uOttawa

L'Université canadienne
Canada's university

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

Chelsea Battell

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of
Master of Science in Mathematics[1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa

[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

Hybrid is a two-level logical framework that supports higher-order abstract syntax (HOAS), where a specification logic (SL) extends the class of object logics (OLs) we can reason about. We develop a new Hybrid SL and formalize its metatheory, proving weakening, contraction, exchange, and cut admissibility; results that greatly simplify reasoning about OLs in systems providing HOAS. The SL is a sequent calculus defined as an inductive type in Coq and we prove properties by structural induction over SL sequents. We also present a generalized SL and metatheory statement, allowing us to prove many cases of such theorems in a general way and understand how to identify and prove the difficult cases. We make a concrete and measurable improvement to Hybrid with the new SL formalization and provide a technique for abstracting such proofs, leading to a condensed presentation, greater understanding, and a generalization that may be instantiated to other logics.

Dedications

To Amy Felty, my advisor and role model.

Acknowledgement

Thanks and acknowledgment go to the Natural Sciences and Engineering Research Council (NSERC) of Canada, Professor Amy Felty, and the University of Ottawa for the financial support provided while completing this research.

I am extremely grateful for the guidance provided by Professor Felty to help me through the many challenges faced while completing this degree and for her feedback and assistance in editing all of the presentations and written documents related to this research.

I would also like to thank Alberto Momigliano for the initial encoding of the data structures for the specification logic and discussions through the course of this work.

Contents

List of Figures **vii**

1 Introduction **1**
1.1 Mechanized Reasoning 1
1.2 Hybrid 2
1.3 Outline 3

I Background **4**

2 Coq **5**
2.1 Calculus of Constructions Type System 6
2.2 Simply Typed Lambda Calculus 7
2.3 Dependent Types 7
2.3.1 Predicates 8
2.4 Higher-Order Types 8
2.4.1 Polymorphism 8
2.4.2 Type Operators 9
2.5 Interactive Proving in Coq 9
2.5.1 Some Coq Tactics, Tacticals, and Commands 9
2.5.2 Proof State 11
2.5.3 Example Proof 11
2.6 Induction in Coq 12
2.6.1 Example Proof 13
2.6.2 Mutually Inductive Types 14
2.7 Conclusion 15

3 Hybrid **17**
3.1 Object Logic (OL) 18
3.1.1 Example OL 18
3.2 Ambient Logic 19

3.3 Representing Higher-Order Abstract Syntax (HOAS) in Hybrid . . . 19
3.4 Specification Logic (SL) . . . 20
3.5 Example OL Implementation . . . 23
3.6 Comparison to Other Architectures . . . 25

II Contributions 26

4 Specification Logic 27
4.1 Contexts in Coq . . . 28
4.2 Hereditary Harrop Specification Logic in Coq . . . 29
4.3 Mutual Structural Induction . . . 31

5 Specification Logic Metatheory 34
5.1 Structural Rules . . . 35
5.2 Cut Admissibility . . . 40
5.2.1 Subcase for *g_dyn*: Alternate Proof Attempt . . . 42
5.2.2 Subcase for *g_dyn*: Original Proof Structure . . . 46

6 Generalized Specification Logic 51
6.1 SL Rules from GSL Rules . . . 52

7 Generalized Specification Logic Metatheory 55
7.1 GSL Induction Part I: A Restricted Theorem . . . 55
7.1.1 Sequent Subgoals . . . 56
7.1.2 Non-Sequent Subgoals . . . 58
7.2 GSL Induction Part II: The Structural Rules Hold . . . 61
7.2.1 Sequent Subgoals . . . 61
7.2.2 Non-Sequent Subgoals . . . 62
7.3 GSL Induction Part III: Cut Rule Proven Admissible . . . 63
7.3.1 Sequent Subgoals . . . 63
7.3.2 Non-Sequent Subgoals . . . 64

8 Conclusion 66
8.1 Related Work . . . 66
8.2 Future Work . . . 67

Bibliography 70

Index 70

List of Figures

3.1 Architecture of the Hybrid system 17
3.2 Terms in Hybrid 20
3.3 Typing of λ-calculus Abstractions 21
3.4 Type of SL Formulas 21
3.5 Induction Principle for `oo` 22
3.6 Example OL: Encoding Syntax in Hybrid 24
3.7 Example OL: Encoding OL Inference Rules in Hybrid 24

4.1 Goal-Reduction Rules, `grseq` : `context` $\rightarrow$ `oo` $\rightarrow$ `Prop` 29
4.2 Backchaining Rules, `bcseq` : `context` $\rightarrow$ `oo` $\rightarrow$ `atm` $\rightarrow$ `Prop` 30
4.3 SL Sequent Mutual Induction Principle 32

5.1 Coq proof of `monotone` (Theorem 5.7) 50
5.2 Coq proof of 98/105 cases of `cut_admissible` (Theorem 5.8) 50

7.1 Proof state of GSL induction after rule application 57
7.2 Incomplete proof branches for sequent premises 58
7.3 Incomplete proof branch (*g_dyn* case) 60

List of Theorems

3.1 Theorem (`hodb_det1`) 23
3.2 Theorem (`hodb_det3`) 23

4.1 Lemma (`elem_inv`) 28
4.2 Lemma (`elem_sub`) 28
4.3 Lemma (`elem_self`) 28
4.4 Lemma (`elem_rep`) 28
4.5 Lemma (`context_swap`) 28
4.6 Lemma (`context_sub_sup`) 28

