Reasoning About Higher-Order Relational Specifications

Yuting Wang
University of Minnesota, USA
yuting@cs.umn.edu

Kaustuv Chaudhuri
INRIA, France
kaustuv.chaudhuri@inria.fr

Andrew Gacek
Rockwell Collins, USA
andrew.gacek@gmail.com

Gopalan Nadathur
University of Minnesota, USA
gopalan@cs.umn.edu

ABSTRACT

The logic of hereditary Harrop formulas (HH) has proven useful for specifying a wide range of formal systems that are commonly presented via syntax-directed rules that make use of contexts and side-conditions. The two-level logic approach, as implemented in the Abella theorem prover, embeds the HH specification logic within a rich reasoning logic that supports inductive and co-inductive definitions, an equality predicate, and generic quantification. Properties of the encoded systems can then be proved through the embedding, with special benefit being extracted from the transparent correspondence between HH derivations and those in the encoded formal systems. The versatility of HH relies on the free use of nested implications, leading to dynamically changing assumption sets in derivations. Realizing an induction principle in this situation is nontrivial and the original Abella system uses only a subset of HH for this reason. We develop a method here for supporting inductive reasoning over all of HH. Our approach relies on the ability to characterize dynamically changing contexts through finite inductive definitions, and on a modified encoding of backchaining for HH that allows these finite characterizations to be used in inductive arguments. We demonstrate the effectiveness of our approach through examples of formal reasoning on specifications with nested implications in an extended version of Abella.

Categories and Subject Descriptors

F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Specification Techniques*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Proof Theory*

Keywords

formal specifications, meta-theoretic reasoning, higher-order abstract syntax, induction over higher-order specifications.

PPDP'13, September 16–18 2013, Madrid, Spain.

http://dx.doi.org/10.1145/2505879.2505889.

1. INTRODUCTION

We are concerned in this paper with the task of reasoning about formal systems such as programming languages, proof systems and process calculi. The data objects that are of interest within such systems often embody binding constructs. Higher-order abstract syntax (HOAS) provides an effective means for representing such structure. In an HOAS representation, which is based on using a well-calibrated λ-calculus as a metalanguage, the binding structure of object language expressions is encoded using abstractions in λ-terms. For example, consider an object language that is itself a λ-calculus. Letting `tm` be a type for the representation of these terms, their HOAS encoding can be built around two constructors, $\texttt{app} : \texttt{tm} \to \texttt{tm} \to \texttt{tm}$ and $\texttt{abs} : (\texttt{tm} \to \texttt{tm}) \to \texttt{tm}$: the object term $\lambda x.\, \lambda y.\, y\ x$ would, for instance, be represented as $\texttt{abs}\ (\lambda x.\, \texttt{abs}\ (\lambda y.\, \texttt{app}\ y\ x))$. Observe that there is no constructor for variables in this encoding; object-level variables are directly represented by the variables of the meta-language, bound by an appropriate abstraction. The virtue of HOAS is that if the metalanguage is properly chosen, *i.e.*, if it incorporates λ-conversion but is otherwise weak in a computational sense, then it provides a succinct and logically precise treatment of object-level operations such as substitution and analysis of binding structure.

Formal systems are usually defined by the relations that hold between the data objects that constitute them. Such relations are conveniently presented through syntax-directed rules. When they pertain to data embodying binding structure, these specifications naturally tend to be higher-order, *i.e.*, their rule-based presentation involves the use of contexts. Moreover, these contexts can contain conditional assertions whose use may require the construction of sub-derivations. Towards understanding this issue, consider the alternative notation for λ-terms due to De Bruijn in which bound variables are not named and their occurrences are represented instead by indexes that count the abstractions up to the one binding them [6]. Using the type `dtm` for the representation of λ-terms in this form, we can encode them via the constructors $\texttt{dvar} : \texttt{nat} \to \texttt{dtm}$ (for variables), $\texttt{dapp} : \texttt{dtm} \to \texttt{dtm} \to \texttt{dtm}$ and $\texttt{dabs} : \texttt{dtm} \to \texttt{dtm}$. Now, there is a natural bijection between the named and nameless representation of λ-terms. Writing $\Gamma \vdash m \equiv_h d$ to denote the correspondence between the HOAS-encoded term m that occurs at *depth* h (*i.e.*, under h λ-abstractions) and the De Bruijn term d where Γ determines the mapping between free variables in the two representations, we can define this relation via these rules:

$$\frac{\Gamma \vdash m \equiv_h d \quad \Gamma \vdash n \equiv_h e}{\Gamma \vdash \texttt{app}\ m\ n \equiv_h \texttt{dapp}\ d\ e} \tag{1}$$

$$\frac{\Gamma, \forall i, k.\, ((h + k = i) \supset x \equiv_i \texttt{dvar}\ k) \vdash m \equiv_{h+1} d}{\Gamma \vdash \texttt{abs}\ (\lambda x.\, m) \equiv_h \texttt{dabs}\ d} \quad (2)$$

$$\frac{\forall i, k.\, ((h + k = i) \supset x \equiv_i \texttt{dvar}\ k) \in \Gamma \quad \vdash h + k = i}{\Gamma \vdash x \equiv_i \texttt{dvar}\ k} \quad (3)$$

The rule for relating applications is straightforward. To relate $\texttt{abs}\ (\lambda x.\, m)$ to a De Bruijn term at depth h, we must relate each occurrence of x in m, which must be at a depth $h+k$ for some $k > 0$, to the De Bruijn term $\texttt{dvar}\ k$. To encode this correspondence, the context is extended in the premise of rule (2) with a (universally quantified) implicational formula. Note also that this rule carries with it the implicit assumption that the name x used for the bound variable is fresh to Γ, the context for the concluding judgment. Eventually, when the HOAS term on the right of $\vdash$ is a variable, rule (3) provides the means to complete the derivation by using the relevant assumption from Γ. Observe that the use of this rule entails a construction of an auxiliary derivation for $\vdash h + k = i$.

Our ultimate interest is in reasoning about such higher-order relational specifications. For example, we might be interested in showing that the relation that we have defined above identifies a bijective mapping between the two representations of λ-terms. One part of establishing this fact is proving that the relation is deterministic from left to right, *i.e.*, that every term in the named notation is related to at most one term in the nameless notation. Writing $\{\Gamma \vdash m \equiv_h d\}$ to denote derivability of the judgment $\Gamma \vdash m \equiv_h d$ by virtue of the rules (1), (2) and (3), this involves providing a proof for the following assertion:

$$\forall \Gamma, m, h, d, e.\, \{\Gamma \vdash m \equiv_h d\} \supset \{\Gamma \vdash m \equiv_h e\} \supset d = e. \quad (4)$$

Note that $\forall$ and $\supset$ in (4) are logical constants at the reasoning level in contrast to the ones in (1) – (3) that are at the object level. Such a proof must obviously be based on an analysis of derivability using the rules that define the relation. To formalize such reasoning, we need a logic that can encode these rules in a way that allows case analysis to be carried out over their structure. Furthermore, the logic must embody an induction principle since proofs of general theorems of the kind we are interested in must be inductive over the structure of object-level derivations. A particular difficulty in articulating such inductive arguments relative to higher-order relational specifications is that they may need to take into account derivations in the object system that rely on hypotheses in changing contexts. For example, a proof of (4) must accommodate the fact that Γ can be dynamically extended in a derivation of $\Gamma \vdash m \equiv_h d$ and that the particular content of Γ influences the derivation in the variable case via the rule (3).

In this paper we develop a framework that provides an elegant solution to this reasoning problem. Formally, our framework is a realization of the two-level logic approach [11, 13], which is based on embedding a *specification logic* inside a *reasoning logic*. Within this setup, we take our specification logic to be that of hereditary Harrop formulas (HH). This logic extends the well-known logic of Horn clauses essentially by employing simply typed λ-terms as a means for representing data objects and by permitting universal quantification and implications in the bodies of clauses. As such, it provides an excellent basis for encoding rule-based higher-order specifications over HOAS representations [14]. Moreover, these formulas can be given a proof-theoretic interpretation that simultaneously is complete with respect to intuitionistic logic and reflects the structure of derivations based on the object-level rules they encode. For the reasoning logic we use the system $\mathcal{G}$ from [10]. This logic permits atomic predicates to be defined through clauses in a way that allows case analysis based reasoning to be carried out over them. The treatment of definitions in $\mathcal{G}$ can also be specialized to interpret them inductively. The capability for formally proving properties about relational specifications is realized in this setting by first encoding HH derivability in $\mathcal{G}$ via an inductive definition and then using this encoding to reflect reasoning based on object-level rules into reasoning over HH derivations that formalize these rules.

The two-level logic approach has previously been implemented in the Abella system and has been used successfully in several reasoning tasks [9]. However, the original version of Abella uses a fragment of HH that is capable of treating syntax-directed rules only when the dynamic additions to their contexts is restricted to atomic formulas. There is an inherent difficulty in structuring the reasoning when contexts can be extended with formulas having an implicational structure. For example, as already noted, case analysis over the derivation of $\Gamma \vdash m \equiv_h d$ in a proof of (4) must take into account the fact that the derivation can proceed by using a hypothesis that was dynamically added to Γ. Without well-defined constraints on Γ, it is difficult to predict how such hypotheses might be used and indeed the assertion may not even be true.

In the example under consideration, there is an easy resolution to the dilemma described above. We are not interested in proving assertion (4) for arbitrary Γ but only for those Γs that result from additions made through the rule (2). The elements of Γ must therefore all be of the form $\forall i, k.\, ((h + k = i) \supset x \equiv_i \texttt{dvar}\ k)$ where h is some depth and x is some variable not otherwise present in Γ. Moreover, the use of such assumptions in derivations can occur only through rule (3) that is in fact another instance of a backchaining step that is manifest explicitly in the rules (1) and (2). Thus, the structure of Γ can be encoded into an inductive definition in $\mathcal{G}$ and treated in a finitary fashion by the machinery that $\mathcal{G}$ already provides for reasoning about backchaining steps.

The key insight underlying this paper is that the above observation generalizes cleanly to other reasoning situations that involve contexts with higher-order hypotheses. Concretely, the contexts that need to be considered in these situations are completely determined by the additions that can be made to them. Further, the structure of such additions must already be manifest in the original specifications and can therefore always be encapsulated in an inductive definition. To take advantage of this observation we modify the encoding of HH derivations in Abella to support reasoning also over the backchaining steps that result from using dynamically added assumptions. We then demonstrate the power of this extension through its use in explicitly proving the bijectivity property discussed above as well as another non-trivial property about paths in λ-terms and their relation to reduction. These exercises also show the benefits of using a logic for specifications: the meta-theoretic properties of this logic greatly simplify the reasoning process.

