Separating Functional Computation from Relations

Ulysse Gérard, Dale Miller

Separating Functional Computation from Relations

Ulysse Gérard and Dale Miller

Inria Saclay & LIX/École Polytechnique, Palaiseau, France

Abstract

The logical foundations of arithmetic generally start with a quantificational logic of relations. Of course, one often wishes to have a formal treatment of functions within this setting. Both Hilbert and Church added to logic choice operators (such as the epsilon operator) in order to coerce relations that happen to encode functions into actual functions. Others have extended the term language with confluent term rewriting in order to encode functional computation as rewriting to a normal form. We take a different approach that does not extend the underlying logic with either choice principles or with an equality theory. Instead, we use the familiar two-phase construction of focused proofs and capture functional computation entirely within one of these phases. As a result, our logic remains purely relational even when it is computing functions.

1 Introduction

The development of the logical foundations of arithmetic generally starts with the first-order logic of relations to which constructors for zero and successor have been added. Various axioms (such as Peano's axioms) are then added to that framework in order to define the natural numbers and various relations among them. Of course, it is often natural to think of some computations, such as say addition and multiplications of natural numbers, as being *functions* instead of relations.

A common way to introduce functions into the relational setting is to enhance the equality theory. For example, Troelstra in [24, Section I.3] presents an intuitionistic theory of arithmetic in which all primitive recursive functions are treated as black-boxes and every one of their instances, for example $23 + 756 = 779$, is simply added as an equation. A modern and more structured version of this approach is that of $\lambda\Pi$-*calculus modulo* framework proposed by Cousineau and Dowek [7]: in that framework, the dependently typed λ-calculus (a presentation of intuitionistic predicate logic) is extended with a rich set of terms and rewriting rules on them. When rewriting is confluent, it can be given a functional programming implementation: the Dedukti proof checker [2] is based on this hybrid approach to treating functions in a relational setting.

A predicate can, of course, encode a function. For example, assume that we have a predicate R of $n+1$-ary ($n \geqslant 0$) and that we can prove that the first n arguments of R uniquely determines the value of its last argument. That is, assume that the following formula (where $\bar{x}$ denotes a list of variables $x_1, \ldots, x_n$) is provable:

$$(*) \qquad \forall \bar{x}([\exists y.R(\bar{x}, y)] \wedge \forall y \forall z[R(\bar{x}, y) \supset R(\bar{x}, z) \supset y = z]).$$

In this situation, an n-ary function f_R exists such that $f_R(\bar{x}) = y$ if and only if $R(\bar{x}, y)$. In order to formally describe the function f_R, Hilbert [16] and Church [6] evoked *choice operators* such as ϵ and ι which (along with appropriate axioms) are able to take a singleton set and return the unique element in that set. For example, in Church's Simple Theory of Types [6], the expression $\lambda x_1 \ldots \lambda x_n \iota(\lambda y.R(x_1, \ldots, x_n, y))$ provides a definition of f_R.

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we take a different approach to separating functional computations from more general reasoning with relations. We shall not extend the equational theory beyond the minimal equality on terms and we shall not use choice principles.

Although our approach to separating functions from relations is novel, it does not need any new theoretical results: we simply make direct use of several recent results in proof theory. In particular, our paper follows the following outline.

1. We formulate a sequent calculus proof system for Heyting arithmetic where fixed points and term equality are *logical connectives*: that is, they are defined via their left and right introduction rules. This work builds on earlier work by McDowell & Miller [19] and Momigliano & Tiu [22].
2. We replace Gentzen's sequent proofs with *focused proof systems* as developed by Andreoli, Baelde, and Liang & Miller [1, 3, 17]. Such inference systems structure proofs into two *phases*: the *negative* phase organizes *don't-care nondeterminism* while the *positive* phase organizes *don't-know nondeterminism*. In this way, the construction of a negative phase (reading it as a mapping from its endsequent to its premises) determines a function and the construction of the positive phase determines a more general nondeterminism relation.
3. Since $\forall x[P(x) \supset Q(x)] \equiv \exists x[P(x) \wedge Q(x)]$ whenever predicate P denotes a singleton set, the resulting *ambiguity of polarity* makes it possible to position such singleton predicates always into the negative phase. As mentioned above, a suitable treatment of singleton sets allows for a direct treatment of functions.
4. We exploit focused proof systems in another sense. If we view proofs of simply types as denoting typed terms, then the usual representation of terms as function-applied-to-arguments occurs when primitive types are polarized negatively. If we flip, however, the polarity of primitive types to positive polarity, then we flip around term structure into something similar to *administrative normal form* [9]. Using such a term representation, we will be able to translate common arithmetic expressions using functions into appropriate sequences of the relations that compute those functions. This approach to term representation builds on the $\lambda\kappa$-term calculus of Brock-Nannestad, Guenot, & Gustafsson [4] as well as the LJQ' calculus of Dyckhoff & Lengrand [8].
5. Finally, the proof system provides a means to take the specification of a relation and use it directly to compute a function (something that is not available directly when applying choice operators).

2 The basics of focusing in quantificational intuitionistic logic

We present a proof system for an intuitionistic theory of arithmetic by dividing it into three parts: Section 2.1 presents the proof system for the propositional fragment, Section 2.2 introduces quantification and equality of terms (at all types), and Section 3 introduces various inference rules for treating the fixed point connective.

2.1 Propositional intuitionistic logic

In this section, we present propositional intuitionistic logic and a focused proof system for it. Propositional intuitionistic logic formulas are given by the logical connectives $\wedge$, $\vee$, and $\supset$, the logical constants t and f, and atomic formulas. The focused system in Figure 1 contains not formulas but *polarized* formulas. Such polarized formulas differ from unpolarized formulas in two ways. First, the conjunction is replaced with two conjunctions $\wedge^+$ and $\wedge^-$ and the unit of conjunction t with t^+ and t^-. Second, every atomic formula A is

Structural rules

$$\frac{\Gamma, N \Downarrow N \vdash \cdot \Downarrow E}{\Gamma, N \Uparrow \cdot \vdash \cdot \Uparrow E}\, D_l \qquad \frac{C, \Gamma \Uparrow \Theta \vdash \Delta_1 \Uparrow \Delta_2}{\Gamma \Uparrow C, \Theta \vdash \Delta_1 \Uparrow \Delta_2}\, S_l \qquad \frac{\Gamma \Uparrow P \vdash \cdot \Uparrow E}{\Gamma \Downarrow P \vdash \cdot \Downarrow E}\, R_l$$

$$\frac{\Gamma \Downarrow \cdot \vdash P \Downarrow \cdot}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow P}\, D_r \qquad \frac{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow E}{\Gamma \Uparrow \cdot \vdash E \Uparrow \cdot}\, S_r \qquad \frac{\Gamma \Uparrow \cdot \vdash N \Uparrow \cdot}{\Gamma \Downarrow \cdot \vdash N \Downarrow \cdot}\, R_r$$