5.1 Theorem (`gr_weakening`) 35
5.2 Theorem (`bc_weakening`) 35
5.3 Theorem (`gr_contraction`) 35
5.4 Theorem (`bc_contraction`) 35
5.5 Theorem (`gr_exchange`) 35
5.6 Theorem (`bc_exchange`) 35
5.7 Theorem (`monotone`) 35
5.8 Theorem (`cut_admissible`) 40

Chapter 1

Introduction

The goal of this research is to increase the reasoning abilities of an existing system that is intended to help mechanize programming language metatheory. The system that this work contributes to is called Hybrid [6] and is part of the research program carried out by the Software Correctness and Safety Research Laboratory at the University of Ottawa under the supervision of Professor Amy Felty. Hybrid is implemented both in Coq [20] and Isabelle/HOL [14], interactive proof assistants used for applications such as formalizing mathematics, certifying compilers, and proving correctness of programs. Our application of Coq is proving metatheory of formal systems efficiently. The contributions to Hybrid described in this thesis are to the Coq implementation.

1.1 Mechanized Reasoning

Proof is essential in modern mathematics and in logic in particular. We trust and build on the work of others when we can see that they have presented a rigorous argument supporting their work. When writing proofs with many cases or details to manage, it is easy to make errors and these proofs are tedious to check. So we must trust the proof writer (and all proofs that their work builds on) or else spend an exorbitant amount of time checking proofs (and still possibly miss errors). Proof assistants such as Coq provide proof terms that can be independently checked. To trust all proof terms of theorems proven in Coq, one only needs to trust the underlying proof theory and the implementation of the system checking the proof.

Manually checking "paper and pencil" (non-formalized) proofs is not a scalable or practical technique for applications to software development in industry where financial concerns may be prioritized over software correctness and safety. The area of formal methods for software engineering is focused on (im)proving the correctness of software. Even if we develop techniques to allow software developers to prove correctness of software without needing expertise in the research area, we still need to be sure that the languages in which the programs are defined cause the expected behaviour.

We need a mechanized solution to studying programming language metatheory.

The POPLmark challenge [1] was introduced to merge the concerns of the proof theory and programming languages communities. It provides a set of challenge problems to explore how to mechanize programming language metatheory using a variety of systems and techniques and illustrates the importance of formalized reasoning in programming language research. In fact, it is standard for papers presented at programming language conferences to be accompanied by formal proofs of the metatheory, again for reasons related to confidence in the correctness of the work.

Implementing Hybrid in an existing trusted general-purpose theorem prover makes it possible to easily make modular changes to the system.

1.2 Hybrid

Hybrid uses the technique of *higher-order abstract syntax* (HOAS) [16], also known as λ-tree syntax [13], and an intermediate reasoning layer called a *specification logic* (SL) to make the proofs of programming language theorems more efficient.

HOAS is a technique for representing formal systems or *object logics* (OLs) that we wish to reason about. It simplifies reasoning about OLs by allowing object-level name binding structures to be encoded in the binding structures of the meta-language that the system is defined in. Hybrid is implemented in Coq, so the meta-language is a λ-calculus. We can encode object-level substitution and renaming as meta-level β-reduction and α-conversion, respectively. This avoids the requirement of implementing infrastructure of libraries of definitions and lemmas to deal with issues surrounding binders and variable naming when studying an OL.

Adding a SL extends the class of OLs that we can reason about efficiently using a system supporting HOAS. The relationship between these levels is the reason Hybrid is considered a *two-level* logical framework. This approach was introduced by McDowell and Miller in [11] with the $FO\lambda^{\Delta\mathbb{N}}$ logic.

There are many features of Hybrid that help work toward the goal of efficiently mechanizing programming language metatheory, but this thesis is focused on the SL layer. Here we makes two contributions: an extension to the reasoning power of Hybrid and new insight into proofs of properties of certain kinds of sequent calculi.

First, a new SL is implemented and structural properties of this logic are formalized and proven in the Coq proof assistant. The new SL presented here is a sequent calculus based on the logic of hereditary Harrop formulas as presented in [12]. We prove weakening, contraction, exchange and admissibility of the cut rule for this logic. The structural rules can then be used in proofs of OL theorems that we wish to prove. This is a concrete extension to an existing computing tool (namely Hybrid) since it improves the reasoning abilities of this system in a fully formalized way.

The second contribution is more theoretical and educational. We present a generalization of the specification logic and form of theorem statement to encapsulate the

implemented SL and desired structural rules, respectively. This presentation allows us to see the structural proofs in a more condensed but still comprehensive way. We are also able to gain a deeper understanding of these proofs and isolate the difficult cases. It is our hope that this presentation will give others insight into the kind of proofs we work through and that this general framework may find some use in other applications.

1.3 Outline

This thesis is broken up into two parts: background and contributions. To understand the research described here, it is necessary to first ensure that the reader understands the logical foundations of the ambient reasoning system for this work, namely Coq, and the basics of Hybrid. In Chapter 2 we review the Coq type system and introduce the reader to using it as a proof assistant. We present an overview of Hybrid in Chapter 3. Next we move to the contributions of this research. As stated above, this is focused on a new intermediate reasoning logic for Hybrid. Chapter 4 presents this logic and its metatheory is studied in Chapter 5. From here we abstract the specification logic of Chapter 4 with a generalized specification logic in Chapter 6 and prove properties of this logic in a general way in Chapter 7. We conclude in Chapter 8 with a review of the results presented and look at related and future work.

The research presented in this thesis is published in [2]. The files of the Coq formalization are available at `www.eecs.uottawa.ca/~afelty/BattellThesis/`.