In summary, we make three contributions through this work: we propose a methodology for reasoning about higher-order relational specifications, we present an implemented system for supporting this methodology and we show its

effectiveness through actual reasoning tasks. The framework we describe exploits the HOAS representation style to structure and simplify the reasoning process. To the best of our knowledge, the only other systems that use such an approach to similar effect are Twelf [18] and Beluga [19]. In contrast to these systems, the one we develop here provides a rich language for stating meta-theoretic properties of specifications and an explicit logic for articulating their proofs. We elaborate on these comparisons in a later section.

The rest of the paper is structured as follows. In the next two sections, we present the specification logic HH, the reasoning logic $\mathcal{G}$, and the two-level logic approach that is built out of their combination. Section 4 illustrates the use of the resulting framework and the associated methodology for a novel and non-trivial example. The focus in this example is on specifications that have a rich higher-order character and on showing how context definitions and context relations can be used to structure and realize the reasoning process. The last two sections discuss related work and conclude the paper by providing a perspective on its technical contributions.

The extended Abella system that is the outcome of this work is available at [1]. Besides the examples described in this paper, this version of Abella also contains a number of other examples of reasoning about higher-order relational specifications that illustrate our approach.

2. THE SPECIFICATION LOGIC

In this section, we present the specification logic HH, show how it can be used to encode rule-based descriptions, and discuss some of its meta-theoretic properties that turn out to be useful in reasoning about specifications developed in it.

2.1 The HH Proof System

The logic HH of hereditary Harrop formulas is a predicative fragment of Church's Simple Theory of Types [5] whose expressions are simply typed λ-terms. Types are built freely from primitive types, which must include the type o of formulas, and the function type constructor $\to$. Terms are built from a user-provided signature of typed constants, and are considered identical up to $\alpha\beta\eta$-conversion. We write $\Sigma \vdash t : \tau$ to denote that t is a well-formed term of type τ relative to Σ. Well-formed terms of type o relative to Σ are called Σ-*formulas* or just *formulas* when Σ is implicit.

Logic is introduced into this background via a countable family of constants containing: $\Rightarrow, \& : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$ (written infix, and associating to the right and left, respectively), and for every type τ not containing o, the (generalized) universal quantifier $\Pi_\tau : (\tau \to \mathsf{o}) \to \mathsf{o}$. An atomic formula, denoted by A possibly with a subscript, is one that does not have a logical constant as its head symbol. We use the abbreviations $\Pi x{:}\tau.\, F$ for $\Pi\ (\lambda x{:}\tau.\, F)$, $\Pi x_1{:}\tau_1, \ldots, x_n{:}\tau_n.\, F$ for $\Pi x_1{:}\tau_1.\ \ldots \Pi x_n{:}\tau_n.F$, and $\Pi \bar{x}{:}\bar{\tau}.\, F$ where $\bar{x} = x_1, \ldots, x_n$ and $\bar{\tau} = \tau_1, \ldots, \tau_n$ for $\Pi x_1{:}\tau_1.\ \ldots \Pi x_n{:}\tau_n.\, F$. We will omit the types when they are irrelevant or can be inferred from context. Finally, we will often write $G \Leftarrow F$ (with "$\Leftarrow$" associating to the left and pronounced "if") to mean $F \Rightarrow G$.

The HH proof system has two kinds of sequents:

$$\Sigma; \Theta; \Gamma \vdash G \qquad \textit{goal-reduction sequent}$$
$$\Sigma; \Theta; \Gamma, [F] \vdash A \qquad \textit{backchaining sequent}$$

In these sequent forms, Σ is a signature; Γ and Θ are multisets of Σ-formulas; G is a Σ-formula and A is an atomic Σ-formula.

Goal reduction rules

$$\frac{\Sigma; \Theta; \Gamma, F \vdash G}{\Sigma; \Theta; \Gamma \vdash F \Rightarrow G} \Rightarrow_R \qquad \frac{\Sigma; \Theta; \Gamma \vdash G_1 \quad \Sigma; \Theta; \Gamma \vdash G_2}{\Sigma; \Theta; \Gamma \vdash G_1 \,\&\, G_2} \&_R$$

$$\frac{(c \notin \Sigma) \quad \Sigma, c{:}\tau; \Theta; \Gamma \vdash (G\ c)}{\Sigma; \Theta; \Gamma \vdash \Pi_\tau G} \Pi_R$$

Backchaining rules

$$\frac{\Sigma; \Theta; \Gamma \vdash G \quad \Sigma; \Theta; \Gamma, [F] \vdash A}{\Sigma; \Theta; \Gamma, [G \Rightarrow F] \vdash A} \Rightarrow_L \qquad \frac{\Sigma; \Theta; \Gamma, [F_i] \vdash A}{\Sigma; \Theta; \Gamma, [F_1 \,\&\, F_2] \vdash A} \&_L$$

$$\frac{\Sigma \vdash t : \tau \quad \Sigma; \Theta; \Gamma, [(F\ t)] \vdash A}{\Sigma; \Theta; \Gamma, [\Pi_\tau F] \vdash A} \Pi_L$$

Structural rules

$$\frac{}{\Sigma; \Theta; \Gamma, [A] \vdash A}\ \mathsf{match}$$

$$\frac{(F \in \Theta) \quad \Sigma; \Theta; \Gamma, [F] \vdash A}{\Sigma; \Theta; \Gamma \vdash A}\ \mathsf{prog} \qquad \frac{(F \in \Gamma) \quad \Sigma; \Theta; \Gamma, [F] \vdash A}{\Sigma; \Theta; \Gamma \vdash A}\ \mathsf{dyn}$$

Figure 1: Rules for HH. In $\&_L$, $i \in \{1, 2\}$.

The context Θ is called the *static context* because it contains a finite and unchanging HH *program*. The context Γ, called the *dynamic context*, contains the assumptions introduced during the goal reduction procedure, and can therefore grow. The members of Θ and Γ are called the *static clauses* and the *dynamic clauses* respectively.

Figure 1 contains the inference rules of HH. Reading the rules as a computation of premise sequents from goal sequents, the *goal reduction rules* decompose the goal on the right of $\vdash$ until it becomes atomic. The $\Rightarrow_R$ rule extends the dynamic context with the antecedent of the implication, while the Π_R rule extends the signature with a fresh constant for the universally quantified variable.

Once the goal becomes atomic, the only rules that apply are the final two structural rules that *select* a backchaining clause. The prog rule selects a static clause, while the dyn rule selects a dynamic clause. In either case, the premise is a backchaining sequent with the selected clause indicated by $[-]$. The HH proof system does not prescribe a strategy for selecting clauses, so to reason about HH derivations we will have to consider every possibility.

While the selected clause is non-atomic, the *backchaining rules* are used to reduce it. The $\Rightarrow_L$ rule changes the selection to the succedent of the implication, moving in the direction of the *head* of the clause, and additionally checks that the antecedent is derivable. The $\&_L$ rules change the selection to one of the operands of a $\&$. The Π_L rule changes the selection to some instance of a universally quantified clause. When the selected clause has been reduced to atomic form, the corresponding branch of the proof finishes by the rule match which requires that the atomic clause match the atomic goal. Therefore, if the right hand side does not match, then this branch of the proof is invalid and some choice made earlier in the proof needs to be revisited.

In the common case of a clause with the form $\Pi \bar{x}{:}\bar{\tau}.\, G_1 \Rightarrow \cdots \Rightarrow G_n \Rightarrow A$, the match rules and the backchaining rules compose to give this derived rule:

$$\frac{\Sigma \vdash \bar{t} : \bar{\tau} \quad \Sigma; \Theta; \Gamma \vdash [\bar{t}/\bar{x}]G_1 \quad \cdots \quad \Sigma; \Theta; \Gamma \vdash [\bar{t}/\bar{x}]G_n}{\Sigma; \Theta; \Gamma, [\Pi \bar{x}{:}\bar{\tau}.\, G_1 \Rightarrow \cdots \Rightarrow G_n \Rightarrow A] \vdash [\bar{t}/\bar{x}]A}$$

This derived form can readily be seen as implementing the backchaining procedure: the goal on the right of $\vdash$ is matched against the head of a selected clause, and then new goals are generated corresponding to the body of the clause.

2.2 Example: HOAS vs. De Bruijn λ-terms

As a concrete example of a higher-order relational specification in HH, let us consider the example in the introduction of λ-terms represented in two different ways, one with higher-order abstract syntax (HOAS) and the other using De Bruijn indexes. The signature of this specification consists of the following basic types: `nat` (for natural numbers), `tm` (for HOAS terms) and `dtm` (for De Bruijn terms), together with the following constants.

`nat`	HOAS (`tm`)	De Bruijn (`dtm`)
`z : nat`	`app : tm → tm → tm`	`dapp : dtm → dtm → dtm`
`s : nat → nat`	`abs : (tm → tm) → tm`	`dabs : dtm → dtm`
		`dvar : nat → dtm`

The static context specifies two relations, `add : nat → nat → nat → o` and `hodb : tm → nat → dtm → o`, that define addition relationally and relate the two encodings of terms at a given depth. These relations are given by the following static clauses.

$$\texttt{add z } X\ X. \qquad (R_{\text{addz}})$$

$$\texttt{add (s } X\texttt{) } Y\ \texttt{(s } Z\texttt{)} \Leftarrow \texttt{add } X\ Y\ Z. \qquad (R_{\text{adds}})$$

$$\texttt{hodb (app } M\ N\texttt{) } H\ \texttt{(dapp } D\ E\texttt{)} \Leftarrow \texttt{hodb } M\ H\ D\ \&\ \texttt{hodb } N\ H\ E. \qquad (R_{\text{app}})$$

$$\texttt{hodb (abs } M\texttt{) } H\ \texttt{(dabs } D\texttt{)} \Leftarrow \Pi x.\, \texttt{hodb } (M\ x)\ \texttt{(s } H\texttt{)}\ D \Leftarrow \Pi i, k.\, \texttt{hodb } x\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add } H\ k\ i. \qquad (R_{\text{abs}})$$

The clauses are written using the standard convention of indicating variables that are universally quantified using capital letters; that is, the clause R_{addz} stands for $\Pi X.\, \texttt{add z } X\ X$, *etc.* The clauses R_{app} and R_{abs} provide a transparent encoding of rules (1) and (2) relative to the HH proof system. Note especially the embedded implication in the body of R_{abs}: as we see in more detail in the example derivation below, when combined with the derived backchaining and the goal reduction rules, this implication leads to proving a sequent with an extended dynamic context that closely resembles the premise of (2). There is no clause corresponding to rule (3); it will arise from clauses in the dynamic context as part of the backchaining mechanism of HH.

