Combining Deduction Modulo and Logics of Fixed-Point Definitions

David Baelde
IT University of Copenhagen

Gopalan Nadathur
University of Minnesota

Abstract—Inductive and coinductive specifications are widely used in formalizing computational systems. Such specifications have a natural rendition in logics that support fixed-point definitions. Another useful formalization device is that of recursive specifications. These specifications are not directly complemented by fixed-point reasoning techniques and, correspondingly, do not have to satisfy strong monotonicity restrictions. We show how to incorporate a rewriting capability into logics of fixed-point definitions towards additionally supporting recursive specifications. In particular, we describe a natural deduction calculus that adds a form of “closed-world” equality—a key ingredient to supporting fixed-point definitions—to deduction modulo, a framework for extending a logic with a rewriting layer operating on formulas. We show that our calculus enjoys strong normalizability when the rewrite system satisfies general properties and we demonstrate its usefulness in specifying and reasoning about syntax-based rewrite system satisfies general properties and we demonstrate its usefulness in specifying and reasoning about syntax-based rewrite system.

I. INTRODUCTION

Fixed-point definitions constitute a widely used specification device in computational settings. The process of reasoning about such definitions can be formalized within a logic by including a proof rule for introducing predicates from their definition, and a case analysis rule for eliminating such predicates in favor of the definitions through which they might have been derived. For example, given the following definition of natural numbers

\[ \text{nat} \, 0 \triangleq \top \quad \text{nat} \, (s \, x) \triangleq \text{nat} \, x \]

the introduction and elimination rules would respectively build in the capabilities of recognizing natural numbers and of reasoning by case analysis over them. When definitional clauses are positive, they are guaranteed to admit a fixed point and the logic can be proved to be consistent. Further, least (resp. greatest) fixed points can be characterized by adding an induction (resp. coinduction) rule to the logic. These kinds of treatments have been added to second-order logic [13], [15], type theory [17] and first-order logics [14], [18], [20], [22].

The case analysis rule, which corresponds under the Curry-Howard isomorphism to pattern matching in computations, is complex in many formulations of the above ideas, and the (co)induction rules are even more so. By identifying and utilizing a suitable notion of equality, it is possible to give these rules a simple and elegant rendition. For example, the two clauses for \text{nat} can be transformed into the following form:

\[ \text{nat} \, x \triangleq x = 0 \vee \exists y. \, x = s \, y \land \text{nat} \, y \]

The case analysis rule can then be derived by unfolding a \text{nat} hypothesis into its single defining clause and using elimination rules for disjunction and equality. However, to obtain the expected behavior, equality elimination has to internalize aspects of term equality such as disjointness of constructors; e.g., the 0 branch should be closed immediately if the instantiation of \( x \) has the form \( s \, n \). The introduction of this separate notion of equality, which we refer to as \textit{closed-world equality}, has been central to the concise formulation of generic (co)induction rules [20]. Further, fixed-point combinators can be introduced to make the structure of (co)inductive predicates explicit rather than relying on a side table of definitions. Thus, the (inductive) definition of natural numbers may simply be rendered as \( \mu (\lambda x. \, x = 0 \vee \exists y. \, x = s \, y \land N \, y) \). Fixed point combinators simplify and generalize the theory, notably enabling mutual (co)induction schemes from the natural (co)induction rules [2], [3]. The logics resulting from this line of work, which we refer to as logics of fixed-point definitions from now on, have a simple structure that is well-adapted to automated and interactive proof-search [4], [5]. Moreover, they can be combined with features such as generic quantification that are useful in capturing binding structure to yield calculi that are well-suited to formalizing the meta-theory of computational and logical systems [10], [11], [16].

Logics featuring (co)inductive definitions can be made more powerful by adding another genre of definitions: recursive definitions based on inductive sets. A motivating context for such definitions is provided by the Tait-style strong normalizability argument [19], which figures often in the meta-theory of computational systems. For the simply typed \( \lambda \)-calculus, this argument relies on a reducibility relation specified by the following clauses:

\[ \text{red} \, \epsilon \, e \triangleq \text{sn} \, e \]
\[ \text{red} \, (t_1 \rightarrow t_2) \, e \triangleq \forall e'. \, \text{red} \, t_1 \, e' \supset \text{red} \, t_2 \, (e \, e') \]

We assume here that \( \epsilon \) is the sole atomic type and that \( \text{sn} \) is a predicate that recognizes strong normalizability. The specification of \text{red} looks deceptively like a fixed-point definition. However, treating it as such is problematic because the second clause in the definition does not satisfy the positivity condition. More importantly, the Tait-style argument does
not involve reasoning on red like we reason on fixed-point definitions. Instead of performing case-analysis or induction on red, properties are proved about it using an (external) induction on types and the clauses for red mainly support an unfolding of the definition once the structure of a type is known [12]. Generally, recursive definitions are distinguished by the fact that they embody computations or rewriting within proofs rather than the case analysis and speculative rewriting that is characteristic of fixed-point based reasoning.

In this paper, we show how to incorporate the capability of recursive definitions into logics of fixed-point definitions. At a technical level, we do this by introducing least and greatest fixed points and the idea of closed-world equality into deduction modulo [7], a framework for extending a logic with a rewriting layer that operates on formulas and terms. This rewriting layer allows for a transparent treatment of recursive definitions, but a satisfactory encoding of closed-world equality (and thus fixed-point definitions) seems outside its reach. This dichotomy actually highlights the different strengths of logics of fixed-point definitions and deduction modulo: while the former constitute excellent vehicles for dealing with (co)inductive definitions, the rewriting capability of the latter is ideally suited for supporting recursive definitions. By extending deduction modulo with closed-world equality and fixed points, we achieve a combination of these strengths. This combination also clarifies the status of our equality: we show that it is compatible with a theory on terms.

The rest of the paper is structured as follows. In Section II, we motivate and present our logical system. Section III describes reductions on proofs. Section IV provides a proof of strong normalizability that is modular in the rewrite rules being considered. We use this result to facilitate recursive definitions in Section V and we illustrate their use in formalizing the meta-theory of programming languages. Section VI discusses related and future work.

II. Deduction Modulo with Fixed-Points and Equality

We present our extension to deduction modulo in the form of a typing calculus for appropriately structured proof terms. This gives us a convenient tool for defining proof reductions and proving strong normalizability in later sections.

A. Formalizing closed-world equality

We first provide an intuition into our formalization of the desired form of equality. The rule for introducing an equality is the expected one: two terms are equal if they are congruent modulo the operative rewriting relation. Denoting the congruence by ≡, this rule can simply be

\[ \Gamma \vdash t = t' \quad t \equiv t' \]

The novelty is in the elimination rule that must encapsulate the closed-world interpretation. This can be captured in the form of a case analysis over all unifiers of the eliminated equality; the unifiers that are relevant to consider here would instantiate variables of universal strength, called eigenvariables, in the terms. One formulation of this idea that has been commonly used in the literature is the following:

\[ \frac{\Gamma \vdash t = t' \quad [\Gamma \theta \vdash P \theta_i | \theta_i \in csu(t, t')] }{ \Gamma \vdash P } \]

The notation csu(t, t') is used here to denote a complete set of unifiers for t and t' modulo ≡, i.e., a set of unifiers such that every unifier for the two terms is subsumed by a member of the set. The closed world assumption is expressed in the fact that \( \Gamma \vdash P \) needs to be proved under only these substitutions. Note in particular that the set of right premises here is empty when t and t' are not unifiable, i.e., have no common instances.

The equality elimination rule could have simply used the set of all unifiers for t and t'. Basing it on csu instead allows the cardinality of the premise set to be controlled, typically permitting it to be reduced to a finite collection from an infinite one. However, a problem with the way this rule is formulated is that this property is not stable under substitution. For example, consider the following derivation in which x and y are variables:

\[ p \cdot x, x = y \vdash x = y \\
 p \cdot x, x = y \vdash p \cdot y \]

If we were to apply the substitution \([t_1/x, t_2/y]\) to it, the branching structure of the derivation would have to be changed to reflect the nature of a csu for t_1 and t_2: this could well be an infinite set. A related problem manifests itself when we need to substitute a proof \(\pi\) for an assumption into the derivation. If we were to work the proof substitution eagerly through each of the premises in the equality elimination rule, it would be necessary to modify the structure of \(\pi\) to accord with the term substitution that indexes each of the premise derivations. In the context of deduction modulo, the instantiation in \(\pi\) can create new opportunities for rewriting formulas. Since the choice of the "right" premise cannot be determined upfront, the eager propagation of proof substitutions into equality eliminations...
can lead to a form of speculative rewriting which, as we shall see, is problematic when recursive definitions are included.

We avoid these problems by formulating equality elimination in a way that allows for the suspension of term and proof substitutions. Specifically, this rule is

\[
\Gamma' \vdash \theta = \ell \theta \quad \Gamma' \vdash \Gamma \theta \quad \{ \Gamma \theta_i \vdash P \theta_i \mid \theta_i \in \text{csu}(t, t') \}
\]

Here, \( \Gamma' \vdash \theta = \ell \theta \) means that there is a derivation of \( \Gamma' \vdash Q \) for any \( Q \in \Gamma \theta \). This premise, that introduces a form of cut, allows us to delay the propagation of proof substitutions over the premises that represent the case analysis part of the rule. Notice also that we consider \( \text{csu} \) for \( t \) and \( t' \) and not \( \theta \) and \( \ell \theta \) over these premises, i.e., the application of the substitution \( \theta \) is also suspended. Of course, these substitutions must eventually be applied. Forcing the application becomes the task of the reduction rule for equality that also simultaneously selects the right branch in the case analysis.

Our equality elimination rule also has the pleasing property of allowing the structure of proofs to be preserved under substitutions. For example, the proof

\[
p \ x, x = y \vdash \ x = y \quad p \ x, x = y \vdash p \ x \quad (p \ x)[y/x] \vdash (p \ y)[y/x]
\]

under the substitution \( \theta := [t_1/x, t_2/y] \) becomes

\[
\Gamma' \vdash (x = y)\theta \quad \Gamma' \vdash (p \ x)\theta \quad (p \ x)[y/x] \vdash (p \ y)[y/x]
\]

where \( \Gamma = (p \ t_1, t_1 = t_2) \).