Part I

Background

Chapter 2

Coq

Coq [3, 20] is an implementation of the Calculus of Inductive Constructions (CIC), an extension of the Calculus of Constructions (CoC) [5], the typed lambda calculus at the top of the lambda cube. Its position on the lambda cube means it allows dependent types, polymorphism, and type operators. Originally created by Thierry Coquand, CoC and its extensions have spurred the development of a variety of proof assistants and interactive theorem proving systems currently used in the research areas of automated deduction and formal methods for software engineering.

Coq can be used both as a theorem proving system and a functional programming language. Its type system allows for a correspondence to be observed between theorems and function specifications; between proofs and function definitions (the Curry-Howard correspondence [10]).

The calculus of constructions has the strong normalization property, meaning all terms of CoC will be reduced to an irreducible form by any sequence of reductions. Other than the standard β-reduction we have δ-reductions, which replace an identifier with its definition, and ι-reductions, which handle computations in recursive programs.

In this chapter we will explore the Coq type system by first looking at the simply typed λ-calculus, followed by the increased expressive power added by each axis of refinement on the lambda-cube (dependent types, polymorphism, and type operators) in Sections 2.2 and 2.4. We see how to use Coq as an interactive proof assistant in Section 2.5 followed by information on writing inductive proofs in Section 2.6. The notation and style used to illustrate the concepts follows the presentation in [3] and [20]. The discussion is motivated by [3] and [5].

2.1 Calculus of Constructions Type System

In CoC, every term has a type, which is itself a term. There is no syntactic difference between terms and types. We will use "term" and "type" interchangeably according to what is most reasonable for the current discussion.

The notation $e : t$ means e has type t. We distinguish two special types, `Set` and `Prop` with `Set` : $\texttt{Type}_0$ and `Prop` : $\texttt{Type}_0$. This yields an infinite type hierarchy $s \leq \texttt{Type}_0 \leq ... \leq \texttt{Type}_j \leq \texttt{Type}_{j+1}...$ with $s \in \{\texttt{Set}, \texttt{Prop}\}$ and for all $i : \mathbb{N}$, $\texttt{Type}_i : \texttt{Type}_{i+1}$. `Set` and `Prop` are the lowest types in the hierarchy that we typically consider to be types of types, so we call them "base sorts". Terms of type `Prop` are meant to be logical formulas. Terms of type `Set` are meant to be data types. For example, we may construct a term $\mathbb{N} \to \mathbb{N}$, representing the type of functions from natural numbers to natural numbers, which has type `Set`. We can define a term representing the successor function with type $\mathbb{N} \to \mathbb{N}$. So $\mathbb{N} \to \mathbb{N}$ is simultaneously thought of as a term and a type. Hence `Set` is a type of a type.

A context is a list of type declarations, written $e_1 : e_2$, and definitions, written $x := e_1 : e_2$ (identifier x has value e_1 of type e_2). We write [ ] for the empty context and $\Gamma :: (t_1 : t_2)$ for adding an element to the context. It can be useful to have two contexts available, one for local scope and one for global scope, but we will only use a single context as it is sufficient for the contents of this discussion.

Definition: A type T is *inhabited* in a context Γ if there exists t such that $\Gamma \vdash t : T$.

Definition: If T : `Set`, then T is a *specification*. If $t : T$, then t is a *realization* of the specification T.

Definition: If T : `Prop`, then T is a formula. If $t : T$, then T is a *theorem* and t is a *proof* of theorem T.

Term construction rules will be presented as inference rules which contain typing judgments of the form $\Gamma \vdash t_1 : t_2$. This says that in context Γ, the term t_1 has type t_2.

Consider the following product type construction rule from CoC:

$$\frac{\Gamma \vdash T : s \quad \Gamma :: (t : T) \vdash U : s'}{\Gamma \vdash \forall (t : T), U : s''} \; \textit{Prod(s, s', s'')}$$

The possible tuples (s, s', s'') that we allow can give us simple types, dependent types, and higher-order types and cause CoC to have a very expressive type system. For example, if we require s is `Set` and s' is `Prop`, then the rule *Prod(s,s',s')* is a rule to build propositions with universal quantification:

$$\frac{\Gamma \vdash T : \texttt{Set} \quad \Gamma :: (t : T) \vdash U : \texttt{Prop}}{\Gamma \vdash \forall t : T, U : \texttt{Prop}} \; \textit{Prod(Set, Prop, Prop)}$$

2.2 Simply Typed Lambda Calculus

The first types we will construct are simple types. This includes atomic types, referred to by their identifier (e.g. $\mathbb{N}$, $\mathbb{Z}$), and arrow types $A \to B$ where A and B are simple types. Observe that by the Curry-Howard correspondence we may view $A \to B$ as either a specification (function type) or a theorem (implication), depending on the sort of the arrow type. In either case, if $t : A \to B$, then t maps either data of type A, or proofs of A, to data of type B, or proofs of B, respectively.

Any type declaration in the context can be proven. For example, if we have defined $x : T$ in Γ, then $\Gamma \vdash x : T$. This is illustrated by the following rule:

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T} \; \textit{Var}$$

Standard rules are also available for function application and abstraction:

$$\frac{\Gamma \vdash t_1 : A \to B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 \; t_2 : B} \; \textit{App-ST}$$

$$\frac{\Gamma :: (x : A) \vdash t : B}{\Gamma \vdash \lambda(x : A).t : A \to B} \; \textit{Lam-ST}$$

The rule *Lam-ST* makes it possible to construct a term of type $A \to B$, but we also need to be able to build such a realization. This is accomplished via the product rule for simple types:

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : s}{\Gamma \vdash A \to B : s} \; \textit{Prod-ST}$$

where $s \in \{\texttt{Set}, \texttt{Prop}\}$. Notice *Prod-ST* is the same rule as *Prod*(s, s, s) where $s \in \{\texttt{Set}, \texttt{Prop}\}$ and $A \to B$ is shorthand for $\forall (x : A), B$ (given x does not occur in B).