Let Σ be the signature above and Θ be R_{addz}, R_{adds}, R_{app}, R_{abs}. Let us try to show that the term $\lambda x.\, \lambda y.\, (y\ x)$ corresponds to the De Bruijn term $\lambda.\, \lambda.\, (1\ 2)$. This amounts to proving the following HH sequent:

$$\Sigma; \Theta; \cdot \vdash \texttt{hodb (abs } (\lambda x.\, \texttt{abs } (\lambda y.\, \texttt{app } y\ x))\texttt{) z (dabs (dabs (dapp (dvar (s z)) (dvar (s (s z)))))))}.$$

The dynamic context is empty and the goal is atomic, so only the prog rule is applicable. Selecting R_{addz} or R_{adds} will fail because the heads are different predicates, and selecting R_{app} will also fail because the first-argument of `hodb` is `abs`, which does not unify with `app`. Therefore, the only choice is backchaining R_{abs}, which changes the proof obligation to:

$$\Sigma, x{:}\texttt{nat}; \Theta; (\Pi i, k.\, \texttt{hodb } x\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add z } i\ k) \vdash \texttt{hodb (abs } (\lambda y.\, \texttt{app } y\ x)\texttt{) (s z) (dabs (dapp (dvar (s z)) (dvar (s (s z)))))}.$$

Attempting to backchain the new dynamic clause using dyn will fail because the new signature constant x does not unify with `abs`. Hence, the sole possibility that remains is backchaining R_{abs} again, yielding:

$$\Sigma, x, y{:}\texttt{nat}; \Theta; (\Pi i, k.\, \texttt{hodb } x\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add z } k\ i), (\Pi i, k.\, \texttt{hodb } y\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add (s z) } k\ i) \vdash \texttt{hodb (app } y\ x\texttt{) (s (s z)) (dapp (dvar (s z)) (dvar (s (s z))))}.$$

Now we can only backchain R_{app} to yield two new proof obligations, the first of which is:

$$\Sigma, x, y{:}\texttt{nat}; \Theta; (\Pi i, k.\, \texttt{hodb } x\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add z } k\ i), (\Pi i, k.\, \texttt{hodb } y\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add (s z) } k\ i) \vdash \texttt{hodb } y\ \texttt{(s (s z)) (dvar (s z))}.$$

The only clause that we can select for backchaining is the second dynamic clause for y using dyn; none of the other clauses have a matching head. This modifies the goal to:

$$\Sigma, x, y{:}\texttt{nat}; \Theta; (\Pi i, k.\, \texttt{hodb } x\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add z } k\ i), (\Pi i, k.\, \texttt{hodb } y\ i\ \texttt{(dvar } k\texttt{)} \Leftarrow \texttt{add (s z) } k\ i) \vdash \texttt{add (s z) (s z) (s (s z))}.$$

This sequent is then proved by backchaining R_{adds} and R_{addz}. The other proof obligation is handled similarly.

2.3 Meta-theorems of ~~HH~~

As a logic, HH possesses several properties that can be useful in analyzing derivability and therefore in reasoning about specifications written in it. The following meta-theorems will be specifically useful in the examples we consider.

THEOREM 1 (META THEOREMS OF HH).

1. *If* $\Sigma; \Theta; \Gamma \vdash F$ *and* $\Sigma; \Theta; \Gamma, F \vdash G$ *are derivable, then so is* $\Sigma; \Theta; \Gamma \vdash G$ *(cut).*
2. *If* $\Sigma \vdash t : \tau$ *and* $\Sigma, c{:}\tau; \Theta; \Gamma \vdash G$ *(where* c *is not free in* Θ*) is derivable, then so is* $\Sigma; \Theta; [t/c]\Gamma \vdash [t/c]G$*, where* $[t/c]$ *stands for the capture-avoiding substitution of* t *for* c *(instantiation).*
3. *If* $\Sigma; \Theta; \Gamma \vdash G$ *is derivable, and* $F \in \Gamma$ *implies* $F \in \Delta$*, then* $\Sigma; \Theta; \Delta \vdash G$ *is also derivable (monotonicity).*

PROOF. Each theorem follows by a straightforward inductive argument. See also Thm. 2. □

A direct corollary of the monotonicity theorem is that weakening and contraction are admissible for the dynamic context. Observe that the static context Θ never changes, even in the case of cut and instantiation. Obviously this theorem holds even if Θ is empty, so a variant proof system that combines the static and dynamic contexts into a single context will also enjoy the same properties. However, when reasoning about the specification of a computational system, we are almost never interested in considering situations where the static rules of the system change.

3. THE TWO-LEVEL LOGIC APPROACH

We describe now the reasoning logic $\mathcal{G}$ and outline the encoding of HH in $\mathcal{G}$ that underlies our particular use of the two-level logic approach. We then illustrate the resulting framework by using it to formalize and prove the bijectivity property of the relation between HOAS and De Bruijn representations of λ-terms.

$$\frac{(B \approx B')}{\Xi; \Delta, B \Vdash B'}\,\mathsf{id} \qquad \frac{\Xi; \Delta \Vdash B \quad \Xi; \Delta, B' \Vdash C \quad (B \approx B')}{\Xi; \Delta \Vdash C}\,\mathsf{cut}$$

$$\frac{\Xi, \Sigma, \mathcal{C} \vdash t : \tau \quad \Xi; \Delta, B\ t \Vdash C}{\Xi; \Delta, \forall_\tau B \Vdash C}\,\forall_L$$

$$\frac{(h \notin \Xi) \quad (\bar{c} = \mathrm{supp}(B)) \quad \Xi, h; \Delta \Vdash B\ (h\ \bar{c})}{\Xi; \Delta \Vdash \forall_\tau\ B}\,\forall_R$$

$$\frac{(a \in \mathcal{C} \setminus \mathrm{supp}(B)) \quad \Xi; \Delta, (B\ a) \Vdash C}{\Xi; \Delta, \nabla_\tau\ B \Vdash C}\,\nabla_L$$

$$\frac{(a \in \mathcal{C} \setminus \mathrm{supp}(B)) \quad \Xi; \Delta \Vdash (B\ a)}{\Xi; \Delta \Vdash \nabla_\tau\ B}\,\nabla_R$$

Figure 2: Selected rules of $\mathcal{G}$.

3.1 The Reasoning Logic $\mathcal{G}$

Specifications based on derivation rules are usually given a *closed-world* reading, where relations are considered to be characterized fully by the rules that describe them. For instance, the rules that assign simple types to λ-terms can be used not only to identify types with well-formed terms, but also to argue that a term such as $\lambda x.\, x\ x$ cannot be typed. The HH logic can be used to realize only the positive part of such specifications. To completely formalize the intended meaning of rule-based specifications, we use the logic $\mathcal{G}$ [10] that supports inductive fixed-point definitions.

The basis for $\mathcal{G}$ is also an intuitionistic and predicative version of Church's Simple Theory of Types. Types are determined in $\mathcal{G}$ as in HH except that the type of formulas is `prop` rather than `o`. We assume a fixed collection Σ of logical and non-logical constants none of whose members other than the ones mentioned below contains `prop` in its argument types. The logical constants of $\mathcal{G}$ consist initially of $\top$ and $\bot$ of type `prop`; $\wedge$, $\vee$ and $\supset$ of type $\texttt{prop} \to \texttt{prop} \to \texttt{prop}$; for every type τ not containing `prop`, the quantifiers $\forall_\tau$ and $\exists_\tau$ of type $(\tau \to \texttt{prop}) \to \texttt{prop}$; and the equality symbol $=_\tau$ of type $\tau \to \tau \to \texttt{prop}$. To provide the capability of reasoning about *open* λ-terms, which is necessary in many arguments about HOAS, $\mathcal{G}$ also supports *generic* reasoning. Specifically, for every type τ not containing `prop`, $\mathcal{G}$ includes an infinite set of *nominal constants* of type τ, and a *generic quantifier* ∇_τ of type $(\tau \to \texttt{prop}) \to \texttt{prop}$ [16]. Like with HH, we often omit types and adopt the usual syntactic conventions for displaying the logical connectives.

The proof system for $\mathcal{G}$ is presented as a sequent calculus with sequents of the form $\Xi; \Delta \Vdash C$ where Δ is a set of formulas (*i.e.*, terms of type `prop`), C is a formula, and Ξ contains the free eigenvariables in Δ and C. The treatment of fixed-point definitions in $\mathcal{G}$ results in the eigenvariables being given an extensional interpretation; in other words, unfolding a definition on the left may instantiate some of the eigenvariables and introduce other eigenvariables. We write $\Xi, \Sigma, \mathcal{C} \vdash t : \tau$ to mean that t is a well-formed term of type τ all of whose free variables, constants, and nominal constants are drawn from the respective sets to the left of $\vdash$. Here and elsewhere, we use $\mathcal{C}$ to denote the collection of all nominal constants that we assume to be disjoint from the eigenvariables contained in Ξ and the (logical and non-logical) constants contained in the signature, Σ.

Nominal constants are used to simplify generic judgments in the course of proof search. A correct formalization of this idea needs two provisos: that quantifier scopes be respected and that judgments that differ only in the names of nominal constants be identified. Figure 2 contains a few rules of $\mathcal{G}$ that show how these conditions are realized; the full system can be found in [10]. The essential feature of nominal constants is *equivariance*: two terms B and B' are considered to be equal, written $B \approx B'$, if they are λ-convertible modulo a permutation of the nominal constants. We write $\mathrm{supp}(B)$—called the *support* of B—for the (finite) collection of nominal constants occurring in B. The rules for ∇ are the same on both sides of the sequent; in each case a nominal constant that doesn't already exist in the support of the principal formula is chosen to replace the ∇-quantified variable. In the $\forall_R$ rule of Fig. 2, the eigenvariable is *raised* over the support of the principal formula; this is needed to express permitted dependencies on these nominal constants in a situation where later substitutions for eigenvariables will not be allowed to contain them. Note, however, that nominal constants may be used in witnesses in the $\forall_L$ rule.

To accommodate fixed-point definitions, $\mathcal{G}$ is parameterized by sets of *definitional clauses*. Each such clause has the form $\forall \bar{x}.\, (\nabla \bar{z}.\, A) \triangleq B$ where A is an atomic formula (called the *head*) whose free variables are drawn from $\bar{x}$ and $\bar{z}$, and B is an arbitrary formula (called the *body*) whose free variables are also free in $\nabla \bar{z}.\, A$. Each clause partially defines a relation named by the predicate in the head. In every definitional clause $\forall \bar{x}.\, (\nabla \bar{z}.\, A) \triangleq B$, we require that $\mathrm{supp}(\nabla \bar{z}.\, A)$ and $\mathrm{supp}(B)$ are both empty. Consistency of $\mathcal{G}$ also requires predicate occurrences in the body of a clause to also satisfy certain *stratification* conditions, explained in [10].