B. The logic \( \mu NJ \) modulo

The syntax of our formulas is based on a language of typed \( \lambda \)-terms. We do not describe this language in detail and assume only that it is equipped with standard notions of variables and substitutions. We distinguish \( o \) as the type of propositions. Term types, denoted by \( \gamma \), are ones that do not contain \( o \). Predicates are expressions of type \( \gamma_1 \rightarrow \ldots \rightarrow \gamma_n \rightarrow o \). Both formulas and predicates are denoted by \( P \) or \( Q \). We use \( p \) or \( q \) for predicate variables and \( a \) for predicate constants. Terms are expressions of term types, and shall be denoted by \( t, u \) or \( v \). We use \( x, y \) or \( z \) for term variables. All expressions are considered up to \( \beta \) - and \( \eta \)-conversion. In addition to that basic syntactic equality, we assume a congruence relation \( = \). In Section V, we will describe conditions on such a congruence relation that are sufficient for ensuring the consistency of the logic.

**Definition 1.** A unifier of \( u \) and \( v \) is a substitution \( \theta \) such that \( u \theta \equiv v \theta \). A complete set of unifiers for \( u \) and \( v \), written \( \text{csu}(u, v) \), is a set \( \{ \theta \} \) of unifiers of \( u \) and \( v \), such that any other unifier of \( u \) and \( v \) is of the form \( \theta \theta' \) for some \( i \) and \( \theta' \). Note that complete sets of unifiers may not be unique. However, this ambiguity will be harmless in our setting.

**Definition 2.** Formulas are built as follows:

\[
\begin{align*}
\pi &::= \alpha \mid \ell \pi \mid \delta_c(\pi) \\
& \mid \lambda \alpha \cdot \pi \mid (\pi \pi') \\
& \mid (\pi, \pi) \mid \text{proj}_1(\pi) \mid \text{proj}_2(\pi) \\
& \mid \text{in}_1(\pi) \mid \text{in}_2(\pi) \mid \delta_\ell(\pi_1, \alpha \pi_2, \beta \pi_3) \\
& \mid \lambda x. \pi \mid (\pi t) \\
& \mid (t, \pi) \mid \delta_\ell(\pi, x, \alpha \pi') \\
& \mid \text{refl} \mid \delta_\ell(\Gamma, \theta, \sigma, u, v, P, \pi, (\theta', \pi)) \\
& \mid \mu(B, \bar{\tilde{i}}, \pi) \mid \delta_\ell(\pi, x \alpha \pi') \\
& \mid \nu(\pi, \alpha \pi') \mid \delta_\ell(\bar{\tilde{i}}, \pi)
\end{align*}
\]

Here and later, we use \( \alpha, \beta, \gamma \) to denote proof variables, and \( \sigma \) to denote substitutions for proof variables. The notation \((\theta', \pi)\), in the equality elimination construct stands for a finite, possibly empty, collection of subterms. In the expression \( \theta, \pi \), all free variables of \( \pi \) must be in the range of the substitution \( \theta \). Finally, the notation \( x, \pi \) or \( \alpha, \pi \) denotes a binding construct, i.e., \( x \) (resp. \( \alpha \)) is bound in \( \pi \). As usual, terms are identified up to a renaming of bound variables, and renaming is used to avoid capture when propagating a substitution under a binder.

Typing judgments are relativized to contexts that are assignments of types to finite sets of proof variables. We denote contexts by \( \Gamma \), written perhaps with subscripts and superscripts.

**Definition 4.** A proof term \( \pi \) has type \( P \) under the context \( \Gamma \) if \( \Gamma \vdash \pi : P \) is derivable using the rules in Figure 1. We also
say that \( \Gamma' \vdash \sigma : \Gamma \) holds if \( \Gamma \) and \( \sigma \) have the same domain and \( \Gamma' \vdash \sigma(\alpha) : \Gamma(\alpha) \) holds for each \( \alpha \) in that domain.

C. Expressiveness of the logic

The logic \( \muNJ \) modulo inherits from logics of fixed-point definitions a simplicity in the treatment of (co)inductive sets and relations and from deduction modulo the ability to blend computation and deduction in the course of reasoning. We illustrate this aspect through a few simple examples here.

Natural numbers may be specified through the following lemma show that this treatment of substitution is coherent.

\[
\begin{align*}
\Gamma \vdash \alpha : P & \quad P \equiv Q, (\alpha : Q) \in \Gamma \\
\Gamma, \alpha : P_1 \vdash \pi : P_2 & \quad P \equiv P_1 \supset P_2 \\
\Gamma \vdash \alpha, \pi : P & \quad P \equiv P_1 \wedge P_2 \\
\Gamma \vdash \pi_1 : P_1 \quad \Gamma \vdash \pi_2 : P_2 & \quad \Gamma \vdash \pi_1 \wedge \pi_2 \quad P \equiv P_1 \vee P_2 \\
\Gamma \vdash \pi : P_1 & \quad \Gamma \vdash \pi : P_1 \wedge \pi_2 : P_2 & \quad \Gamma \vdash \pi : P_1 \vee \pi_2 : P_2 \\
\Gamma \vdash \pi : Q & \quad \Gamma \vdash \alpha, \pi : P & \quad P \equiv \forall x. Q \\
\Gamma \vdash \alpha, \pi : P & \quad \Gamma \vdash \pi : Q & \quad P \equiv \exists x. Q \\
\Gamma \vdash \pi : Q[t/x] & \quad \Gamma \vdash (t, \pi) : P & \quad P \equiv t = i \\
\Gamma \vdash \pi_1 : B (\mu B) \mathrel{\hat{\top}} & \quad \Gamma \vdash \mu B \mathrel{\hat{\top}} & \quad P \equiv B \mathrel{\hat{\top}} \\
\Gamma \vdash \pi : S \mathrel{\hat{\top}} & \quad \Gamma, \alpha : S \mathrel{\hat{\top}} \pi' : B \mathrel{\hat{\top}} & \quad P \equiv B \mathrel{\hat{\top}} \\
\Gamma \vdash \forall \pi, \forall \pi, \pi' : P & \quad P \equiv \forall \pi, \forall \pi, \pi' : P \\
\end{align*}
\]

Variables bound in proof terms are assumed to be new in instances of typing rules, i.e., they should not occur free in the base sequent. Specifically, \( \alpha, \beta, x \) are assumed to be new in the introduction rules for implication, universal quantification and greatest fixed-point, as well as elimination rules for disjunction, existential quantification, equality and least fixed-point.

Fig. 1. \( \muNJ \): Natural deduction modulo with equality and least and greatest fixed points.
Lemma 1. Term-level substitution preserves type assignment:  
$$\Gamma \vdash \pi : P \implies \Gamma \vdash \pi \theta : P \theta.$$  
Proof: This is easily checked by induction on the typing derivation. An interesting case is that of equality elimination. Consider the following derivation:

$$\Gamma \vdash \pi : \sigma \theta \quad \Gamma \vdash \sigma : \Gamma \theta' \quad (\Gamma \theta' \vdash \pi ; : P \theta') \quad P \equiv P \theta'$$

By the induction hypothesis, $$\Gamma \vdash \pi \theta : \sigma \theta'$$ and $$\Gamma \vdash \sigma' : \Gamma \theta'$$ have derivations. From these we build the derivation

$$\Gamma \vdash \sigma \theta \theta' : \Gamma \theta' \quad \Gamma \vdash \sigma' \theta : \Gamma \theta' \quad (\Gamma \theta' \vdash \pi ; : P \theta') \quad P \equiv P \theta'$$

Lemma 2. If $$\Gamma \vdash \pi : P$$ and $$\Gamma \vdash \sigma : \Gamma$$ then $$\Gamma \vdash \pi \sigma : P.$$  
Proof: This is shown also by induction on the typing derivation. An interesting case, again, is that of equality elimination. Consider the following derivation:

$$\Gamma \vdash \pi : u \theta = v \theta \quad \Gamma \vdash \sigma : \Gamma \theta' \quad (\Gamma \theta' \vdash \pi ; : P \theta') \quad P \equiv P \theta'$$

By the induction hypothesis, $$\Gamma \vdash \pi \theta = v \theta$$ and $$\Gamma \vdash \sigma' \theta' : \Gamma \theta'$$ have derivations. From this we build the derivation

$$\Gamma \vdash \sigma \theta \theta' : \Gamma \theta' \quad \Gamma \vdash \sigma' \theta' : \Gamma \theta' \quad (\Gamma \theta' \vdash \pi ; : P \theta') \quad P \equiv P \theta'$$

The most interesting reduction rules are those for the least and greatest fixed-point operators. In the former case, the rule must apply to a proof of the form

$$\Gamma \vdash \pi : \mu \pi B \pi \quad \Gamma \vdash \theta \pi B \pi \theta \quad \Gamma \vdash \alpha : B S \pi \quad \Gamma \vdash \delta_\alpha (\theta \pi B \pi \theta) : S \pi$$

This can be eliminated by generating a proof of $$\Gamma \vdash \pi$$ directly from the derivation of $$\Gamma \vdash \pi : \mu \pi B \pi$$; doing this effectively means that we move the reduction (cut) deeper into the iteration that introduces the least fixed point. To realize this transformation, we proceed as follows:

- Using the derivation $$\pi',$$ we can get a proof of $$\pi$$ from $$\Gamma \vdash \pi.$$
- It should be noted that this transformation requires that the proof be a proof of the form $$\Gamma \vdash \pi.$$
from it terminates in a normal proof term. The set of strongly
normalizable proof terms is denoted by \( SN \). The normaliz-
ability of proof terms can be coupled with the following
observation to show the (conditional) consistency of the logic.

**Lemma 3.** If \( \equiv \) is defined by a confluent rewrite system
that rewrites terms to terms and atomic propositions to pro-
tositions, then \( \vdash \pi : \bot \) is not derivable for any normal \( \pi \).

**Proof:** This standard observation is not affected by the
rewriting layer, since \( \bot \) cannot be equated with another logical
connective under the assumptions on \( \equiv \), and it is not affected
either by our new constructs, for which progress is ensured:
eliminations followed by introductions can always be reduced.
More details may be found in Appendix I.