2.3 Dependent Types

A dependent type is a term of Cic that depends on a choice of a realization of a specification (for parametric types) or a choice of proof (in the logical case).

Example 1: The type of a tuple of size n depends on the value of n.

Example 2: The type of a characteristic function $\chi_S : S \to \texttt{Prop}$ depends on the set S.

Definition: A term of the form $\forall t : T, U$ is called a *dependent product.*

If $s \in \{\texttt{Set}, \texttt{Prop}\}$, then the rule *Prod(s, Type$_i$, Type$_i$)* allows us to build a term $\forall(t : T), U$, where t may occur freely in U. If t is not a free variable of U, then $\forall(t : T), U$ is a non-dependent type abbreviated as $T \to U$ (since for any x of type T, $U \equiv U[x/t]$). The application and abstraction rules shown for simply-typed λ-calculus may be generalized:

$$\frac{\Gamma \vdash t_1 : \forall(x : A), B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : B[t_2/x]}\ App$$

$$\frac{\Gamma :: (x : A) \vdash t : B}{\Gamma \vdash \lambda(x : A).t : \forall(x : A), B}\ Lam$$

The *Lam* rule builds terms whose type is a dependent product, and *App* builds applications.

2.3.1 Predicates

Let s be Set and U be Prop. Then the rule *Prod(Set, Type$_0$, Type$_0$)* becomes

$$\frac{\Gamma \vdash T : \texttt{Set} \quad \Gamma :: (t : T) \vdash \texttt{Prop} : \texttt{Type}_0}{\Gamma \vdash \forall(t : T), \texttt{Prop} : \texttt{Type}_0}\ Prod(\texttt{Set}, \texttt{Type}_0, \texttt{Type}_0)$$

We can use this rule to build the type of unary predicates $T \to \texttt{Prop}$ and extend this to n-ary predicates by repeated uses of the rule.

2.4 Higher-Order Types

So far there is no method to build dependent products with quantification over higher-order types; this is given by the rule *Prod(Type$_i$, Type$_j$, Type$_k$)* where $i \leq j$ and $j \leq k$.

2.4.1 Polymorphism

Let T be Set, then we can use the *Prod(Type$_0$, Type$_0$, Type$_0$)* rule to build specifications of polymorphic functions:

$$\frac{\Gamma \vdash \texttt{Set} : \texttt{Type}_0 \quad \Gamma :: (t : \texttt{Set}) \vdash U : \texttt{Type}_0}{\Gamma \vdash \forall (t : \texttt{Set}), U : \texttt{Type}_0} \; \textit{Prod(}\texttt{Type}_0\textit{, }\texttt{Type}_0\textit{, }\texttt{Type}_0\textit{)}$$

For example, it can be shown that $\Gamma \vdash \forall t : \texttt{Set}, (t \to t) \to \mathbb{N} \to t \to t : \texttt{Type}_0$ holds in CIC. This means we can specify a function for the n-th iterate of a unary function on some type t with sort `Set`.

2.4.2 Type Operators

The use of the *Prod(*$\texttt{Type}_i$*,* $\texttt{Type}_j$*,* $\texttt{Type}_k$*)* rule in building higher-order types is what also allows us to express type operators in CIC. For example, let T be `Prop`, then we have a rule to build the type of logical connectives.

$$\frac{\Gamma \vdash \texttt{Prop} : \texttt{Type}_0 \quad \Gamma :: (t : \texttt{Prop}) \vdash U : \texttt{Type}_0}{\Gamma \vdash \forall t : \texttt{Prop}, U : \texttt{Type}_0} \; \textit{Prod(}\texttt{Type}_0\textit{, }\texttt{Type}_0\textit{, }\texttt{Type}_0\textit{)}$$

The infix binary connectives representing "or" and "and" can be declared as $\vee : \texttt{Prop} \to \texttt{Prop} \to \texttt{Prop}$ and $\wedge : \texttt{Prop} \to \texttt{Prop} \to \texttt{Prop}$, respectively. Then we can construct dependent products corresponding to natural deduction rules using these types.

2.5 Interactive Proving in Coq

As described in Section 2.1, to prove a statement $P : \texttt{Prop}$, we define $t : P$. Then t, a lambda term, is a proof of P. As an alternative to stating t and allowing the type checker to verify that t is a proof of P, Coq provides an interactive proof mode where *tactics* are used to interactively work through a proof. These proofs start with the theorem statement as the goal and work backward reducing to subgoals at each step and eventually to axioms. Once all goals have been discharged, the system builds the proof term t. The names of some of these tactics will be mentioned throughout this document, so we collect descriptions of the relevant tactics here. Included in this list are other Coq commands and tacticals (i.e. operators that take tactic arguments to build a tactic). More information can be found in the Coq Reference Manual [20].

