systems that is suitable for programming, proof search also has a rich "programmable" operational behavior.

More specifically, the research arc on Concurrent LF and related systems tries to studies a wider range of logics to capture the operational behavior of interesting systems, typically concurrent systems with several interacting actors or processes. Lollimon uses a mix of intuitionistic logic and linear logic – linear logic is suitable to represent consumable resources and, thus, essential to the modeling of systems with modifiable state.

In Lollimon, as in our case, forward search comes from the behavior of the left-focusing rule with positive conclusion, that is the forward-chaining rule of the logic. This forward search ingredient provides an elegant way to describe behaviors that are asynchronous (they do not necessarily rely on a communication between independent parts of a formula) but non-invertible – one example is the computation of a future alongside the rest of the program. Furthermore, when the forward search strategy performs forward search until saturation is reached, Lollimon can easily describe algorithms that rely on saturation, such as computing the transitive closure of a graph.

Because of this focus on representing the operation behavior of a variety of system, the Lollimon logic is not prescriptive: it does not actually enforce saturating or any other forward-search strategy, it is their implementation of the proof search algorithm that made specific implementation choices. In contrast, saturated logic is formulated in a strongly prescriptive way: while the choice of the saturation function gives some leeway, the logic enforces saturation phase as long as new hypotheses are present, and a form of completeness for provability through the SELECT-SPECIFIC restriction.

Saturated logic is prescriptive because we can afford it: in the more limited applications that we are interested in, either the search of a unique inhabitant or equivalence checking, there is a natural choice of selection function that allows some form of "full saturation" and yet remains terminating, so enforcing (restricted) saturation is practical.

We believe that the consideration of program terms – the type-theoretic rather than proof-theoretic setting – also gives some intuitions that would be harder to acquire in the Lollimon setting. Our distinction between "old" and "new" formulas would be possible in a purely logical setting, but the idea of only saturating on the neutrals that use the "new" formulas relies on the intuition of considering proof terms as programs – those new neutral may have new values that we did not know about yet. The saturation selection strategy used in our unicity-checking algorithm, the "two or more" criterion (we can keep at most two variables of each type to find out if two distinct programs are possible), would not at all be natural in a purely proof-theoretic setting.

9.1.3 Equivalence of terms in presence of sums

Ghani [1995] first proved the decidability of equivalence of lambda-terms with sums, using sophisticated rewriting techniques. The two works that followed [Altenkirch, Dybjer, Hofmann, and Scott, 2001, Balat, Di Cosmo, and Fiore, 2004] used normalization-by-evaluation instead. Finally, Lindley [2007] was inspired by Balat, Di Cosmo, and Fiore [2004] to re-explain equivalence through rewriting. Our idea of "cutting sums as early as possible" was inspired from Lindley [2007], but in retrospect it could be seen in the "restriction (A)" in the normal forms of Balat, Di Cosmo, and Fiore [2004], or directly in the "maximal conversions" of Ghani [1995].

Note that the existence of unknown atoms is an important aspect of our calculus. Without them (starting only from base types 0 and 1), all types would be finitely inhabited. This observation is the basis of the promising unpublished work of Ahmad, Licata, and Harper [2010], also strongly relying on (higher-order) focusing. Finiteness hypotheses also play an important role in Ilik [2014], where they are used to reason on type *isomorphisms* in presence of sums.

In Munch-Maccagnoni and Scherer [2015], we collaborated with Guillaume Munch-Maccagnoni to rephrase the problem of sum equivalence in a notational framework of *abstract machine calculi* called System L. Historically this work comes from both the search for a term notation that would give a clear computational meaning to classical logic, and the fine-grained study of weak reduction strategies, notably the duality between call-by-name and call-by-value reduction. It subsumes both by using a "polarized" reduction strategy. In a typed setting – System L can also be studied as an untyped calculus – this "polarization" can be seen as going beyond focusing. In particular, the relation between System L's reduction and cut-elimination in strongly focused systems is similar to the relation between reduction in a direct-style effectful λ-calculus and an indirect-style monadic calculus.

9.1.4 Elaboration of implicits

The most visible uses of typed-directed code inference for functional languages are *type-classes* [Wadler and Blott, 1989] and *implicits* [Oliveira, Moors, and Odersky, 2010]. Type classes elaboration is traditionally presented as a satisfiability problem (or constraint solving problem [Stuckey and Sulzmann, 2002]) that happens to have operational consequences. Implicits recast the feature as elaboration of a programming *term*, which is closer to our methodology. Type-classes traditionally try (to various degrees of success) to ensure *coherence*, namely that a given elaboration goal always give the same dynamic semantics wherever it happens in the program – often by making instance declarations a toplevel-only construct. Implicits allow a more modular construction of the elaboration environment, but have to resort to priorities to preserve determinism [Oliveira, Schrijvers, Choi, Lee, Yi, and Wadler, 2014].