$\mathcal{G}$ also includes special rules for interpreting definitional clauses. When an atom occurs on the right of a sequent, then any of the clauses with a matching head may be used to replace the atom by the corresponding body of the clause; in other words, clauses may be backchained. Matching the head of a clause requires some care with regard to the quantifiers. To match the head of a clause $\forall \bar{x}.\, (\nabla \bar{z}.\, A) \triangleq B$ against the atom A', we look for a collection of distinct nominal constants $\bar{c}$ and witness terms $\bar{t}$ that do not contain any of the elements of $\bar{c}$ such that $[\bar{t}/\bar{x}, \bar{c}/\bar{z}]A \approx A'$. If these can be found, then A' is replaced on the right by $[\bar{t}/\bar{x}]B$. When an atom A occurs on the left in a sequent, for every clause and every way of *unifying* the head of the clause to that atom, a new premise is created with the corresponding body added to the context. This amounts to a *case analysis* over the clauses in a definition. Note that substitutions into the clause must respect the order of the $\forall$ and ∇ quantifiers at its head and that different unifiers may result from considering different distinct nominal constant instantiations for the ∇ quantifiers. Some of the eigenvariables may be instantiated in the premises thus created so the eigenvariable context should be modified to reflect the resulting changes.

The final crucial component derived from $\mathcal{G}$ that we use in this paper is the ability to mark certain predicates as being *inductive*, whereby the set of clauses for that predicate is interpreted as a least fixed-point definition. When deriving a sequent of the form

$$\Xi; \Delta \Vdash \forall \bar{x}.\, F_1 \supset \cdots \supset A \supset \cdots \supset F_n \supset G$$

by induction on the atom A, $\mathcal{G}$ produces this premise:

$$\Xi, \bar{x}; \Delta, (\forall \bar{x}.\, F_1 \supset \cdots \supset A^* \supset \cdots \supset F_n \supset G),$$
$$F_1, \ldots, A^{@}, \ldots, F_n \Vdash G$$

Here, A^* and $A^{@}$ are simply annotated versions of A standing for *strictly smaller* and *equal sized* measures respectively. If

$A^{@}$ is unfolded using a definitional clause, the predicates in the body of the corresponding clause are given the * annotation; thus, the inductive hypothesis (containing A^*) only becomes usable after at least one unfolding of $A^{@}$. For each following use of induction, a new set of annotations is produced (*e.g.*, ** and $^{@@}$). This use of annotations is justified by using $\lambda\bar{x}.F_1 \supset \cdots \supset A \supset \cdots \supset F_n \supset G$ as inductive invariant in a more general (and also more abstract) rule that codifies a least fixed-point treatment of the definition of A. A formal development of the connection and a correctness argument can be found in [9].

3.2 Encoding HH in $\mathcal{G}$

The logic $\mathcal{G}$ has the necessary ingredients to represent the HH proof system as an inductive definition. Formally, the type `o` of HH is imported as an *uninterpreted* type in $\mathcal{G}$. Two HH formulas H, G : `o` may be compared only for syntactic equality (or unifiability) in $\mathcal{G}$; in other words, the $\mathcal{G}$ formula $H = G$ does not check for logical equivalence (in HH) of H and G. The connectives of HH are thus treated as *constructors* of `o`, so $(F \Rightarrow G) = (F' \Rightarrow G')$ in $\mathcal{G}$ would entail that $F = F'$ and $G = G'$ because of congruence (*i.e.*, injectivity of constructors), and $(F \Rightarrow G) = G$ would not hold, even if F were known to be derivable in HH, because $F \Rightarrow G$ and G are not unifiable.

To encode HH sequents in $\mathcal{G}$, we first note that $\mathcal{G}$ and HH share the same type system. The HH signature can therefore be imported transparently into $\mathcal{G}$, so the signatures of HH sequents will not be explicitly encoded. The contexts of HH are represented in $\mathcal{G}$ as lists of HH formulas (*i.e.*, lists of terms of type `o`). The type `olist` with constructors `nil` : `olist` and (`::`) : `o` → `olist` → `olist` is used for these lists, and, per tradition, the `::` constructor is written infix. Membership in a context is defined inductively as a predicate `member` : `o` → `olist` → `prop` with these clauses:

$$\texttt{member}\ E\ (E :: L) \triangleq \top$$
$$\texttt{member}\ E\ (F :: L) \triangleq \texttt{member}\ E\ L.$$

Observe that the two clauses have overlapping heads; there will be as many ways to show `member` E L as there are occurrences of E in L. This validates the view of HH contexts as multisets.

The sequents of HH are then encoded in $\mathcal{G}$ using the predicates `seq` and `bch`.

HH	$\mathcal{G}$	notation
$\Theta; \Gamma \vdash G$	`seq` L G	$\{L \vdash G\}$
$\Theta; \Gamma, [F] \vdash A$	`bch` L F A	$\{L, [F] \vdash A\}$

Here, L is an `olist` representation of Γ. The third column contains a convenient and evocative notation for the equivalent $\mathcal{G}$ atom in the second column; we shall often use this notation in the rest of this paper. Note that while the HH contexts are unordered multisets, the `olist` representations are ordered. This is not a limitation because we will always reason about the contexts using `member`.

The static program clauses in Θ are not part of the $\mathcal{G}$ encoding of sequents. Rather, we use the inductively defined predicate `prog` : `o` → `prop` that has one clause of the form `prog` $F \triangleq \top$ for each $F \in \Theta$.

The rules of the HH proof system in Fig. 1 are used to build mutually inductive definitions of the `seq` and `bch` predicates. This definition is depicted in Fig. 3; each clause of the definition corresponds to a single rule of HH. The goal reduction rules are systematically translated into the clauses, the only novelty being that universally quantified variables of the specification logic are represented as nominal constants in $\mathcal{G}$ using the ∇ quantifier. This use of ∇ is necessary because the encoding must completely characterize provability in HH. In particular, in HH the sequent $\cdot; (\Pi x.\, \texttt{eq}\ x\ x) \vdash \Pi y, z.\, \texttt{eq}\ y\ z$ is *not* derivable, meaning that the $\mathcal{G}$ formula `seq` $((\Pi x.\, \texttt{eq}\ x\ x)\ ::\ \texttt{nil})\ ((\Pi y, z.\, \texttt{eq}\ y\ z) \supset \bot)$ should be true. This is achievable since it unfolds to $(\nabla y, z.\, \texttt{seq}\ ((\Pi x.\, \texttt{eq}\ x\ x)\ ::\ \texttt{nil})\ (\texttt{eq}\ y\ z)) \supset \bot$. As a point of comparison, if we were to use this clause instead:

$$\texttt{seq}\ L\ (\Pi_\tau\ G) \triangleq \forall x{:}\tau.\, \texttt{seq}\ L\ (G\ x)$$

then the non-derivability property of the HH sequent above, now encoded as $(\forall y, z.\, \texttt{seq}\ ((\Pi x.\, \texttt{eq}\ x\ x)\texttt{::nil})\ (\texttt{eq}\ y\ z)) \supset \bot$, would not be true. (In particular, the antecedent is satisfiable in models with only a single inhabitant.)

The backchaining rules of HH are encoded as clauses of `bch` in a straightforward manner.

For the structural rules of HH, we have to enforce the invariant that the right hand side of the sequent is atomic. This is achieved by means of a predicate `atomic` : `o` → `prop` defined by the following clause:

$$\begin{aligned}\texttt{atomic}\ F \triangleq\ & (\forall G.\, (F = \Pi_\tau\ G) \supset \bot) \\ & \wedge\ (\forall G_1, G_2.\, (F = (G_1\ \&\ G_2)) \supset \bot) \\ & \wedge\ (\forall G_1, G_2.\, (F = (G_1 \Rightarrow G_2)) \supset \bot).\end{aligned}$$

Effectively, `atomic` characterizes atomic formulas negatively by saying that an atomic formula cannot be constructed with a HH connective. It is important to note that there is a small issue with all three of `seq`, `bch`, and `atomic`: they treat $\Pi_\tau\ G$ as if it were a single object, but, since the reasoning and specification logics share the type system, it actually stands for all instances for the type τ. To keep these definitions finite, we would require polymorphism, which $\mathcal{G}$ currently lacks. In the Abella implementation, therefore, these definitions are treated specially. Note that the meta-theory of $\mathcal{G}$ does not require that inductive definition have finitely many clauses, so even an infinitary interpretation of the clauses of Fig. 3, as was done in [11], is compatible with our approach.

The faithfulness of our encoding allows us to state and prove known properties of HH in $\mathcal{G}$. For example, the meta-theoretic properties discussed in Thm. 1 have the following counterparts relative to the encoding in $\mathcal{G}$. Having proved them in $\mathcal{G}$, we can use the `cut` rule to invoke them as lemmas in arguments concerning particular specifications.

THEOREM 2. *The earlier discussed meta-theoretic properties of* HH *are validated by their encoding in* $\mathcal{G}$. *In other words, each of the following is provable in* $\mathcal{G}$.