Negative Phase Introduction Rules

$$\frac{\Gamma \Uparrow \Theta \vdash \Delta_1 \Uparrow \Delta_2}{\Gamma \Uparrow t^+, \Theta \vdash \Delta_1 \Uparrow \Delta_2} \qquad \frac{\Gamma \Uparrow \cdot \vdash B_1 \Uparrow \cdot \quad \Gamma \Uparrow \cdot \vdash B_2 \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash B_1 \wedge^- B_2 \Uparrow \cdot} \qquad \frac{}{\Gamma \Uparrow \cdot \vdash t^- \Uparrow \cdot} \qquad \frac{}{\Gamma \Uparrow f^+, \Theta \vdash \Delta_1 \Uparrow \Delta_2}$$

$$\frac{\Gamma \Uparrow B_1, B_2, \Theta \vdash \Delta_1 \Uparrow \Delta_2}{\Gamma \Uparrow B_1 \wedge^+ B_2, \Theta \vdash \Delta_1 \Uparrow \Delta_2} \qquad \frac{\Gamma \Uparrow B_1, \Theta \vdash \Delta_1 \Uparrow \Delta_2 \quad \Gamma \Uparrow B_2, \Theta \vdash \Delta_1 \Uparrow \Delta_2}{\Gamma \Uparrow B_1 \vee B_2, \Theta \vdash \Delta_1 \Uparrow \Delta_2} \qquad \frac{\Gamma \Uparrow B_1 \vdash B_2 \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash B_1 \supset B_2 \Uparrow \cdot}$$

Positive Phase Introduction Rules

$$\frac{\Gamma \Downarrow \cdot \vdash B_1 \Downarrow \cdot \quad \Gamma \Downarrow B_2 \vdash \cdot \Downarrow E}{\Gamma \Downarrow B_1 \supset B_2 \vdash \cdot \Downarrow E} \qquad \frac{}{\Gamma \Downarrow \cdot \vdash t^+ \Downarrow \cdot} \qquad \frac{\Gamma \Downarrow \cdot \vdash B_1 \Downarrow \cdot \quad \Gamma \Downarrow \cdot \vdash B_2 \Downarrow \cdot}{\Gamma \Downarrow \cdot \vdash B_1 \wedge^+ B_2 \Downarrow \cdot}$$

$$\frac{\Gamma \Downarrow \cdot \vdash B_i \Downarrow \cdot}{\Gamma \Downarrow \cdot \vdash B_1 \vee B_2 \Downarrow \cdot}\ i \in \{1, 2\} \qquad \frac{\Gamma \Downarrow B_i \vdash \cdot \Downarrow E}{\Gamma \Downarrow B_1 \wedge^- B_2 \vdash \cdot \Downarrow E}\ i \in \{1, 2\}$$

Figure 1 The propositional fragment of cut-free LJF

assigned a polarity of either positive or negative in an arbitrary but fixed fashion. Thus, one can polarize atomic formulas (propositional variables) as being all positive or all negative or mixed. A polarized formula is *positive* if it is a positive atomic formula or its top-level logical connective is either t^+, $\wedge^+$, or $\vee$. A polarized formula is *negative* if it is a negative atomic formula or its top-level logical connective is either t^-, $\wedge^-$, or $\supset$.

Figure 1 contains the structural and introduction rules for the propositional fragment of the LJF focused proof system [17]. That proof system uses the following two kinds of sequents: *unfocused* sequents have the form $\Gamma \Uparrow \Theta \vdash \Delta_1 \Uparrow \Delta_2$, while *focused* sequents have the form $\Gamma \Downarrow \Theta \vdash \Delta_1 \Downarrow \Delta_2$. In those inference rules, the syntactic variables Δ, Θ, and Γ (possibly with subscripts) range over multisets of polarized formulas; P denotes a positive formula; N denotes a negative formula; C denotes either a negative formula or a positive atom; E denotes either a positive formula or a negative atom; and B denotes an unrestricted polarized formula. Since we are working with an intuitionistic sequent system, we require that all sequents in a focus proof have exactly one formula on the right: that is, the multiset union of Δ_1 and Δ_2 is a singleton. Since we are considering only *single focused* proof systems, we also require that sequents of the form $\Gamma \Downarrow \Theta \vdash \Delta_1 \Downarrow \Delta_2$ also have the property that the multiset union of Θ and Δ_1 is always a singleton. An unfocused sequent of the form $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow E$ is also called a *border* sequent.

A *derivation* is a tree structure of occurrences of inference rules: a derivation has one endsequent and possibly several premises. A derivation with no premises is a (focused) proof. A derivation that contains only negative sequents is a *negative phase*: such a phase contains introduction rules for negative connectives, and the storage rules (S_l and S_r). A derivation that contains only positive sequents is a *positive phase*: such a phase contains introduction rules for positive connectives. A *bipole* is a derivation whose endsequent and premises sequents are all border sequents: also, when reading the inference rules from the bottom up, the first inference rule is a decide rule (either D_l and D_r); the next rules are

Typed first-order quantification rules

$$\frac{\Sigma \vdash t : \tau \qquad \Sigma : \Gamma \Downarrow [t/x]B \vdash \cdot \Downarrow E}{\Sigma : \Gamma \Downarrow \forall x_\tau . B \vdash \cdot \Downarrow E} \qquad \frac{y : \tau, \Sigma : \Gamma \Uparrow \cdot \vdash [y/x]B \Uparrow \cdot}{\Sigma : \Gamma \Uparrow \cdot \vdash \forall x_\tau . B \Uparrow \cdot}$$

$$\frac{y : \tau, \Sigma : \Gamma \Uparrow [y/x]B, \Theta \vdash \Delta_1 \Uparrow \Delta_2}{\Sigma : \Gamma \Uparrow \exists x_\tau . B, \Theta \vdash \Delta_1 \Uparrow \Delta_2} \qquad \frac{\Sigma \Uparrow \cdot \vdash t : \tau \Uparrow \cdot \quad \Sigma : \Gamma \Downarrow \cdot \vdash [t/x]B \Downarrow \cdot}{\Sigma : \Gamma \Downarrow \cdot \vdash \exists x_\tau . B \Downarrow \cdot}$$

Equality rules

$$\frac{\Sigma\theta : \Gamma\theta \Uparrow \Theta\theta \vdash \Delta_1\theta \Uparrow \Delta_2\theta}{\Sigma : \Gamma \Uparrow s = t, \Theta \vdash \Delta_1 \Uparrow \Delta_2}\,\dagger \qquad \frac{}{\Sigma : \Gamma \Uparrow s = t, \Theta \vdash \Delta_1 \Uparrow \Delta_2}\,\ddagger \qquad \frac{}{\Sigma : \Gamma \Downarrow \cdot \vdash t = t \Downarrow \cdot}$$

There are two provisos: (†) θ is the mgu of s and t. (‡) t and s are not unifiable.

Figure 2 Focused proof rules for quantification and term equality.

positive introduction rules; then there is a release rule (either R_l or R_r); followed by negative introduction rules and storage rules (either S_l or S_r).

Figure 1 contains neither the initial rule nor the cut rule. The cut rule plays a minor role in this paper (for example, it is not part of our notion of computation). The initial rule will be important but not globally: we introduce it later when we need (variants of) it.

2.2 Quantification and term equality

In order to treat first-order quantification, sequents are extended to permit the binding of *eigenvariables* [10]. To that end, we prefix all $\Uparrow$ and $\Downarrow$ sequents with Σ :, where Σ is a list of variables that are considered bound over the sequent. When we write a prefix as $y : \tau, \Sigma$, we imply that y does not appear as one of the variables in Σ. The inference rules for term equality and quantification are displayed in Figure 2. Formulas with a top-level $\forall$ are polarized negatively while formulas with a top-level $\exists$ or equality are polarized positively.

Here, the expression $[t/x]B$ denotes the $\beta\eta$-long normal form of $(\lambda x.B)t$ and the judgment $\Sigma \vdash t : \tau$ denotes the fact that t is a term in $\beta\eta$-long form and with type τ. Given the discussion in Section 6.2, this later judgment can be replaced by $\Gamma \Uparrow \cdot \vdash t : \tau \Uparrow \cdot$. This identification assumes that all primitive types are given a negative bias.

