The Logic of Hereditary Harrop Formulas as a Specification Logic for Hybrid

Chelsea Battell
Department of Mathematics and Statistics
University of Ottawa, Canada
cbattell@uottawa.ca

Amy Felty
School of Electrical Engineering and Computer Science
and Department of Mathematics and Statistics
University of Ottawa, Canada
afelty@uottawa.ca

Abstract

Hybrid is a logical framework that supports the use of higher-order abstract syntax (HOAS) in representing formal systems or "object logics" (OLs). It is implemented in Coq and follows a two-level approach, where a specification logic (SL) is implemented as an inductive type and used to concisely and elegantly encode the inference rules of the formal systems of interest. In this paper, we develop a new higher-order specification logic for Hybrid. By increasing the expressive power of the SL beyond what was considered previously, we increase the flexibility of encoding OLs and thus extend the class of formal systems for which we can reason about efficiently. We focus on formalizing the meta-theory of the SL. We develop an abstract way in which to present an important class of meta-theorems. This class includes properties such as weakening, contraction, exchange, and the admissibility of the cut rule. The cut admissibility theorem establishes consistency and also provides justification for substituting a formula for an assumption in a context of assumptions. It can greatly simplify reasoning about OLs in systems that provide HOAS. We present the abstraction and show how it is used to prove all of these theorems.

Categories and Subject Descriptors F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs; F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic

Keywords logical frameworks, higher-order abstract syntax, cut admissibility, structural rules, interactive theorem proving, Coq

1. Introduction

Logical frameworks provide general languages in which it is possible to represent a wide variety of logics, programming languages, and other formal systems. They are designed to capture uniformities of the syntax and inference systems of these *object logics* (OLs) and to provide support for implementing and reasoning about them. Hybrid (Felty and Momigliano 2012) is a logical framework that provides support for encoding OLs via *higher-order abstract syntax* (HOAS), also referred to as *lambda-tree syntax*. Using HOAS, binding constructs in the OL are encoded using the binding constructs provided by an underlying λ-calculus or function space of the logical framework (the *meta-language*). Using such a representation allows us to delegate to the meta-language α-conversion and capture-avoiding substitution. Further, object logic substitution can be rendered as meta-level β-conversion. HOAS encodings aim to relieve users from having to build up common (and often large) infrastructure implementing operations dealing with variables, such as capture-avoiding substitution, renaming, and fresh name generation. In addition, in such logical frameworks, embedded implication and universal quantification are often used to represent *hypothetical* and *parametric* judgments, also called *generic* judgments, which allow elegant and succinct specifications of OL inference rules.

An important goal of Hybrid is to exploit the advantages of HOAS within the well-understood setting of higher-order logic as implemented by systems such as Isabelle and Coq.[1] Building on such a system allows us to easily experiment with new specification logics. It also provides a high degree of trust; for instance proof terms in Coq serve as proof certificates, which can be independently checked. In addition, Hybrid in Coq inherits Coq's full recursive function space as well as its extensive set of libraries.

Hybrid is implemented as a two-level system, an approach first introduced in the $FO\lambda^{\Delta N}$ logic (McDowell and Miller 2002), and now applied within a variety of logics and systems, such as the Abella interactive theorem prover (Gacek 2008). In a two-level system, the *specification* and (inductive) *meta-reasoning* are done within a single system but at different *levels*. An intermediate level is introduced by inductively defining a *specification logic* (SL) in Coq, and OL judgments (including hypothetical and parametric judgments) are encoded in the SL. Several meta-theoretic properties about the SL provide powerful tools for reasoning about OLs. For example, the cut admissibility theorem provides a direct and convenient way to substitute a formula for an assumption in a context of assumptions. Structural properties of the SL, such as weakening, contraction, and exchange, also provide tools that can be directly applied to reasoning in any OL.

In this paper, we introduce an intuitionistic higher-order SL, namely the logic of hereditary Harrop formulas (HH). HH is a sublogic of the logic of higher-order hereditary Harrop formulas as presented in (Miller and Nadathur 2012). Two kinds of "order" can be seen in this logic, the domain of quantification and the implicational complexity. In terms of the former, HH allows quantification over second-order types, and in terms of the latter, HH is higher-order, allowing any level of nested implications. Previous SLs considered for Hybrid include the fragment of HH with

[1] Although Hybrid has been implemented in both Coq and Isabelle/HOL, we use the Coq version in this paper.

LFMTP '16 June 23, 2016, Porto, Portugal

ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00
DOI: http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

second-order implicational complexity and an ordered linear logic (Felty and Momigliano 2012).

We adopt a minor variation of the inference rules for HH used as an SL in recent versions of the Abella interactive theorem prover (Wang et al. 2013). We present our encoding in Coq, and discuss the proofs of meta-theoretic properties in some detail. In particular, we develop an abstraction to capture uniformities across proofs of different meta-theoretic properties. Cut admissibility in particular relies on a fairly complex inductive argument, involving mutual inductions and sub-inductions. Our proof follows the tradition of many other syntactic proofs of cut admissibility for various logics that first induct on the formula depth and then on the proof structure, e.g. (Girard et al. 1989). Furthermore, our proof is *structural* in the sense of (Pfenning 2000), in our case using structural induction principles generated by Coq. We present the proofs via our abstraction, with the goal of providing a deeper insight into the proofs and the formalization process. Variants of the properties we prove have also been proved in Abella. They are mentioned in (Wang et al. 2013), but proofs are not presented there. We briefly discuss some differences.

Our overall goal is to extend the reasoning power of Hybrid. Implementing HH as a new SL in Hybrid now allows us to directly encode, for example, the two OLs in the case studies considered in (Wang et al. 2013). The first involves reasoning about the correspondence between an HOAS encoding and a de Bruijn representation of the terms of the untyped λ-calculus, while the second involves reasoning about a structural characterization of reductions on untyped λ-terms, and is originally posed in (Miller and Nadathur 2012). Other examples we intend to study include the elegant algorithmic specification of bounded subtype polymorphism in System F in (Pientka 2007), which comes from the *PoplMark* challenge (Aydemir et al. 2005), as well as specifications of continuation-passing transformations in functional programs. The specification of the main judgments of all of these OLs will benefit from the availability of embedded implication in HH (in particular, using two or three levels).

We also note that in addition to the advantages mentioned above with regard to implementing Hybrid inside a well-established theorem prover, Hybrid also provides an ideal setting in which to quickly prototype and experiment with new SLs. Each one is developed as a library and a user can choose and import one that is best-suited to the task at hand and/or move between them easily. For example, case studies that don't require the expressiveness of HH can use the second-order fragment, likely leading to simpler proofs. Case studies that are better suited to a linear logic can directly import and use a linear SL, etc. In contrast, in Abella, a slight extension of HH replaced the SL used in earlier versions of the system. Fixing the SL allows developers to focus more on adding powerful automation for a particular SL, and thus proofs in Hybrid currently require more interaction.

In Section 2, we give a brief introduction of Hybrid. In Section 3, we introduce HH as an example specification layer and describe its implementation in Coq. Highlights of the the mutual structural induction used in later proofs is found in Section 4 followed by the presentation of a generalized SL in Section 5 and a proof technique using this generalized SL in Section 6. Section 7 outlines proofs of the structural rules of HH, while Section 8 details the proof of cut admissibility. Finally, Section 9 concludes and discusses related and future work. The files of the Coq formalization are available at `www.eecs.uottawa.ca/~afelty/lfmtp16/`.

2. Hybrid

The Calculus of Inductive Constructions (CIC) (Paulin-Mohring 1993; Bertot and Castéran 2004) as implemented by Coq is the *reasoning logic* (RL) of Hybrid. Both SLs and OLs are implemented as inductive types in Coq, and Coq is used to carry out all formal reasoning. Hybrid is implemented as a Coq library. This library first introduces a special type `expr` that encodes a de Bruijn representation of λ-terms. It is defined inductively and its definition appears in Figure 1. Here, `VAR` and `BND` represent bound and free variables, respectively, and `var` and `bnd` are defined to be the natural numbers. The type `con` is a parameter to be filled in when defining the constants used to represent an OL. The library then includes a series of definitions used to define the operator `LAM` of type $(\texttt{expr} \to \texttt{expr}) \to \texttt{expr}$, which provides the capability to express OL syntax using HOAS. Expanding its definition fully down to primitives gives the low-level de Bruijn representation, which is hidden from the user when reasoning about meta-theory. In fact, the user only needs `CON`, `VAR`, `APP`, and `LAM` to define operators for OL syntax. One other predicate from the Hybrid library will appear in the proof development: $\texttt{proper} : \texttt{expr} \to \texttt{Prop}$. This predicate rules out terms that have occurrences of bound variables that do not have a corresponding binder (*dangling indices*).

```
Inductive expr : Set :=
| CON : con -> expr
| VAR : var -> expr
| BND : bnd -> expr
| APP : expr -> expr -> expr
| ABS : expr -> expr.
```