![Fig. 3. Definition of functionality](image)

\[
\Gamma, \beta : \mu (B P) \vdash \gamma : B P (\mu (B P')) \quad \vdash F_{B P} (\mu (B P')) (x, \alpha, \pi) \\
\Gamma, \beta : \mu (B P) \vdash \mu (B P') \quad \vdash \mu (B P', \ldots) : \mu (B P') \biota
\]

![Fig. 4. Typing functionality for least-fixed points](image)

\[
(\lambda x. \pi) \pi' \to [\pi'[\alpha/\pi]] \quad \text{proj} (\pi_1, \pi_2) \to \pi_1 \\
(\lambda x. \pi) t \to [\pi[t/x]] \quad \delta (t, \pi, x, \alpha, \pi') \to \pi'[t/x][\pi/\alpha] \\
\delta_p (\mu (B P), \pi, \alpha, \pi') \to \pi'[\pi'][\pi] (F_{B P} (\mu (B P), \pi, \beta, \alpha, \pi')) \pi/\alpha \\
\delta (\beta, \pi, \alpha, \pi') \to F_{B P} (\mu (B P), \pi, \beta, \alpha, \pi') (\pi'[\pi][\pi/\alpha]) \\
\delta_x (\Gamma, \sigma, u, v, P, \text{refl} (\theta, \pi)) \to \pi \theta' \sigma \quad \text{where } \theta \equiv \theta' \theta'
\]

![Fig. 5. Reduction rules for \( \mu NJ \) proof terms](image)

\[
F_{B P} (\underline{\alpha}, \pi, \pi) = \lambda \alpha. \pi [\underline{\alpha}] \quad F_{B P} (\underline{\alpha}, \pi) = \lambda \beta \beta \beta \ if \ does \ not \ occur \ in \ Q \\
F_{B P} (\underline{\alpha}, \pi) = \lambda \beta \ (F_{B P} (\underline{\alpha}, \pi) (\text{proj}_1 (\beta)), F_{B P} (\underline{\alpha}, \pi) (\text{proj}_2 (\beta))) \\
F_{B P} (\underline{\alpha}, \pi) = \lambda \beta \delta (\beta, \gamma, \text{in}_1 (F_{B P} (\underline{\alpha}, \pi) \gamma), \text{in}_2 (F_{B P} (\underline{\alpha}, \pi) \gamma)) \\
F_{B P} (\underline{\alpha}, \pi) = \lambda \beta \lambda \gamma F_{B P} (\underline{\alpha}, \pi) (\beta (F_{B P} (\underline{\alpha}, \pi) \gamma)) \\
F_{B P} (\underline{\alpha}, \pi) = \lambda \beta \lambda \gamma F_{B P} (\underline{\alpha}, \pi) (\beta (F_{B P} (\underline{\alpha}, \pi) \gamma)) \\
F_{B P} (\underline{\alpha}, \pi) = \lambda \beta \lambda \gamma F_{B P} (\underline{\alpha}, \pi) (\beta (F_{B P} (\underline{\alpha}, \pi) \gamma)) \\
F_{B P} (\underline{\alpha}, \pi) = \lambda \beta \lambda \gamma F_{B P} (\underline{\alpha}, \pi) (\beta (F_{B P} (\underline{\alpha}, \pi) \gamma))
\]

IV. Strong Normalizability

In a fashion similar to [8], we now establish strong normal-
izability for proof reductions when the congruence relation
satisfies certain general conditions. The proof is based on
the framework of reducibility candidates, and borrows elements
from earlier work in linear logic [3] regarding fixed-points.

**Definition 6.** A proof term is neutral iff it is not an intro-
duction, i.e., it is a variable or an elimination construct.

**Definition 7.** A set \( R \) of proof terms is a reducibility candidate
if (1) \( R \subseteq SN \); (2) \( \pi \in R \) and \( \pi \to \pi' \) implies \( \pi' \in R \); and (3)
if \( \pi \) is neutral and all of its one-step reducts are in \( R \), then
\( \pi \in R \). We denote by \( C \) the set of all reducibility candidates.

Conditions (2,3) are positive and compatible with (1) so that
for any subset \( S \) of \( SN \) there is a least candidate containing
\( S \). We refer to the operation that yields this set as satur-
ation. Reducibility candidates, equipped with inclusion, form a com-
plete lattice: the intersection of a family of candidates gives
their infimum and the saturated union gives their supremum.
Having a complete lattice, we can define least and greatest
reducibility candidates, called the

**Definition 8.** A pre-model \( M \) is an assignment of a function
\( \ast \) from \( [\gamma_1] \times \cdots \times [\gamma_n] \) to \( C \) to each predicate constant \( a \) of
type \( \gamma_1 \to \ldots \rightarrow \gamma_n \rightarrow a \). Here, \( [\gamma] \) denotes the set of (potentially
open) terms of type \( \gamma \).

**Definition 9.** Let \( M \) be a pre-model, let \( P \) be a formula and
let \( E \) be a context assigning predicate candidates of the right
types to at least the free predicate variables in \( P \). We define
the candidate \( P^C \), called the interpretation of \( P \), by recursion
on the structure of \( P \) as shown in Figure 6.

To justify this definition, we show simultaneously by an
induction on \( P \) that \( P^C \) is a candidate and that it is monotonic.
|\{1\}^E = |\top|^E = \{u = v\}^E = SN |p_1, \ldots, p_n|^E = \mathcal{E}(p)(t_1, \ldots, t_n) |\mu_1 t_1, \ldots, t_n|^E = d(t_1, \ldots, t_n)

\[\begin{align*}
|P \cup Q|^E &= \{ \pi \in SN | P \rightarrow_\pi \alpha, \pi_1 \text{ implies } \pi_1(\pi'/\alpha) \in |Q|^E \text{ for any } \pi' \in |P|^E \} \\
|P \land Q|^E &= \{ \pi \in SN | P \rightarrow_\pi (\pi_1, \pi_2) \text{ implies } \pi_1 \in |P|^E \text{ and } \pi_2 \in |Q|^E \} \\
|P \lor Q|^E &= \{ \pi \in SN | \pi \rightarrow_\pi \pi \text{ implies } \pi' \in |P|^E \} \\
|\forall x. P|^E &= \{ \pi \in SN | P \rightarrow_\pi \lambda x. \pi' \text{ implies } \pi' \in |P[x]/\pi|^E \text{ for any } \pi \} \\
|\exists x. P|^E &= \{ \pi \in SN | P \rightarrow_\pi \langle \pi, \pi \rangle \text{ implies } \pi' \in |P[x]/\pi|^E \} \\
|\mu B|^E &= \lambda f(\phi(t)) \text{ where } \phi(X) = \pi \rightarrow [\pi \in \mu B(t, \pi) \text{ implies } \pi' \in |B[p]^E|_{\pi}(X)] \\
|\nu B|^E &= \lambda f(\phi(t)) \text{ where } \phi(X) = \pi \rightarrow [\pi \in |B[p]^E|_{\pi}(X)] \\
\end{align*}\]

Fig. 6. Interpretation of formulas as candidates

(resp. anti-monotonic) in \(\mathcal{E}(p)\) for any variable \(p\) that only occurs positively (resp. negatively) in \(P\); the latter two facts ensure that the fixed points assumed in the definition actually exist, anti-monotonicity being needed because of the covariance in implication formulas. Preservation of (anti)monotonicity and satisfaction of the conditions for reducibility candidates are readily verified in all but the fixed point cases. For the least fixed point case, \(\mu B t|^E\) is easily seen to be a candidate provided it is well-defined, i.e., if \(\lambda f(\phi)\) exists for \(\phi\) as in the definition. But this must be so: the induction hypothesis applied to \(B p t\) ensures that \(\phi\) is a monotonic mapping, hence it has a least fixed point in the lattice of predicate candidates. For monotonicity, consider \(S\) and \(\mathcal{E}\) differing only on a variable \(p\) that occurs only positively in \(\mu B t\), with \(\mathcal{E}(p) \subseteq \mathcal{E}'(p)\). Let \(\mu B t|^E = \lambda f(\phi(t))\). Unfolding and using the induction hypothesis, we have \(\phi(X) \subseteq \phi'(X)\) for any candidate \(X\), and in particular \(\phi((\mu B t|^E)) \subseteq \phi((\mu B t|^E)) = \mu B t|^E\). The least fixed point being contained in all prefixed points, we obtain the expected result: \(\mu B t|^E = \lambda f(\phi(t)) \subseteq |\mu B t|^E\). Antimonomonicity is established in a symmetric fashion. The treatment of the greatest fixed point case is similar.

Notation 1. If \(P\) is a predicate of type \(\gamma \rightarrow \alpha\), \(|P|^E\) denotes the mapping \(\gamma \mapsto |P|^E\). If \(B\) is of type \(\gamma \rightarrow \alpha\) \(\rightarrow \alpha\), \(|B|^E\) denotes the mapping \(\gamma \rightarrow |B|^E|_{\pi}(X)\). If \(B\) is a predicate operator of type \(\gamma \rightarrow \alpha\) \(\rightarrow \alpha\), \(|B|^E\) denotes the mapping \(\gamma \rightarrow B |p|^E|_{\pi}(X)\). For conciseness we write directly \(|B X|^E|_{\pi}(X)\) for |\lambda p. B p|^E X or, equivalently, \(|B|^E|_{\pi}(X)\).

Lemma 4. Interpretation commutes with second-order substitution: \(|B[p]\|^E = |B|^E|_{\pi}(\cdot)^E|_{\pi}(\cdot)|p|^E|_{\pi}\).

Proof: Straightforward, by induction on \(B\).

We naturally extend the interpretation to typing contexts:
if \(\Gamma = (a_1 : P_1, \ldots, a_n : P_n)\), \(|\Gamma|^E = (a_1 : |P_1|^E, \ldots, a_n : |P_n|^E)\). We also write \(\sigma \in |\Gamma|^E\) when \(\sigma\) is of the form \(\pi_1/\alpha_1, \ldots, \pi_n/\alpha_n\) with \(\pi_i \in |\Gamma|^E\) for all \(i\).

Definition 10. If \(\Gamma\) is a proof term with free variables \(\alpha_1, \ldots, \alpha_n, \gamma, X_1, \ldots, X_m\) are reducibility candidates, we say that \(\Gamma\) is \((\alpha_1 : X_1, \ldots, \alpha_n : X_n \cup \gamma)\)-reducible if \(\pi_i/\alpha_i \in \gamma\) for any \(\pi_i \in X_i\). When it is not ambiguous, we omit the variables and simply say that \(\pi\) is \((\alpha_1 : X_1, \ldots, X_n \cup \gamma)\)-reducible.

Definition 11. A pre-model \(\mathcal{M}\) is a pre-model of \(\equiv\) iff it accords the same interpretation to formulas that are congruent.

In the rest of this section, we assume that \(\mathcal{M}\) is a pre-model of the congruence, and we show that if \(\Gamma \vdash \pi : P\) has a proof then \(\pi\) is \((|\pi|^E \cup |\pi|^E)\)-reducible. In order to do so, we prove adequacy lemmas which show that each typing rule can be simulated in the calculus.

Lemma 5. The following properties hold for any context \(E\).

\((\Rightarrow)\) – if \(\pi = (\alpha : |P|^E \cup |Q|^E)\)-reducible, then \(\lambda x. \pi \in P \supset Q\).

\((\Leftarrow)\) – if \(\pi \in P \supset Q\) and \(\pi' \in |P|^E\), then \(\pi' \in |Q|^E\).

\((\wedge)\) – if \(\pi_1 \in |P|^E\) and \(\pi_2 \in |P|^E\), then \(\pi_1 \wedge \pi_2 \in |P|^E\).

\((\Lambda)\) – if \(\pi \in P \supset Q\), then \(\pi \in P \supset Q\).