1. $\forall L, F, G.\, \{L \vdash F\} \supset \{L, F \vdash G\} \supset \{L \vdash G\}$ *(cut).*
2. $\forall L, G.\, \nabla x.\, \{L\ x \vdash G\ x\} \supset \forall t.\, \{L\ t \vdash G\ t\}$ *(instntiation).*
3. $\forall L, L', G.\, \{L \vdash G\} \supset (\forall F.\, \texttt{member}\ F\ L \supset \texttt{member}\ F\ L') \supset \{L' \vdash G\}$ *(monotonicity).*

PROOF. These are fairly straightforward inductive theorems of $\mathcal{G}$. We have proved them formally in the Abella [1] implementation of $\mathcal{G}$; the proofs can be found in the file `hh_meta.thm`. □

3.3 Example: HOAS vs. De Bruijn Revisited

We are now in a position to formally verify that the relation presented in the introduction between the encodings of the

$$
\begin{array}{lll}
\texttt{seq}\ L\ (G_1\ \&\ G_2) & \triangleq & \texttt{seq}\ L\ G_1 \wedge \texttt{seq}\ L\ G_2 \\
\texttt{seq}\ L\ (F \Rightarrow G) & \triangleq & \texttt{seq}\ (F :: L)\ G \\
\texttt{seq}\ L\ (\Pi_\tau\ G) & \triangleq & \nabla x{:}\tau.\, \texttt{seq}\ L\ (G\ x) \\
\texttt{seq}\ L\ A & \triangleq & \texttt{atomic}\ A \wedge \texttt{member}\ F\ L \wedge \texttt{bch}\ L\ F\ A \\
\texttt{seq}\ L\ A & \triangleq & \texttt{atomic}\ A \wedge \texttt{prog}\ F \wedge \texttt{bch}\ L\ F\ A
\end{array}
\qquad
\begin{array}{lll}
\texttt{bch}\ L\ (F_1\ \&\ F_2)\ A & \triangleq & \texttt{bch}\ L\ F_1\ A \vee \texttt{bch}\ L\ F_2\ A \\
\texttt{bch}\ L\ (G \Rightarrow F)\ A & \triangleq & \texttt{seq}\ L\ G \wedge \texttt{bch}\ L\ F\ A \\
\texttt{bch}\ L\ (\Pi_\tau\ F)\ A & \triangleq & \exists t{:}\tau.\, \texttt{bch}\ L\ (F\ t)\ A \\
\texttt{bch}\ L\ A\ A & \triangleq & \top
\end{array}
$$

Figure 3: Encoding of HH rules as inductive definitions in $\mathcal{G}$.

named and nameless representations of λ-terms actually specifies an isomorphism. We do this by showing that its rendition in HH described in Sec. 2.2 is deterministic in both its first and third arguments. As expected, we work within $\mathcal{G}$ with the encoding of HH described in the previous section. We also assume that the (static) clauses R_{addz}, R_{adds}, R_{app}, and R_{abs} have been reflected into the definition of `prog` in this context.

As mentioned in the introduction, we will need to finitely characterize the possible dynamic context extensions during the derivation of `hodb`. The inductive definition of these dynamic contexts of `hodb` has the following pair of clauses.

$$
\begin{array}{l}
\texttt{ctx}\ \texttt{nil} \triangleq \top \\
\Big(\nabla x.\, \texttt{ctx}\ ((\Pi i, k.\, \texttt{hodb}\ x\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i) :: L)\Big) \triangleq \texttt{ctx}\ L.
\end{array}
$$

As usual, the capitalized variables H and L are universally quantified over the entire clause. Note the occurrence of ∇x at the head of the second clause of the definition: it guarantees that x does not occur in H or L. Therefore, in any L for which $\texttt{ctx}\ L$ holds, it must be the case that there is exactly one such dynamic clause for each such $x \in \text{supp}(L)$. It is easy to establish this fact in terms of a pair of lemmas.

The first of these lemmas characterizes the dynamic clauses.

$$
\begin{array}{l}
\forall L, E.\, \texttt{ctx}\ L \supset \texttt{member}\ E\ L \supset \\
\quad \exists x, H.\, E = (\Pi i, k.\, \texttt{hodb}\ x\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i) \\
\qquad \wedge\, \texttt{name}\ x.
\end{array}
\tag{5}
$$

Here, $\texttt{name}\ x$ is a predicate that asserts that x is a nominal constant; this predicate can be defined in $\mathcal{G}$ with the clause $(\nabla x.\, \texttt{name}\ x) \triangleq \top$. To prove (5), we proceed by induction on the first hypothesis, $\texttt{ctx}\ L$. As mentioned in Sec. 3.1, this is achieved by assuming a new *inductive hypothesis* IH:

$$
\begin{array}{l}
\forall L, E.\, (\texttt{ctx}\ L)^* \supset \texttt{member}\ E\ L \supset \\
\quad \exists x, H.\, E = (\Pi i, k.\, \texttt{hodb}\ x\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i) \\
\qquad \wedge\, \texttt{name}\ x.
\end{array}
\tag{IH}
$$

Moreover, the proof obligation is modified to the following $\mathcal{G}$ sequent, where L and E are promoted to eigenvariables, and the assumptions of the lemma are converted to hypotheses.

$$
\begin{array}{l}
L, E; (\texttt{ctx}\ L)^{@}, \texttt{member}\ E\ L \Vdash \\
\quad \exists x, H.\, E = (\Pi i, k.\, \texttt{hodb}\ x\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i) \wedge \texttt{name}\ x.
\end{array}
$$

The IH cannot be immediately used because the annotations of $\texttt{ctx}\ L$ do not match. To make progress, the definition of $\texttt{ctx}\ L$ needs to be *unfolded*. As explained in Sec. 3.1, this amounts to finding all ways of unifying $\texttt{ctx}\ L$ with the heads of the clauses in the definition of `ctx`. The complete set of unifiers is characterized by $L = \texttt{nil}$ and $L = (\Pi i, k.\, \texttt{hodb}\ \texttt{n}\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i) :: L'$ for new eigenvariables H and L' and a nominal constant `n`. In the latter case we also have a new hypothesis, $(\texttt{ctx}\ L')^*$, that comes from the body of the second clause for `ctx`. There are two things to note: first, the ∇ at the head of the second clause of `ctx` is turned into a nominal constant in the proof obligation, and the second is that the new hypothesis in the second case is annotated with *, which suits the IH.

In each case for L, the argument proceeds by analyzing the second hypothesis, $\texttt{member}\ E\ L$. The case of $L = \texttt{nil}$ is vacuous, because there is no way to infer $\texttt{member}\ E\ \texttt{nil}$, making that hypothesis equivalent to false. In the case of $L = (\Pi i, k.\, \texttt{hodb}\ \texttt{n}\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i) :: L'$, we have two possibilities for $\texttt{member}\ E\ L$: either

$$
E = (\Pi i, k.\, \texttt{hodb}\ \texttt{n}\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i),
$$

or $\texttt{member}\ E\ L'$. The former possibility is exactly the conclusion that we seek, so this branch of the proof finishes. The latter possibility lets us apply IH to the hypotheses $(\texttt{ctx}\ L')^*$ and $\texttt{member}\ E\ L'$, which also yields the desired conclusion.

The second necessary lemma asserts that there is at most a single clause for each variable in the dynamic context.

$$
\begin{array}{l}
\forall L, x,\ H_1, H_2.\, \texttt{ctx}\ L \supset \\
\quad \texttt{member}\ (\Pi i, k.\, \texttt{hodb}\ x\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H_1\ k\ i)\ L \supset \\
\quad \texttt{member}\ (\Pi i, k.\, \texttt{hodb}\ x\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H_2\ k\ i)\ L \supset \\
\quad H_1 = H_2.
\end{array}
\tag{6}
$$

Note that from $H_1 = H_2$, we are able to conclude that the two dynamic clauses relating x to a De Bruijn term must be the same. Like the previous lemma, it is proved by induction on the hypothesis $\texttt{ctx}\ L$.

Armed with these lemmas, we can then show both directions of determinacy for `hodb`. In the forward direction the statement is as follows.

$$
\begin{array}{l}
\forall L, M, H, D, E.\, \texttt{ctx}\ L \supset \\
\qquad \{L \vdash \texttt{hodb}\ M\ H\ D\} \supset \{L \vdash \texttt{hodb}\ M\ H\ E\} \supset D = E.
\end{array}
$$

We prove this by induction on $\{L \vdash \texttt{hodb}\ M\ H\ D\}$; this amounts to assuming the lemma IH below:

$$
\begin{array}{l}
\forall L, M, H, D, E.\, \texttt{ctx}\ L \supset \{L \vdash \texttt{hodb}\ M\ H\ D\}^* \supset \\
\qquad \{L \vdash \texttt{hodb}\ M\ H\ E\} \supset D = E
\end{array}
\tag{IH}
$$

and proving the $\mathcal{G}$ sequent

$$
\begin{array}{l}
L, M, H, D, E; \texttt{ctx}\ L, \{L \vdash \texttt{hodb}\ M\ H\ D\}^{@}, \\
\qquad \{L \vdash \texttt{hodb}\ M\ H\ E\} \Vdash D = E.
\end{array}
$$

Now, $\{L \vdash \texttt{hodb}\ M\ H\ D\}^{@}$ is just a notation for the $\mathcal{G}$ atom $\texttt{seq}\ L\ (\texttt{hodb}\ M\ H\ D)^{@}$ whose definition is given by the clauses in Fig. 3. Unfolding the definition amounts to finding all the clauses in Fig. 3 whose heads match

$$
\texttt{seq}\ L\ (\texttt{hodb}\ M\ H\ D).
$$

Only the final two clauses of `seq`, corresponding to the rules `dyn` and `prog` of HH, are therefore relevant.

Let us consider backchaining the static clauses first, *i.e.*, the applications of the `prog` rule. There are only a finite number

of them, so the assumption $\texttt{prog}\ F$ can be immediately turned into a branched tree with one case for every static program clause. For the first static clause, we are left with a new assumption:

$$\Big\{L, \Big[\Pi M', N', H', D', E'.\, \texttt{hodb}\ (\texttt{app}\ M'\ N')\ H'\ (\texttt{dapp}\ D'\ E') \Leftarrow \texttt{hodb}\ M'\ H'\ D' \,\&\, \texttt{hodb}\ N'\ H'\ E'\Big] \vdash \texttt{hodb}\ M\ H\ D\Big\}^*$$

The annotation * here was obtained from unfolding the definition of a $^@$-annotated atom per the technique outlined in Sec. 3.1. Note that this is just a backchaining sequent (`bch`) whose definition in Fig. 3 can be unfolded. Doing this instantiates the Π prefix in the bakchaining clause in such a way that the head $\texttt{hodb}\ (\texttt{app}\ M'\ N')\ H'\ (\texttt{dapp}\ D'\ E')$ unifies with the HH formula on the right, $\texttt{hodb}\ M\ H\ D$; this produces the substitutions $M = \texttt{app}\ M'\ N'$, $H = H'$, and $D = \texttt{dapp}\ D'\ E'$ for fresh eigenvariables M', N', H', D', E'. Moreover, by the second clause for `bch` in Fig. 3, we get this goal reduction sequent as a fresh hypothesis:

$$\{L \vdash \texttt{hodb}\ M'\ H'\ D' \,\&\, \texttt{hodb}\ N'\ H'\ E'\}^*$$

which is reduced by the first clause for `seq` to:

$$\{L \vdash \texttt{hodb}\ M'\ H'\ D'\}^* \text{ and } \{L \vdash \texttt{hodb}\ N'\ H'\ E'\}^*$$

We can almost apply the induction hypothesis IH—we know $\texttt{ctx}\ L$ and $\{L \vdash \texttt{hodb}\ M'\ H'\ D'\}^*$ already—but we still must find the third argument. To get this argument we need to case analyze the other hypothesis, $\{L \vdash \texttt{hodb}\ M\ H\ E\}$, which becomes $\{L \vdash \texttt{hodb}\ (\texttt{app}\ M'\ N')\ H'\ E\}$ as a result of the previous unification. It has no size annotations because the induction was on the first hypothesis. Nevertheless, we can perform a case analysis of its structure by unfolding its definition (using the clauses in Fig. 3). Once again, we have a choice of using a static program clause or a dynamic clause from L. If we use a static clause, then by a similar argument to the above we will get the following fresh hypotheses, for new eigenvariables D'' and E'' such that $E = \texttt{dapp}\ D''\ E''$:

$$\{L \vdash \texttt{hodb}\ M'\ H'\ D''\} \text{ and } \{L \vdash \texttt{hodb}\ N'\ H'\ E''\}$$

We can now apply the IH twice, yielding $D' = D''$ and $E' = E''$, so $D = \texttt{dapp}\ D'\ E' = \texttt{dapp}\ D''\ E'' = E$.