Figure 1. Terms in Hybrid

```
Inductive con : Set :=
| hAPP : con | hABS : con
| dAPP : con | dABS : con | dVAR : nat -> con.

Definition hApp : expr -> expr -> expr :=
 fun (t1 : expr) => fun (t2 : expr) =>
  APP (APP (CON hAPP) t1) t2.
Definition hAbs : (expr -> expr) -> expr :=
 fun (f : expr -> expr) =>
  APP (CON hABS) (LAM f).

Definition dApp : expr -> expr -> expr :=
 fun (t1 : expr) => fun (t2 : expr) =>
  APP (APP (CON dAPP) t1) t2.
Definition dAbs : expr -> expr :=
 fun (d : expr) => APP (CON cdABS) d.
Definition dVar : nat -> expr :=
 fun (n : nat) => (CON (dVAR n)).
```

Figure 2. Encoding OL Syntax in Hybrid

To illustrate the encoding of OL syntax, we consider the example mentioned earlier of reasoning about the correspondence between a HOAS encoding and a de Bruijn representation of the terms of the untyped λ-calculus.[2] We fill in the definition of `con`, and define operators `hApp` and `hAbs` for the HOAS encoding and operators of the untyped λ-calculus, and `dApp`, `dAbs`, and `dVar` for object-level de Bruijn terms. Figure 2 defines this encoding. As mentioned, the type `con` is actually a parameter in the Hybrid library. This will be explicit when discussing Coq proofs, where we write $(\texttt{expr con})$ as the type used to express OL terms. Also, the keyword `fun` is Coq's abstraction operator. In this paper, we will sometimes use the standard notation from the λ-calculus for function abstraction. For example, the defi-

[2] In (Wang et al. 2013), inference rules and proofs of properties of this OL are also discussed.

```
Inductive oo : Type :=
| atom : atm -> oo
| T : oo
| Conj : oo -> oo -> oo
| Imp : oo -> oo -> oo
| All : (expr con -> oo) -> oo
| Allx : (X -> oo) -> oo
| Some : (expr con -> oo) -> oo.
```

Figure 3. Type of SL Formulas

nition of hApp can also be written as λ(t1 : expr con)(t2 : expr con) . APP (APP (CON hAPP) t1) t2.

3. The Specification Logic

At the specification level, the terms of HH are the terms of the simply-typed λ-calculus. We assume a set of primitive types that includes expr as well as the special symbol oo to denote formulas. Types are built from the primitive types and the function arrow $\rightarrow$ as usual. Logical connectives and quantifiers are introduced as constants with their corresponding types as in (Church 1940). For example, conjunction has type oo $\rightarrow$ oo $\rightarrow$ oo and the quantifiers have type $(\tau \rightarrow \texttt{oo}) \rightarrow \texttt{oo}$, with some restrictions on τ described below. Predicates are function symbols whose target type is oo. Following (Miller and Nadathur 2012), the grammars below for G (goals) and D (clauses) define the formulas of the logic, while Γ describes contexts of hypotheses.

$$\begin{array}{lcl} G & ::= & \top \mid A \mid G \,\&\, G \mid G \vee G \mid D \longrightarrow G \mid \forall_\tau x.G \mid \exists_\tau x.G \\ D & ::= & A \mid G \longrightarrow D \mid D \,\&\, D \mid \forall_\tau x.D \\ \Gamma & ::= & \emptyset \mid \Gamma, D \end{array}$$

In goal formulas, we restrict τ to be a primitive type not containing oo. In clauses, τ also cannot contain oo, and is either primitive or has the form $\tau_1 \rightarrow \tau_2$ where both τ_1 and τ_2 are primitive types. We note that there is no restriction on the implicational complexity.[3] We follow the presentation in (Wang et al. 2013) to define the inference rules of HH, using two sequent judgments that distinguish between *goal-reduction rules* and *backchaining rules*. These sequents have the forms $\Gamma \rhd G$ and $\Gamma, [D] \rhd A$, respectively, where the latter is a left focusing judgment with D the formula under (left) focus. In these sequents, there is also an implicit fixed context Δ, called the *static program clauses*, containing closed clauses of the form $\forall_{\tau_1} \ldots \forall_{\tau_n}.G \longrightarrow A$ with $n \geq 0$. These clauses represent the inference rules of an OL.

Our encoding of the formulas of the SL in Coq is shown in Figure 3. In this implementation, the type atm is a parameter to the definition of oo and is used to define the predicates needed for reasoning about a particular OL. For instance, our above example might include a predicate hodb : (expr con) $\rightarrow$ nat $\rightarrow$ (expr con) relating the higher-order and de Bruijn encodings at a given depth. The constant atom coerces an atomic formula (a predicate applied to its arguments) to an SL formula. Also, note that in this implementation, we restrict the type of universal quantification to two types, (expr con) and X, where X is a parameter that can be instantiated with any primitive type; in our running example, X would become nat for the depth of binding in a de Bruijn term. We also leave out disjunction. It is not difficult to extend our implementation to include disjunction and quantification (universal or existential) over other primitive types, but these have not been needed in reasoning about OLs.

We write $\langle\ \alpha\ \rangle$, $(\beta_1\ \&\ \beta_2)$, and $(\beta_1 \longrightarrow \beta_2)$ as notation for (atom α), (Conj β_1 β_2), and (Imp β_1 β_2), respectively. Note that

[3] In the second-order SL in (Felty and Momigliano 2012), implication in goals is restricted to the form $A \longrightarrow G$.

$$\frac{A :\!- \ G \quad \Gamma \rhd G}{\Gamma \rhd \langle\ A\ \rangle}\ g\_prog \qquad \frac{D \in \Gamma \quad \Gamma, [D] \rhd A}{\Gamma \rhd \langle\ A\ \rangle}\ g\_dyn$$

$$\frac{\Gamma \rhd G_1 \quad \Gamma \rhd G_2}{\Gamma \rhd G_1 \,\&\, G_2}\ g\_and \qquad \frac{\Gamma\ ,\ D \rhd G}{\Gamma \rhd D \longrightarrow G}\ g\_imp$$

$$\frac{}{\Gamma \rhd \texttt{T}}\ g\_tt \qquad \frac{\forall(E : \texttt{expr con}), (\texttt{proper}\ E \rightarrow \Gamma \rhd G\,E)}{\Gamma \rhd \texttt{All}\ G}\ g\_all$$

$$\frac{\forall(E : \texttt{X}), (\Gamma \rhd G\,E)}{\Gamma \rhd \texttt{Allx}\ G}\ g\_allx \qquad \frac{\texttt{proper}\ E \quad \Gamma \rhd G\,E}{\Gamma \rhd \texttt{Some}\ G}\ g\_some$$

Figure 4. Goal-Reduction Rules, grseq

$$\frac{}{\Gamma, [\langle\ A\ \rangle] \rhd A}\ b\_match \qquad \frac{\Gamma, [D_1] \rhd A}{\Gamma, [D_1 \,\&\, D_2] \rhd A}\ b\_and_1$$

$$\frac{\Gamma, [D_2] \rhd A}{\Gamma, [D_1 \,\&\, D_2] \rhd A}\ b\_and_2 \qquad \frac{\Gamma \rhd G \quad \Gamma, [D] \rhd A}{\Gamma, [G \longrightarrow D] \rhd A}\ b\_imp$$

$$\frac{\texttt{proper}\ E \quad \Gamma, [D\,E] \rhd A}{\Gamma, [\texttt{All}\ D] \rhd A}\ b\_all \qquad \frac{\Gamma, [D\,E] \rhd A}{\Gamma, [\texttt{Allx}\ D] \rhd A}\ b\_allx$$

$$\frac{\forall(E : \texttt{expr con}), (\texttt{proper}\ E \rightarrow \Gamma, [D\,E] \rhd A)}{\Gamma, [\texttt{Some}\ D] \rhd A}\ b\_some$$

Figure 5. Backchaining Rules, bcseq

we write β or δ for formulas (type oo), and α for elements of type atm, possibly with subscripts. When discussing proofs, we also write o for formulas and a for atoms. When we want to make explicit when a formula is a goal or clause, we write G or D, respectively. Formulas quantified by All are written (All β) or (All $\lambda(x$: expr con) . $\beta x)$. The latter is the η-long form with types included explicitly. The other quantifiers are treated similarly.

Figures 4 and 5 define the inference rules of the SL. They are encoded in Coq as two mutually inductive types, one each for goal-reduction and backchaining sequents. The syntax used in the figures is a pretty-printed version of the Coq definition. Coq's dependent products are written $\forall(x_1 : t_1) \cdots (x_n : t_n), M$, where $n \geq 0$ and for $i = 1, \ldots, n$, x_i may appear free in $x_{i+1}, \ldots, x_n, M$. If it doesn't, implication can be used as an abbreviation, e.g., the premise of the *g_all* rule is an abbreviation for $\forall(E : \texttt{expr con})(H : \texttt{proper}\ E), (\Gamma \rhd G\,E)$.