\((\forall)\) – if \(\pi \in |P|^E\) for \(i \in \{1, 2\}\), then \(\pi_1 \in |P \supset Q|^E\).

\((\exists)\) – if \(\pi \in |P|^E\), then \(\pi \in |P \supset Q|^E\).

\(\Rightarrow\) – refl \(\in \equiv\).

\((\Rightarrow)\) – if \(\pi \in \equiv\), then \(\pi \in \equiv\).

\((\Leftarrow)\) – if \(\pi \in \equiv\), then \(\pi \in \equiv\).

\((\wedge)\) – if \(\pi \in \equiv\) and \(\pi \land \pi' \equiv \equiv\), then \(\pi \equiv \equiv\).

\((\Lambda)\) – if \(\pi \in \equiv\), then \(\pi \equiv \equiv\).

\((\forall)\) – if \(\pi \in \equiv\), then \(\pi \equiv \equiv\).

\(\Rightarrow\) – refl \(\in \equiv\).

\((\Rightarrow)\) – if \(\pi \in \equiv\), then \(\pi \equiv \equiv\).

\((\Leftarrow)\) – if \(\pi \in \equiv\), then \(\pi \equiv \equiv\).

\((\wedge)\) – if \(\pi \equiv \equiv\) and \(\pi \equiv \equiv\), then \(\pi \equiv \equiv\).

\((\Lambda)\) – if \(\pi \in \equiv\), then \(\pi \equiv \equiv\).

\((\forall)\) – if \(\pi \in \equiv\), then \(\pi \equiv \equiv\).

Proof: These observations are proved easily using standard proof techniques on candidates. We illustrate only a few cases here; more details may be found in Appendix I. For the case of least fixed point introductions, we have \(\mu B|^E = \lambda f(\phi(t)) = \phi(|B|^E(t))\) by Definition 9, and thus \(\mu B|^E = \{ \rho \in SN | \rho \rightarrow_\mu \mu B t|^E \text{ for any } \pi' \in |B|^E\}\) by Lemma 4, from which it is easy to conclude. Similarly, we observe that \(|\nu B|^E = \{ \pi | \delta(B, \pi, \pi') \in |B|^E\}\) from which the greatest fixed point elimination case follows immediately. Finally, the equality elimination case is proved by induction on the strong normalisability of the subderivations \(\pi, \sigma\) and \(\pi_1\). In order to show that a neutral term belongs to a candidate, it suffices to consider all its one-step reduces. Reductions occurring inside subterms are handled by induction hypothesis. We may also have a toplevel redex when \(\theta \equiv \theta\) and \(\pi = refl\).
reducing to $\pi, \theta \sigma$ where $\theta'$ is such that $\theta' \equiv \theta$. By hypothesis, $\pi, \theta'$ is ($\Gamma, \theta' \theta' \theta'$ [P]) reducible and $\sigma \in [\Gamma, \theta'] \theta'$, and thus we have $\pi, \theta' \sigma \in [P, \theta'] \theta'$ as expected.

Although adequacy is easily proved for our new equality formulation, a few important observations should be made here. First, the proof crucially relies on the fact that we are considering only syntactic pre-models, and not the general notion of pre-model of Dowek and Werner where terms may be interpreted in arbitrary structures. This requirement makes sense conceptually, since closed-world equality internalizes the fact that equality can only hold when the congruence allows it, and is thus incompatible with further equalities that could hold in non-trivial semantic interpretations. Second, the suspension of proof-level substitutions in equality elimination goes hand in hand with the independence of interpretations for different predicate instances, which in turn is necessary to interpret recursive definitions. Indeed, when applying a proof-level substitution $\sigma \in [\Gamma, \theta']$ on an eager equality elimination, we are forced to apply the csu substitutions on $\sigma$, and we need $\sigma \in [\Gamma, \theta'] \theta'$ which essentially forces us to have a term-independent interpretation [3].

We now address the adequacy of functoriality, induction and coinduction.

**Lemma 6.** Let $\pi$ be a proof, and let $X$ and $X'$ be predicate candidates such that $\pi(\bar{t}, \bar{x})$ is $(\alpha : X \leftarrow \bar{X})$-reducible for any $\bar{t}$. If $B$ is a monotonic (resp. antimonic) operator, then $F_B(\bar{x}, \alpha.\pi) \in [B \leftarrow X]$ (resp. $F_B(\bar{x}, \alpha.\pi) \in [B \leftarrow X]$).

**Lemma 7.** Let $\pi$ be a proof and $X$ a predicate candidate. If $\pi(\bar{t}, \bar{x})$ is $(\alpha : B \leftarrow \bar{X})$-reducible for any $\bar{t}$, then $\delta(\beta, \bar{x}, \alpha.\pi)$ is $(\beta : \mu \bar{B}) \leftarrow \bar{X}$-reducible for any $\bar{t}$.

**Lemma 8.** Let $\pi$ be a proof and $X$ a predicate candidate. If $\pi(\bar{t}, \bar{x})$ is $(\alpha : B \leftarrow \bar{X})$-reducible for any $\bar{t}$, then $\nu(\beta, \bar{x}, \alpha.\pi)$ is $(\beta : \nu \bar{B}) \leftarrow \bar{X}$-reducible for any $\bar{t}$.

**Proof:** These lemmas must be proved simultaneously, in a generalized form that is detailed in Appendix I. There is no essential difficulty in proving the functoriality lemma, using previously proved adequacy properties as well as the other two lemmas for the fixed point cases. The next two lemmas are the interesting ones, since they involve using the properties of the fixed point interpretations to justify the (co)induction rules. In the case of induction, we need to establish that $\delta(\beta, \bar{x}, \alpha.\pi) \in \bar{X}$ when $\rho \in \mu \bar{B}$. In order to do this, it suffices to show that $\mathcal{Y} \models \bar{t} \rightarrow \{ \rho \mid \delta(\beta, \bar{x}, \alpha.\pi) \in \bar{X} \} \subset \mu \bar{B}$. This follows from the fact that $\mathcal{Y}$ is a pre-fixed point of the operator $\phi$ such that $[\mu \bar{B}] = \lfp(\phi)$, which can be proved easily using the adequacy property for functoriality. We proceed similarly for the coinduction rule, showing that $\mathcal{Y} \models \bar{t} \rightarrow \{ \pi \in \Sigma \leftarrow \bar{X} \leftarrow \{ \rho \mid \nu(\beta, \bar{x}, \alpha.\pi) \in \bar{X} \} \} \subset \mu \bar{B}$. This is a post-fixed point of the operator $\phi$ such that $[\nu \bar{B}] = \gfp(\phi)$. In both cases, note that the candidate $\mathcal{Y}$ is _a priori_ not the interpretation of any predicate; this is where we use the power of reducibility candidates.

**Theorem 2** (Adequacy). Let $\equiv$ be a congruence, $M$ be a pre-model of $\equiv$ and $\Gamma : \pi \vdash P$ be a derivable judgment. Then $\pi \sigma \in [P]$ for any substitution $\sigma \in [\Gamma]$.

**Proof:** By induction on the height of $\pi$, using the previous adequacy properties.

The usual corollaries hold. Since variables belong to any candidate by condition (3), we can take $\sigma$ to be the identity substitution, and obtain that any well-typed proof is strongly normalizable. Together with Lemma 3, this means that our logic is consistent. Note that the suspended computations in the (co)induction and equality elimination rules do not affect these corollaries, because they can only occur in normal forms of specific types. For instance, equality elimination cannot hide a non-terminating computation if there is no equality assumption in the context.

V. **Recursive Definitions**

We now identify a class of rewrite rules relative to which we can always build a pre-model. This class supports recursive definitions whose use we illustrate through a sound formalization of a Tait-style argument.

A. **Recursive rewriting that admits a pre-model**

The essential idea behind recursive definitions is that they are formed gradually, following the inductive structure of one of their arguments, or more generally a well-founded order on arguments. In order to reflect this idea into a pre-model construction, we need to identify all the atom interpretations that could be involved in the interpretation of a given formula. This is the purpose of the next definition.

**Definition 12.** We say that $P$ may occur in $Q$ when $P \equiv P'$, $P'$ occurs in $Q$, and $\theta$ is a substitution for variables quantified over in $Q$.

For example, $(a t)$ may occur in $(a' x \land \exists y. a y)$ for any $t$.

**Theorem 3.** Let $\equiv$ be a congruence defined by a rewrite system rewriting terms to terms and atomic propositions to propositions, and let $M$ be a pre-model of $\equiv$. Consider the addition of new predicate symbols $a_1, \ldots, a_n$ in the language, together with the extension of the congruence resulting from the addition of rewrite rules of the form $a\bar{t} \implies B$. There is a pre-model of the extended congruence in the extended language, provided that the following conditions hold.

1. If $(a_1 \equiv a_2)$, then $a_1 \leftarrow B$ and $a_2 \leftarrow B'$, then $B \equiv B'$.
2. There exists a well-founded order $<$ such that $a_1 \leftarrow B$ whenever $a_1 \leftarrow B$ and $a_1 \leftarrow B'$ may occur in $B'$.

Note that condition (1) is not obviously satisfied, even when there is a single rule per atom. Consider, for example, $a(0 \times x) \leftarrow a' x$ in a setting where $0 \times x \equiv 0$: our condition requires that $a' x \equiv a' y$ for any $x$ and $y$, which is an _a priori_ not guaranteed. Condition (2) restricts the use of quantifiers but still allows useful constructions. Consider for example the Ackermann relation, built using a double induction on its first two parameters: $\text{ack } 0 x (s x) \leftarrow \top$, $\text{ack } (s x) 0 y \leftarrow$
ack x (s 0) y and ack (s x) (s y) z \leadsto \exists r. \ack x y \land \ack x z.

The third rule requires that \( \ack x z \leadsto \ack x y \) for any \( x, y, z \) and \( r \), which is indeed satisfied with a lexicographic ordering.

**Proof:** We only present the main idea here; a detailed proof may be found in Appendix I. We first build pre-models \( \mathcal{M}^{\text{pre}} \) that are compatible with instances \( a_i t_j \leadsto B \) of the new rewrite rules for \( a_i t_j \leq a_i t_j' \). This is done gradually following the order \( < \), using a well-founded induction on \( a_i t_j \). We build \( \mathcal{M}^{\text{pre}} \) by aggregating smaller pre-models \( \mathcal{M}_{\text{pre}} \) for \( a_i t_j < a_i t_j' \) and adding the interpretation \( \hat{a}_i \). To define it, we consider rule instances of the form \( a_i t_j \leadsto B \). If there is none we use a dummy interpretation: \( \hat{a}_i = SN \). Otherwise, condition (1) imposes that there is essentially a single possible such rewrite modulo the congruence, so it suffices to choose \( [B] \) as the interpretation \( \hat{a}_i \) to satisfy the new rewrite rules. Finally, we aggregate interpretations from all the pre-models \( \mathcal{M}^{\text{pre}} \) to obtain a pre-model of the full extended congruence.