2.5.1 Some Coq Tactics, Tacticals, and Commands

`intros`

introduces variables and assumptions from the goal to the context of assumptions; a meta-level backward reasoning step of implication introduction (arguments optional)

induction
: applies the appropriate induction principle for the type we induct over (one argument)

reflexivity
: solves a goal when it is an equality with both sides equal

simpl
: applies $\beta\iota$-reduction then expands constants from their definitions and again tries $\beta\iota$-reduction

rewrite
: rewrites from an equality that is either an assumption, a local definition, or a theorem; optionally use either `<-` or `->` to give the rewrite direction

constructor
: applies an appropriate constructor for the type of the goal without having to name the constructor; constructors are the names of the clauses of an inductive definition

apply
: used either for forward reasoning on assumptions in the context, or backward reasoning on a goal, also known as *backchaining* (one required argument, one optional argument)

inversion
: all conditions derived for each constructor of the type of the argument are new assumptions; for each constructor matched, the proof has one new subgoal with the premises of that clause as new assumptions in the context

assumption
: used when the goal matches an assumption to complete the proof of a goal

auto
: attempts to prove the goal automatically using results in a hints database

Hint
: a Coq command; using `Hint Resolve` *theorem_name* adds *theorem_name* to a list of hints used by `auto`

try
: a tactical that tries to apply the tactic given as an argument and if it fails does not cause an error

;
applies tactics in sequence

Many of the tactics can be replaced with the same tactic name prefixed with the letter e (e.g. `eapply`). This provides placeholders of appropriate type that act as logical variables that can be filled in by unification. They are used where we would otherwise need to provide a witness in cases of application as backward reasoning on the goal.

2.5.2 Proof State

When working through a proof, the state of the proof is changing. In each proof state we can have assumptions in the context of assumptions (a collection of type names and their types) and a goal. It is possible to have multiple incomplete proof states at one time. Visually we will write a proof state in a vertical form with the assumptions in the context above a horizontal line and the goal below. For example:

$$\begin{array}{c} H_1 : P_1 \\ \vdots \\ H_k : P_k \\ \hline G \end{array}$$

where H_1, … , H_k are variables names with types P_1, … , P_k, respectively. Each P_i may be a data type or a formula. The goal G is a formula.

Unlike in Coq, when we have multiple goals with the same context of assumptions we will write them all below the horizontal line, separated by commas.

$$\begin{array}{c} H_1 : P_1 \\ \vdots \\ H_k : P_k \\ \hline G_1, \ldots, G_j \end{array}$$

We also sometimes refer to the goal as a subgoal as a reminder that it was acquired from a previous goal and there may be other subgoals.

2.5.3 Example Proof

Note that in Coq, conjunction is defined by the clause

$$\forall (P\ Q : Prop), P \to Q \to P \wedge Q$$

To illustrate the Coq interactive theorem proving system, we will prove the conjunction elimination rule $\frac{P \wedge Q}{P} \wedge_{e_1}$. Initially the context of assumptions is empty.

$$\forall (P\ Q : \texttt{Prop}), P \wedge Q \rightarrow P$$

We backchain with a meta-level use of implication introduction using `intros`.

$$\begin{array}{l} P : \texttt{Prop} \\ Q : \texttt{Prop} \\ H : P \wedge Q \\ \hline P \end{array}$$

Now we use `inversion` on H. Since assumption H is a witness of the conjunction $P \wedge Q$, by the definition of conjunction it must be the case that we can also assume both P and Q.

$$\begin{array}{l} P : \texttt{Prop} \\ Q : \texttt{Prop} \\ H : P \wedge Q \\ H_1 : P \\ H_2 : Q \\ \hline P \end{array}$$

Now the goal matches H_1 and we finish this proof with `assumption`.

2.6 Induction in Coq

CIC extends CoC by adding inductive type definitions. An inductive type is a type with constructors that may take arguments of that type, so it is self-referential. For example, the type of natural numbers is inductive and can be defined in Coq as:

```
Inductive nat : Set :=
| Z : nat
| S : nat -> nat.
```

We declare that we are defining an inductive type with the keyword `Inductive`. The name of the type is `nat` and its type is `Set`. This inductive type has two *constructors*, `Z` (to represent zero) and `S` (to represent the successor function). We can understand

this type as saying that any natural number can be constructed either as zero or the successor of some other natural number.

From an inductive type, Coq automatically generates an *induction principle* whose target type is `Prop`. To prove a property of all elements of a type, proofs using these induction principles have one subcase for each constructor of the type. The induction principle for `nat` is

$$\begin{aligned}&\texttt{nat\_ind} : \forall(P : \texttt{nat} \to \texttt{Prop}),\\ &\quad (*\texttt{Z}*) \qquad P\ \texttt{Z} \to\\ &\quad (*\texttt{S}*) \qquad (\forall(m : \texttt{nat}), P\ m \to P\ (\texttt{S}\ m)) \to\\ &\qquad\quad \forall(n : \texttt{nat}), P\ n\end{aligned}$$

where P is the property to be proven of all natural numbers. We sometimes refer to P as the *induction property*.

Constructors that have recursive occurrences of the type being defined will have corresponding induction subcases with *induction hypotheses* . In the case of `nat`, the constructor `S` requires a `nat` argument. Notice the corresponding induction subcase is to prove $\forall(m : \texttt{nat}), P\ m \to P\ (\texttt{S}\ m)$. The variable $m : \texttt{nat}$ and the premise $H : P\ m$ will be introduced into the context of the proof state. So we will then have assumption $H : P\ m$ which we call an induction hypothesis.

2.6.1 Example Proof

We will see how to prove the statement $\forall(n : \texttt{nat}), n = n + \texttt{Z}$ by induction interactively in Coq, also pointing out the induction property. Initially the context is empty.

$$\frac{}{\forall(n : \texttt{nat}), n = n + \texttt{Z}}$$

The tactic `induction` n is used to backchain with the induction principle for natural numbers. This proof has two subcases, one corresponding to each constructor of `nat`. These are to prove $P\ \texttt{Z}$ and $\forall(m : \texttt{nat}), P\ m \to P\ (\texttt{S}\ m)$ where

$$P \coloneqq \lambda(n : \texttt{nat})\ .\ n = n + \texttt{Z}$$

is the induction property.