Goal-reduction sequents have type grseq : context $\rightarrow$ oo $\rightarrow$ Prop, and we write $\Gamma \rhd \beta$ as notation for grseq Γ β. Backchaining sequents have type bcseq : context $\rightarrow$ oo $\rightarrow$ atm $\rightarrow$ Prop and we write $\Gamma, [\beta] \rhd \alpha$ as notation for bcseq Γ β α, understanding β to be the focused formula from Γ. The rule names in the figures are the constructor names in the inductive definitions. The premises and conclusion of a rule are the argument types and the target type, respectively, of one clause in the definition. Quantification at the outer level is implicit. Inner quantification is written explicitly in the premises. For example, the linear format of the *g_dyn* rule from Figure 4 with all quantifiers explicit is $\forall(\Gamma : \texttt{context})(D : \texttt{oo})(A : \texttt{atm}), D \in \Gamma \rightarrow \Gamma, [D] \rhd A \rightarrow \Gamma \rhd \langle\ A\ \rangle$. This is the type of the *g_dyn* constructor in the inductive definition of grseq. (See the definition of grseq in the Coq files.)

The type `context` is used to represent contexts of assumptions in sequents and is defined as a Coq `ensemble oo` since we want contexts to behave as sets. We write (Γ, β) as notation for (`context_cons` Γ β). We write write c or Γ to denote contexts when discussing formalized proofs. The following context lemmas will be mentioned in the proofs in this paper:

Lemma 1 (`elem_inv`). $$\frac{\texttt{elem}\ \beta_1\ (\Gamma, \beta_2)}{(\texttt{elem}\ \beta_1\ \Gamma) \vee (\beta_1 = \beta_2)}$$

Lemma 2 (`context_sub_sup`). $$\frac{\Gamma_1 \subseteq \Gamma_2}{(\Gamma_1, \beta) \subseteq (\Gamma_2, \beta)}$$

We use the `ensemble` axiom `Extensionality_Ensembles` : $\forall(E_1\ E_2 : \texttt{ensemble}), (\texttt{Same\_set}\ E_1\ E_2) \rightarrow E_1 = E_2$, where `Same_set` is defined in the `Ensemble` library, and `elem` is `context` membership.

The goal-reduction rules are the right-introduction rules of this sequent calculus. If we consider building a proof bottom-up from the root, these rules "reduce" the formula on the right to atomic formulas. The rules *g_prog* and *g_dyn* are the only goal-reduction rules with an atomic principal formula.

The rule *g_prog* is used to backchain over the static program clauses Δ, which are defined for each new OL as an inductive type called `prog` of type `atm` $\rightarrow$ `oo` $\rightarrow$ `Prop`, and represent the inference rules of the OL. This rule is the interface between the SL and OL layers and we say that the SL is parametric in OL provability. We write $A :- G$ for (`prog` A G) to suggest backward implication. Recall that clauses in Δ may have outermost universal quantification. The premise $A :- G$ actually represents an instance of a clause in Δ.

The rule *g_dyn* allows backchaining over dynamic assumptions (i.e. a formula from Γ). To use this rule to prove $\Gamma \rhd \langle\ A\ \rangle$, we need to show $D \in \Gamma$ and $\Gamma, [D] \rhd A$. Formula D is chosen from, or shown to be in, the dynamic context Γ and we use the backchaining rules to show $\Gamma, [D] \rhd A$ (where D is the focused formula).

The backchaining rules are the standard focused left rules for conjunction, implication, and universal and existential quantification. Considered bottom up, they provide backchaining over the focused formula. In using the backchaining rules, each branch is either completed by *b_match* where the focused formula is an atomic formula identical to the goal of the sequent, or *b_imp* is used resulting in one branch switching back to using goal-reduction rules.

We mention several Coq tactics when presenting proofs. The main one is the `constructor` tactic, which applies a clause of an inductive definition in a backward direction (a step of meta-level backchaining), determining automatically which clause to apply.

4. Mutual Structural Induction

Our theorem statements will often have the form

$$(\forall\ (c : \texttt{context})\ (o : \texttt{oo}), (c \rhd o) \rightarrow (P_1\ c\ o))\ \wedge$$
$$(\forall\ (c : \texttt{context})\ (o : \texttt{oo})\ (a : \texttt{atm}), (c, [o] \rhd a) \rightarrow (P_2\ c\ o\ a))$$

where we extract predicates P_1 : `context` $\rightarrow$ `oo` $\rightarrow$ `Prop` and P_2 : `context` $\rightarrow$ `oo` $\rightarrow$ `atm` $\rightarrow$ `Prop` from the statement to be proven. We can generate an induction principle over the mutually inductive sequent types to allow proof by mutual structural induction. This is done using the Coq `Scheme` command.

To prove a statement of the above form by mutual structural induction over $c \rhd o$ and $c, [o] \rhd a$, 15 subcases must be proven, one corresponding to each inference rule of the SL. The proof state of each subcase of this induction is constructed from an inference rule of the system. We can see a snippet of the sequent mutual induction principle in Figure 6, where each antecedent (clause of the induction principle defining the cases) corresponds to a rule of the SL and a subcase for an induction using this

$$\begin{aligned}
&\texttt{seq_mutind} : \forall(P_1 : \texttt{context} \rightarrow \texttt{oo} \rightarrow \texttt{Prop})\\
&\qquad (P_2 : \texttt{context} \rightarrow \texttt{oo} \rightarrow \texttt{atm} \rightarrow \texttt{Prop}),\\
&(*g_dyn*)\quad (\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}),\\
&\qquad o \in c \rightarrow c, [o] \rhd a \rightarrow P_2\ c\ o\ a \rightarrow P_1\ c\ \langle\ a\ \rangle) \rightarrow\\
&(*g_all*)\quad (\forall(c : \texttt{context})(o : \texttt{expr con} \rightarrow \texttt{oo}),\\
&\qquad (\forall(e : \texttt{expr con}), \texttt{proper}\ e \rightarrow c \rhd o\ e \rightarrow\\
&\qquad (\forall(e : \texttt{expr con}), \texttt{proper}\ e \rightarrow P_1\ c\ (o\ e) \rightarrow\\
&\qquad P_1\ c\ (\texttt{All}\ o)) \rightarrow\\
&(*b_imp*)\quad (\forall(c : \texttt{context})(o_1\ o_2 : \texttt{oo})(a : \texttt{atm}),\\
&\qquad c \rhd o_1 \rightarrow P_1\ c\ o_1 \rightarrow c, [o_2] \rhd a \rightarrow P_2\ c\ o_2\ a \rightarrow\\
&\qquad P_2\ c\ (o_1 \longrightarrow o_2)\ a) \rightarrow\\
&\quad \ldots\\
&\quad (\forall(c :\ \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow P_1\ c\ o) \wedge\\
&\quad (\forall(c :\ \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow P_2\ c\ o\ a)
\end{aligned}$$

Figure 6. Sequent Mutual Induction Principle Snippet

technique. After applying the induction principle, the subcases are generated and externally quantified variables in each antecedent are introduced to the context of assumptions of the proof state and are then considered *signature variables*.

This induction principle is automatically generated following the description shown below, with examples from the figure given in each point.

- Non-sequent premises are assumptions of the induction subcase (e.g. $o \in c$ from the *g_dyn* rule).
- For every rule premise that is a goal-reduction sequent (with possible local quantifiers) of the form $\forall(x_1 : T_1) \cdots (x_n : T_n), \Gamma \rhd \beta$ where $n \geq 0$, the induction subcase has assumptions $(\forall(x_1 : T_1) \cdots (x_n : T_n), \Gamma \rhd \beta)$ and $(\forall(x_1 : T_1) \cdots (x_n : T_n), P_1\ \Gamma\ \beta)$ (e.g. $\forall(e : \texttt{expr con}), \texttt{proper}\ e \rightarrow c \rhd o\ e$ and $\forall(e : \texttt{expr con}), \texttt{proper}\ e \rightarrow P_1\ c\ (o\ e)$ from the *g_all* rule with $n = 2$ and unabbreviated prefix $\forall(e : \texttt{expr con})(H : \texttt{proper}\ e)$).
- For every rule premise that is a backchaining sequent (with possible local quantifiers) of the form $\forall(x_1 : T_1) \cdots (x_n : T_n), \Gamma, [\beta] \rhd \alpha$ where $n \geq 0$, the induction subcase has assumptions $(\forall(x_1 : T_1) \cdots (x_n : T_n), \Gamma, [\beta] \rhd \alpha)$ and $(\forall(x_1 : T_1) \cdots (x_n : T_n), P_2\ \Gamma\ \beta\ \alpha)$ (e.g. $c, [o_2] \rhd a$ and $(P_2\ c\ o_2\ a)$ from the *b_imp* rule).
- If the rule conclusion is a goal-reduction sequent of the form $\Gamma \rhd \beta$, then the subcase goal is $P_1\ \Gamma\ \beta$ (e.g. $(P_1\ c\ \langle\ a\ \rangle)$ from the *g_dyn* rule).
- If the rule conclusion is a backchaining sequent of the form $\Gamma, [\beta] \rhd \alpha$, then the subcase goal is $P_2\ \Gamma\ \beta\ \alpha$ (e.g. $(P_2\ c\ (o_1 \longrightarrow o_2)\ a)$ from the *b_imp* rule).

Implicit in these last two points is the possible introduction of more assumptions, in the case when P_1 and P_2 are dependent products themselves (i.e. contain quantification and/or implication). We will refer to assumptions introduced this way as *induction assumptions* in future proofs, since they are from a predicate that is used to construct induction hypotheses. That is, assumptions of the form $(P_1\ \Gamma\ \beta)$ or $(P_2\ \Gamma\ \beta\ \alpha)$ are induction hypotheses for any proof subcase for a rule with premises $\Gamma \rhd \beta$ or $\Gamma, [\beta] \rhd \alpha$. In this SL, exactly two cases of this induction principle have more than one induction hypothesis (*b_imp* and *g_and*).