This result can be used to obtain pre-models for complex definition schemes, such as ones that iterate and interleave groups of fixed-point and recursive definitions. Consider, for example, \( a (s n) \leadsto a \sqsupseteq a (s n) \). While this rewrite rule does not directly satisfy the conditions of Theorem 3, it can be rewritten into the form \( a (s n) \leadsto \mu Q. a \sqsupseteq Q \), which does satisfy these conditions.

**B. An application of recursive definitions**

Our example application is the formalization of the Tait-style argument of strong normalizability for the simply typed \( \lambda \)-calculus. We assume term-level sorts \( \text{tm} \) and \( \text{ty} \) corresponding to representations of \( \lambda \)-terms and simple types, and symbols \( t : \text{ty}, \text{arrow} : \text{ty} \to \text{ty} \to \text{ty}, \text{app} : \text{tm} \to \text{tm} \to \text{tm} \) and \( \text{abs} : (\text{tm} \to \text{tm}) \to \text{tm} \). We identify well-formed types through an inductive predicate:

\[
\text{isty} (\text{def}) = \mu \alpha (\lambda T. \text{ty}. \text{ty} \lor \exists t' B. t = \text{arrow} t' t' \land T t' \land T t')
\]

We assume a definition of term reduction and strong normalization, denoting the latter predicate by \( \text{sn} \). Finally, we define \( \text{red} m t \) expressing that \( m \) is a reducible \( \lambda \)-term of type \( t \), by the following rewrite rules:

\[
\text{red} m t \leadsto \text{sn} m \\
\text{red} (\text{arrow} t t') \leadsto \forall n. \text{red} n t \supset \text{red} (\text{app} m n) t'
\]

This definition satisfies the conditions of Theorem 3, taking as \( < \) the order induced by the subterm ordering on the second argument of \( \text{red} \). We can thus safely use it.

With these definitions, our logic allows us to mirror very closely the strong normalization proof presented in [12]. For instance, consider proving that reducible terms are strongly normalizing:

\[
\forall m \forall t. \text{isty} t \supset \text{red} m t \supset \text{sn} m
\]

The paper proof is by induction on types, which corresponds in the formalization to an elimination on \( \text{isty} t \). In the base case, we have to derive \( \text{red} m t \supset \text{sn} m \) which is simply an instance of \( P \supset P \) modulo our congruence. In the arrow case, we must prove \( \text{red} m (\text{arrow} t t') \supset \text{sn} m \). The hypothesis \( \text{red} m (\text{arrow} t t') \) is congruent to \( \forall n. \text{red} n t \supset \text{red} (\text{app} m n) t' \) and we can show that variables are always reducible, which gives us \( \text{red} (\text{app} m x) t' \). From there, we obtain \( \text{sn} (\text{app} m x) \) by induction hypothesis, from which we can deduce \( \text{sn} m \) with a little more work.

The full formalization, which is too detailed to present here, is shown in Appendix II. It has been tested using the proof assistant Abella [9]. The logic that underlies Abella features fixed-point definitions, closed-world equality and generic quantification. The last notion is useful when dealing with binding structures, and we have employed it in our formalization although it is not available yet in our logic. Abella does not actually support recursive definitions. To get around this fact, we have entered the one we need as an inductive definition, and ignored the warning provided about the non-monotonic clause while making sure to use an unfolding of this inductive definition in the proof only when this is allowed for recursive definitions. In the future, we plan to extend Abella to support recursive definitions based on the theory developed in this paper. This would mean allowing such definitions as a separate class, building in a test that they satisfy the criterion described in Theorem 3 and properly restricting the use of these definitions in proofs. Such an extension is obviously compatible with all the current capabilities of Abella and would support additional reasoning that is justifiably sound.

**VI. Related and Future Work**

The logical system that we have developed is obviously related to deduction modulo. In essence, it extends that system with a simple yet powerful treatment of fixed-point definitions. The additional power is obtained from two new features: fixed-point combinators and closed-world equality. If our focus is only on provability, the capabilities arising from these features may perhaps be encoded in deduction modulo. Dowek and Werner provide an encoding of arithmetic in deduction modulo, and also show how to build pre-models for some more general fixed-point constructs [8]. Regarding equality, Allali [1] has shown that a more algorithmic version of equality may be defined through the congruence, which allows to simplify some equations by computing. Thus, it simulates some aspects of closed-world equality. However, the principle of substitutivity has to be recovered through a complex encoding involving inductions on the term structures. In any case, our concern here is not simply with provability; in general, we do not follow the project of deduction modulo to have a logic as basic as possible in which stronger systems are then encoded. Rather, we seek to obtain meaningful proof structures, whose study can reveal useful information. For instance, in the context of proof-search, it has been shown that a direct treatment of fixed-point definitions allows for stronger focused proof systems [3] which have served as a basis for

---

1This actually has to be proved simultaneously with \( \text{red} m t \supset \text{sn} m \), but we ignore it for the simplicity of the presentation.
several proof-search implementations [4], [5]. This goal also justifies why we do not simply use powerful systems such as the Calculus of Inductive Constructions [17] which obviously supports inductive as well as recursive definitions; here again we highlight the simplicity of our (co)induction rules and of our rich equality elimination principle.

Our logic is also related to logics of fixed-point definitions [14], [18], [22]. The system we have described represents an advance over these logics in that it adds to them a rewriting capability. As we have seen, this capability can be used to blend computation and deduction in natural ways and add support for recursive definitions — a similar support may also be obtained in other ways [21]. Our work also makes important contributions to the understanding of closed-world equality. We have shown that it is compatible with an equational theory on terms. We have, in addition, resolved some problematic issues related to this notion that affect the stability of finite proofs under reduction. This has allowed us to prove for the first time a strong normalizability result for logics of fixed-point definitions. Our calculus is, at this stage, missing a treatment of generic quantification present in some of the alternative logics [10], [11], [16]. We plan to include this feature in the future, and do not foresee any difficulty in doing so since it has typically been added in a modular fashion to such logics. This addition would make our logic an excellent choice for formalizing the meta-theory of computational and logical systems.

An important topic for further investigation of our system is proof search. The distinction between computation and deduction is critical for theorem proving with fixed point definitions. For instance, in the Tac system [5], which is based on logics of definitions, automated (co)inductive theorem proving relies heavily on ad-hoc annotations that identify computations. In that context, our treatment of recursive definitions seems like a good candidate more a more principled separation of computation and deduction. Finally, now that we have refactored equality rules to simplify the proof normalization process, we should study their proof search behavior. The new equality elimination rule seems difficult to analyze at first. However, we hope to gain some insights from studying its use in settings where the old rule (which it subsumes) is practically satisfactory, progressively moving to newer contexts where it offers advantages. We note in this regard that the new complexity is in fact welcome: the earlier infinitely branching treatments of closed-world equality had a simple proof-search treatment in theory, but did not provide a useful handle to study the practical difficulties of automated theorem proving with complex equalities.

Acknowledgements. The authors thank Gilles Dowek, Benjamin Werner and the reviewers for helpful discussions and comments. This work has been funded by the NSF grant CCF-0917140; opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation. Nadathur and Baelde are currently receiving support respectively from the Velux Fonden and grant 10-092309 from the Danish Council for Strategic Research, Programme Commission on Strategic Growth Technologies.

References

[1] L. Allali, “Algorithmic equality in heyting arithmetic modulo,” in TYPES, ser. LNCS, M. Miculan, I. Scagnetto, and F. Honsell, Eds., vol. 4941. Springer, 2007, pp. 1–17.
[2] D. Baelde, “A linear approach to the proof-theory of least and greatest fixed points,” Ph.D. dissertation, Ecole Polytechnique, Dec. 2008.
[3] ———, “Least and greatest fixed points in linear logic,” vol. 13, no. 1, Jan. 2012. ACM Transactions on Computational Logic.
[4] D. Baelde, A. Gacek, D. Miller, G. Nadathur, and A. Tiu, “The Bedwyr system for model checking over syntactic expressions,” in 21th Conf on Automated Deduction (CADE), ser. LNLI, F. Pfenning, Ed., no. 4603. New York: Springer, 2007, pp. 391–397.
[5] D. Baelde, D. Miller, and Z. Snow, “Focused inductive theorem proving,” in Fifth International Joint Conference on Automated Reasoning, ser. LNCS, J. Giesl and R. Hähnle, Eds., no. 6173, 2010, pp. 278–292.
[6] G. Burel, “Unbounded proof-length speed-up in deduction modulo,” in CSI 2007: Computer Science Logic, ser. LNCS, J. Duparc and T. A. Henzinger, Eds., vol. 4646. Springer, 2007, pp. 496–511.
[7] G. Dowek, T. Hardin, and C. Kirchner, “Theorem proving modulo,” J. of Automated Reasoning, vol. 31, no. 1, pp. 31–72, 2003.
[8] G. Dowek and B. Werner, “Proof normalization modulo,” Journal of Symbolic Logic, vol. 68, no. 4, pp. 1289–1316, 2003.
[9] A. Gacek, “The Abella interactive theorem prover (system description),” in Fourth International Joint Conference on Automated Reasoning, ser. LNCS, A. Armando, P. Baumeister, and G. Dowek, Eds., vol. 3195. Springer, 2008, pp. 154–161.
[10] ———, “A framework for specifying, prototyping, and reasoning about computational systems,” Ph.D. dissertation, University of Minnesota, 1999.
[11] A. Gacek, D. Miller, and G. Nadathur, “Nominal abstraction,” Information and Computation, vol. 209, no. 1, pp. 48–73, 2011.
[12] J.-Y. Girard, P. Taylor, and Y. Lafont, Proofs and Types. Cambridge University Press, 1989.
[13] R. Matthes, “Monotone fixed-point types and strong normalization,” in CSL 1998: Computer Science Logic, ser. LNCS, G. Goltlob, E. Grandjean, and K. Seryl, Eds., Berlin, 1999, vol. 1584, pp. 298–312.
[14] R. McDowell and D. Miller, “Cut-elimination for a logic with definitions and induction,” Theoretical Computer Science, vol. 232, pp. 91–119, 2000.
[15] N. P. Mendler, “Inductive types and type constraints in the second order lambda calculus,” Annals of Pure and Applied Logic, vol. 51, no. 1, pp. 159–172, 1991.
[16] D. Miller and A. Tiu, “A proof theory for generic judgments,” ACM Trans. on Computational Logic, vol. 6, no. 4, pp. 749–783, Oct. 2005.
[17] C. Paulin-Mohring, “Inductive definitions in the system Coq: Rules and properties,” in Proceedings of the International Conference on Typed Lambda Calculi and Applications, M. Bezem and J. F. Groote, Eds. Utrecht, The Netherlands: Springer LNCS 664, Mar. 1993, pp. 328–345.
[18] P. Schroeder-Heister, “Rules of definitional reflection,” in 8th Symp. on Logic in Computer Science, M. Vardi, Ed., IEEE Computer Society Press. IEEE, Jun. 1993, pp. 222–232.
[19] W. W. Tait, “Intensional interpretations of functionals of finite type I,” J. of Symbolic Logic, vol. 32, no. 2, pp. 198–212, 1967.
[20] A. Tiu, “A logical framework for reasoning about logical specifications,” Ph.D. dissertation, Pennsylvania State University, May 2004.
[21] ———, “Stratification in logics of definitions,” in IJCAR, 2012, (to appear).
[22] A. Tiu and A. Momigliano, “Induction and co-induction in sequent calculi,” CoRR, vol. abs/0812.4727, 2008.
Appendix I: Proofs of Lemmas and Theorems