We will first prove $P\ \texttt{Z}$ (usually called the "base case"):

$$\frac{}{\texttt{Z} = \texttt{Z} + \texttt{Z}}$$

This is done by reducing $\texttt{Z}+\texttt{Z}$ to `Z` by the definition of $+$ and then with `reflexivity`.

To complete this proof we need to show $\forall(m : \texttt{nat}), P\ m \rightarrow P\ (\texttt{S}\ m)$ (the "inductive step"):

$$\forall(m : \texttt{nat}), m = m + \texttt{Z} \rightarrow \texttt{S}\ m = (\texttt{S}\ m) + \texttt{Z}$$

We make introductions into the context with `intros`.

$$\begin{array}{l} m : \texttt{nat} \\ H : m = m + \texttt{Z} \\ \hline \texttt{S}\ m = (\texttt{S}\ m) + \texttt{Z} \end{array}$$

The right side of the goal equality can be reduced by `simpl`, again based on the definition of $+$.

$$\begin{array}{l} m : \texttt{nat} \\ H : m = m + \texttt{Z} \\ \hline \texttt{S}\ m = \texttt{S}\ (m + \texttt{Z}) \end{array}$$

Now we can use `rewrite <- H` to replace $m + \texttt{Z}$ with m on the right side of the goal equality.

$$\begin{array}{l} m : \texttt{nat} \\ H : m = m + \texttt{Z} \\ \hline \texttt{S}\ m = \texttt{S}\ m \end{array}$$

The goal is an equality with both sides equal, so this proof is finished with `reflexivity`.

2.6.2 Mutually Inductive Types

A type may be built using types that are already defined. When two types have dependencies on each other, they cannot both be defined before the other. In this case we define a mutually inductive type. An example of where this is useful is in defining two types `even` and `odd` which are unary relations to identify even and odd natural numbers, respectively. In Coq these can be defined as:

```
Inductive even : nat -> Prop :=
| e_Z : even Z
| e_S : forall (n : nat), odd n -> even (S n)
with odd : nat -> Prop :=
| o_S : forall (n : nat), even n -> odd (S n).
```

Intuitively this says that Z (meaning zero) is even and the successor of any odd number is even. Also, the successor of any even number is odd.

Coq automatically generates an induction principle for each of these types. For `even`, this is

$$\begin{array}{l}\texttt{even\_ind} : \forall(P : \texttt{nat} \to \texttt{Prop}),\\ \quad (*\texttt{e\_Z}*) \qquad (P\ \texttt{Z}) \to\\ \quad (*\texttt{e\_S}*) \qquad (\forall(n : \texttt{nat}), \texttt{odd}\ n \to P\ (S\ n)) \to\\ \qquad\qquad \forall(n : \texttt{nat}), \texttt{even}\ n \to P\ n\end{array}$$

where P is the induction property for even natural numbers. Notice that a proof using this induction principle will have one subcase for each constructor of `even`. Also, in the case corresponding to the constructor `e_S`, there is no induction hypothesis about the premise `odd` n. So for some types and some theorems to prove, the generated induction principle is insufficient.

The command `Scheme` may be used to generate induction principles over mutually inductive types. These induction principles will have subcases for every constructor in every type in the mutually inductive type. Continuing the example above, we can get the following mutual induction principle over `even`:

$$\begin{array}{l}\texttt{even\_mutind} : \forall(P_1\ P_2 : \texttt{nat} \to \texttt{Prop}),\\ \quad (*\texttt{e\_Z}*) \qquad (P_1\ \texttt{Z}) \to\\ \quad (*\texttt{e\_S}*) \qquad (\forall(n : \texttt{nat}), \texttt{odd}\ n \to P_2\ n \to P_1\ (S\ n)) \to\\ \quad (*\texttt{o\_S}*) \qquad (\forall(n : \texttt{nat}), \texttt{even}\ n \to P_1\ n \to P_2\ (S\ n)) \to\\ \qquad\qquad \forall(n : \texttt{nat}), \texttt{even}\ n \to P_1\ n\end{array}$$

This induction principle has more cases but provides more powerful assumptions in each inductive case. Notice that now the subcase corresponding to the constructor `e_S` also has the induction hypothesis $P_2\ n$ and we also have a subcase for the constructor `o_S`.

2.7 Conclusion

The Cic inference system implements a type checker, so it can be used both for proof verification and checking that a function satisfies its specification (i.e. it is a realization of the required type). It can also be used to construct a proof. By appropriately instantiating the *Prod* rule, the system is made more expressive as a functional language and theorem proving system while still maintaining many desirable properties including strong normalization (or at least weak normalization in some extensions) and consistency.

Coq can be used to prove formalized statements interactively. A large library of tactics are available to assist in proof development. Since Coq is an implementation of CIC, it is possible to define inductive types and then prove statements by induction over these types using automatically generated induction principles.

The upcoming presentation will make use of all of the concepts just presented: the rich type system of CIC, interactive proofs, and inductive types, culminating in proofs by structural induction over mutually inductive dependent types.

Chapter 3

Hybrid

Hybrid is a two-level logical framework implementing *higher-order abstract syntax* (HOAS). The purpose of such systems is to reason efficiently about properties of formal systems (e.g. programming languages or logics) that have common structures.