We propose to reformulate the question of determinism or ambiguity by presenting elaboration as a *typing* problem, and proving that the elaborated problems intrinsically have unique inhabitants. This point of view does not by itself solve the difficult questions of which are the good policies to avoid ambiguity, but it provides a more declarative setting to expose a given strategy; for example, priority to the more recently introduced implicit would translate to an explicit weakening construct, removing older candidates at introduction time, or a restricted variable lookup semantics.

(The global coherence issue is elegantly solved, independently of our work, by using a dependent type system where the values that semantically depend on specific elaboration choices (for example a balanced tree ordered with respect to some specific order) have a type that syntactically depends on the elaboration witness. This approach meshes very well with our view, especially in systems with explicit equality proofs between terms, where features that grow the implicit environment could require proofs from the user that unicity is preserved.)

9.1.5 Smart completion and program synthesis

Type-directed program synthesis has seen sophisticated work in the recent years, notably Perelman, Gulwani, Ball, and Grossman [2012], Gvero, Kuncak, Kuraj, and Piskac [2013]. Type information is used to fill missing holes in partial expressions given by the users,

typically among the many choices proposed by a large software library. Many potential completions are proposed interactively to the user and ordered by ranking heuristics.

Our uniqueness criterion is much more rigid: restrictive (it has far less potential applications) and principled (there are no heuristics or subjective preferences at play). Complementary, it aims for application in richer type systems, and in *programming constructs* (implicits, etc.) rather than *tooling* with interactive feedback.

An aspect of interaction which could be interesting in our system is the *failure* case were at least two distinct inhabitants are found. A first question is, among all the possible counter-examples our algorithm could provide, which will be the more beneficial to the user? We suspect that having a computationally-observable difference as *early* in the terms as possible is preferable. A second is whether the user could interact with the system to refine the search space, possibly navigating between alternatives proposed by the system – for now the only refinement tools are type annotations.

Synthesis of glue code interfacing whole modules has been presented as a type-directed search, using type isomorphisms [Aponte and Di Cosmo, 1996] or inhabitation search in combinatory logics with intersection types [Düdder et al., 2014].

Focusing and program synthesis We were very interested in the recent work on Myth, presented in Osera and Zdancewic [2015], which generates code from both expected type and input/output examples. It is based on bidirectional type-checking, but we believe that it is in fact using focusing. The works are complementary: they have interesting proposals for data-structures and algorithm to make term search efficient, while we bring a deeper connection to proof-theoretic methods. They independently discovered the idea that saturation must use the "new" context, and present it as an algorithmic improvement called "relevant term generation".

This work has been expanded upon in Frankle, Osera, Walker, and Zdancewic [2016], and at the time of writing there is work underway to strengthen the connection to focusing. We hope to be able to study the connections in more details. This work, notably, seems more advanced in terms of study of applicability to real scenarios, so a cooperation could be very fruitful.

9.2 Future work

9.2.1 Pushing the application front

Despite some interesting experiments with our software prototype, we have not yet pushed in the direction of practical application of this work to real-world programming language. We think that supporting richer type systems would help to make it more widely applicable, but it may already be possible to provide the current capabilities as a code inference tool for typed functional languages, and thus gather some usage experience.

9.2.2 Substructural logics

Instead of moving to more polymorphic type systems, one could move to substructural logics. We could expect to refine a type annotation using, for example, linear arrows, to get a unique inhabitant. We observed, however, that linearity is often disappointing in getting

"unique enough" types. Take the polymorphic type of mapping on lists, for example: $\forall\alpha\beta.(\alpha \to \beta) \to (\texttt{List}\ \alpha \to \texttt{List}\ \beta)$. Its inhabitants are the expected map composed with any function that can reorder, duplicate or drop elements from a list. Changing the two inner arrows to be linear gives us the set of functions that may only reorder the mapped elements: still not unique. An idea to get a unique type is to request a mapping from $(\alpha \leq \beta)$ to $(\texttt{List}\ \alpha \leq \texttt{List}\ \beta)$, where the subtyping relation $(\leq)$ is seen as a substructural arrow type.

(Dependent types also allow to capture `List.map`, as the unique inhabitant of the dependent induction principle on lists is unique.)