A. Proof of Lemma 3

We first observe that typed normal forms are characterized as usual: no introduction term is ever found as the main parameter of an elimination. This standard property is not affected by our new constructs. For example, consider the case of equality: $\delta_i(\ldots, refl(\theta, \pi_i), \ldots)$ can always be reduced by definition of complete sets of unifiers. The rest of the argument follows the usual lines: the proof cannot end with an elimination, otherwise it would have to be a chain of eliminated terminations with a proof variable, but the context is empty; it also cannot end with an introduction since there is no introduction for $\bot$ and the congruence cannot equate it with another connective.

B. Proof of Lemma 5

All introduction rules are treated in a similar fashion:

- If $\pi$ is $(\alpha : [P] + [Q])$-reducible, then $\lambda \alpha. \pi \in [P \supset Q]$. First, $\lambda \alpha. \pi$ is SN, like all reducible proof-terms, because variables belong to all candidates, and candidates are sets of SN proofs. Now, assuming $\lambda \alpha. \pi \to^* \lambda \alpha. \pi'$, we seek to establish that $\pi'[\pi'/\alpha] \in [Q]$ for any $\pi' \in [P]$. By definition of reducibility, $\pi'[\pi'/\alpha]$ belongs to $[Q]$, and we conclude by stability of candidates under reduction since $\pi'[\pi'/\alpha] \to^* \pi'[\pi'/\alpha]$.

- The cases for $\land$, $\lor$ and $\exists$ are proved similarly.

- The cases for $\forall$ and $\equiv$ are trivial.

- If $\pi[t/x]\in[P[t/x]]$ for any $t$, then $\lambda \pi \in [\forall x. P]$. Assume $\lambda \pi \to^* \lambda \pi'$. It must be the case that $\pi \to^* \pi'$, and for any $t$ we have $\pi[t/x] \to^* \pi'[t/x]$ by Proposition 1 and thus $\pi'[t/x] \in [P[t/x]]$ as needed.

- If $\pi \in [B(\mu B)]$, then $\mu(B, \vec{t}, \pi) \in [\mu B]$.

Using Definition 9, we have $[\mu B] = \{ \rho \in SN \mid \rho \to^* \mu(B, \vec{t}, \pi) \}$, and it is now easy to see that $\pi \in [B(\mu B)]$ implies $\mu(B, \vec{t}, \pi) \in [\mu B]$. For any reduction $\mu(B, \vec{t}, \pi) \to^* \mu(B, \vec{t}, \pi')$ it must be the case that $\pi \to^* \pi'$ and thus $\pi' \in [B(\mu B)]$.

Elimination rules also follow a common scheme:

- If $\pi \in [P \supset Q]$ and $\pi' \in [P]$, then $\pi' \to^* [Q]$.

We proceed by induction on the strong normalizability of $\pi$ and $\pi'$. By the candidate of reducibility condition on neutral terms, it suffices to show that all immediate reducts $\pi \to^* \pi'$ belong to $[Q]$. If $\pi''$ is obtained by a reduction inside $\pi$ or $\pi'$, then we conclude by induction hypothesis since the resulting subterm still belongs to the expected interpretation. Otherwise, it must be that $\pi = \lambda \alpha. \rho$ and the reduct is $\rho[\pi'/\alpha]$. In that case we conclude by definition of $\pi \in [P \supset Q]$.

- The cases of $\land$, $\lor$ and $\bot$ are treated similarly.

- If $\pi \in [\forall x. P]$, then $\pi \to^* [P[t/x]]$.

We proceed by induction on the strong normalizability of $\pi$, considering all one-step reducts of the neutral term $\pi t$. Internal reductions are handled by induction hypothesis.

If $\pi = \lambda \lambda. \pi'$, our term may reduce at toplevel into $\pi'[t/x]$. In that case we conclude by definition of $[\forall x. P]$.

- If $\pi \in [\exists x. P$ and $\pi'[t/x]$ is $(\alpha : [P[t/x]] + [Q])$-reducible for any $t$, then $\delta_\pi(x, \alpha. \pi') \in [Q]$.

We proceed by induction on the strong normalizability of $\pi$ and $\pi'$, considering all one-step reducts. The internal reductions are handled by induction hypothesis. A toplevel reduction into $\pi'[t/x][\pi''/\alpha]$ may occur when $\pi = (t, \pi'')$ in which case we have $\pi'' \in [P[t/x]]$ by hypothesis on $\pi$ and definition of $[\exists x. P]$. We conclude by hypothesis on $\pi'[t/x]$.

- If $\pi \in [\pi \equiv t, \sigma \in [\pi \equiv t' \theta'] and \pi[\theta]$ is $(\pi[\theta], \theta') \to^* [P[\theta]'\to]^*$-reducible for any $i$ and $\theta'$, then $\delta_\sigma(\pi, \pi, \sigma, t, t', i, \pi, \sigma, \pi_i) \in [P\theta]$.

We proceed by induction on the strong normalizability of the subderivations $\sigma$, $\sigma'$ and $\pi_i$. In order to show that a neutral term belongs to a candidate, it suffices to consider all its one-step reducts. Reductions occurring inside subterms are handled by induction hypothesis. We may also have a toplevel redux when $\theta \equiv t \theta'$ and $\pi = refl$, reducing to $\pi[\theta]'\sigma'$ where $\sigma'$ is such that $\theta \equiv \theta' \equiv \theta'$. By hypothesis, $\pi[\theta]'$ is $(\pi[\theta]', \theta') \to^* [P[\theta]'\to]^*$-reducible, $\sigma' \in [\pi[\theta]'\to]$ and thus we have $\pi[\theta]'\sigma' \in [\pi[\theta]'\to]$ as expected.

- The case of $\delta_\sigma$ is singular, as it follows directly from the definition of the interpretation of greatest fixed points. Indeed, we obtain exactly $[\forall B] = \{ \pi \mid \delta_\sigma(B, \vec{t}, \pi) \in [B(\forall B)] \}$ by unfolding the interpretation of greatest fixed points like we did for the least fixed point case above, using Definition 9 and Lemma 4.

C. Proof of Lemmas 6, 7 and 8

Let us first introduce the following notation for conciseness: we say that $\pi$ is $(\bar{x}, X, X\bar{x} + Y\bar{x})$-reducible when $\pi[\vec{t}/x]$ is $(\bar{x} \equiv Y\bar{x})$-reducible for any $\vec{t}$.

We prove the three lemmas simultaneously, generalized as follows for a predicate operator $B$ of second-order arity\footnote{In (1) and (2), $B$ has type $\sigma^1 \rightarrow o$. In (3) and (4) we are considering $B$ of type $\sigma^0 \rightarrow (\theta \\rightarrow o) \rightarrow (\theta \\rightarrow o)$.} $n + 1$, predicates $\bar{A}$ and predicate candidates $\bar{Z}$:

1. (For any $\bar{x}, X, X\bar{x} + Y\bar{x})$-reducible $\pi$, $\bar{x} = \bar{A}$, $\bar{x}. \alpha. \pi \in [B\bar{Z} \supset B\bar{Z}]$.

2. (For any $\bar{x}, X, X\bar{x} + Y\bar{x})$-reducible $\pi$, $\bar{x} = \bar{B}$, $\bar{x}. \alpha. \pi \in [B\bar{Z} \supset B\bar{Z}]$.

3. (For any $\bar{x}, X, X\bar{x} + Y\bar{x})$-reducible $\pi$, $\bar{x} = \bar{B}$, $\bar{x}. \alpha. \pi \in [B\bar{Z} \supset B\bar{Z}]$.

4. (For any $\bar{x}, X, X\bar{x} + Y\bar{x})$-reducible $\pi$, $\bar{x} = \bar{B}$, $\bar{x}. \alpha. \pi \in [B\bar{Z} \supset B\bar{Z}]$.)

We proceed by induction on the number of logical connectives in $B$. The purpose of the generalization is to keep formulas $\bar{A}$ out of the picture: those are potentially large but are treated atomically in the definition of functionality, moreover they will be interpreted by candidates $\bar{Z}$ which may not be interpretations of formulas. We first prove (3) and (4) by relying on smaller instances of (1), then we show (1) and
(2) by relying on smaller instances of all four properties but also instances of (3) and (4) for an operator of the same size.

(1) We proceed by case analysis on $B$. When $B = \lambda \vec{p} q \vec{q}.\vec{q}'$, we have to establish that $F^+_{\lambda \vec{p} q \vec{q}.\vec{q}'}(\vec{x}.\alpha.\pi) : B P (\nu (B P)) \vec{x} \Rightarrow B P' (\nu (B P)) \vec{x}$.

We proceed by induction on the strong normalizability of $\vec{x}$. It simply follows from Lemma 5 and the hypothesis on $\pi$. When $B = \lambda \vec{p} q \vec{q}.\vec{q}'$, we have to show $F^+_{\lambda \vec{p} q \vec{q}.\vec{q}'}(\vec{x}.\alpha.\pi) \Rightarrow \lambda \vec{p} \vec{q}.\vec{q}' \vec{x}$. Assuming that every one-step reduct of a neutral $B P (\nu (B P)) \vec{x}$ is trivial, we have to show $B P (\nu (B P)) \vec{x} \Rightarrow B P' (\nu (B P)) \vec{x}$.

This is done by induction on the strong normalizability of $\vec{x}$ and $\pi$. It simply follows from Lemma 5 and the hypothesis.