In describing proofs, we will follow the Coq style and write the proof state in a vertical format with the assumptions above a horizontal line and the goal below it. For example, the *g_dyn* subcase will have the following form:

$$\frac{H_1 : o \in c \quad H_2 : c, [o] \rhd a \quad IH : P_2\ c\ o\ a}{P_1\ c\ \langle\ a\ \rangle}$$

As in Coq, we provide hypothesis names (so that we can refer to them as needed). Also, we often omit the type declarations of signature variables, in this case c : `context`, o : `oo`, and a : `atm`, when they can be easily inferred from context. Unlike in Coq, when we have multiple subcases to prove with the same context of assumptions we will write them all under the horizontal line in the same proof state, separated by commas.

5. Generalized SL Part I: Abstract Rules

Here we present generalized specification logic rules to reduce the number of induction cases and allow us to partition cases of the original SL based on rule structure. Our goal is to gain insight into the high-level structure of such inductive proofs, providing the proof writer and reader with the ability to understand where the difficult cases are and how similar cases can be handled in a general way.

All rules of the SL have some number of premises that are either non-sequent predicates, goal-reduction sequents, or backchaining sequents. Also, all rule conclusions are sequents; this is necessary to encode these rules in inductive types `grseq` and `bcseq`. With this observation, we can generalize the rules of the SL inference system and say that all rules have one of the following forms:

$$\frac{\begin{array}{c}\overline{Q_m}\langle\!\langle c, o\rangle\!\rangle \\ \forall\overline{(x_{n,s_n} : R_{n,s_n})}, (c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle) \\ \forall\overline{(y_{p,t_p} : S_{p,t_p})}, (c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p})\end{array}}{c \rhd o}\ \textit{gr\_rule}$$

$$\frac{\begin{array}{c}\overline{Q_m}\langle\!\langle c, o\rangle\!\rangle \\ \forall\overline{(x_{n,s_n} : R_{n,s_n})}, (c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle) \\ \forall\overline{(y_{p,t_p} : S_{p,t_p})}, (c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p})\end{array}}{c, [o] \rhd a}\ \textit{bc\_rule}$$

where m, n, p represent the (possibly zero) number of non-sequent premises, goal-reduction sequent premises, and backchaining sequent premises, respectively. Note that for all rules in our implemented SL, $0 \le m \le 1$, $0 \le n \le 2$, and $0 \le p \le 1$.

We call this collection of inference rules consisting of *gr_rule* and *bc_rule* the generalized specification logic (GSL). This is *not* implemented in Coq as the previously described SL is; but rather all rules of the SL can be instantiated from the two rules of the GSL (see Subsection 5.1). The GSL allows us to investigate the SL without needing to consider each of the 15 rules of the SL separately. This makes it possible to more efficiently study and explain the metatheory of the SL.

Much of the notation used in these rules requires further explanation. A horizontal bar above an element with some subscript index, say z, means we have a collection of such items indexed from 1 to z. For example, the "premise" $\overline{Q_m}\langle\!\langle c, o\rangle\!\rangle$ represents the m premises $Q_1\langle\!\langle c, o\rangle\!\rangle, \ldots, Q_m\langle\!\langle c, o\rangle\!\rangle$. The premises with sequents can possibly have local quantification. For $i = 1, \ldots, n$, $\overline{(x_{i,s_i} : R_{i,s_i})}$ represents the prefix $(x_{i,1} : R_{i,1}) \cdots (x_{i,s_i} : R_{i,s_i})$.

The notation $\langle\!\langle\cdot\rangle\!\rangle$ is used to list arguments from the conclusion that may be used by a function or predicate. We wish to show how elements of the rule conclusion propagate through a proof.

Given types $T_0, T_1, \ldots, T_z$, when we write $F\langle\!\langle a_1 : T_1, \ldots, a_z : T_z\rangle\!\rangle : T_0$, we mean a term of type T_0 that may contain any (sub)terms appearing in conclusion terms $a_1, \ldots, a_z$. For example, given $\gamma_1\langle\!\langle D \longrightarrow G : \texttt{oo}\rangle\!\rangle$: `context`, we may "instantiate" this expression to $\{D\}$. We often omit types and use definitional notation, e.g., in this case we may write $\gamma_1\langle\!\langle D \longrightarrow G\rangle\!\rangle \coloneqq \{D\}$.

We infer the following typing judgments from the GSL rules:

- For $i = 1, \ldots, m$, the definition of Q_i may use the context and formula of the conclusion, so with full typing information, $Q_i\langle\!\langle c : \texttt{context}, o : \texttt{oo}\rangle\!\rangle$: `Prop`
- For $j = 1, \ldots, n$, SL context γ_j may use the formula of the conclusion and SL formula F_j may use the formula of the conclusion and locally quantified variables. So with full typing information, $\gamma_j\langle\!\langle o : \texttt{oo}\rangle\!\rangle$: `context` and $F_j\langle\!\langle o : \texttt{oo}, x_{j,1} : R_{j,1}, \ldots, x_{j,s_j} : R_{j,s_j}\rangle\!\rangle$: `oo`
- For $k = 1, \ldots, p$, SL context γ'_k may use the formula of the conclusion and SL formula F'_k may use the formula of the conclusion and locally quantified variables. So with full typing information $\gamma'_k\langle\!\langle o : \texttt{oo}\rangle\!\rangle$: `context` and $F'_k\langle\!\langle o : \texttt{oo}, y_{k,1} : S_{k,1}, \ldots, y_{k,t_k} : S_{k,t_k}\rangle\!\rangle$: `oo`

5.1 SL Rules from GSL Rules

The rules of the GSL can be instantiated to obtain the SL by specifying the values of the variables in the GSL rules. We first fill in m, n, and p. Then for $i = 1, \ldots, m$, we specify Q_i. For $j = 1, \ldots, n$, we specify s_j, γ_j, F_j, x_{j,s_j}, and R_{j,s_j}. For $k = 1, \ldots, p$, we specify γ'_k, F'_k, y_{k,t_k}, and S_{k,t_k}. Below are examples for SL rules *g_dyn*, *g_allx* and *b_imp*.

Rule	m	n	p	c	o
$\dfrac{D \in \Gamma \quad \Gamma, [D] \rhd A}{\Gamma \rhd \langle\ A\ \rangle}$ *g_dyn*	1	0	1	Γ	$\langle\ A\ \rangle$
$t_1 \coloneqq 0$ $Q_1\langle\!\langle \Gamma, \langle\ A\ \rangle\rangle\!\rangle \coloneqq D \in \Gamma \quad \gamma'_1\langle\!\langle \langle\ A\ \rangle\rangle\!\rangle \coloneqq \emptyset \quad F'_1\langle\!\langle \langle\ A\ \rangle\rangle\!\rangle \coloneqq D$					
$\dfrac{\forall (E : \texttt{X}), (\Gamma \rhd G\ E)}{\Gamma \rhd \texttt{Allx}\ G}$ *g_allx*	0	1	0	Γ	$\texttt{Allx}\ G$
$s_1 \coloneqq 1 \quad x_{1,1} \coloneqq E \quad R_{1,1} \coloneqq \texttt{X}$ $\gamma_1\langle\!\langle \texttt{Allx}\ G\rangle\!\rangle \coloneqq \emptyset \quad F_1\langle\!\langle \texttt{Allx}\ G, E\rangle\!\rangle \coloneqq G\ E$					
$\dfrac{\Gamma \rhd G \quad \Gamma, [D] \rhd A}{\Gamma, [G \longrightarrow D] \rhd A}$ *b_imp*	0	1	1	Γ	$G \longrightarrow D$
$s_1 \coloneqq 0 \quad t_1 \coloneqq 0$ $\gamma_1\langle\!\langle G \longrightarrow D\rangle\!\rangle \coloneqq \emptyset \quad F_1\langle\!\langle G \longrightarrow D\rangle\!\rangle \coloneqq G$ $\gamma'_1\langle\!\langle G \longrightarrow D\rangle\!\rangle \coloneqq \emptyset \quad F'_1\langle\!\langle G \longrightarrow D\rangle\!\rangle \coloneqq D$					

Notice that for the *g_dyn* rule, D appears in Q_1, even though it is not in the argument list of Q_1. The notation $\langle\!\langle\cdot\rangle\!\rangle$ only specifies arguments from the rule conclusion. Any variables that only appear in the premises of a rule of the SL are also permitted to appear in the propositions, formulas, and contexts when specializing the premises of a GSL rule to obtain the premises of a specific SL rule.