9.2.3 Equational reasoning

We have only considered pure, strongly terminating programs so far. One could hope to find monadic types that uniquely defined transformations of impure programs (e.g. $(\alpha \to \beta) \to \texttt{M}\ \alpha \to \texttt{M}\ \beta$). Unfortunately, this approach would not work by simply adding the unit and bind of the monad as formal parameters to the context, because many programs that are only equal up to the monadic laws would be returned by the system. It could be interesting to enrich the search process to also normalize by the monadic laws.[3] In the more general case, can the search process be extended to additional rewrite systems?

9.2.4 Unique inhabitation with polymorphism or dependent types

We have started experimenting with an extension of saturated proof search to System F, with no strong results so far.

The general problem with polymorphism is the loss of the subformula property, and thus the loss of termination in our algorithm – or any algorithm, as the problem becomes undecidable as shown by reducing unicity to inhabitation.[4] In the details, this appears when trying to build a negative neutral out of $\forall$-quantified formula during a left-focusing phase: there is an infinite space of possible instantiations choices.

First, remark that the algorithm of Section 5 (Saturation logic for canonicity) directly extends to the sub-system where $\forall$-quantifiers are only present in positive subformulas occurrences – this is the easy subset where no instantiation choices have to be made. Gilles Dowek and Ying Jiang studied this almost-non-polymorphic fragment in Dowek and Jiang [2009]; it gives a precise formal status to our handling of prenex polymorphism in our experiments. Note that formulas with positive $\forall$ occurrences are a more general fragment than just prenex polymorphism, although type systems such as Mitchell's $\mathsf{F}\eta$ [Mitchell, 1988] bridge the gap by allowing to lift positive quantifiers into prenex position by subtyping/containment.

[3] Sam Lindley remarked that the specific case of monad laws should be relatively easy, as monad laws can be seen as a weaker form of sum laws. If we consider an abstract monad $\texttt{M}\ A$ as a sum $0+A$, with 0 being an empty type and with the expected implementations of `bind` and `return`, the reduction and *weak* η-expansion on sums suffice to recover the usual monad laws – the equational theory of Eugenio Moggi's computational λ-calculus.

[4] Undecidability of inhabitation in System F is an old result recalled in Wells [1994] – an article that is itself related to the different issue of decidability of typability of a term.

Second, our suggestion for future work would be to replace the problem of "at a use site, how to instantiate this polymorphic neutral to make further progress", which leads to a natural explosion of the saturation dynamics – there will often be infinitely many strict positives to deduce – by the different question of "at the abstraction site, is there a set of instantiations that summarizes the polymorphic value in its full generality?".

For example, if the polymorphic type $\forall\alpha, (X^+ \to \alpha) \to (Y^+ \to \alpha) \to \alpha$ is in an invertible context, we could in a sense "invertibly decompose" it by instantiating it either with X^+ or with Y^+, as we can easily prove that no other instantiation leads to an inhabited type. Note that we are taking a "closed world" view here: we are assuming that the context has no other way to build a value of this type that we have ourselves, and thus that we can reason on the possible values that were passed to us by enumerating the terms we could build ourselves at this type.

In a more general setting, this suggests a generalization of Noam Zeilberger's higher-order focusing rule [Zeilberger, 2009] that "decomposes" polymorphic hypotheses that could look like

$$\text{DRAFT-POLYMORPHIC-HIGHER-ORDER-RULE}\quad \frac{\forall \Sigma', \qquad \Sigma', \alpha \Vdash A(\alpha) \quad \Longrightarrow \quad \Gamma^{\mathsf{at}}; \Sigma, \Sigma' \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}}{\Gamma^{\mathsf{at}}; \Sigma, \forall\alpha, A(\alpha) \vdash_{\mathsf{inv}} N \mid P^{\mathsf{at}}}$$

where the $\Sigma \Vdash A$ relation ranges over the minimal set of contexts that must be inhabited for A to be inhabitable.

We have been trying to find a way to enumerate those "most general contexts" by reusing our (unicity-aware) proof search procedure on $A(\alpha)$, in a mode that would collect inhabitation constraints (the minimal context is an output, rather than an input, of the enumeration procedure). If this succeeded, it would give a new understanding of parametricity results in terms of syntactic proof search.

Note that the interaction between this idea of closed-world proof search and focusing is unknown and quite likely to be a delicate issue. The fact that $\forall$-quantifiers in positive position are invertibly introduced would suggest to consider polymorphic types as negatives, but our higher-order focusing approach instead consider them (in negative position) as positives.