In this chapter each layer of the system will be explored to provide more intuition on how it is constructed and used. This explanation will be driven by an analogy, for use as an aid to both memory and understanding of the system.

The orientation of the layers is as in Figure 3.1 from [6] describing an earlier version of Hybrid. We will first consider the top layer, the object logic, in Section 3.1 with an example to motivate what we are trying to accomplish. Next we will consider each layer bottom-up, beginning with the ambient logic in Section 3.2, then the higher-order abstract syntax layer in Section 3.3. Continuing up the stack we next come to the specification logic. Since much of the work presented later is on the implementation and metatheory of the specification logic required for our motivating example, we will not see details of the specification logic here. Rather, Section 3.4 will illustrate the benefit a specification logic adds to Hybrid and reinforce its necessity. This will be followed by another look at the object logic in Section 3.5, but this time

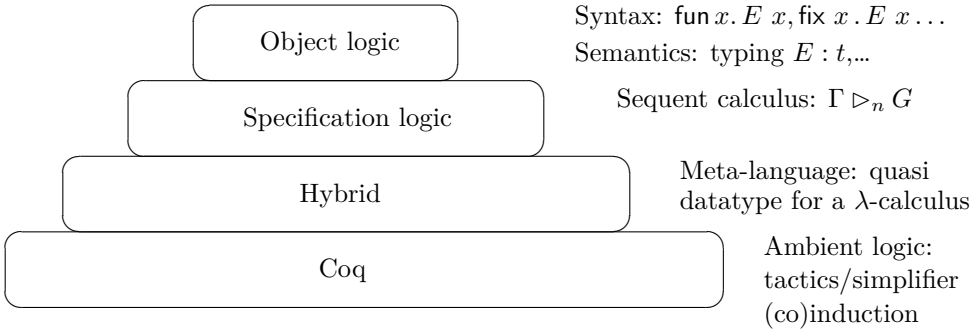

Figure 3.1: Architecture of the Hybrid system

we will be focusing on implementation details with the rest of the system in place. To conclude this chapter, Section 3.6 will compare Hybrid with alternative architectures for systems intended to reason about object logics using HOAS.

3.1 Object Logic (OL)

Suppose we wish to study flowers and create things with them. Then we need to be able to grow flowers.

Suppose we wish to prove something about a programming language or logic, the OL. This language will have rules expressing syntax and semantics that we need to encode in some proof assistant so that we can reason about it. It is also necessary to define the judgments of this language so that we can make claims about the OL.

3.1.1 Example OL

We can define a syntax and rules expressing direct and de Bruijn representations of untyped λ-terms. An example property we might want to prove is that these two representations are equivalent (or seen another way, to construct equivalent λ-terms in these different forms). Let n represent a natural number, x a variable, and e and d represent direct and de Bruijn representations, respectively. Then the following are grammars for these λ-terms:

$$e ::= x \mid \lambda x.e \mid e\ e$$
$$d ::= n \mid \lambda d \mid d\ d$$

There are three inference rules expressing equivalence of these two kinds of terms, one for each of application, abstraction, and variables, seen below. This logic has a judgment to say that λ-term e is equivalent to de Bruijn term d at depth n, written $e \equiv_n d$.

$$\frac{\Gamma \vdash e_1 \equiv_n d_1 \quad \Gamma \vdash e_2 \equiv_n d_2}{\Gamma \vdash e_1\ e_2 \equiv_n d_1\ d_2}\ \mathit{hodb\_app}$$

$$\frac{\Gamma, x \equiv_{n+k} k \vdash e \equiv_{n+1} d}{\Gamma \vdash \lambda x.e \equiv_n \lambda d}\ \mathit{hodb\_abs}$$

$$\frac{x \equiv_{n+k} k \in \Gamma}{\Gamma \vdash x \equiv_{n+k} k}\ \mathit{hodb\_var}$$

Notice that there is a binding operator in the conclusion of the rule *hodb_abs*. This observation will be important when we see how to represent untyped λ-terms using HOAS in Hybrid in Section 3.3 and then implement this OL in Section 3.5.

3.2 Ambient Logic

We can plant seeds in the ground and use the natural resources around us to reach our goal. The sun will provide energy and rain will give water.

The ambient logic (also known as the reasoning logic or the meta-meta-logic) is the layer of the system that everything else is defined in. It is an implementation of a logic and so has its own reasoning rules and allows us to define other reasoning systems within it. In our case, this is CIC and its implementation in Coq. This is the lowest reasoning level we consider carefully as part of our system.

Existing theorem proving systems are an ideal tool to allow a language and its judgments to be encoded without building extra infrastructure. This development is a Coq library so it is relatively easy to make modular updates to the system and to add new intermediate reasoning layers called specification logics, as will be explained in Section 3.4. Hybrid can also make use of the inductive and interactive reasoning tools of Coq as well as existing Coq libraries.

3.3 Representing Higher-Order Abstract Syntax (HOAS) in Hybrid

As our aspirations continue to grow, we find it difficult to scale up our flower production. When the rain doesn't fall as we require, we manually make up for the shortfall. The task of watering every plant every day is tedious. A dedicated plot of land with an organized arrangement and an irrigation system is a solution to this problem.

Many tedious computations are necessary for each encoding of an OL with binding structures. Since Hybrid is implemented in an ambient logic that is a typed λ-calculus, the technique of *higher-order abstract syntax* (HOAS) can be used for representing OL expressions. Using HOAS one can avoid implementing logic to reason about variable naming concepts such as fresh name generation and capture-avoiding substitution by encoding OL binding constructs as meta-level binding constructs, thus inheriting the meta-level solutions to these challenges. In addition, OL renaming and substitution are handled as meta-level α-conversion and β-reduction, respectively.