6. Proof by Induction over the Generalized Rules

The induction subcase corresponding to *gr_rule* (resp. *bc_rule*) requires a proof of:

$$\overline{H_m} : \overline{Q_m}\langle\!\langle c, o\rangle\!\rangle$$
$$\overline{Hg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}, (c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle)$$
$$\overline{IHg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}, P_1\ (c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle)\ (\overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle)$$
$$\overline{Hb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}, (c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p})$$
$$\overline{IHb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}, P_2\ (c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle)\ (\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle)\ \overline{a_p}$$

$$P_1\ c\ o\ (resp.\ P_2\ c\ o\ a)$$

Given specific P_1 and P_2, we could unfold uses of these predicates and continue the proof. Suppose

$$P_1 := \lambda c\ o.\forall(\Gamma' : \texttt{context}),$$
$$IA_1\langle\!\langle c, o, \Gamma'\rangle\!\rangle \to \cdots \to IA_w\langle\!\langle c, o, \Gamma'\rangle\!\rangle \to \underline{\Gamma' \rhd o} \qquad \text{and}$$
$$P_2 := \lambda c\ o\ a.\forall(\Gamma' : \texttt{context}),$$
$$IA_1\langle\!\langle c, o, \Gamma'\rangle\!\rangle \to \cdots \to IA_w\langle\!\langle c, o, \Gamma'\rangle\!\rangle \to \underline{\Gamma', [o] \rhd a}$$

The underlining of sequents in the definitions of P_1 and P_2 is to highlight that these are the sequents we apply the generalized rules to (following introductions). In particular, we unfold uses of P_1 and P_2 in the proof state and introduce the variables and induction assumptions. Then the goal is either $\Gamma' \rhd o$ or $\Gamma', [o] \rhd a$. Apply *gr_rule* or *bc_rule* as appropriate, and either will give $(m + n + p)$ new subgoals which come from the three premise forms in these rules, with appropriate instantiations for the externally quantified variables. Now the proof state is

$$\overline{H_m} : \overline{Q_m}\langle\!\langle c, o\rangle\!\rangle$$
$$\overline{Hg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}, (c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle)$$
$$\overline{IHg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}(\Gamma' : \texttt{context}),$$
$$IA_1\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle, \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \cdots \to$$
$$IA_w\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle, \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \Gamma' \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle$$
$$\overline{Hb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}, (c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p})$$
$$\overline{IHb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}(\Gamma' : \texttt{context}),$$
$$IA_1\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, \overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \cdots \to$$
$$IA_w\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, \overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \Gamma', [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p}$$
$$\overline{IP_w} : \overline{IA_w}\langle\!\langle c, o, \Gamma'\rangle\!\rangle$$

$$\overline{Q_m}\langle\!\langle \Gamma', o\rangle\!\rangle,$$
$$\forall\overline{(x_{n,s_n} : R_{n,s_n})}, (\Gamma' \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle),$$
$$\forall\overline{(y_{p,t_p} : S_{p,t_p})}, (\Gamma' \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p})$$

where Γ' is a new signature variable.

6.1 Subproofs for Sequent Premises

To prove the last $(n + p)$ subgoals (the "second" and "third" subgoals above) we first introduce any locally quantified variables as signature variables. For the goal-reduction (resp. backchaining) subgoals, for $j = 1, \ldots, n$ (resp. $k = 1, \ldots, p$), we apply induction hypothesis IHg_j (resp. IHb_k), instantiating Γ' in the induction hypothesis with $\Gamma' \cup \gamma_j\langle\!\langle o\rangle\!\rangle$ (resp. $\Gamma' \cup \gamma'_k\langle\!\langle o\rangle\!\rangle$). This yields the proof state in Figure 7 for goal-reduction premises (resp. backchaining premises).

6.2 Subproofs for Non-Sequent Premises

The proof of the first m subgoals depends on the definition of Q_i for $i = 1 \ldots m$. If the first argument (a `context`) is not used in its definition, then $Q_i\langle\!\langle \Gamma', o\rangle\!\rangle$ is provable by assumption H_i, since we will have $Q_i\langle\!\langle \Gamma', o\rangle\!\rangle = Q_i\langle\!\langle c, o\rangle\!\rangle$. Any other dependencies on signature variables can be ignored since we can assign the variables

$$\overline{H_m} : \overline{Q_m}\langle\!\langle c, o\rangle\!\rangle$$
$$\overline{Hg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}, (c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle)$$
$$\overline{IHg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}(\Gamma' : \texttt{context}),$$
$$IA_1\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle, \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \cdots \to$$
$$IA_w\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle, \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \Gamma' \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle$$
$$\overline{Hb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}, (c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p})$$
$$\overline{IHb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}(\Gamma' : \texttt{context}),$$
$$IA_1\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, \overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \cdots \to$$
$$IA_w\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, \overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle, \Gamma'\rangle\!\rangle \to \Gamma', [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle] \rhd \overline{a_p}$$
$$\overline{IP_w} : \overline{IA_w}\langle\!\langle c, o, \Gamma'\rangle\!\rangle$$

$$\overline{IA_w}\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle, \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}}\rangle\!\rangle, \Gamma' \cup \overline{\gamma_n}\langle\!\langle o\rangle\!\rangle\rangle\!\rangle$$
$$(resp.\ \overline{IA_w}\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle, \overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}}\rangle\!\rangle, \Gamma' \cup \overline{\gamma'_p}\langle\!\langle o\rangle\!\rangle\rangle\!\rangle)$$

Figure 7. Incomplete proof branches for sequent premises

$$H_1 : D \in \Gamma$$
$$Hb_1 : \Gamma, [D] \rhd a_1$$
$$IHb_1 : \forall(\Gamma' : \texttt{context}), IA_1\langle\!\langle \Gamma, D, \Gamma'\rangle\!\rangle \to \cdots \to$$
$$IA_w\langle\!\langle \Gamma, D, \Gamma'\rangle\!\rangle \to \Gamma', [D] \rhd a_1$$
$$\overline{IP_w} : \overline{IA_w}\langle\!\langle \Gamma, \langle\ A\ \rangle, \Gamma'\rangle\!\rangle$$

$$D \in \Gamma'$$

Figure 8. Incomplete proof branch (*g_dyn* case)

as we choose when applying the generalized rule. We will illustrate this by considering each rule with non-sequent premises, starting from the second proof state in Section 6 and, for $(i = 1, \ldots, m)$, $(j = 1, \ldots, n)$, $(k = 1, \ldots, p)$, show how to define Q_i, γ_j, F_j, γ'_k, and F'_k and finish the subproofs where possible.

Case g_prog : This rule has one non-sequent premise and one goal-reduction sequent premise with no local quantification, so $m = n = 1, p = 0, o = \langle\ A\ \rangle$, and $c = \Gamma$. Define $Q_1\langle\!\langle \Gamma, \langle\ A\ \rangle\rangle\!\rangle := A :- G$, $\gamma_1\langle\!\langle \langle\ A\ \rangle\rangle\!\rangle := \emptyset$, and $F_1\langle\!\langle \langle\ A\ \rangle\rangle\!\rangle := G$, where G : `oo` is a signature variable. Then we are proving the following:

$$H_1 : A :- G$$
$$Hg_1 : \Gamma \rhd G$$
$$IHg_1 : \forall(\Gamma' : \texttt{context}), IA_1\langle\!\langle \Gamma, G, \Gamma'\rangle\!\rangle \to \cdots \to$$
$$IA_w\langle\!\langle \Gamma, G, \Gamma'\rangle\!\rangle \to \Gamma' \rhd G$$
$$\overline{IP_w} : \overline{IA_w}\langle\!\langle \Gamma, \langle\ A\ \rangle, \Gamma'\rangle\!\rangle$$

$$A :- G$$

which is completed by assumption H_1.

Case g_dyn : This rule has one non-sequent premise and one backchaining sequent premise with no local quantification, so $m = p = 1$, $n = 0$, $c = \Gamma$, and $o = \langle\ A\ \rangle$. Define $Q_1\langle\!\langle \Gamma, \langle\ A\ \rangle\rangle\!\rangle := D \in \Gamma$, $\gamma'_1\langle\!\langle \langle\ A\ \rangle\rangle\!\rangle := \emptyset$, and $F'_1\langle\!\langle \langle\ A\ \rangle\rangle\!\rangle := D$, where D : `oo` is a signature variable. Then we need to prove what is displayed in Figure 8. Here we do not have enough information to finish this

branch of the proof. An induction assumption may be of use, but we will need specific P_1 and P_2.

Case g_some : This rule has one non-sequent premise and one goal-reduction sequent premise with no local quantification, so $m = n = 1$, $p = 0$, $c = \Gamma$, and $o = \texttt{Some}\ G$. Define $Q_1(\!|\Gamma, \texttt{Some}\ G|\!) \coloneqq \texttt{proper}\ E$, $\gamma_1(\!|\texttt{Some}\ G|\!) \coloneqq \emptyset$, and $F_1(\!|\texttt{Some}\ G|\!) \coloneqq G\ E$ where $E : \texttt{expr con}$ is a signature variable. Then we are proving the following:

$$\frac{\begin{array}{l} H_1 : \texttt{proper}\ E \\ Hg_1 : \Gamma \rhd G\ E \\ IHg_1 : \forall(\Gamma' : \texttt{context}), IA_1(\!|\Gamma, G\ E, \Gamma'|\!) \to \cdots \to \\ \qquad IA_w(\!|\Gamma, G\ E, \Gamma'|\!) \to \Gamma' \rhd G\ E \\ \overline{IP_w} : \overline{IA_w}(\!|\Gamma, \texttt{Some}\ G, \Gamma'|\!) \end{array}}{\texttt{proper}\ E}$$

which is completed by assumption H_1.