While provability in the propositional fragment is known to be decidable [10], it has been shown in [25] that adding these rules for term equality and quantification results in an undecidable logic even if we restrict to just first-order terms and quantifiers and even without any predicate symbols (and, hence, without atomic formulas).

3 Inference rules for the fixed point connective

We shall now add to our collection of logical connectives a fixed point operator. There have been many treatments of fixed points and induction within proof systems such as those involving Peano's axioms and induction schemes or those using a specially designed proof system such as Scott induction [13]. Here, we restrict our attention to the rather minimalistic setting where the fixed point operator is treated as a logical connective in the sense that it will have left and right introduction rules. For most of this paper, we shall introduce one fixed point operator, say, μ and unfolding is used to define left and right introduction rules. While the resulting fixed point operator is self dual and rather weak, it can still play a useful role in proving some weak theorems of arithmetic [12, 19, 20] and it can explain significant aspects of the proof theory of model checking [14, 23]. It is possible to describe a more powerful proof system for fixed points that uses induction and co-induction rules to describe the introduction rules for the *least* and *greatest* fixed points [19, 22].

The logical constant μ is actually parametrized by a list of typed constants as follows:

$$\mu^n_{\tau_1,\ldots,\tau_n} : (\tau_1 \to \cdots \to \tau_n \to o) \to \tau_1 \to \cdots \to \tau_n \to o$$

where $n \geqslant 0$ and $\tau_1, \ldots, \tau_n$ are simple types. (Following Church [6], we use o to denote the type of formulas.) Expressions of the form $\mu^n_{\tau_1,\ldots,\tau_n} B t_1 \ldots t_n$ will be abbreviated as simply $\mu B \bar{t}$ (where $\bar{t}$ denotes the list of terms $t_1 \ldots t_n$). We shall also restrict fixed point expressions to use only *monotonic* higher-order abstraction: that is, in the expression $\mu^n_{\tau_1,\ldots,\tau_n} B t_1 \ldots t_n$ the expression B is equivalent (via $\beta\eta$-conversion) to $\lambda P_{\tau_1 \to \cdots \to \tau_n \to o}\ \lambda x^1_{\tau_1} \ldots \lambda x^n_{\tau_n}\ B'$ and where all occurrences of the variable P in B' occur to the left of an implication an even number of times. The unfolding of the fixed point expressions $\mu B\, \bar{t}$ yields $B(\mu B)\, \bar{t}$ and the introduction rules for μ state the logical equivalence of these two expressions.

▶ **Example 1.** Assume that we have a primitive type i and that there are two typed constants $z : i$ and $s : i \to i$. We shall abbreviate the terms z, $(s\ z)$, $(s\ (s\ z))$, $(s\ (s\ (s\ z)))$, etc by **0**, **1**, **2**, **3**, etc. The following two named fixed point expressions define the natural number predicate and the ternary relation of addition.

$$\begin{aligned} \text{nat} &= \mu\lambda N \lambda n (n = \mathbf{0} \vee \exists n' (n = s\ n' \wedge^+ N\ n')) \\ \text{plus} &= \mu\lambda P \lambda n \lambda m \lambda p ((n = \mathbf{0} \wedge^+ m = p) \vee \exists n' \exists p' (n = s\ n' \wedge^+ p = s\ p' \wedge^+ P\ n'\ m\ p')) \end{aligned}$$

The following theorem, proved using induction, states that the plus relation describes a (total) functional dependency between its first two arguments and its third.

$$\forall m, n (\text{nat } M \supset \exists k (\text{plus } m\ n\ k)) \wedge \forall m, n, p, q (plus\ m\ n\ p \supset plus\ m\ n\ q \supset p = q)$$

3.1 Focusing and unfolding

Given that μ-expressions (with the inference rules provided so far) are self dual, they can be polarized either negatively or positively. We shall pick the positive polarity since this is consistent with the polarity assignment that is most natural in the setting where the induction rule is added. Thus, the natural rules for unfolding such expressions is given via the first two inference rules of Figure 3.

Focused sequent calculus proof systems were originally developed for quantificational logic—as opposed to arithmetic—and in that setting, the bottom-up construction of the negative phase causes sequents to get strictly smaller (counting, for example, the number of occurrences of logic connectives). As a result, it was possible to design focused proof systems in which decide rules were not applied until all invertible rules were applied. We shall say that such proofs systems are *strongly* focused proof systems: examples of such systems can be found in [1, 17].

As is obvious from the first inference rules in Figure 3, the size of the formulas in the negative phase can increase when μ-expressions are unfolded. Thus, a more flexible approach to building negative phases should be considered. Some focused proof systems have been designed in which a decide rule can be applied without consideration of whether all or some of the invertible rules have been applied. Following [21], such proof systems are called *weak* focused proof systems: an early example of such a proof system is Girard's LC [11]. Since we wish to use the negative phase to do functional style, determinate computation, a

Fixed point rules

$$\frac{\Sigma\colon \Gamma \Uparrow B(\mu B)\bar{t}, \Delta \vdash \cdot \Uparrow E}{\Sigma\colon \Gamma \Uparrow \mu B\,\bar{t}, \Delta \vdash \cdot \Uparrow E}\ \mathtt{unfoldL}\dagger \qquad \frac{\Sigma\colon \Gamma \Downarrow \cdot \vdash B(\mu B)\bar{t} \Downarrow \cdot}{\Sigma\colon \Gamma \Downarrow \cdot \vdash \mu B\,\bar{t} \Downarrow \cdot}\ \mathtt{unfoldR}$$

Modified versions of the decide and release rules

$$\frac{\Sigma\colon \Gamma, N \Downarrow N; \Omega \vdash \cdot \Downarrow E}{\Sigma\colon \Gamma, N \Uparrow \Omega \vdash \cdot \Uparrow E}\ D_l\ddagger \quad \frac{\Sigma\colon \Gamma \Downarrow \cdot; \Omega \vdash P \Downarrow \cdot}{\Sigma\colon \Gamma \Uparrow \Omega \vdash \cdot \Uparrow P}\ D_r\ddagger \quad \frac{\Sigma\colon \Gamma \Uparrow P, \Omega \vdash \cdot \Uparrow E}{\Sigma\colon \Gamma \Downarrow P; \Omega \vdash \cdot \Downarrow E}\ R_l \quad \frac{\Sigma\colon \Gamma \Uparrow \cdot \vdash N \Uparrow \cdot}{\Sigma\colon \Gamma \Downarrow \cdot \vdash N \Downarrow \cdot}\ R_r$$

Initial rule

$$\frac{P \in \Omega}{\Sigma\colon \Gamma \Downarrow \cdot; \Omega \vdash P \Downarrow \cdot}\ I_r$$

The proviso † requires that $\mu B\,\bar{t}$ does not satisfy $\mathcal{S}$. The proviso ‡ requires Ω to be a multiset of μ-expressions that satisfy $\mathcal{S}$.

Figure 3 Rules governing fixed point unfolding, suspensions, and initial sequents.

weak focused system—with its possibility to stop in many different configurations—cannot provide the foundations that we need.