In all other cases, we use the adequacy properties and conclude by induction hypothesis. Most cases are straightforward, relying on the adequacy properties. In the implication case, i.e., $B = \vec{p} \vec{q} \vec{q}'$, we use induction hypothesis (2) on $B_1$ and (1) on $B_2$. Let us only detail the least fixed point case:

$$F^+_{\lambda \vec{p} q \vec{q}.\vec{q}'}(\vec{x}.\alpha.\pi) = \lambda \vec{p} \vec{q}.\vec{q}' \vec{x} \Rightarrow \lambda \vec{p} \vec{q}.\vec{q}' \vec{x}$$

By induction hypothesis (1), we obtain $F^+_{\lambda \vec{p} q \vec{q}.\vec{q}'}(\vec{x}.\alpha.\pi) \Rightarrow \lambda \vec{p} \vec{q}.\vec{q}' \vec{x}$. Thus, we conclude by induction hypothesis (1) on $\vec{x}$.

We can now apply the $\beta\gamma\delta$-elimination and $\mu$-introduction principles to obtain that $\mu (B P (\nu (B P)) \vec{x}) \Rightarrow (\vec{y} : [\vec{B} \vec{Z} X (\mu (B \vec{Z} X')) \vec{x}]$-reducible. Finally, we conclude using induction hypothesis (3) with $B := \lambda \vec{p} q \vec{q}.\vec{q}' \vec{x} \Rightarrow \lambda \vec{p} q \vec{q}.\vec{q}' \vec{x}$. It simply follows from Lemma 5 and the hypothesis.

We shall show that $[\mu (B \vec{Z} X')]$ is included in the set of proofs for which this holds, by showing that (a) this set is a candidate and (b) it is a prefixed point of $\phi$ such that $[\mu (B \vec{Z} X')] \Rightarrow \phi (\vec{y}) (\phi)$.

Let us consider $\mu (B \vec{Z} X') : \vec{y} \Rightarrow \vec{y}$.

First, $\vec{y}$ is a candidate for any $\vec{y}$ conditions (1) and (2) are inherited from $\vec{X}'$, only condition (3) is non-trivial. Assuming that every one-step reduct of a neutral derivation $\rho$ belongs to $\vec{y}$, we prove $\delta (\rho, \vec{x}.\alpha.\pi) \vec{x} \Rightarrow \vec{X}'$. This is done by induction on the strong normalizability of $\pi$. Using condition (3) on $\vec{X}'$, it suffices to consider one-step reducts: if the reduction takes place in $\rho$ we conclude by hypothesis; if it takes place in $\pi$ we conclude by induction hypothesis; finally, it cannot take place at toplevel because $\rho$ is neutral.

We now establish that $\phi (\vec{y}) \subseteq \vec{y}$: assuming $\rho \in \phi (\vec{y})$, we show that $\delta (\rho, \vec{x}.\alpha.\pi) \vec{x} \Rightarrow \vec{X}'$. This is done by induction on the strong normalizability of $\rho$ and $\pi$, and it suffices to show that each one step reduct belongs to $\vec{X}'$, with internal reductions handled simply by induction hypothesis. Therefore we consider the case where $\rho = (\mu (B \vec{Z} X') \vec{x})$ and our derivation reduces to $\pi [\vec{y}] \vec{x} \Rightarrow (F_{\lambda \vec{p} q \vec{q}.\vec{q}'}(\vec{x}.\alpha.\pi)) \vec{x}$. Now, recall that $[\pi [\vec{y}] \vec{x}]$ is ((BZ)X')-reducible. Since $\mu (B \vec{Z} X') \vec{x}$ and we have $\vec{y} \in [B Z X']\vec{x}$. By induction hypothesis (1) we obtain $F_{\lambda \vec{p} q \vec{q}.\vec{q}'}(\vec{x}.\alpha.\pi) \Rightarrow (BZ)X'$-reducible, since $\delta (\rho, \vec{x}.\alpha.\pi) \vec{x} \Rightarrow (\vec{x}.\beta : X')$-reducible by definition of $\vec{y}$. We conclude by composing all that.

(4) Coinduction is similar to induction. Let us consider $\vec{Y} := \vec{y} \Rightarrow (\pi \in SN \mid \pi \Rightarrow \nu (\vec{x}.\alpha.\pi) \Rightarrow \rho \in \vec{X}' \vec{x} \Rightarrow \vec{X}' \vec{x})$ implies $\rho \in \vec{X}' \vec{x} \Rightarrow \vec{X}' \vec{x}$.

It is easy to show that $\vec{Y}$ is a predicate candidate, and if we show that $\vec{Y} \subseteq \nu (\vec{B} \vec{Z})$ we can conclude because the properties on $\rho$ and $\pi$ are preserved by reduction.

We have $\nu (\vec{B} \vec{Z}) = \vec{y} (\vec{y})$, so it suffices to establish that $\vec{Y}$ is a post-fixed point of $\phi$, or in other words that for any $\vec{y}$ and $\vec{y} \in \vec{Y}$, $\delta (\vec{y}, \vec{x} : \vec{X}) \vec{x} \in [B Z Y \vec{x}]$. We do this as usual by induction on the strong normalizability of $\pi$ and the only interesting case to consider is the toplevel reduction, which can occur when $\pi = \nu (\vec{x}.\alpha.\pi)$. The reduce is $F_{\lambda \vec{p} q \vec{q}.\vec{q}'}(\vec{x}.\alpha.\pi) \vec{x} \Rightarrow (\nu (\vec{y}) \vec{x}) \vec{x} \Rightarrow (\nu (\vec{y}) \vec{x}) \vec{x} \Rightarrow (\nu (\vec{y}) \vec{x}) \vec{x}$. It does belong to $[B Z Y \vec{x}]$ because $\rho \in \vec{X}' \vec{x}$ by definition of $\pi \in \vec{Y}$; $\nu (\vec{y}) \vec{x}$ is a candidate for any $\vec{x}$ conditions (1) and (2) (P \subseteq P_1 \subseteq P_2 and $\vec{P} \in P \subseteq P_1 \subseteq P_2$). By induction hypothesis,
Next, we observe that and property (b) of our pre-models, we have to check it separately for each rewrite rule. An instance simple to show that it satisfies (b). To check that it verifies of a rule defining the initial congruence cannot involve the new other instances.

E. Proof of Theorem 3

We define $\equiv_{a,i}$ (resp. $\equiv_{a,a}$) to be the congruence resulting from the extension of $\equiv$ with rule instances $a_i \rightsquigarrow B$ for $a_i \neq a_i$ (resp. $a_i \neq a_i$). Let us also write $P \equiv a_i$ (resp. $P \equiv a_i$) when $a_i \equiv a_i$ (resp. $a_i \equiv a_i$) for any $a_i$ which may occur in $P$. We shall build a family of pre-models $M^{a,i}$ such that:

(a) for any $a_i \equiv a_i$, $[a_i]_{M^{a,i}} = SN$;
(b) for any $P \equiv a_i$ and $a_i \equiv a_i$, $[P]_{M^{a,i}} = [P]_{M^{a,i}}$;
(c) $M^{a,i}$ is a pre-model of $\equiv_{a,i}$.

We proceed by well-founded induction. Assuming that $M^{a,i}$ is defined for all $a_i \equiv a_i$, we shall thus build $M^{a,i}$.

We first define $M^{a,i}$ by taking each $a_i$ to be the same as in $M^{a,i}$ when $a_i \equiv a_i$ and $SN$ otherwise. By this definition and property (b) of our pre-models, we have

$$[P]_{M^{a,i}} = [P]_{M^{a,i}}$$

for any $P \equiv a_i$ and $a_i \equiv a_i$.

Next, we observe that $M^{a,i}$ is a pre-model of $\equiv_{a,i}$. It suffices to check it separately for each rewrite rule. An instance $P \rightsquigarrow Q$ of a rule defining the initial congruence cannot involve the new predicates, so in that case we do have

$$[P]_{M^{a,i}} = [P]_{M} = [Q]_{M} = [Q]_{M^{a,i}}.$$

For a rule instance $a_i \rightsquigarrow B$ with $a_i \equiv a_i$, the property is similarly inherited from $M^{a,i}$ because $B \equiv a_i$ by (2):

$$[a_i]_{M^{a,i}} = [a_i]_{M^{a,i}} = [B]_{M^{a,i}} = [B]_{M^{a,i}}.$$

We finally build $M^{a,i}$ to be the same as $M^{a,i}$ except for $a_i$ which is defined as follows:

- If there is no rule $a_i \rightsquigarrow B$ such that $a_i \equiv a_i$, we define $\hat{a_i}$ to be $SN$.
- Otherwise, pick any such $B$, and define $\hat{a_i}$ to be $[B]_{M^{a,i}}$.

This is uniquely defined: for any other $a_i \rightsquigarrow B'$ such that $a_i \equiv a_i$, we have $B \equiv B'$ by (1), and thus $[B]_{M^{a,i}}$. Since $M^{a,i}$ is a fortiori a pre-model of $\equiv_{a,i}$, it is easy to check that it is compatible with all rewrite rules.

APPENDIX II: Formalization of Strong Normalizability

We detail below the formalization of Tait’s strong normalizability argument described in Section V. The full Abella scripts are available at http://www.lix.polytechnique.fr/~dbaeldel/lics12.

Following the two-level reasoning methodology facilitated by Abella, we first define the objects and judgments of interest in a module file shown on Figure 8. The specification is given by means of hereditary Harrop clauses. Adequate representations are obtained by considering uniform proofs for the corresponding clauses. For instance, uniform proofs of $\Gamma \vdash M T$ are in bijection with typing derivations in simply typed $\lambda$-calculus. Abella allows one to reason over the specified objects through this representation methodology. Derivability is inductively defined as a built-in predicate in Abella, written in a concise notation: $\{ C \vdash M \}$ corresponds to the derivability of $C \vdash M T$. More details on the methodology and syntax of Abella, refer to http://abella.cs.umn.edu.

A. Preliminaries

We first prove that $\text{steps}$ is transitive and that it is a congruence.

Theorem steps_steps : forall M M N P, {steps M M} -> {steps M N} -> {steps M P}.

Theorem steps_app_left : forall M M' N, {steps M M'} -> {steps (app M N) (app M' N)}.

Theorem steps_app_right : forall M M' N, {steps M M'} -> {steps (app M N) (app M' N)}.

Theorem steps_app : forall M M' N, {steps M M'} -> {steps (app M N) (app M' N)}.

Theorem steps_abs : forall M M', nabla x, {steps (M x) (M' x)} -> {steps (abs M) (abs M')}.

Next, we define open terms, which cannot be done at the specification level like, for example, the definition of $\text{isty}$. We prove a few basic properties of open terms.

Define isotm : tm -> prop by

$$\text{istotm} \equiv \text{prop by} \ nablax, \ \text{isotm} x ;$$

isotm (app M N) := isotm M \ \text{ unpaid} isotm N ;

isotm (abs M) := nablax, isotm M x.