Case b_all : This case is proven as above but with $m = p = 1$, $n = 0$, $c = \Gamma$, and $o = \texttt{All}\ D$. Define $Q_1(\!|\Gamma, \texttt{All}\ D|\!) \coloneqq \texttt{proper}\ E$, $\gamma_1'(\!|\texttt{All}\ D|\!) \coloneqq \emptyset$, and $F_1'(\!|\texttt{All}\ D|\!) \coloneqq D\ E$ where $E : \texttt{expr con}$ is a signature variable. The goal $\texttt{proper}\ E$ is provable by the assumption of the same form as in the previous case.

In the next two sections we will return to this idea of proofs about a specification logic from a generalized form of SL rule to prove properties of the SL once we have fully defined P_1 and P_2. The proof states in Figures 7 and 8 (the incomplete branches) will be roots of these explanations.

7. Structural Rules

For our intuitionistic SL we prove the standard structural rules of weakening, contraction, and exchange:

Theorem (`weakening`)**.**

$$\frac{\Gamma \rhd \beta_2}{\Gamma, \beta_1 \rhd \beta_2} \wedge \frac{\Gamma, [\beta_2] \rhd \alpha}{\Gamma, \beta_1, [\beta_2] \rhd \alpha}$$

Theorem (`contraction`)**.**

$$\frac{\Gamma, \beta_1, \beta_1 \rhd \beta_2}{\Gamma, \beta_1 \rhd \beta_2} \wedge \frac{\Gamma, \beta_1, \beta_1, [\beta_2] \rhd \alpha}{\Gamma, \beta_1, [\beta_2] \rhd \alpha}$$

Theorem (`exchange`)**.**

$$\frac{\Gamma, \beta_2, \beta_1 \rhd \beta_3}{\Gamma, \beta_1, \beta_2 \rhd \beta_3} \wedge \frac{\Gamma, \beta_2, \beta_1, [\beta_3] \rhd \alpha}{\Gamma, \beta_1, \beta_2, [\beta_3] \rhd \alpha}$$

These are all corollaries of a general theorem:

Theorem 1 (`monotone`)**.**

$$\frac{\Gamma \subseteq \Gamma' \quad \Gamma \rhd \beta}{\Gamma' \rhd \beta} \wedge \frac{\Gamma \subseteq \Gamma' \quad \Gamma, [\beta] \rhd \alpha}{\Gamma', [\beta] \rhd \alpha}$$

The proofs of all of these theorems are automated in the Coq implementation. Here, we continue our explanation using our generalized proof states.

Proof. Theorem 1 is proven by mutual structural induction over the premises $\Gamma \rhd \beta$ and $\Gamma, [\beta] \rhd \alpha$. Defining P_1 and P_2 as

$$\begin{aligned} P_1 \coloneqq &\lambda\ (c : \texttt{context})(o : \texttt{oo})\ . \\ &\forall\ (\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \underline{\Gamma' \rhd o} \\ P_2 \coloneqq &\lambda\ (c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm})\ . \\ &\forall\ (\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \underline{\Gamma', [o] \rhd a} \end{aligned}$$

we are proving

$$\begin{aligned} &(\forall\ (c : \texttt{context})\ (o : \texttt{oo}), (c \rhd o) \to (P_1\ c\ o))\ \wedge \\ &(\forall\ (c : \texttt{context})\ (o : \texttt{oo})\ (a : \texttt{atm}), (c, [o] \rhd a) \to (P_2\ c\ o\ a)) \end{aligned}$$

which has the form discussed in Section 4, so the mutual structural induction principle may be used.

7.1 Generalized SL Part II: The Structural Rules Hold

We build on the inductive proof in Section 6 over the GSL. Recall that when we took the proof as far as we could; we had three remaining groups of branches to finish ($m + n + p$ subgoals), one group for rules with non-sequent premises depending on the context of the rule conclusion, and one for each kind of sequent premise (see Figures 7 and 8). We will continue this effort below, using the P_1 and P_2 defined for this theorem. This means we will have one induction assumption (i.e., $w = 1$) which is $IA_1(\!|c, o, \Gamma'|\!) \coloneqq c \subseteq \Gamma'$.

7.1.1 Subproofs for Sequent Premises

First we will prove the subgoals coming from the sequent premises, building on Figure 7 and using IA_1 as defined above. The proof state for goal-reduction (resp. backchaining) premises is

$$\frac{\begin{array}{l} \overline{H_m} : \overline{Q_m}(\!|c, o|\!) \\ \overline{Hg_n} : \forall(\overline{x_{n,s_n} : R_{n,s_n}}), (c \cup \overline{\gamma_n}(\!|o|\!) \rhd \overline{F_n}(\!|o, \overline{x_{n,s_n}}|\!)) \\ \overline{IHg_n} : \forall(\overline{x_{n,s_n} : R_{n,s_n}})(\Gamma' : \texttt{context}), \\ \qquad (c \cup \overline{\gamma_n}(\!|o|\!)) \subseteq \Gamma' \to \Gamma' \rhd \overline{F_n}(\!|o, \overline{x_{n,s_n}}|\!) \\ \overline{Hb_p} : \forall(\overline{y_{p,t_p} : S_{p,t_p}}), (c \cup \overline{\gamma_p'}(\!|o|\!), [\overline{F_p'}(\!|o, \overline{y_{p,t_p}}|\!)] \rhd \overline{a_p}) \\ \overline{IHb_p} : \forall(\overline{y_{p,t_p} : S_{p,t_p}})(\Gamma' : \texttt{context}), \\ \qquad (c \cup \overline{\gamma_p'}(\!|o|\!)) \subseteq \Gamma' \to \Gamma', [\overline{F_p'}(\!|o, \overline{y_{p,t_p}}|\!)] \rhd \overline{a_p} \\ IP_1 : c \subseteq \Gamma' \end{array}}{(c \cup \overline{\gamma_n}(\!|o|\!)) \subseteq (\Gamma' \cup \overline{\gamma_n}(\!|o|\!))\ (\textit{resp.}\ (c \cup \overline{\gamma_p'}(\!|o|\!)) \subseteq (\Gamma' \cup \overline{\gamma_p'}(\!|o|\!)))}$$

The goal is provable by context lemma `context_sub_sup` and assumption IP_1.

7.1.2 Subproofs for Non-Sequent Premises

Still to be proven are the subgoals for non-sequent premises. As seen in Subsection 6.2, the only rule of the SL whose corresponding subcase still needs to be proven is *g_dyn*. From Figure 8 and using P_1 and P_2 as defined here, we are proving

$$\frac{\begin{array}{l} H_1 : D \in \Gamma \\ Hb_1 : \Gamma, [D] \rhd a_1 \\ IHb_1 : \forall(\Gamma' : \texttt{context}), \Gamma \subseteq \Gamma' \to \Gamma', [D] \rhd a_1 \\ IP_1 : \Gamma \subseteq \Gamma' \end{array}}{D \in \Gamma'}$$

Unfolding the definition of context subset in IP_1 it becomes $\forall(o : \texttt{oo}), o \in \Gamma \to o \in \Gamma'$. Backchaining on this form of the goal gives $D \in \Gamma$, provable by assumption H_1.

In Section 6, we explored how to prove statements about the GSL for a restricted form of theorem statement. There were three classes of incomplete proof branches that had a final form shown in Figures 7 and 8. In Section 5.1 we saw how to derive the SL from the GSL. So here we have proven a structural theorem for the rules of the GSL in a general way that can be followed for any SL rule. □

8. Cut Admissibility

The cut rule is shown to be admissible in this specification logic by proving the following:

Theorem 2 (`cut_admissible`)**.**

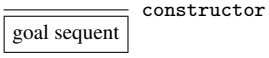
$$\frac{\Gamma, \delta \rhd \beta \quad \Gamma \rhd \delta}{\Gamma \rhd \beta} \wedge \frac{\Gamma, \delta, [\beta] \rhd \alpha \quad \Gamma \rhd \delta}{\Gamma, [\beta] \rhd \alpha}$$

Since our specification logic makes use of two kinds of sequents, we prove two cut rules. These correspond to the two conjuncts above, where the first is for goal-reduction sequents and the second is for backchaining sequents.

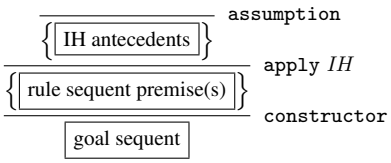
Outline. This proof will be a nested induction, first over the cut formula δ, then over the sequent premises with δ in their contexts. Since there are seven rules for constructing formulas and 15 SL rules, this will result in 105 subcases. These can be partitioned into five classes with the same proof structure, four of which we briefly illustrate presently.

Technical details based on the particular statement to be proven will be seen in the main proof where we again consider the generalized form of SL rule and also see what the proof state will look like for specific subcases.

The cases for the axioms *g_tt* and *b_match* are proven by one use of `constructor` (7 formulas * 2 rules = 14 subcases).