Instead of strong and weak focused proof systems, we modify strong focusing by allowing certain explicitly described μ-expressions appearing in the negative phase to be *suspended*. In that case, one can switch from a negative phase to a positive phase (using a decide rule) when the only remaining formulas in the negative phase are suspendable. In that case, those formulas are "put aside" during the processing of the positive phase and are reinstated when the positive phase switches to the negative phase (using a release rule). In more detail, let $\mathcal{S}$ denote a *suspension* predicate: this predicate is defined only on μ-expression and if $\mathcal{S}$ holds for $(\mu B\bar{t})$ then we say that this expression is suspended. The $\mathtt{unfoldL}$ rule in Figure 3 has the proviso that $\mathcal{S}$ does not hold of the μ-expression that is the subject of that inference rule. In order to accommodate suspended formulas, $\Downarrow$-sequents need to contain a new multiset zone, denoted by the syntactic variable Ω: in particular, they now have the structure $\Gamma \Downarrow \Theta; \Omega \vdash \Delta_1 \Downarrow \Delta_2$. All positive introduction rules ignore this new zone: for example, the left-introduction of $\wedge^-$ will now be written as

$$\frac{\Gamma \Downarrow B_i; \Omega \vdash \cdot \Downarrow E}{\Gamma \Downarrow B_1 \wedge^- B_2; \Omega \vdash \cdot \Downarrow E}\ i \in \{1, 2\}.$$

The suspension property $\mathcal{S}$ is defined at the mathematics level and, as a result, can make use of syntactic details about μ-expressions. For example, this property could be that the μ-expression contains more than, say, 100 symbols or that the first term in the list $\bar{t}$ is an eigenvariable. However, in order to guarantee that the negative phase is determinate, we need to require the following property:

$(*)$ For all μ-expressions $(\mu B\bar{t})$ and for all substitutions θ defined on the eigenvariables free in that expression, if $\mathcal{S}$ holds for $(\mu B\bar{t})\theta$ then $\mathcal{S}$ holds for $(\mu B\bar{t})$.

That is, if an instance of a μ-expression satisfies $\mathcal{S}$ after a substitution is applied, it must satisfy $\mathcal{S}$ before it was applied. This condition rules out the possible suspension condition "holds if it contains 100 symbols" but it allows the condition "holds if the first term in $\bar{t}$ is an eigenvariable". Furthermore, suspension properties should not, in general, be invariant under substitution since otherwise a suspended formula will remain suspended during the duration of proof construction (only be involved in the initial rule).

▶ **Example 2.** Consider the suspension predicate that is true of μ-expressions $\mu B\, t_1 \ldots t_n$ if and only if $n \geqslant 2$ and t_1 and t_2 are the same variable. Clearly, property $(*)$ does not hold and the construction of the negative phase can be non-confluent. For example, let A be $\mu p \lambda x \lambda y. x = a$ (where a is a constant) and consider the sequent $\Gamma \Uparrow u = v, Auv \vdash \cdot \Uparrow (E\ u)$. Since Auv is a μ-expression for which $\mathcal{S}$ does not hold, unfolding is applicable and yields the sequent $\Gamma \Uparrow u = v, u = a \vdash \cdot \Uparrow (E\ u)$ which then leads to the border sequent $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow (E\ a)$. However, the first step in the negative phase of the original sequent could have been the equality introduction, which yields $\Gamma \Uparrow Auu \vdash \cdot \Uparrow (E\ u)$ and this must mark the end of the negative phase since $A\ u\ u$ is a suspended formula.

Fortunately, this non-confluent behavior is ruled out by the $(*)$ property above.

▶ **Definition 3** (Purely positive formula)**.** A polarized formula in which all occurrences of logical connectives are polarized positively is called a *purely positive* formula. A μ-expression that is also purely positive will also be called a purely positive fixed point expression.

Horn clauses (Prolog) can provide immediate examples of purely positive fixed points as illustrated in Example 1. Let B be a purely positive formula. If $\Sigma\colon \Gamma \Downarrow \cdot \vdash B \Downarrow \cdot$ is provable then all proofs of that sequent are built of only positive right-introduction rules for t^+, $\wedge^+$, $\vee$, $\exists$, μ (unfolding) and equality. Similarly, If $\Sigma\colon \Gamma \Uparrow B \vdash \cdot \Uparrow \cdot$ is provable then all proofs of that sequent are built of only negative left-introduction rules for t^+, $\wedge^+$, $\vee$, $\exists$, μ (unfolding) and equality. Thus, focused proofs of B and $B \supset f^+$ are achieved by using only one phase. In particular, such proofs do not contain structural rules and the initial rule. As a result, synthetic inference rules are not decidable since they can encode arbitrary Horn clause specifications.

3.2 Phases as abstractions

Focused proof systems make it possible to define new inference rules by abstracting away details used in the construction of phases. The positive phase allows a simple abstraction since there is exactly one formula under focus in a positive sequent. A positive phase can be seen as the (derived) inference rule with a conclusion that is a border sequent and with premises that are marked by release rules.

There are, however, at least two challenges to making abstractions of negative phases. First, the premises of a negative phase may repeat the same sequents many times since there can be many paths to compute the result of a function. We shall choose to denote as the premises of the negative phase the *set* of border sequents. Second, there are many ways to process the don't care non-determinism that is possible when applying invertible rules. We will abstract away from those differences by simply ignoring *how* a phase is constructed since all constructions yield the same border sequents.

This second abstract is essentially the same motivation used in confluent rewriting systems: once one finds a path to a normal form, no other paths need to be considered since all other paths must yield the same normal form.

4 The polarity ambiguity of singleton sets

As we mentioned in the introduction, singleton sets can be used to help convert relations to functions: if the $n+1$-ary relation R describes a function from its first n arguments to its argument then the expression $(\lambda y. R(x_1, \ldots, x_n, y))$ denotes a singleton set (given fixed values for $x_1, \ldots, x_n$). The choice operators of ϵ or ι can then be applied to this singleton set to extract that element, resulting in a proper function $\lambda x_1 \ldots \lambda x_n \iota(\lambda y. R(x_1, \ldots, x_n, y))$.

Singleton sets play a role here as well. In fact, let P be a predicate of one argument so that it is provable that P is a singleton, namely,

$$(\exists x.P(x)) \wedge (\forall x, y.P(x) \supset P(y) \supset x = y)$$

As a consequence, the formulas $\exists x.P(x) \wedge Q(x)$ and $\forall x.P(x) \supset Q(x)$ are equivalent. If we used the ι-choice operator, these formulas would also be equivalent to $Q(\iota P)$.

Note that the sequent calculus treatment of $\exists x.P(x) \wedge Q(x)$ and $\forall x.P(x) \supset Q(x)$ are strikingly different. In particular, a proof of $\Sigma\colon \Gamma \Downarrow \cdot \vdash \exists x.P(x) \wedge Q(x) \Downarrow \cdot$ proceeds by guessing a term t and then attempting to prove $\Sigma\colon \Gamma \Downarrow \cdot \vdash P(t) \Downarrow \cdot$ and $\Sigma\colon \Gamma \Downarrow \cdot \vdash Q(t) \Downarrow \cdot$. Of course, since P denotes a singleton, there is at most one correct guess t and that guess is confirmed after it is inserted into the proof. On the other hand, a proof of $\Sigma\colon \Gamma \Uparrow \cdot \vdash \forall x.P(x) \supset Q(x) \Uparrow \cdot$ can be seen as computing the value that satisfies P. Proof construction for that sequent leads to proving $y, \Sigma : \Gamma \Uparrow P(y) \vdash Q(y) \Uparrow \cdot$. As mentioned in Section 3.1, this phase will move to completion by repeatedly unfolding fixed points and if the phase completes, the eigenvariable y will be instantiated to be the unique term t. Thus, the premises of this completed phase will have the shape $\Sigma\colon \Gamma \Uparrow \vdash \cdot \Uparrow Q(t)$ (assuming for the sake of argument that $Q(t)$ is a positive formula).