If, on the other hand, we use a dynamic clause in L, then the two fresh hypotheses we get are:

$$\texttt{member}\ F\ L \text{ and } \{L, [F] \vdash \texttt{hodb}\ (\texttt{app}\ M'\ N')\ H\ E\}.$$

for some new eigenvariable F. This is the first place where the context characterization hypothesis $\texttt{ctx}\ L$ becomes useful. By Lem. (5) above, we should be able to conclude that F is of the form $\big(\Pi i, k.\, \texttt{hodb}\ \texttt{n}\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ \tilde{H}\ k\ i\big)$ for some term $\tilde{H}$ and nominal constant $\texttt{n}$. By looking at the clauses for `bch` in Fig. 3, it is clear that there is no way to prove the sequent $\{L, [F] \vdash \texttt{hodb}\ (\texttt{app}\ M'\ N')\ H\ E\}$, because the term $\texttt{n}$ will never unify with $\texttt{app}\ M'\ N'$. Hence this hypothesis is vacuous, which closes this branch. We have now accounted for all the cases of backchaining a static clause for the inductive assumption $\{L \vdash \texttt{hodb}\ M\ H\ D\}^@$.

This leaves only the dynamic clauses in L—which are backchained using the `dyn` rule—which corresponds to the following pair of new hypotheses:

$$\texttt{member}\ F\ L^* \text{ and } \{L, [F] \vdash \texttt{hodb}\ M\ H\ D\}^*$$

```
Theorem ctx_inv : ∀L, E. ctx L ⊃ member E L ⊃
  ∃x, H. E = (Πi, k. hodb x i (dvar k) ⇐ add H k i) ∧ name x.
Theorem ctx_unique : ∀L, x, H1, H2. ctx L ⊃
  member (Πi, k. hodb x i (dvar K) ⇐ add H1 k i) L ⊃
  member (Πi, k. hodb x i (dvar K) ⇐ add H2 k i) L ⊃ H1 = H2.
Theorem add_det2 : ∀L, X, Y1, Y2, Z. ctx L ⊃
  {L ⊢ add X Y1 Z} ⊃ {L ⊢ add X Y2 Z} ⊃ Y1 = Y2.
Theorem hodb_det3 : ∀L, M, D, E, H. ctx L ⊃
  {L ⊢ hodb M H D} ⊃ {L ⊢ hodb M H E} ⊃ D = E.
induction on 2. intros cH dH eH. dcH:case dH.
  % case of M = app M1 M2
  ecH:case eH.
    apply IH to cH dcH ecH.
    apply IH to cH dcH1 ecH1. search.
    bcH:apply ctx_inv to cH. case bcH. case ecH.
  % case of M = abs M'
  ecH:case eH.
    apply IH to _ dcH ecH. search.
    bcH:apply ctx_inv to cH ecH1. case bcH. case ecH.
  % backchaining on L
  bcH:apply ctx_inv to cH dcH1. aH:case dcH.
    ecH:case eH. case bcH. case bcH.
    uH:apply ctx_inv to cH bH1. case uH.
    bH:case ecH. apply ctx_unique to cH dcH1 ecH1.
    apply add_det2 to cH aH bH search.
```

Figure 4: Abella proof that `hodb` is deterministic in its third argument

As these hypotheses come from unfolding an inductive assumption, they are *-annotated. Once again, we can apply lem. (5) to conclude that

$$F \quad = \quad (\Pi i, k.\, \texttt{hodb}\ \texttt{n}\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ H\ k\ i)$$

where $\texttt{n}$ denotes a nominal constant, We then continue using the definitional clauses for `bch` to get the fresh assumption

$$\{L \vdash \texttt{add}\ H\ k\ i\}^*$$

for new eigenvariables k and i, and the equations $M = \texttt{n}$ and $D = \texttt{dvar}\ k$. Then, since $\texttt{n}$ does not occur in the static clauses Θ, the only way to prove the second hypothesis $\{L \vdash \texttt{hodb}\ \texttt{n}\ H\ E\}$ would be to use a dynamic clause in L. Once again, by lem (5) and unfolding the definition of `bch` as above, we see that this clause must have been of the form $\Pi i, k.\, \texttt{hodb}\ \texttt{n}\ i\ (\texttt{dvar}\ k) \Leftarrow \texttt{add}\ \hat{H}\ k\ i$ for some eigenvariable $\hat{H}$. We can now use the other lemma (6) to show that $H = \hat{H}$. Hence, $\{L \vdash \texttt{hodb}\ \texttt{n}\ H\ E\}$ backchains on the same clause in L as $\{L \vdash \texttt{hodb}\ \texttt{n}\ H\ D\}$, so it must be that $E = \texttt{dvar}\ k$ as well, *i.e.*, $D = E$.

This proof, which has been explained here in some detail, is concisely expressed using the tactics language of Abella [1] as shown in Fig. 4. Induction and case analyses are indicated explicitly using the `induction` and `case` tactics, while lemmas are applied using the `apply` tactic. The `seq` and `bch` definitions are used implicitly by the `case` and `search` tactics; in particular `case` handles reasoning on backchaining sequents. The tactics language of Abella therefore remains unchanged from earlier versions that were designed to support only second-order hereditary Harrop formulas.

The `hodb` relation is also deterministic in its first argument—*i.e.*, given a De Bruijn indexed term, there is at most a single HOAS term it corresponds to—which is proved in a similar fashion. Thus, the `hodb` relation is manifestly an isomorphism

between the two representations of λ-terms. If we were to specify the translations functionally, then we would not only have to repeat the clauses for both directions of the translation, but we would also have to prove separately that they are injective and inverses. We do not sacrifice any of the executable power of a functional specification: the program `hodb` is directly executable in the language λProlog [17].

4. CASE STUDY: RELATING MARKED REDUCTION TO LAMBDA PATHS

The example of the previous section was simple enough that the dynamic context could always be characterized directly by an inductive definition. In the general case, we will need to prove properties about a collection of higher-order relations where each relation has its own separate form of dynamic context. We will therefore need to generalize unary definitions such as `ctx` of the previous section to *context relations* of higher arities. This section contains a case study of such an example, which is independently novel.

The example is drawn from [14, Sec. 7.4.2] and involves a structural characterization of reductions on λ-terms. A *path* through a λ-term is a way to reach any non-binding occurrence of a variable in the term [14, Sec. 4.2]. In HH, we can use a basic type `p` for paths with the following constructors: $\texttt{left}, \texttt{right} : \texttt{p} \to \texttt{p}$ to descend to the function or the argument sub-trees in an application, and $\texttt{bnd} : (\texttt{p} \to \texttt{p}) \to \texttt{p}$ to descend through a λ-abstraction. Crucially, `bnd` has the same binding structure as the λ-abstractions encountered along the path. The predicate $\texttt{path} : \texttt{tm} \to \texttt{path} \to \texttt{o}$ asserts that a given λ-term contains a given path; it is defined by the following three HH clauses.

$$\begin{aligned}
&\texttt{path}\ (\texttt{app}\ M\ N)\ (\texttt{left}\ P) \Leftarrow \texttt{path}\ M\ P.\\
&\texttt{path}\ (\texttt{app}\ M\ N)\ (\texttt{right}\ P) \Leftarrow \texttt{path}\ N\ P.\\
&\texttt{path}\ (\texttt{abs}\ M)\ (\texttt{bnd}\ P) \Leftarrow\\
&\quad \Pi x, p.\, \texttt{path}\ (M\ x)\ (P\ p) \Leftarrow \texttt{path}\ x\ p.
\end{aligned}$$

As these paths record the specific structure of a λ-term, β-reduction changes the paths in the term. On the other hand, a path through *the result of reducing* $\texttt{app}\,(\texttt{abs}\,(\lambda x.\, Mx))\,N$ would be a path through $M\,x$ with the additional proviso that any path through N is also a path through x. Paths are a useful tool for structural characterization of terms. For instance, if two terms have the same paths, then they must be identical; this corresponds to the following theorem of $\mathcal{G}$:

$$\begin{aligned}
&\forall M, N.\\
&\quad (\forall P.\, \{\vdash \texttt{path}\ M\ P\} \supset \{\vdash \texttt{path}\ N\ P\}) \supset \qquad (7)\\
&\quad\quad M = N.
\end{aligned}$$

This theorem is provable in the version of Abella described in [11] that only has the second-order fragment of HH as its specification logic.