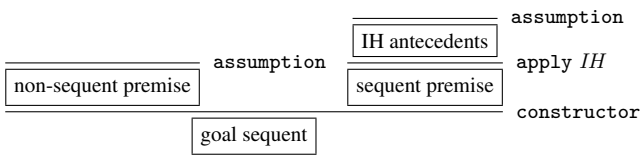


Cases for rules with only sequent premises, including those with inner quantification, with the same context as the conclusion have the same proof structure. Note that by *same context*, we include rules modifying the focused formula. The rules in this class are *g_and*, *g_all*, *g_allx*, b_and_1, b_and_2, *b_imp*, *b_allx*, and *b_some* (7 formulas * 8 rules = 56 subcases). We apply `constructor` to the goal sequent which, after any introductions, will give a sequent subgoal for each sequent premise of the rule. To each of the new subgoals we apply the appropriate induction hypothesis, giving new subgoals for each antecedent of each induction hypothesis used. Now all goals can be proven by assumption (hypotheses from the induction principle and induction assumptions).



Only one rule modifies the context of the sequent, *g_imp* (7 formulas * 1 rule = 7 subcases). The proof of the subcase for this rule is similar to above, but requires the use of another structural rule, `weakening`, before the sequent subgoal will match the sequent assumption introduced from the goal.

The remaining four rules have both a non-sequent premise and a sequent premise. Of these, the subcases for *g_prog*, *g_some*, and *b_all* have a similar proof structure; apply `constructor` to the goal so that the non-sequent premise is provable by assumption, then prove the branch for the sequent premise as above (7 formulas * 3 rules = 21 subcases).



The proof of the subcase for *g_dyn* is more complicated due to the form of the non-sequent premise, $D \in \Gamma$, which depends on the context in the goal sequent, $\Gamma \rhd \langle\ A\ \rangle$. We need more details to analyse the subcases for this rule further.

For seven formula constructions and 14 SL rule subcases, we are able to automate 98 of 105 subcases of this proof in the Coq implementation, as seen below (where `o1` is the cut formula in the implementation, in place of δ).

```
Hint Resolve gr_weakening context_swap.
induction o1; eapply seq_mutind; intros;
subst; try (econstructor; eauto; eassumption).
```

(end outline) □

The cut admissibility theorem stated above is a simple corollary of the following theorem (with explicit quantification):

$$\begin{aligned}
&\forall(\delta : \texttt{oo}),\\
&\quad(\forall(c : \texttt{context})(o : \texttt{oo}),\ c \rhd o \rightarrow\\
&\qquad\forall(\Gamma' : \texttt{context}), c = \Gamma', \delta \rightarrow \Gamma' \rhd \delta \rightarrow \Gamma' \rhd o) \wedge\\
&\quad(\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}),\ c, [o] \rhd a \rightarrow\\
&\qquad\forall(\Gamma' : \texttt{context}), c = \Gamma', \delta \rightarrow \Gamma' \rhd \delta \rightarrow \Gamma', [o] \rhd a)
\end{aligned}$$

Proof. We begin with an induction over δ, so we are proving $\forall(\delta : \texttt{oo}), P\ \delta$ with P defined as

$$\begin{aligned}
&P : \texttt{oo} \rightarrow \texttt{Prop} := \lambda(\delta : \texttt{oo})\ .\\
&\quad(\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow P_1\ c\ o) \wedge\\
&\quad(\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow P_2\ c\ o\ a),
\end{aligned}$$

where

$$\begin{aligned}
&P_1 : \texttt{context} \rightarrow \texttt{oo} \rightarrow \texttt{Prop} := \lambda(c : \texttt{context})(o : \texttt{oo})\ .\\
&\qquad\forall(\Gamma' : \texttt{context}), c = (\Gamma', \delta) \rightarrow \Gamma' \rhd \delta \rightarrow \underline{\Gamma' \rhd o}\\
&P_2 : \texttt{context} \rightarrow \texttt{oo} \rightarrow \texttt{atm} \rightarrow \texttt{Prop} :=\\
&\quad\lambda(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm})\ .\\
&\qquad\forall(\Gamma' : \texttt{context}), c = (\Gamma', \delta) \rightarrow \Gamma' \rhd \delta \rightarrow \underline{\Gamma', [o] \rhd a}
\end{aligned}$$

P, P_1, and P_2 will provide the induction hypotheses used in this proof. Next is a nested induction, which is a mutual structural induction over $c \rhd o$ and $c, [o] \rhd a$ using P_1 and P_2 as above.

8.1 Generalized SL Part III: Cut Rule Proven Admissible

As in the proof of Theorem 1, we build on the inductive proof in Section 6, unfolding P_1 and P_2 as defined here. Recall that we have now introduced assumptions and applied the appropriate generalized SL rule to the underlined sequents in the definition of P_1 and P_2. For the proof of cut admissibility, there are two induction assumptions from P_1 and P_2 (so $w = 2$). Define $IA_1(\!|c, o, \Gamma'|\!) := (c = (\Gamma', \delta))$ and $IA_2(\!|c, o, \Gamma'|\!) := \Gamma' \rhd \delta$, where δ is the cut formula in the cut rule and is in the signature of variables of the proof state.

8.1.1 Subproofs for Sequent Premises

First we will prove the subgoals coming from the sequent premises, building on Figure 7 and using IA_1 and IA_2 as defined above. For a moment we will ignore the outer induction over the cut formula δ. By ignore we mean let $\delta := \eta$ where $\eta : \texttt{oo}$, and we will not display the induction hypothesis for this induction. The proof state

for goal-reduction premises (resp. backchaining premises) is

$$\overline{H_m} : \overline{Q_m}\langle\!\langle c, o \rangle\!\rangle$$
$$\overline{Hg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}, (c \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}} \rangle\!\rangle)$$
$$\overline{IHg_n} : \forall\overline{(x_{n,s_n} : R_{n,s_n})}(\Gamma' : \texttt{context}),$$
$$(c \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle) = (\Gamma', \eta) \to \Gamma' \rhd \eta \to \Gamma' \rhd \overline{F_n}\langle\!\langle o, \overline{x_{n,s_n}} \rangle\!\rangle$$
$$\overline{Hb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}, (c \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle, [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}} \rangle\!\rangle] \rhd \overline{a_p})$$
$$\overline{IHb_p} : \forall\overline{(y_{p,t_p} : S_{p,t_p})}(\Gamma' : \texttt{context}),$$
$$(c \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle) = (\Gamma', \eta) \to \Gamma' \rhd \eta \to \Gamma', [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}} \rangle\!\rangle] \rhd \overline{a_p}$$
$$IP_1 : c = (\Gamma', \eta)$$
$$IP_2 : \Gamma' \rhd \eta$$

$$(c \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle = ((\Gamma' \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle), \eta)), (\Gamma' \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle \rhd \eta)$$
$$(resp.\ (c \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle = ((\Gamma' \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle), \eta)), (\Gamma' \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle \rhd \eta))$$

The subgoals concerning context equality are proven by context lemmas and assumption IP_1. To prove the sequent subgoals, apply `weakening` then assumption IP_2.

8.1.2 Subproofs for Non-Sequent Premises

Recall that before the induction over sequent premises, we had induction over the cut formula δ. To finish this proof we need to consider the subcases corresponding to the *g_dyn* rule for each form of δ. Below is a proof of the *g_dyn* subcase where $\delta = o_1 \longrightarrow o_2$. The *g_dyn* subcases for other formula constructions follow similarly.

Case $\delta = o_1 \longrightarrow o_2$ **:** The antecedent of the `oo` induction principle for this case is $\forall(o_1\ o_2 : \texttt{oo}), P\ o_1 \to P\ o_2 \to P\ (o_1 \longrightarrow o_2)$, where $P\ o_1$ and $P\ o_2$ are induction hypotheses and P is as defined at the start of this proof. Expanding the goal (we will wait to expand the premises), the proof state is

$$IH_1 : P\ o_1$$
$$IH_2 : P\ o_2$$

$$(\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \to \forall(\Gamma' : \texttt{context}),$$
$$c = (\Gamma', (o_1 \longrightarrow o_2)) \to \Gamma' \rhd (o_1 \longrightarrow o_2) \to \Gamma' \rhd o) \wedge$$
$$(\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \to \forall(\Gamma' : \texttt{context}),$$
$$c = (\Gamma', (o_1 \longrightarrow o_2)) \to \Gamma' \rhd (o_1 \longrightarrow o_2) \to \Gamma', [o] \rhd a)$$

Next we have the mutual induction over sequents.

Subcase g_dyn : Expanding and making introductions building on Figure 8, we want:

$$IH_1 : P\ o_1$$
$$IH_2 : P\ o_2$$
$$H_1 : D \in (\Gamma', o_1 \longrightarrow o_2)$$
$$H_2 : \Gamma', o_1 \longrightarrow o_2, [D] \rhd a$$
$$IH_3 : \forall(\Gamma_0 : \texttt{context}), (\Gamma', o_1 \longrightarrow o_2) = (\Gamma_0, o_1 \longrightarrow o_2) \to$$
$$\Gamma_0 \rhd (o_1 \longrightarrow o_2) \to \Gamma_0, [D] \rhd a$$
$$IP_1 : \Gamma = \Gamma', o_1 \longrightarrow o_2$$
$$IP_2 : \Gamma' \rhd o_1 \longrightarrow o_2$$

$$\Gamma' \rhd \langle\ a\ \rangle$$

with $(\Gamma', o_1 \longrightarrow o_2)$ substituted for Γ using IP_1 and renaming in IH_3 to avoid variable capture. We can specialize IH_3 with Γ', a reflexivity lemma and IP_2 to get the new premise $P_3 : \Gamma', [D] \rhd a$ and apply `elem_inv` to H_1 to get $(D \in \Gamma') \vee (D = o_1 \longrightarrow o_2)$. Inverting H_1, we get two new subgoals with different sets of assumptions. In the second we substitute $o_1 \longrightarrow o_2$ for D using H_1 in that proof state.