► **Example 4.** Using the definitions in Example 1, consider the construction of a negative phase of the form $x, \Sigma : \Gamma \Uparrow \text{plus } \mathbf{2}\ \mathbf{3}\ x \vdash \cdot \Uparrow (Q\ x)$ Since plus is a μ-expression, this sequent is proved by an unfoldL inference rule (assuming that $\mathcal{S}$ is false for all μ-expressions, i.e., nothing should be suspended). Unfolding yields an expression with a top-level disjunction, namely, $x, \Sigma : \Gamma \Uparrow ((\mathbf{2} = \mathbf{0} \wedge^+ \mathbf{3} = x) \vee \exists n' \exists x' (\mathbf{2} = s\ n' \wedge^+ x = s\ x' \wedge^+ \text{plus } n'\ \mathbf{3}\ x')) \vdash \cdot \Uparrow (Q\ x)$. Following the left introduction for that disjunction, we are left with proving two sequents: the left premises, $x, \Sigma : \Gamma \Uparrow ((\mathbf{2} = \mathbf{0} \wedge^+ \mathbf{3} = x) \vdash \cdot \Uparrow (Q\ x)$ is proved immediately since $\mathbf{2} = \mathbf{0}$ is not unifiable (Figure 2). A proof of the second premise must proceed as follows

$$\cfrac{\cfrac{x', \Sigma : \Gamma \Uparrow \text{plus } \mathbf{1}\ \mathbf{3}\ x' \vdash \cdot \Uparrow (Q\ (s\ x'))}{x, n', x', \Sigma : \Gamma \Uparrow (\mathbf{2} = s\ n' \wedge^+ x = s\ x' \wedge^+ \text{plus } n'\ \mathbf{3}\ x') \vdash \cdot \Uparrow (Q\ x)}}{x, \Sigma : \Gamma \Uparrow (\exists n' \exists x' (\mathbf{2} = s\ n' \wedge^+ x = s\ x' \wedge^+ \text{plus } n'\ \mathbf{3}\ x')) \vdash \cdot \Uparrow (Q\ x)}$$

(Here, the double line between sequents denotes the application of possibly several inference rules.) After several more inference steps, the negative phase terminates with the border premise $\Sigma\colon \Gamma \Uparrow \cdot \vdash \cdot \Uparrow (Q\ \mathbf{5})$. By ignoring the internal structure of phases, we have just the synthetic inference rule

$$\frac{\Sigma\colon \Gamma \Uparrow \cdot \vdash \cdot \Uparrow (Q\ \mathbf{5})}{x, \Sigma : \Gamma \Uparrow \text{plus } \mathbf{2}\ \mathbf{3}\ x \vdash \cdot \Uparrow (Q\ x)}$$

Furthermore, there were no choices involved in computing this phase. Note that the actual specification of the relation plus is used to compute the addition as a function. Later in Section 6 we shall show how we can use that synthetic inference rule to capture the more familiar looking rule

$$\frac{\Sigma\colon \Gamma \Uparrow \cdot \vdash \cdot \Uparrow (Q\ \mathbf{5})}{\Sigma\colon \Gamma \Uparrow \cdot \vdash \cdot \Uparrow (Q\ (\mathbf{2} + \mathbf{3}))}$$

► **Example 5.** Employing the suspension mechanism makes it possible for functional computation to be mixed with symbolic computation. In particular,

$$\text{times} = \mu\lambda P \lambda n \lambda m \lambda p((n = \mathbf{0} \wedge^+ p = \mathbf{0}) \vee \exists n' \exists p' (n = s\ n' \wedge^+ P\ n'\ m\ p' \wedge^+ \text{plus } p'\ m\ p))$$

The theorem that states that $(0 \times (x+1)) + y = y$ can be encoded and proved in this setting by taking two steps. First we translate this expression into the following sequent (using a technique described in Section 6).

$$\Gamma \Uparrow \cdot \vdash \forall u.\ \text{times}\ \mathbf{0}\ (s\ x)\ u \supset \forall v.\ \text{plus}\ u\ y\ v \supset v = y \Uparrow \cdot$$

Here, we assume (a rather typical) suspension mechanism: assume that μ-expressions that are built from plus and times are suspended if their first argument is an eigenvariable. Thus, when this sequent is reduced to

$$y, u, v, \Sigma : \Gamma \Uparrow \text{times}\ \mathbf{0}\ (s\ x)\ u,\ \text{plus}\ u\ y\ v \vdash v = y \Uparrow \cdot$$

only the times expression can be unfolded. After that unfolding, the eigenvariable u will be instantiated and the plus μ-expression can then also be unfolded. Finally, the negative phase ends with the border sequent $y, \Sigma : \Gamma \Uparrow \cdot \vdash \cdot \Uparrow y = y$ which is proved by a D_r rule followed by the right introduction for equality.

5 Equivalence classes

Equivalence relations play important roles in computation and reasoning. Occasionally, we have a relation that is not functional but all the possible outcomes are equivalent, for some specific equivalence relation. For example, if two lists are considered equivalent when they are permutations of each other, then the equivalence class of lists modulo that relation encodes multisets. Similarly, if two pair of integers (x, y) and (w, z) (where the y and z are not zero) are considered equivalent when $xz = wy$ then equivalence classes encode rational numbers.

The ambiguity of singletons can be lifted to computing with equivalence classes in the following sense. Let ρ be an equivalence relation. The familiar notion $[x]_\rho$ for the ρ-equivalence class containing x is just syntactic sugar for $\lambda y.x\rho y$.

Assume that ρ is an equivalence relation and that the following holds for $Q : i \to o$.

$$\forall x \forall y.\ x\ \rho\ y \supset [Q(x) \equiv Q(y)]$$

(Note that this theorem is immediate for all $Q : i \to o$ when ρ is equality.) The following equivalence holds.

$$[\forall x \in [y]_\rho \supset Q(x)] \equiv [\exists x \in [y]_\rho \wedge Q(x)]$$

In a more informal mathematical notation, one might replace either the above existential or universal expression with $Q([y]_\rho)$. While we shall not use this expression (it involves a typing error), it conveys the usual mathematical sense of this ambiguity: if we show that one member of an equivalence class satisfies such a property Q then all members of that equivalence class satisfies Q.

Obviously, we can generalize the notion of functional dependency to the following

$$\forall \bar{x}([\exists y.R(\bar{x}, y)] \wedge \forall y \forall z[R(\bar{x}, y) \supset R(\bar{x}, z) \supset y\rho z])$$

which states that the n-ary relation is a total function up to ρ. Thus, during the construction of a proof where one is asked to pick a term t that makes $R(x_1, \ldots, x_n, t)$ true, one can instead compute just any term t' such that $R(x_1, \ldots, x_n, t')$ instead (as long as the property established—Q above—is ρ-invariant). In that setting, we can also extend the phase-abstraction mechanism to exclude border premises that differ up to ρ.