Finally, on a more technical level, we think that extending our proof search procedure to System F (and beyond) would benefit from an explicit handling of metavariables as done in Lengrand, Dyckhoff, and McKinna [2011]. Explicit meta-variables let us explicitly represent the state of proof search as a derivation, and this let us explore a richer setting of proof search strategies – choices metavariable instantiation order – notably breadth-first search strategies. Without this explicit representation of search state, the natural approach is to have a recursive proof search procedure that provides complete proof of each judgment when called, so it imposes a depth-first approach. This inflexibility is acceptable in a simply-typed setting where each search branch terminates, but in a undecidable setting it makes the system halt as soon as some subspace becomes infinite – we would hope for a better behavior in this case.

Bibliography

Arbob Ahmad, Daniel R. Licata, and Robert Harper. Deciding coproduct equality with focusing. Online draft, 2010. 89

Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *LICS*, 2001. 4, 49, 89

Jean-Marc Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation*, 2(3), 1992. 3, 6, 15, 87

Takahito Aoto. Uniqueness of normal proofs in implicational intuitionistic logic. *Journal of Logic, Language and Information*, 1999. 85

Takahito Aoto and Hiroakira Ono. Non-Uniqueness of Normal Proofs for Minimal Formulas in Implication-Conjunction Fragment of BCK. *Bulletin of the Section of Logic*, 1994. 84, 85

Maria-Virginia Aponte and Roberto Di Cosmo. Type isomorphisms for module signatures. In *PLILP*, 1996. 91

Ali Babaev and Sergei Soloviev. A coherence theorem for canonical morphisms in cartesian closed categories. *Journal of Soviet Mathematics*, 1982. 85

Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In *POPL*, 2004. 4, 8, 49, 89

Pierre Bourreau and Sylvain Salvati. Game semantics and uniqueness of type inhabitance in the simply-typed λ-calculus. In *TLCA*, 2011. 85

Taus Brock-Nannestad and Carsten Schürmann. Focused natural deduction. In *LPAR-17*, 2010. 25, 87

Sabine Broda and Luís Damas. On long normal inhabitants of a type. *J. Log. Comput.*, 2005. 85, 86

Kaustuv Chaudhuri. Magically constraining the inverse method using dynamic polarity assignment. In *LPAR*, October 2010. URL https://hal.inria.fr/inria-T00535948. 60

Kaustuv Chaudhuri and Frank Pfenning. Focusing the inverse method for linear logic. In *CSL*, 2005. 87

Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical sequent proofs via multi-focusing. In *IFIP TCS*, 2008a. 3, 9, 88

Kaustuv Chaudhuri, Frank Pfenning, and Greg Price. A logical characterization of forward and backward chaining in the inverse method. volume 40, 2008b. 18, 60

Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. A Systematic Approach to Canonicity in the Classical Sequent Calculus. In *CSL*, 2012. 88

Anatoli Degtyarev and Andrei Voronkov. Introduction to the inverse method. In *Handbook of Automated Reasoning*. 2001. 87

Kosta Dosen. Identity of proofs based on normalization and generality. *Bulletin of Symbolic Logic*, 2003. 86

Gilles Dowek and Ying Jiang. Enumerating proofs of positive formulae. *Comput. J.*, 52 (7), 2009. 92

Gilles Dowek and Ying Jiang. On the expressive power of schemes. *Inf. Comput.*, 2011. 86

Boris Düdder, Moritz Martens, and Jakob Rehof. Staged composition synthesis. In *ESOP*, 2014. 91

Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *J. Symb. Log.*, 1992. 87

Roy Dyckhoff. Intuitionistic decision procedures since gentzen, 2013. Talk notes. 87

Mahfuza Farooque, Stéphane Graham-Lengrand, and Assia Mahboubi. A bisimulation between dpll(T) and a proof-search strategy for the focused sequent calculus. In *LFTMP*, 2013. 60

Jonathan Frankle, Peter-Michael Osera, David Walker, and Steve Zdancewic. Example-directed synthesis: a type-theoretic interpretation. In *POPL*, 2016. 91

Didier Galmiche and Daniel Méry. A connection-based characterization of bi-intuitionistic validity. *J. Autom. Reasoning*, 2013. 87

Neil Ghani. Beta-Eta Equality for Coproducts. In *TLCA*, 1995. 4, 49, 89

Tihomir Gvero, Viktor Kuncak, Ivan Kuraj, and Ruzica Piskac. Complete completion using types and weights. In *PLDI*, 2013. 2, 90

Hugo Herbelin. A Lambda-calculus Structure Isomorphic to Gentzen-style Sequent Calculus Structure. In *CSL*, 1994. URL https://hal.inria.fr/inria-T00381525. 18, 86

Danko Ilik. Axioms and decidability for type isomorphism in the presence of sums. *CoRR*, abs/1401.2567, 2014. URL http://arxiv.org/abs/1401.2567. 89