$IH_1 : P\ o_1$	$IH_1 : P\ o_1$
$IH_2 : P\ o_2$	$IH_2 : P\ o_2$
$H_1 : D \in \Gamma'$	$H_1 : D = o_1 \longrightarrow o_2$
$P_3 : \Gamma', [D] \rhd a$	$P_3 : \Gamma', [o_1 \longrightarrow o_2] \rhd a$
$IP_2 : \Gamma' \rhd o_1 \longrightarrow o_2$	$IP_2 : \Gamma' \rhd o_1 \longrightarrow o_2$
$\Gamma' \rhd \langle\ a\ \rangle$	$\Gamma' \rhd \langle\ a\ \rangle$

To prove the first, we apply *g_dyn* to the goal, then need to prove $D \in \Gamma'$ and $\Gamma', [D] \rhd a$ which are both provable by assumption.

For the second (right) subgoal, it will be necessary to apply `inversion` to some assumptions to get structurally simpler assumptions, before being able to apply the induction hypotheses IH_1 and IH_2. Inverting P_3 and IP_2, and unfolding P, we have:

$$IH_1 : (\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \to$$
$$\forall(\Gamma' : \texttt{context}), c = (\Gamma', o_1) \to \Gamma' \rhd o_1 \to \Gamma' \rhd o) \wedge$$
$$(\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \to$$
$$\forall(\Gamma' : \texttt{context}), c = (\Gamma', o_1) \to \Gamma' \rhd o_1 \to \Gamma', [o] \rhd a)$$
$$IH_2 : (\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \to$$
$$\forall(\Gamma' : \texttt{context}), c = (\Gamma', o_2) \to \Gamma' \rhd o_2 \to \Gamma' \rhd o) \wedge$$
$$(\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \to$$
$$\forall(\Gamma' : \texttt{context}), c = (\Gamma', o_2) \to \Gamma' \rhd o_2 \to \Gamma', [o] \rhd a)$$
$$P_{3_1} : \Gamma' \rhd o_1$$
$$P_{3_2} : \Gamma', [o_2] \rhd a$$
$$IP_2 : \Gamma', o_1 \rhd o_2$$

$$\Gamma' \rhd \langle\ a\ \rangle$$

Applying the first conjunct of IH_2 to the goal gives three new subgoals $\Gamma', o_2 \rhd \langle\ a\ \rangle$, $(\Gamma', o_2) = (\Gamma', o_2)$ and $\Gamma' \rhd o_2$. For the first, apply *g_dyn*, then we need to prove $o_2 \in (\Gamma', o_2)$ (proven by a context lemma) and $\Gamma', o_2, [o_2] \rhd a$ (proven by `weakening` and assumption P_{3_2}). The second is proven by `reflexivity`. For the third, we apply IH_1 and get new subgoals $(\Gamma', o_1 \rhd o_2)$, $(\Gamma', o_1) = (\Gamma', o_1)$, and $(\Gamma' \rhd o_1)$. The sequent subgoals are proven by assumption and the other by `reflexivity`.

The other six *g_dyn* cases follow a similar argument requiring `inversion` on hypotheses and induction hypothesis specialization.

In summary, the outer induction over δ gave seven cases for seven `oo` constructors. For each of these, an inner induction over sequents gave 15 new subgoals for 15 rules. We saw that for 14 of 15 rules, each rule has the same proof structure for every form of δ. The remaining subgoals were all for the rule *g_dyn* and were more challenging due to the presence of a non-sequent premise that depends on the context of the conclusion.

□

9. Conclusion

We have described the extension of Hybrid to a new more expressive SL, and focused on presenting the proofs of important metatheoretic properties in a general manner. With the metatheory of this SL completed, the next step is to illustrate the benefits of the extra expressive power via case studies.

In the GSL we have made the rules general enough to capture the rules of the SL, but it could be generalized further to explore

other specification logics that do not fit the restrictions here. This is a subject of future work, as we explore even more expressive SLs.

There are a variety of other systems supporting HOAS, such as Beluga (Pientka and Dunfield 2010) and Twelf (Schürmann 2009), to name just two. These two systems, along with Abella and Hybrid were compared on several benchmark problems in (Felty et al. 2015). Hybrid and Abella are similar in the sense that they are based on a proof-theoretic foundation and follow the two-level approach, implementing an SL inside a logic or type theory, while Twelf and Beluga are built on type-theoretic foundations.

Our formulation of sequent rules uses the style developed in (Pfenning 2000), where the cut rule incorporates aspects of weakening and contraction, facilitating the kind of structural induction argument used both there and here. One main difference in the approaches is the representation of inference rules. In that paper, the proofs are formalized in Elf (an early version of Twelf) where rules are generally expressed in a natural deduction style with implicit contexts. In contrast, we represent and reason about contexts directly, and consequently illustrate that doing so is not as difficult as is argued in (Pfenning 2000).[4]

The proof of cut admissibility in Abella uses the same overall structure with induction on the cut formula, with a sub-induction on the structure of the proof of the left premise of the cut rule. A detailed analysis of the differences in the proofs is left as future work. We do note, however, that the statement of the two theorems differs. The Abella version requires the following additional conjunct:

$$\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c \rhd o \to c, [o] \rhd a \to c \rhd \langle\ a\ \rangle.$$

There are also some differences in the rules. Our logic includes existential quantification, while the Abella version does not. Also, without loss of generality, our *g_prog* rule restricts static program clauses to have the form $\forall_{\tau_1} \ldots \forall_{\tau_n}.G \longrightarrow A$. Finally, we restrict universal quantification to second-order, while the Abella version does not. This does not affect the proofs of metatheorems, and we don't expect it to significantly limit the kinds of OLs we can consider. For example, the two case studies in (Wang et al. 2013) do not require more than second-order quantification in the SL.

As mentioned, implementing Hybrid in Coq gives us access to Coq's extensive standard library and other facilities. Having such access allows us, for example, to simplify the encoding of the example discussed in Section 2 in two ways. First, in general, when an OL's syntax can be directly encoded using a first-order representation, we can define it directly as a Coq inductive type instead of as a set of terms of type `expr`. For example, we could remove the three definitions for `dApp`, `dAbs`, and `dVar` and the constants they depend on, and instead define an type `dtm` with three constructors. Second, we can use Coq's library for natural numbers, which allows for a simpler definition of the `hodb` inference rules (mentioned but not shown earlier).

The development of the metatheory of the SL in this paper differs from other Hybrid SLs, in particular, those that appear in (Felty and Momigliano 2012). In the definition of those SLs, sequents had an additional natural number argument, and metatheoretic properties were proved by induction over the height of a proof, rather than by a direct structural induction on the definition of (the two kinds of) sequents, as is done here. Future work includes exploring the connections between these two kinds of induction in the context of the SL in this paper, as well as comparing their use in proving properties of OLs.

[4] See (Felty et al. 2015) for a fuller comparison of these two approaches to reasoning.

Acknowledgments

The authors acknowledge the support of the Natural Sciences and Engineering Research Council of Canada. In addition, special thanks go to Alberto Momigliano for his involvement in the initial coding of the data structures of the specification logic and for his insights and discussions over the course of this work.

References

B. E. Aydemir et al. Mechanized metatheory for the masses: The POPLMARK challenge. In *18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 3603 of *LNCS*, pages 50–65. Springer, 2005.

Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

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

A. P. Felty and A. Momigliano. Hybrid: A definitional two-level approach to reasoning with higher-order abstract syntax. *Journal of Automated Reasoning*, 48(1):43–105, 2012.

A. P. Felty, A. Momigliano, and B. Pientka. The next 700 challenge problems for reasoning with higher-order abstract syntax representations: Part 2—a survey. *Journal of Automated Reasoning*, 55(4):307–372, 2015.

A. Gacek. The Abella interactive theorem prover (system description). In *4th International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *LNCS*, pages 154–161. Springer, 2008.

J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

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

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

C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In *International Conference on Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 328–345. Springer, 1993.

F. Pfenning. Structural cut elimination I: Intuitionistic and classical logic. *Information and Computation*, 157(1/2):84–141, 2000.

B. Pientka. Proof pearl: The power of higher-order encodings in the logical framework lf. In *20th International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 4732 of *LNCS*, pages 246–261. Springer, 2007.

B. Pientka and J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *5th International Joint Conference on Automated Reasoning (IJCAR)*, volume 6173 of *LNCS*, pages 15–21. Springer, 2010.

C. Schürmann. The Twelf proof assistant. In *22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 79–83. Springer, 2009.

Y. Wang, K. Chaudhuri, A. Gacek, and G. Nadathur. Reasoning about higher-order relational specifications. In *15th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP)*, pages 157–168. ACM Press, 2013.