$$\begin{aligned} \mathsf{Terms}: &\quad t, u ::= \lambda x.t \mid x\ k \mid \uparrow p \\ \mathsf{Values}: &\quad p, q ::= x \mid \downarrow t \\ \mathsf{Continuations}: &\quad k ::= \varepsilon \mid p :: k \mid \kappa x.t \end{aligned}$$

$$\frac{\Gamma \Uparrow \cdot \vdash t : N \Uparrow \cdot}{\Gamma \Downarrow \cdot \vdash \downarrow t : N \Downarrow \cdot}\, R_r \qquad \frac{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow t : E}{\Gamma \Uparrow \cdot \vdash t : E \Uparrow \cdot}\, S_r \qquad \frac{\Gamma \Downarrow \cdot \vdash p : P \Downarrow \cdot}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow \uparrow p : P}\, D_r \qquad \frac{}{\Gamma, x : a^+ \Downarrow \cdot \vdash x : a^+ \Downarrow \cdot}\, I_r$$

$$\frac{\Gamma, x : P \Uparrow \cdot \vdash \cdot \Uparrow t : E}{\Gamma \Downarrow P \vdash \cdot \Downarrow \kappa x.t : E}\, R_l/S_l \qquad \frac{\Gamma, x : N \Downarrow N \vdash \cdot \Downarrow k : E}{\Gamma, x : N \Uparrow \cdot \vdash \cdot \Uparrow x\ k : E}\, D_l \qquad \frac{}{\Gamma \Downarrow a^- \vdash \cdot \Downarrow \varepsilon : a^-}\, I_l$$

$$\frac{\Gamma, x : A \Uparrow \cdot \vdash t : B \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash \lambda x.t : A \supset B \Uparrow \cdot}\, \supset_r / S_l \qquad \frac{\Gamma \Downarrow \cdot \vdash p : A \Downarrow \cdot \qquad \Gamma \Downarrow B \vdash \cdot \Downarrow k : E}{\Gamma \Downarrow A \supset B \vdash \cdot \Downarrow p :: k : E}\, \supset_l$$

Figure 4 Cut-free LJF with term annotations

6 Term representation: turning formulas inside-out

6.1 Term annotations for propositional LJF

In Section 2.2 we extended the proof system in Figure 1 with quantifiers and term structures and in Section 3 with recursive definitions. Here we extend that original proof system in two different directions. First, instead of having all predicates (such as nat, plus, and times) be defined, we consider the usual approach to propositional logic where formulas can contain *undefined* atoms. When such atoms appear in polarized formulas, atomic formulas must be provided with an arbitrary but fixed polarity. Following the design of *LJF* [17], we extend the proof system in Figure 1 by adding these two polarized variants of the initial rule. Here, N_a ranges over negatively polarized atoms and P_a ranges over positively polarized atoms. Given that we are working with a propositional logic, it is possible to use a strongly focused version of LJF (as was given in [17]) and to insist that all formulas in the negative phase are processed in a left-to-right discipline. As a result, it is possible to fuse the store rule left (S_l) rule with other rules. The completeness of *LJF* found in [17] states that if B is an (unpolarized) tautology and $\hat{B}$ is *any* polarization of B, then there is an *LJF* proof of $\cdot \Uparrow \cdot \vdash \hat{B} \Uparrow \cdot$. Thus, polarization does not affect provability but, as we shall illustrate, it can affect the shape of proofs.

$$\frac{}{\Gamma \Downarrow N_a \vdash \cdot \Downarrow N_a}\, I_l \qquad \frac{}{\Gamma, P_a \Downarrow \cdot \vdash P_a \Downarrow \cdot}\, I_r$$

Our second extension of the proof system in Figure 1 is meant to harness the resulting variability in proofs in order to provide a rich representation for terms and formulas. Figure 4 contains the propositional *LJF* inference rules annotated with the λκ-term found in [4]. This term calculus contains three syntactic categories: Terms, Values, and Continuations. Note that it is the store-left (S_l) rule that results in bindings in term structures and that such binding can result in either a λ-abstraction or a κ-abstraction.

6.2 Two normal forms for simply typed terms

If all primitive types (atomic formulas) are given a negative polarity, then the terms annotating proofs in the sequents of Figure 4 provide the usual notion of βη-long normal form λ-terms. Recall that terms in βη-*long normal form* are of the form $\lambda x_1 \ldots \lambda x_n.h\ t_1 \ldots t_m$

where h is a variable or constant, where $t_1, \ldots, t_m$ is a list of terms in $\beta\eta$-long normal form, and where the term $(h\ t_1 \ldots t_m)$ has primitive type. In particular, if we use $[\![\cdot]\!]$ to translate such λ-terms into term of the first syntactic category in Figure 4, then

$$[\![\lambda x_1 \ldots \lambda x_n.h\ t_1 \ldots t_m]\!] = \lambda x_1 \ldots \lambda x_n h\ (\downarrow[\![t_1]\!] :: \cdots :: \downarrow[\![t_m]\!] :: \varepsilon)$$

Note that this translation transforms the application of the function h from one argument at a time to the application of h to a list of all its arguments. Such a formal connection between $\beta\eta$-long normal forms and this style of term representation was made by Herbelin using his *LJT* sequent calculus [15]. When all primitive types are given a negative bias, then no formulas are given a positive bias and, as a result, the inference rule named R_l/S_l does not appear in such proofs and terms do not contain the κ binding operator.

▶ **Example 6.** Let i be a primitive type that will be considered negatively biased in the LJF proof system. The only terms t for which $\Gamma \Uparrow \cdot \vdash t : (i \supset i) \supset i \supset i \Uparrow \cdot$ is provable are encodings of the Church numeral. In particular, the terms corresponding to the first three numerals are $\lambda f \lambda x.x\ \varepsilon$, $\lambda f \lambda x.f\ (\downarrow(x\ \varepsilon) :: \varepsilon)$, and $\lambda f \lambda x.f\ (\downarrow(f\ (\downarrow(x\ \varepsilon) :: \varepsilon)) :: \varepsilon)$.

If all primitive types are given a positive bias, then the terms annotating proofs in the sequents in Figure 4 provides a formal definition of a normal form similar to the one described in [9] and which is commonly called *administrative normal form* (ANF).

▶ **Definition 7.** A simply typed λ-term is in administrative normal form (ANF) when written

- as $\lambda x_1 \ldots \lambda x_n.\uparrow h$, where $n \geqslant 0$ and h is a variable of primitive type
- or as $\lambda x_1 \ldots \lambda x_n.h\ (p_1 :: \cdots :: p_m :: \kappa y.t)$ where $n, m \geqslant 0$ and the type of y is primitive.

In both cases t is a simply typed term in ANF and in the later case values $p_1, \cdots, p_m$ are either variables of primitive type or are of the form $\downarrow t$ where t is in ANF.

Note the following: (1) If p_i is not a variable, then it must denote a term of arrow type and, hence, it will be a λ-abstraction: that is, immediately following the $\downarrow\cdot$ there must be a λ-abstraction. (2) A closed term in ANF with a type of order 2 or less is of the form $\lambda x_1 \ldots \lambda x_n.t$ where the types of $x_1, \ldots, x_n$ are either primitive or first-order and where t does not contain any λ. It can be the case, however, that t contains κ bindings. (3) If we ignore the requirements on certain variables being of primitive type, then this definition can be extended to untyped λ-terms.

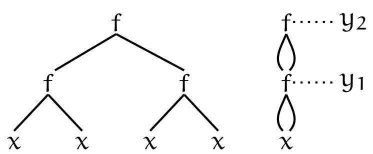
In order to facilitate the presentation of λ-terms in ANF format, we introduce the following convention. Instead of $\lambda x_1 \ldots \lambda x_n.\uparrow h$ we will simply drop the uparrow: $\lambda x_1 \ldots \lambda x_n.h$ (remembering that h is a variable of primitive type). Also, instead of

$$\lambda x_1 \ldots \lambda x_n.h\ (p_1 :: \cdots :: p_m :: \kappa y.t) \quad \text{we write} \quad \lambda x_1 \ldots \lambda x_n.\ \textbf{name}\ y = h\ (p_1, \ldots, p_m)\ \textbf{in}\ t$$

(remember that y is a variable of primitive type) and where $p_1, \ldots, p_m$ is a list of either variables (of primitive types) or λ-abstractions that are also in ANF.