Edward Kmett. Lens, 2012. URL https://github.com/ekmett/lens. 83

Edward Kmett. Lens wiki – types, 2013. URL https://github.com/ekmett/lens/wiki/Types. 83

Olivier Laurent. A proof of the focalization property of linear logic. 2004. 33

Stéphane Lengrand, Roy Dyckhoff, and James McKinna. A focused sequent calculus framework for proof search in Pure Type Systems. *Logical Methods in Computer Science*, 7(1), 2011. 93

Chuck Liang and Dale Miller. Focusing and polarization in intuitionistic logic. *CoRR*, 2007. URL http://arxiv.org/abs/0708.2252. 16, 87

Sam Lindley. Extensional rewriting with sums. In *TLCA*, 2007. 4, 49, 89

Pablo López, Frank Pfenning, Jeff Polakow, and Kevin Watkins. Monadic concurrent linear logic programming. In *PPDP*, 2005. 61, 88

Sean McLaughlin and Frank Pfenning. Imogen: Focusing the polarized inverse method for intuitionistic propositional logic. In *LPAR*, 2008. 82, 87

Dale Miller and Alexis Saurin. From proofs to focused proofs: A modular proof of focalization in linear logic. In *CSL*, 2007. 88

Grigori Mints. Closed categories and the theory of proofs. *Journal of Soviet Mathematics*, 1981. 85

John C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 2/3(76), 1988. 92

Guillaume Munch-Maccagnoni and Gabriel Scherer. Polarised intermediate representation of lambda calculus with sums. In *LICS*, 2015. URL https://hal.inria.fr/hal-T01160579. 90

Bruno C. d. S. Oliveira, Adriaan Moors, and Martin Odersky. Type classes as objects and implicits. In *OOPSLA*, 2010. 90

Bruno C. d. S. Oliveira, Tom Schrijvers, Wontae Choi, Wonchan Lee, Kwangkeun Yi, and Philip Wadler. The implicit calculus: A new foundation for generic programming. 2014. 2, 90

Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. In *PLDI*, 2015. 91

Jens Otten and Christoph Kreitz. A uniform proof procedure for classical and non-classical logics. In *KI Advances in Artificial Intelligence*, 1996. 87

Daniel Perelman, Sumit Gulwani, Thomas Ball, and Dan Grossman. Type-directed completion of partial expressions. In *PLDI*, 2012. 2, 90

Gabriel Scherer. Mining opportunities for unique inhabitants in dependent programs, 2013. 2

Gabriel Scherer. Multi-focusing on extensional rewriting with sums. In *TLCA*, 2015a. URL http://gallium.inria.fr/~scherer/drafts/multifoc_sums.pdf. 62, 88

Gabriel Scherer, 2015b. URL http://gallium.inria.fr/~scherer/research/unique_inhabitants/. 71, 82

Gabriel Scherer and Didier Rémy. Which simple types have a unique inhabitant? In *ICFP*, 2015. URL http://gallium.inria.fr/~scherer/research/unique_inhabitants/unique_stlc_sums-Tlong.pdf. 1, 35, 59, 62

Robert J. Simmons. Structural focalization. *CoRR*, abs/1109.6273, 2011. 33, 35

Colin Stirling. Proof systems for retracts in simply typed lambda calculus. In *Automata, Languages, and Programming - ICALP*, 2013. 86

Peter J. Stuckey and Martin Sulzmann. A theory of overloading. In *ICFP*, 2002. 2, 90

Nikolay Vorob'ev. A new algorithm of derivability in a constructive calculus of statements. In *Problems of the constructive direction in mathematics*, 1958. 87

Philip Wadler and Stephen Blott. How to make ad-hoc polymorphism less ad-hoc. In *POPL*, 1989. 2, 90

Lincoln A. Wallen. Automated proof search in non-classical logics: Efficient matrix proof methods for modal and intuitionistic logic, 1987. 87

Joe B. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *LICS*, July 1994. 92

Joe B. Wells and Boris Yakobowski. Graph-based proof counting and enumeration with applications for program fragment synthesis. In *LOPSTR*, 2004. 86

Marek Zaoinc. Fixpoint technique for counting terms in typed lambda-calculus. Technical report, State University of New York, 1995. 86

Noam Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching*. PhD thesis, Carnegie Mellon University, 2009. 93

Noam Zeilberger. Polarity in proof theory and programming, August 2013. URL http://noamz.org/talks/logpolpro.pdf. Lecture Notes for the Summer School on Linear Logic and Geometry of Interaction in Torino, Italy. 22