Unfortunately, this structural characterization is not preserved by λ-conversion. Suppose we want to compute the paths in a term that result from reducing certain marked β-redexes. Formally, we can add a new constructor for marked redexes, $\texttt{beta} : (\texttt{tm} \to \texttt{tm}) \to \texttt{tm} \to \texttt{tm}$ with the understanding that $\texttt{beta}\,M\,N$ denotes the same λ-term as $\texttt{app}\,(\texttt{abs}\,M)\,N$, except that the redex is marked. We can then define a relation $\texttt{bred} : \texttt{tm} \to \texttt{tm} \to \texttt{o}$ that reduces all the marked β-redexes in a term, with the following clauses.

$$\begin{aligned}
&\texttt{bred}\ (\texttt{app}\ M\ N)\ (\texttt{app}\ U\ V) \Leftarrow \texttt{bred}\ M\ U\ \&\ \texttt{bred}\ N\ V.\\
&\texttt{bred}\ (\texttt{abs}\ M)\ (\texttt{abs}\ U) \Leftarrow\\
&\quad \Pi x.\, \texttt{bred}\ (M\ x)\ (U\ x) \Leftarrow \texttt{bred}\ x\ x.\\
&\texttt{bred}\ (\texttt{beta}\ M\ N)\ V \Leftarrow\\
&\quad \Pi x.\, \texttt{bred}\ (M\ x)\ V \Leftarrow \Pi u.\, \texttt{bred}\ x\ u \Leftarrow \texttt{bred}\ N\ u.
\end{aligned}$$

We also need a static clause for a path in a marked redex.

$$\begin{aligned}
&\texttt{path}\ (\texttt{beta}\ M\ N)\ P \Leftarrow\\
&\quad \Pi x.\, \texttt{path}\ (M\ x)\ P \Leftarrow \Pi q.\, \texttt{path}\ x\ q \Leftarrow \texttt{path}\ N\ q.
\end{aligned}$$

Since different terms can have the same paths as long as they reduce to the same term, the theorem (7) will need to be updated to account for reduction. That is, if two terms have the same paths, then they are *joinable* by `bred`:

$$\begin{aligned}
&\forall M, N, U, V.\\
&\quad (\forall P.\, \{\vdash \texttt{path}\ M\ P\} \supset \{\vdash \texttt{path}\ N\ P\}) \supset \qquad (8)\\
&\quad\quad \{\vdash \texttt{bred}\ M\ U\} \supset \{\vdash \texttt{bred}\ N\ V\} \supset U = V.
\end{aligned}$$

How would one prove (8)? Note that there are two different higher-order predicates: proofs of $\texttt{bred}\,M\,U$ will add dynamic clauses involving `bred`, while proofs of $\texttt{path}\,M\,P$ will add dynamic clauses involving `path`. We would like to prove that `bred` preserves `path`, so the statement of the theorem would have to account for proofs of both kinds, and hence for both kinds of dynamic clauses. The general technique in $\mathcal{G}$ for such situations is to *relate* the two kinds of dynamic contexts for the two different relations. The following definition of $\texttt{ctx2} : \texttt{olist} \to \texttt{olist} \to \texttt{prop}$ achieves this.

$$\begin{aligned}
&\texttt{ctx2}\ \texttt{nil}\ \texttt{nil} \triangleq \top\\
&\Big(\nabla x, p.\, \texttt{ctx2}\ (\texttt{bred}\ x\ x :: K)\ (\texttt{path}\ x\ p :: L)\Big) \triangleq \texttt{ctx2}\ K\ L;\\
&\Big(\nabla x.\, \texttt{ctx2}\ ((\Pi u.\, \texttt{bred}\ N\ u \Rightarrow \texttt{bred}\ x\ u) :: K)\\
&\qquad ((\Pi p.\, \texttt{path}\ N\ p \Rightarrow \texttt{path}\ x\ p) :: L)\Big) \triangleq \texttt{ctx2}\ K\ L.
\end{aligned}$$

It is important to note that the `ctx2` predicate not only says how two such contexts are related, but also contains a specification of the contexts themselves. A hypothesis $\texttt{ctx2}\ K\ L$ where L, say, is not used elsewhere in the theorem is equivalent to assuming just that K is a dynamic context for `bred`. As before, the ∇-bound variables at the head guarantee that every such variable has a unique dynamic clause in both contexts, which we can establish separately using a lemma.

The proof of (8) now proceeds as follows: first we note that if $\texttt{bred}\ M\ N$, then a path in M must also be in N and *vice versa*. Then, we separately show that if $\texttt{bred}\ M\ N$, then it must be that N is free of any subterms involving `beta`. Finally, we prove the lemma that if two `beta`-free terms have the same paths, then they must be identical, which is essentially the same theorem as (7).

Let us consider the first of these lemmas: that `bred` preserves `path`. In the $\mathcal{G}$ encoding of HH, the statement of the theorem is:

$$\begin{aligned}
&\forall K, L, M, U, P.\\
&\quad \texttt{ctx2}\ K\ L \supset \{K \vdash \texttt{bred}\ M\ U\} \supset \qquad (9)\\
&\quad\quad \{L \vdash \texttt{path}\ M\ P\} \supset \{L \vdash \texttt{path}\ U\ P\}.
\end{aligned}$$

This theorem is proved by induction on $\{K \vdash \texttt{bred}\ M\ U\}$. Just as in the inductive proofs in Sec. 3.3, there will be some cases for backchaining static program clauses and some for dynamic clauses. The static cases are fairly straightforward, so we concentrate below on the dynamic cases.

Per the definition in Fig. 3, backchaining a dynamic clause for $\{K \vdash \texttt{bred}\ M\ U\}$ produces the new hypotheses:

$$(\texttt{member}\ E\ K)^* \quad \text{and} \quad \{K, [E] \vdash \texttt{bred}\ M\ U\}^*$$

for some eigenvariable E. From $\texttt{ctx2}\ K\ L$ and $\texttt{member}\ E\ K$, it must follow that:

$$\begin{aligned}&(\exists X.\, (E = \texttt{bred}\ X\ X) \wedge \texttt{name}\ X)\\ &\quad \vee\, (\exists N, X.\, E = (\Pi u.\, \texttt{bred}\ X\ u \Leftarrow \texttt{bred}\ N\ u) \wedge \texttt{name}\ X)\end{aligned}$$

which is itself proven (as a lemma) by induction on the hypothesis $\texttt{ctx2}\ K\ L$. We therefore need to consider only these two cases for the dynamic clause E.

The first case where $E = \texttt{bred}\ X\ X$ is easy to prove. For the second case, we are left with the following problem: although we can characterize the cases for E, this is not enough to reason about `path` because E is a dynamic clause for `bred`. This is where we use the fact that `ctx2` is a relation to prove the following lemma.

$$\begin{aligned}&\forall K, L, N.\, \nabla n.\ \texttt{ctx2}\ (K\ n)\ (L\ n) \supset\\ &\quad \texttt{member}\ (\Pi u.\, \texttt{bred}\ n\ u \Leftarrow \texttt{bred}\ N\ u)\ (K\ n) \supset\\ &\quad\quad \texttt{member}\ (\Pi q.\, \texttt{path}\ n\ q \Leftarrow \texttt{path}\ N\ p)\ (L\ n)\end{aligned} \tag{10}$$

Its proof is by induction on the hypothesis $\texttt{ctx2}\ K\ L$. It can be seen as a kind of translation between the formal relation, given as an inductive definition, to a way of reasoning about the elements of the related contexts. The lemma (10) states, in particular, that a dynamic clause about reduction of marked redexes in the dynamic contexts for `bred` must have a corresponding dynamic clause for paths through a marked redex in the dynamic contexts for `path`.

We now have nearly everything to finish the proof of (9). The only remaining wrinkle is that in the case where the term M is a variable that unifies with a nominal constant n, we will need to look up its dynamic clause in a suitable dynamic context and continue by backchaining it. This amounts to the following *inversion lemma*:

$$\begin{aligned}&\forall K, L, N, P.\, \nabla n.\ \texttt{ctx2}\ (K\ n)\ (L\ n) \supset\\ &\quad \texttt{member}\ (\Pi q.\, \texttt{path}\ n\ q \Leftarrow \texttt{path}\ N\ q)\ (L\ n) \supset\\ &\quad\quad \{(L\ n) \vdash \texttt{path}\ n\ P\} \supset \{(L\ n) \vdash \texttt{path}\ N\ P\}.\end{aligned} \tag{11}$$

Effectively, this lemma says that the only way that $\{(L\ n) \vdash \texttt{path}\ n\ P\}$ could have been proved is by backchaining on the given clause, which has the premise $\{(L\ n) \vdash \texttt{path}\ N\ P\}$. We can show this lemma because we have completely characterized the dynamic context $(L\ n)$, and the static program has no clauses with nominal constants. Note that the nesting order of $\forall$ and ∇ is crucial here: the nominal constant n must not be allowed to occur in N. However, it is obviously allowed to occur in the dynamic context, so we indicate this by means of an explicit dependency, indicated here using the application $(L\ n)$. This punning between the two levels is possibly because HH and $\mathcal{G}$ are both based on a common λ-calculus. The proof of (9) can now be completed by using (11) for the variable case.

The full development of this example in Abella, including the formal proofs, can be found in `examples/hhw/breduce.thm` in the Abella distribution [1].

5. RELATED WORK

The HH proof system presented in Sec. 2 is largely similar to the focused sequent calculus LJF [12] for the fragment of intuitionistic logic containing implication, universal quantification, negatively polarized atoms, and the negatively polarized variant of conjunction. It is also straightforwardly a version of the calculus formalizing *uniform provability* [15]. The term "logic of hereditary Harrop formulas" is often used to indicate an extended logic where disjunction and existential quantification are also allowed in a limited form [14, chap. 3]. Specifications in the full language with these connectives can be compiled into our HH language, possibly with an increase in the number of static clauses in the specifications.

Representational techniques for data with binding can be broadly classified into two styles: first-order and higher-order. Regardless of style, a primary requirement of the representation is that it not distinguish between terms that are α-equivalent. The traditional first-order approach to realizing this requirement is to represent bound variables by De Bruijn indexes, which yields canonical representatives of α-equivalence classes of λ-terms. A very different first-order alternative to De Bruijn indexes is the approach of nominal logic that forgoes canonical representatives of the α-equivalence classes; instead, two terms are considered identical if they are *equivariant*, meaning that the names used in one term can be permuted to the names in the other. This approach is the basis of Nominal Isabelle [4], and there are also a number of libraries for programming with nominal data, such as Fresh OCaml [21] and Alpha Prolog [3, 23].

A drawback with first-order representations, whether of the De Bruijn or the nominal logic kind, is that they typically do not offer support for binding related notions beyond α-equivalence. Typical reasoning applications require a realization of operations such as substitution and analysis of syntactic structure that respects binding. With first-order approaches, these have to be implemented explicitly and the reasoning process must also show their correctness. In particular, the operation of substitution of a term for a free variable, which is at the heart of much of the meta-theory of deductive systems, requires careful book-keeping and fairly detailed correctness arguments (see *e.g.* [20] for a recent example done in Coq). In contrast, higher-order representations reflect binding constructs into the meta-level abstraction operation and thereby absorb arguments about the correctness of binding related operations into a one-time argument, external to the object-level reasoning task, about the correctness of the the meta-language implementation.