We use the keyword "name" here instead of "let" since let-expressions are often considered to be abbreviations for β-redexes: that is, (let $x = s$ in t) is often considered equal to $((\lambda x.t)\ s)$. Here, however, the name-expressions denote normal terms since they are annotations of cut-free sequent calculus proofs.

The figure to the right illustrates two ways of representing a labeled binary tree of height 2. Clearly, the representation on the left takes exponential space as the height increases while the representation on the right increases linearly with the height. Here we assume that x



and f are two bound variables of type i and $i \to i \to i$, respectively. Choosing between these two representation schemes involves assigning either negative or positive polarity to the atomic formula (primitive type) i. For example, if i is polarized negatively, then there is an LJF proof that is annotated with the term $f\ (\downarrow(f\ (\downarrow(x\varepsilon)::\downarrow(x\varepsilon)::\varepsilon))::\downarrow(f\ (\downarrow(x\varepsilon)::\downarrow(x\varepsilon)::\varepsilon))::\varepsilon)$. or in a more friendly syntax: $f\ (f\ (x,\ x),\ f\ (x,\ x))$. On the other hand, when i is polarized positively, the above term is no longer a proper annotation of an LJF proof while the term

$$\textbf{name}\ y_1 = (f\ x\ x)\ \textbf{in}\ \textbf{name}\ y_2 = (f\ y_1\ y_1)\ \textbf{in}\ y_2$$

is one. Since the ANF term format allows subterms to be shared, that format can allow for much smaller term structures. While sharing is a feature of ANF, we shall not require it to be particularly well behaved. For example, it is possible for a term in ANF to have *vacuous naming*—i.e., a named term that is never used in the name's scope—or *redundant naming*—i.e., the same term can be named more than once. For example, the term

$$\textbf{name}\ y_1 = (f\ x\ x)\ \textbf{in}\ \textbf{name}\ y_2 = (f\ y_1\ y_1)\ \textbf{in}\ \textbf{name}\ y_3 = (f\ y_1\ y_1)\ \textbf{in}\ y_2$$

is in ANF even though it has vacuous and redundant naming. One might imagine also that multi-focusing can be used to allow parallel naming, such as in the expression

$$\textbf{name}\ y_1 = (f\ x\ x)\ \textbf{in}\ \textbf{name}\ y_2 = (f\ y_1\ y_1)\ \textbf{and}\ y_3 = (f\ y_1\ y_1)\ \textbf{in}\ y_2.$$

One might also expect that the concept of *maximal multifocusing* [5] could relate to insisting on "maximal sharing". In this paper, we shall not use multifocused proofs nor insist on the absence of vacuous or redundant naming.

6.3 Mixed term representations

The syntax of formulas of arithmetic statements depends on two primitive types: the type of formulas o and of numerals i. We present several examples of term representations below where o is polarized negatively and nat is polarized positively. We also allow the binary infix term constructors $+$ and $*$ of type $i \to i \to i$ as well as the formula constructor $<$ (the less-than relation) of type $i \to i \to o$.

▶ **Example 8.** When the type i for numerals is polarized positively, the λκ-calculus does not allow for expressions of the form $(s \cdots (sz) \cdots)$. Instead, encoding an expression of the form $P(2+2)$ can be done as follows:

$$\textbf{name}\ 1 = (s\ 0)\ \textbf{in}\ \textbf{name}\ 2 = (s\ 1)\ \textbf{in}\ \textbf{name}\ x = 2+2\ \textbf{in}\ P(x)$$

Thus, numerals are really treated as pointers into a sequence of successor terms.

▶ **Example 9.** The formula $\forall x[(x^2+6) = 5x \supset (x = 2 \vee x = 3)]$ can be written as the λκ-term $\forall x[\textbf{name}\ y = x * x\ \textbf{in}\ \textbf{name}\ u = 5 * x\ \textbf{in}\ \textbf{name}\ v = y + 6\ \textbf{in}\ (v = u \supset (x = 2 \vee x = 3))]$.

The inversion of syntax that appears in ANF is familiar to those computing with relations instead of functions, as the following example illustrates.

▶ **Example 10.** To prove $(4*(5+2)) < 8+7$ in a setting with only relations (such as, say, in Prolog) one can rewrite that inequality as the following (equivalent) formulas of arithmetic.

$$\exists x(\text{plus}\ 5\ 2\ x \wedge \exists y(\text{times}\ 4\ x\ y \wedge \exists z(\text{plus}\ 8\ 7\ z \wedge y < z)))$$
$$\forall x(\text{plus}\ 5\ 2\ x \supset \forall y(\text{times}\ 4\ x\ y \supset \forall z(\text{plus}\ 8\ 7\ z \supset y < z)))$$

Here, the binary operators $+$ and $*$ are interpreted by corresponding ternary predicates.

$$\frac{\Sigma\colon \Gamma \Uparrow R_f\ \bar{x}\ y, B, \Theta \vdash \Delta_1 \Uparrow \Delta_2}{\Sigma\colon \Gamma \Uparrow \textbf{name}\ y = f\ \bar{x}\ \textbf{in}\ B, \Theta \vdash \Delta_1 \Uparrow \Delta_2} \qquad \frac{\Sigma\colon \Gamma \Uparrow R_f\ \bar{x}\ y, \Theta \vdash B \Uparrow \cdot}{\Sigma\colon \Gamma \Uparrow \Theta \vdash \textbf{name}\ y = f\ \bar{x}\ \textbf{in}\ B \Uparrow \cdot}$$

$$\frac{\Sigma\colon \Gamma \Uparrow \cdot \vdash \textbf{name}\ x = f\ \bar{x}\ \textbf{in}\ B \Uparrow \cdot}{\Sigma\colon \Gamma \Downarrow \cdot \vdash \textbf{name}\ x = f\ \bar{x}\ \textbf{in}\ B \Downarrow \cdot} \qquad \frac{\Sigma\colon \Gamma \Uparrow \textbf{name}\ x = t\ \textbf{in}\ B \vdash \cdot \Uparrow \Delta}{\Sigma\colon \Gamma \Downarrow \textbf{name}\ x = t\ \textbf{in}\ B \vdash \cdot \Downarrow \Delta}$$

Figure 5 Introduction rules for interpreted constructors

6.4 Interpreting term constructors

As Examples 8 and 9 illustrate, arithmetic formulas can contain a mix of *uninterpreted* term constructors (for example, the constructor for numerals z and s) and *interpreted* term constructors (for example, $+$ and $*$).

Presumably, the formal introduction of a new interpreted term constructor such as $f : i \to \ldots \to i \to i$ of n arguments must be tied to an interpreting μ-expression R_f of $n{+}1$-arity and a formal proof that R_f encodes a function, i.e.,

$$\forall \bar{x}([\exists y.R_f(\bar{x}, y)] \wedge \forall y \forall z[R_f(\bar{x}, y) \supset R_f(\bar{x}, z) \supset y = z]).$$

That is, achieving a proof of this theorem permits the introduction of a new constructor f where $y = f\ x_1 \ldots\ x_n$ is interpreted by $R_f\ x_1 \ldots\ x_n\ y$. In principle, this means that the formula ($\textbf{name}\ y = f\ x_1 \ldots\ x_n\ \textbf{in}\ B$) is interpreted as either $\forall y(R_f\ x_1 \ldots\ x_n\ y \supset B)$ or $\exists y(R_f\ x_1 \ldots\ x_n\ y \wedge^+ B)$. Clearly, the naming construction is a *self dual* operator on formulas in the sense that $\neg(\textbf{name}\ y = f\ x_1 \ldots\ x_n\ \textbf{in}\ B)$ is equivalent to $(\textbf{name}\ y = f\ x_1 \ldots\ x_n\ \textbf{in}\ \neg B)$. As a result, such formulas are said to have an *ambiguous* polarity since they can be coerced to be negative or positive. The introduction rules for interpreted term constructors is given in Figure 5.