Theorem isotm_subst : forall M N, nablax, x,
isty iota.

isty (arrow T T') :- isty T, isty T'.

isty (app M N) :- istm M, istm N.

isty (abs M) :- pi x \ istm x => istm (M x).

of (app M N) T' :- of N T, of M (arrow T T').
of (abs M) (arrow T T') :-
isty T, pi x \ of x T => of (M x) T'.

Theorem isotm_step : forall M M',
isotm M -> {step M M'} -> isotm M'.

Theorem step_osubst_steps : forall M N N', nabla x,
isotm (M x) -> {step N N'} -> {steps (M N) (M N')}.

B. Strong normalizability

We define strong normalizability and prove some basic properties about it.

Define sn : tm -> prop by

  sn M := forall N, {step M N} -> sn N.

Theorem var_sn : nabla x, sn x.

Theorem sn_step_sn : forall M N, sn M -> {step M N} -> sn N.

Theorem sn_preserve : forall M, nabla x, sn (app M x) -> sn M.

Theorem sn_app : forall M N, sn M -> sn N ->

  (forall M', {steps M (abs M')} -> false) ->

  sn (app M N).

C. Reducibility

We now give the definition of reducibility. Abella issues a warning here, because the definition is not monotone, and is thus not formally supported by its underlying theory. However, as explained in Section V, this recursive definition can be justified as rewrite rules in our framework. Below, it is always going to be used following this interpretation.

Define red : tm -> ty -> prop by

  red M iota := sn M ;
  red (arrow T T') := forall N,
isotm N -> red N T -> red (app M N) T'.

We now prove the three conditions defining candidates of reducibility. We first show that reducible terms are SN, and simultaneously that variables are reducible, which requires a generalization to showing that \(x N_1...N_k\) is reducible when the \(N_i\) are SN.

Define vargen : tm -> prop by

  nabla x, vargen x ;
  vargen (app M N) := vargen M \ / \ sn N.

Theorem vargen_step_vargen : forall M N,
vargen M -> {step M N} -> vargen N.

Theorem vargen_steps_noabs : forall M M',
vargen M -> {steps M (abs M')} -> false.

Theorem vargen_sn : forall M, vargen M -> sn M.

Theorem red_sn_gen : forall M T,
{isty T} ->
(red M T -> sn M) \ / \ (vargen M -> red M T).

Theorem var_red : forall T, nabla x,
{isty T} -> red x T.

Theorem red_sn : forall M T,
{isty T} -> red M T -> sn M.

The second condition is that reducts remain in reducibility sets.

Theorem red_step : forall M M' T,
{isty T} -> red M T -> {step M M'} ->
red M' T.

Theorem red_steps : forall M M' T,
{isty T} -> red M T -> {steps M M'} ->
red M' T.

Finally, if all one-step reducts of a neutral term are in a set, then so is the term. We only prove it for neutral terms which are applications. Here, the inner induction is taken care of using an auxiliary lemma.

Theorem cr3_aux : forall M1 M2 N T1 T',

Fig. 8. Module file for the Abella formalization
\{isty \, T1\} \rightarrow \text{sn} \, N \rightarrow \text{isotm} \, N \rightarrow \text{red} \, N \, T1 \rightarrow \\
(\forall M1 \, M2, \\
\text{forall} \, M', \\
\{\text{step} \, (\text{app} \, M1 \, M2) \, M'\} \rightarrow \text{red} \, M' \, T') \rightarrow \\
\text{red} \, (\text{app} \, M1 \, M2) \, T' \\)

\text{red} \, (\text{app} \, M', \text{N} \, T').

\text{Theorem} \, \text{red\_anti} : \forall \, M \, N \, T, \\
\{isty \, T\} \rightarrow \\
(\forall M', \\
\{\text{step} \, (\text{app} \, M \, N) \, M'\} \rightarrow \text{red} \, M' \, T) \rightarrow \\
\text{red} \, (\text{app} \, M \, N) \, T.

\textbf{D. \ Contexts}\n
We characterize the contexts used in derivations of \text{of} \, M \, T and subst \, M \, T that are involved in the proof of adequacy. We also define separately their relationship, using mapctx. This approach requires a fair number of book-keeping lemmas.

\text{Define} \, \text{name} : \text{tm} \rightarrow \text{prop} \text{ by} \, \nabla \, x, \text{name} \, x.

\text{Define} \, \text{ofctx} : \text{olist} \rightarrow \text{prop} \text{ by} \\
\text{ofctx} \, \text{nil} ; \\
\nabla \, x, \text{ofctx} \, (\text{of} \, x \, T \, :: \, C) \, := \\
\{\text{isty} \, T\} \, \setminus \, \text{ofctx} \, C.

\text{Define} \, \text{substctx} : \text{olist} \rightarrow \text{prop} \text{ by} \\
\text{substctx} \, \text{nil} ; \\
\nabla \, x, \text{substctx} \, (\text{subst} \, x \, (M \, x) \, :: \, C) \, := \\
\nabla \, x, \text{isotm} \, (M \, x) \, \setminus \, \text{substctx} \, C.

\text{Define} \, \text{mapctx} : \text{olist} \rightarrow \text{olist} \rightarrow \text{prop} \text{ by} \\
\text{mapctx} \, \text{nil} \, \text{nil} ; \\
\nabla \, x, \text{mapctx} \, (\text{of} \, x \, T \, :: \, C) \, (\text{subst} \, x \, (M \, x) \, :: \, C') \, := \\
\nabla \, x, \\
\{\text{isty} \, T\} \, \setminus \, \text{isotm} \, (M \, x) \, \setminus \, \text{mapctx} \, C \, C'.

\text{Theorem} \, \text{ofctx\_member\_isty} : \forall C \, T, \\
\text{ofctx} \, C \rightarrow \text{member} \, (\text{isty} \, T) \, C \rightarrow \text{false}.

\text{Theorem} \, \text{isty\_weaken} : \forall C \, T, \\
\text{ofctx} \, C \rightarrow \{C \, \vdash \text{isty} \, T\} \rightarrow \{\text{isty} \, T\}.

\text{Theorem} \, \text{ofctx\_member\_isty} : \forall C \, M \, T, \\
\text{ofctx} \, C \rightarrow \text{member} \, (\text{of} \, M \, T) \, C \rightarrow \{\text{isty} \, T\}.

\text{Theorem} \, \text{of\_isty} : \forall C \, M \, T, \\
\text{ofctx} \, C \rightarrow \{C \, \vdash \text{of} \, M \, T\} \rightarrow \{\text{isty} \, T\}.

\text{Theorem} \, \text{mapctx\_of} : \forall G \, G' \, M \, T, \\
\text{mapctx} \, G \, G' \rightarrow \text{member} \, (\text{of} \, M \, T) \, G \rightarrow \\
\text{name} \, M \, \backslash \\
\exists M', \text{red} \, M' \, T \, \backslash \, \text{member} \, (\text{subst} \, M \, M') \, G'.

\text{Theorem} \, \text{mapctx\_subst} : \forall G' \, M \, M', \\
\text{mapctx} \, G' \rightarrow \text{member} \, (\text{subst} \, M \, M') \, G' \rightarrow \\
\text{name} \, M \, \backslash \\
\exists T, \text{red} \, M' \, T \, \backslash \, \text{member} \, (\text{of} \, M \, T) \, G.

\text{Theorem} \, \text{mapctx\_split} : \forall C \, C', \\
\text{mapctx} \, C \, C' \rightarrow \text{ofctx} \, C \, \setminus \, \text{substctx} \, C'.

\text{Theorem} \, \text{ofctx\_member\_name} : \forall C \, M \, T, \\
\text{ofctx} \, C \rightarrow \text{member} \, (\text{of} \, M \, T) \, C \rightarrow \text{name} \, M.

\text{Theorem} \, \text{of\_isotm} : \forall C \, M \, T, \\
\text{ofctx} \, C \rightarrow \{C \, \vdash \text{of} \, M \, T\} \rightarrow \text{isotm} \, M.

\text{Theorem} \, \text{substctx\_member} : \forall C \, M \, M', \\
\text{substctx} \, C \rightarrow \text{member} \, (\text{subst} \, M \, M') \, C \rightarrow \\
\text{name} \, M \, \backslash \, \text{isotm} \, M'.

\text{Theorem} \, \text{subst\_isotm} : \forall C \, M \, M', \\
\text{substctx} \, C \rightarrow \text{isotm} \, M \rightarrow \{C \, \vdash \text{subst} \, M \, M'\} \rightarrow \\
\text{isotm} \, M'.

\text{Theorem} \, \text{member\_not\_fresh} : \\
\forall X \, L, \nabla \, (n:tm), \\
\text{member} \, (X \, n) \, L \rightarrow \exists X', \, X = n \backslash X'.

\text{Theorem} \, \text{substctx\_member\_unique\_aux} : \\
\forall C \, M \, M', \nabla \, x, \\
\text{substctx} \, (C \, x) \rightarrow \\
\text{member} \, (\text{subst} \, x \, (M \, x)) \, (C \, x) \rightarrow \\
\text{member} \, (\text{subst} \, x \, (M' \, x)) \, (C \, x) \rightarrow \\
M = M'.

\text{Theorem} \, \text{substctx\_member\_unique} : \forall C \, X \, M \, M', \\
\text{substctx} \, C \rightarrow \\
\text{member} \, (\text{subst} \, X \, M) \, C \rightarrow \\
\{C \, \vdash \text{subst} \, X \, M'\} \rightarrow M = M'.

\text{E. \ Adequacy \ theorem}\n
\text{Theorem} \, \text{abs\_case} : \forall M \, N \, T', \nabla \, x, \\
\text{isotm} \, (M \, x) \rightarrow \{\text{isty} \, T'\} \rightarrow \\
\text{sn} \, (M \, x) \rightarrow \text{sn} \, N \rightarrow \\
\text{red} \, (M \, N) \, T' \rightarrow \\
\text{red} \, (\text{app} \, (\text{abs} \, M) \, N) \, T'.

\text{Theorem} \, \text{of\_red} : \forall M \, M' \, T \, C, \\
\text{mapctx} \, C \, C' \\
\{C \, \vdash \text{of} \, M \, T\} \rightarrow \\
\{C' \, \vdash \text{of} \, M' \, T\} \rightarrow \\
\text{red} \, M' \, T.
To apply the adequacy result and obtain strong normalizability, it only remains to show that for any typed term we can define the identity substitution with which we have \( \{ C' |- \text{subst } \mathsf{M} \mathsf{M} \} \), from which \text{red } \mathsf{M} \mathsf{T} \text{ and } \text{sn } \mathsf{M} \text{ follow.} \)