Besides Abella, there are three other systems designed to reason about specifications in HOAS: Hybrid [8], Beluga [19], and Twelf [18]. All of these systems are broadly two-level or nested systems, but they make different choices for the specification and reasoning formalisms. Of these, only Hybrid is integrated with popular existing formal reasoning systems (Coq and Isabelle), which allows it to leverage the trusted kernels of the existing systems instead of implementing new trusted components. On the other hand, Hybrid is limited to the second-order hereditary Harrop fragment for the specification level (which makes it similar in this respect to the earlier version of Abella described in [11]) and does not have support for generic reasoning. The second-order restriction is significant when reasoning about higher-order deductive systems: the dynamic clauses of higher-order specifications must be named and transferred to the static program beforehand. For example, the `path` predicate in the second-order fragment requires an auxiliary predicate `jump` and the following clauses

for marked redexes.

$$\texttt{path}\ (\texttt{beta}\ M\ N)\ P \Leftarrow \Pi x.\, \texttt{path}\ (M\ x)\ P \Leftarrow \texttt{jump}\ x\ N.$$

$$\texttt{path}\ X\ P \Leftarrow \texttt{jump}\ X\ N \,\&\, \texttt{path}\ N\ P.$$

Writing such auxiliary predicates is not only error-prone and anti-modular, but they also complicate reasoning about the relations. For instance, in HH it is a direct consequence of cut that $\{L, \texttt{path}\ N\ q \vdash \texttt{path}\ x\ q\}$ and $\{L, (\Pi q.\, \texttt{path}\ x\ q \Leftarrow \texttt{path}\ N\ q) \vdash G\}$ imply $\{L \vdash G\}$. However, for the second-order encoding above, the fact that $\{L, \texttt{path}\ N\ q \vdash \texttt{path}\ x\ q\}$ and $\{L, \texttt{jump}\ x\ N \vdash G\}$ imply $\{L \vdash G\}$ would need a separate inductive proof.

Beluga and Twelf both use the LF dependent type theory for their specification languages. It is known that LF specifications can be systematically and faithfully translated into HH [7, 22]. The encoding of an LF signature in HH uses higher-order features pervasively, and, indeed, was an early motivation for the present work of supporting reasoning over higher-order specifications in Abella. The main difference between LF and HH is their type systems, which directly affects their reasoning principles. Briefly, LF encourages a "combined contexts" reasoning approach, while HH encourages a "context relations" approach. Because LF is dependently typed, the dynamic signature extensions for universally quantified goals cannot be separated from other assumptions; in fact, contexts in LF are interpreted as ordered. It is difficult to place the same LF term in two different contexts.

In both Beluga and Twelf, therefore, the most direct way to reason about different higher-order relations is to use a common dynamic context for the relations. This is achieved formally by specifying contexts *schematically* by means of regular grammars, and using *subordination* analysis on the signature to determine when one regular context may be *subsumed* by another. For example, since there is no way to embed a λ-term value inside a `nat` value using the provided constructors, and the clauses for `add` do not mention λ-terms, it must stand to reason that properties of `add` must hold even in a context of assumptions about λ-terms. Such subsumption properties are often useful; for examples, in the example of Sec. 3.3, if the required properties of `add` are used in a non-empty dynamic context, we must separately prove that earlier theorems still hold, such as the theorem `add_det2` in Fig. 4. In Abella, context definitions are no different from any other inductive definition; there is no automatic subsumption of context relations and such lemmas must be proven manually. On the other hand, reasoning about contexts is not part of the trusted base of Abella, and many properties about arbitrary context relations can be separately proved and used in a modular fashion, as we have done in the examples in Sec. 3.3 and 4.

The differences between the Abella approach and that of Twelf and Beluga taken together can be summarized by the following observations. Firstly, Twelf and Beluga make many kinds of reasoning about context membership, such as (5), automatic and available to the user for free. Explicit reasoning about context members in Abella can be tedious, so it is conceivable that some aspects of the context reasoning of Twelf and Beluga can be imported into Abella in the future. In particular, theorems such as (5), (6), and (10) have entirely predictable proofs that should be easy to automate.

Secondly, the reasoning logic $\mathcal{G}$ has a well-developed proof-theory that includes a sequent calculus with a cut-admissibility result [10]. This logic has a number of features: an equality predicate at all types, generic reasoning, and both inductive and co-inductive fixed-point definitions. Twelf's $\mathcal{M}_2^+$ meta-logic also has a sequent calculus with proof-terms, and the consistency of this logic is proved by giving the proof terms an operational semantics and verifying that they represent total functions under this interpretation. Beluga (as of version 0.5) supports only inductive reasoning in terms of recursive fixed-points, and does not support co-induction. Twelf supports inductive reasoning for Π_0^1 theorems, but also has no support for co-induction. Neither Twelf nor Beluga has a built-in equality predicate. For generic reasoning, Beluga's contextual modal types can achieve many of the same goals as the ∇ quantifier of $\mathcal{G}$, but the global nature of nominal constants and equivariant unification makes it possible to reason about open terms with free variables, unaccompanied by any contexts [2]. Much of the informal meta-theory of the λ-calculus uses open terms in this style, but a first order representation of variables requires an explicit treatment of α-equivalence and substitution. The ∇ quantifier lets us combine the benefits of HOAS and reasoning on open terms.

Finally, the type systems of Twelf and Beluga are endowed with an associated natural induction principle that allows reasoning by induction on the structure of well-typed terms. In Abella, typing is not treated as a definition, so if one wants to induct on the structure of λ-terms, for example, one would have to use a well-formedness predicate $\texttt{is\_tm} : \texttt{tm} \to \texttt{o}$ with the following clauses:

$$\texttt{is\_tm}\ (\texttt{app}\ M\ N) \Leftarrow \texttt{is\_tm}\ M \,\&\, \texttt{is\_tm}\ N.$$

$$\texttt{is\_tm}\ (\texttt{abs}\ M) \Leftarrow \Pi x.\, \texttt{is\_tm}\ (M\ x) \Leftarrow \texttt{is\_tm}\ x.$$

Then, whenever one needs to reason by induction on the structure of a term M, one reasons instead on $\{\vdash \texttt{is\_tm}\ M\}$. Because such predicates essentially reify the well-typedness relation, they will generally need higher-order clauses if the types of the constructors are higher-order. For instance, the `abs` constructor has a second-order type and requires a second-order clause for `is_tm`. Note that such definitions cannot be made in the reasoning logic $\mathcal{G}$ because they are not stratified, *i.e.*, to prove $\texttt{is\_tm}\ M$, one needs to make assumptions of the form $\texttt{is\_tm}\ x$. It is of course possible to automatically generate HH predicates like `is_tm` for a given HH signature, but in any theorem that involves inductive reasoning on the structure of terms one would still need to make hypotheses such as $\{\vdash \texttt{is\_tm}\ M\}$ explicit. Note that $\forall M{:}\texttt{tm}.\, \{\vdash \texttt{is\_tm}\ M\}$ is not a theorem of $\mathcal{G}$.

6. CONCLUSION

We have presented an extension to the two-level logic approach that lets one use the full richness of HH to specify and formally reason about higher-order deductive formalisms. The essence of our method is characterizing the contexts of these higher-order formalisms as inductive relations, and a variant of the backchaining procedure that allows us to use properties of these inductive characterizations in a modular way. We have validated our design and methodology by implementing an extended Abella system and by using it to develop a number of non-trivial examples of reasoning over higher-order specifications.

Acknowledgments: We thank Dale Miller, Olivier Savary-Bélanger and the anonymous reviewers for helpful discussions and comments on earlier drafts. This work has been partially supported by the NSF Grants OISE-1045885 (REUSSI-2) and

CCF-0917140 and by the INRIA Associated Team RAPT. 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.

7. REFERENCES

[1] The Abella prover, 2013. Available at `http://abella-prover.org/`.

[2] B. Accattoli. Proof pearl: Abella formalization of lambda calculus cube property. In C. Hawblitzel and D. Miller, editors, *Second International Conference on Certified Programs and Proofs*, volume 7679 of *LNCS*, pages 173–187. Springer, 2012.

[3] J. Cheney and C. Urban. Alpha-Prolog: A logic programming language with names, binding, and alpha-equivalence. In B. Demoen and V. Lifschitz, editors, *Logic Programming, 20th International Conference*, volume 3132 of *LNCS*, pages 269–283. Springer, 2004.

[4] J. Cheney and C. Urban. Nominal logic programming. *ACM Trans. Program. Lang. Syst.*, 30(5):1–47, 2008.

[5] A. Church. A formulation of the simple theory of types. *J. of Symbolic Logic*, 5:56–68, 1940.

[6] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

[7] A. Felty and D. Miller. Encoding a dependent-type λ-calculus in a logic programming language. In M. Stickel, editor, *Proceedings of the 1990 Conference on Automated Deduction*, volume 449 of *LNAI*, pages 221–235. Springer, 1990.

[8] A. Felty and A. Momigliano. Hybrid: A definitional two-level approach to reasoning with higher-order abstract syntax. *J. of Automated Reasoning*, 48:43–105, 2012.

[9] A. Gacek. *A Framework for Specifying, Prototyping, and Reasoning about Computational Systems*. PhD thesis, University of Minnesota, 2009.

[10] A. Gacek, D. Miller, and G. Nadathur. Nominal abstraction. *Information and Computation*, 209(1):48–73, 2011.

[11] A. Gacek, D. Miller, and G. Nadathur. A two-level logic approach to reasoning about computations. *J. of Automated Reasoning*, 49(2):241–273, 2012.

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

[13] R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Trans. on Computational Logic*, 3(1):80–136, 2002.

[14] D. Miller and G. Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.

[15] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[16] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, Oct. 2005.

[17] G. Nadathur and D. Miller. An Overview of λProlog. In *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Aug. 1988. MIT Press.

[18] F. Pfenning and C. Schürmann. System description: Twelf — A meta-logical framework for deductive systems. In H. Ganzinger, editor, *16th Conf. on Automated Deduction (CADE)*, number 1632 in LNAI, pages 202–206, Trento, 1999. Springer.

[19] B. Pientka and J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In J. Giesl and R. Hähnle, editors, *Fifth International Joint Conference on Automated Reasoning*, number 6173 in LNCS, pages 15–21, 2010.

[20] E. Polonowski. Automatically generated infrastructure for De Bruijn syntaxes. In *Interactive Theorem Proving (ITP)*, volume 7998 of *LNCS*, pages 402–417. Springer, July 2013.

[21] M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. FreshML: Programming with binders made simple. In *Eighth ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden*, pages 263–274. ACM Press, Aug. 2003.

[22] Z. Snow, D. Baelde, and G. Nadathur. A meta-programming approach to realizing dependently typed logic programming. In *ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 187–198, 2010.

[23] C. Urban and J. Cheney. Avoiding equivariance in Alpha-Prolog. In P. Urzyczyn, editor, *Typed Lambda Calculi and Applications, Proceedings*, volume 3461 of *Lecture Notes in Computer Science*, pages 401–416. Springer, 2005.