6.5 A final extension

In order to treat the naming (sharing) of structures built using uninterpreted symbols within proofs and computations, we need to add to our sequents (both $\Uparrow$ and $\Downarrow$) an additional zone (using the Ψ syntactic variable) that explicitly retains the naming relation. We do this by adding the Ψ context to all the previous arithmetic related sequents and inference rules. We also add the inference rules that appear in Figure 6. In the first four of these sequents, the formula-level binder $\textbf{name}\ y = t\ \textbf{in}\ B$ is translated to a proof-level binder by adding the pair $y := t$ to the Ψ context.

The quantifier rules that instantiate their quantifier with terms are modified in Figure 6 so that the naming structure of sequents is respected. In particular, those rules employ the premise $\Sigma, \Sigma(\Psi) \Uparrow \cdot \vdash t : \tau \Uparrow \cdot$. (Here, $\Sigma(\Psi)$ is the set of (typed) variables that are bound in Ψ.) Thus, the term t is, in general, a $\lambda\kappa$-term. The inference rules for equality must also be changed in order to account for the treatment of $\lambda\kappa$-term: with only first-order constructors present (such as in our treatment of natural numbers), the treatment of unification in this setting can be based on the Martelli-Montanari algorithm [18].

7 Conclusion

We have presented a treatment of functional computation based on relations. Principles in proof theory provided both a method of moving expressions denoting embedded computation into naming-combinators of the logic (ANF normal form) and a means of organizing

Name binding rules: the variable x is not bound in Σ nor in Ψ.

$$\frac{\Sigma : x := t, \Psi; \Gamma \Uparrow B, \Theta \vdash \Delta_1 \Uparrow \Delta_2}{\Sigma \colon \Psi; \Gamma \Uparrow \textbf{name}\ x = t\ \textbf{in}\ B, \Theta \vdash \Delta_1 \Uparrow \Delta_2} \qquad \frac{\Sigma : x := t, \Psi; \Gamma \Uparrow \cdot \vdash B \Uparrow \cdot}{\Sigma \colon \Psi; \Gamma \Uparrow \cdot \vdash \textbf{name}\ x = t\ \textbf{in}\ B \Uparrow \cdot}$$

$$\frac{\Sigma : x := t, \Psi; \Gamma \Downarrow \cdot \vdash B \Downarrow \cdot}{\Sigma \colon \Psi; \Gamma \Downarrow \cdot \vdash \textbf{name}\ x = t\ \textbf{in}\ B \Downarrow \cdot} \qquad \frac{\Sigma : x := t, \Psi; \Gamma \Downarrow B \vdash \cdot \Downarrow E}{\Sigma \colon \Psi; \Gamma \Downarrow \textbf{name}\ x = t\ \textbf{in}\ B \vdash \cdot \Downarrow E}$$

Positive phase quantifier rules

$$\frac{\Sigma, \Sigma(\Psi) \Uparrow \cdot \vdash t : \tau \Uparrow \cdot \quad \Sigma \colon \Psi; \Gamma \Downarrow [t/x]B \vdash \cdot \Downarrow E}{\Sigma \colon \Psi; \Gamma \Downarrow \forall x_\tau . B \vdash \cdot \Downarrow E} \qquad \frac{\Sigma, \Sigma(\Psi) \Uparrow \cdot \vdash t : \tau \Uparrow \cdot \quad \Sigma \colon \Psi; \Gamma \Downarrow \cdot \vdash [t/x]B \Downarrow \cdot}{\Sigma \colon \Psi; \Gamma \Downarrow \cdot \vdash \exists x_\tau . B \Downarrow \cdot}$$

Figure 6 The incorporation of the *naming* context Ψ.

Gentzen-style introduction rules so that functional computations can be identified as one specific phase of computation (the negative phase). Since this view of computation is based on the construction of cut-free proof, it is rather different from, say, the Curry-Howard correspondence.

While we have illustrated most of this mechanism using first-order term structures (such as Peano's numerals), the proof theory behind *LJF* works at all finite types. As a result, this approach to encoding functional programming can naturally be extended to treat richer terms than just the first-order ones.

Acknowledgments. We thank Beniamino Accattoli for his comments on an earlier draft of this paper.

References

1. J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.
2. A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant, and R. Saillard. Dedukti: a logical framework based on the $\lambda\Pi$-calculus modulo theory. Unpublished, 2016.
3. D. Baelde. Least and greatest fixed points in linear logic. *ACM Trans. on Computational Logic*, 13(1), Apr. 2012.
4. T. Brock-Nannestad, N. Guenot, and D. Gustafsson. Computation in focused intuitionistic logic. In M. Falaschi and E. Albert, editors, *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14–16, 2015*, pages 43–54, 2015.
5. K. Chaudhuri, D. Miller, and A. Saurin. Canonical sequent proofs via multi-focusing. In G. Ausiello, J. Karhumäki, G. Mauri, and L. Ong, editors, *Fifth International Conference on Theoretical Computer Science*, volume 273 of *IFIP*, pages 383–396. Springer, Sept. 2008.
6. A. Church. A formulation of the Simple Theory of Types. *J. of Symbolic Logic*, 5:56–68, 1940.
7. D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-Pi-calculus modulo. In S. R. D. Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *LNCS*, pages 102–117. Springer, 2007.
8. R. Dyckhoff and S. Lengrand. Call-by-value λ-calculus and LJQ. *J. of Logic and Computation*, 17(6):1109–1134, 2007.

9 C. Flanagan, A. Sabry, B. F. Duba, and M. Felleisen. The essence of compiling with continuations. *ACM SIGPLAN Notices*, 28(6):237–247, 1993.

10 G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1935.

11 J.-Y. Girard. A new constructive logic: classical logic. *Math. Structures in Comp. Science*, 1:255–296, 1991.

12 J.-Y. Girard. A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu, Feb. 1992.

13 M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *LNCS*. Springer, 1979.

14 Q. Heath and D. Miller. A proof theory for model checking: An extended abstract. In I. Cervesato and M. Fernández, editors, *Proceedings Fourth International Workshop on Linearity (LINEARITY 2016)*, volume 238 of *EPTCS*, Jan. 2017.

15 H. Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Computer Science Logic, 8th International Workshop, CSL '94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1995.

16 D. Hilbert and P. Bernays. *Grundlagen der Mathematik II.* Springer Verlag, 1939.

17 C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.

18 A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Lanuages and Systems*, 4(2):258–282, Apr. 1982.

19 R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.

20 P. Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.

21 R. J. Simmons and F. Pfenning. Weak focusing for ordered linear logic. Technical Report CMU-CS-10-147, Carnegie Mellon University, Apr. 2011.

22 A. Tiu and A. Momigliano. Cut elimination for a logic with induction and co-induction. *Journal of Applied Logic*, 10(4):330–367, 2012.

23 A. Tiu, G. Nadathur, and D. Miller. Mixing finite success and finite failure in an automated prover. In *Empirically Successful Automated Reasoning in Higher-Order Logics (ESHOL'05)*, pages 79–98, Dec. 2005.

24 A. S. Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer, 1973.

25 A. Viel and D. Miller. Proof search when equality is a logical connective. Presented to the International Workshop on Proof-Search in Type Theories, July 2010.