At this level we have a type `expr` (see Figure 3.2) used to build expressions of the OL as de Bruijn indices. A parameter `con` represents OL constants, to be defined for each OL. Note that `con` is an implicit parameter in the environment it is defined in; uses outside of this environment must explicitly state this parameter (i.e.

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

Figure 3.2: Terms in Hybrid

as `expr con`). We define `var` and `bnd` to be the natural numbers. Hybrid expressions (`VAR` i) and (`BND` j) represent object-level free and bound variables, respectively. The constructor `APP` is used to build applications and `ABS` to build abstractions in de Bruijn notation.

Object-level binding operators are encoded in HOAS using the Hybrid operator `lambda` : (`expr con` $\rightarrow$ `expr con`) $\rightarrow$ `expr con` which is the meta-level binder defined in the Hybrid library. When using it to encode HOAS, the expanded definition is the underlying de Bruijn notation using only the constructors of `expr`. Although a Hybrid user never sees the expanded form and only works at the HOAS level. As an example, consider the untyped λ-term ($\lambda x.\lambda y.x\ y$). We can represent this using the HOAS of Hybrid as (`lambda` x.(`lambda` $y.x\ y$)) which expands to `ABS` (`ABS` (`APP` (`BND` 1) (`BND` 0))).

3.4 Specification Logic (SL)

Not all flowers will grow in the same conditions. Given any plot of land, there are many plants that will not grow there because they need specific nutrients in their soil. We can create different soil mixes depending on the needs of different classes of flowers.

There are OL judgments that we cannot encode in an inductive type in Coq. One example is a HOAS encoding of inference rules assigning simple types to λ-expressions. The rule for typing abstractions contains negative occurrences of this judgment (see the underlined judgment in Figure 3.3), which is not allowed by the Coq type system. As a solution to this problem Hybrid is a two-level system, meaning an intermediate specification level is introduced between the OL encoding and the meta-levels.

Hybrid is a Coq library and as mentioned earlier, this architectural decision makes quick prototyping of SLs possible. Another important benefit is that one can choose the simplest specification logic necessary for the present task, or possibly a

$$\frac{\forall x, \underline{x : T} \to (E\ x) : T'}{(\lambda x\,.\,E\ x) : (T \to T')}\ tp\_abs$$

Figure 3.3: Typing of λ-calculus Abstractions

```
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 3.4: Type of SL Formulas

combination of more than one depending on the OL to be encoded. Judgments that can be defined inductively do not need to be defined in a SL. This may simplify proofs of OL properties as the user can avoid using a more complicated logic than necessary.

The two levels of the OL and SL interact through a parameter of the SL,

$$\texttt{prog} : \texttt{atm} \to \texttt{oo} \to \texttt{Prop},$$

which is used to encode inference rules for OL judgments (and thus define provability at the OL level). There are two arguments to `prog`; the first is the (atomic) inference rule conclusion of type `atm` and the second a formula of type `oo` representing the premise(s) of the rule.

In this implementation, the type `atm` is a parameter of the SL 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`) $\to$ `nat` $\to$ (`expr con`) relating the higher-order and de Bruijn encodings at a given depth.

The type `oo` is the type of goals and clauses in the SL. The definition of `oo` for the SL defined later is in Figure 3.4. The constant `atom` coerces an atom (a predicate applied to its arguments) to an SL formula. For any α of type `atm`, we may refer to (`atom` α) as an atomic formula. The constructor `Conj` represents conjunction and `Imp` is used to build implications. 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 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.

$$
\begin{aligned}
&\texttt{oo\_ind} : \forall(P : \texttt{oo} \to \texttt{Prop}),\\
(*\texttt{atom}*)\quad &(\forall(a : \texttt{atm}), P(\langle\ a\ \rangle)) \to\\
(*\texttt{T}*)\quad &(P\ \texttt{T}) \to\\
(*\texttt{Conj}*)\quad &(\forall(o_1 : \texttt{oo}), P\ o_1 \to \forall(o_2 : \texttt{oo}), P\ o_2 \to P\ (o_1 \& o_2)) \to\\
(*\texttt{Imp}*)\quad &(\forall(o_1 : \texttt{oo}), P\ o_1 \to \forall(o_2 : \texttt{oo}), P\ o_2 \to P\ (o_1 \longrightarrow o_2)) \to\\
(*\texttt{All}*)\quad &(\forall(o : \texttt{expr con} \to \texttt{oo}), (\forall(e : \texttt{expr con}), P\ (o\ e)) \to P\ (\texttt{All}\ o)) \to\\
(*\texttt{Allx}*)\quad &(\forall(o : \texttt{X} \to \texttt{oo}), (\forall(x : \texttt{X}), P\ (o\ x)) \to P\ (\texttt{Allx}\ o)) \to\\
(*\texttt{Some}*)\quad &(\forall(o : \texttt{expr con} \to \texttt{oo}), (\forall(e : \texttt{expr con}), P\ (o\ e)) \to P\ (\texttt{Some}\ o)) \to\\
&\forall(o : \texttt{oo}), P\ o
\end{aligned}
$$

Figure 3.5: Induction Principle for oo

We write $\langle\ \alpha\ \rangle$, $(\beta_1\ \&\ \beta_2)$, and $(\beta_1 \longrightarrow \beta_2)$ as notation for $(\texttt{atom}\ \alpha)$, $(\texttt{Conj}\ \beta_1\ \beta_2)$, and $(\texttt{Imp}\ \beta_1\ \beta_2)$, respectively. Formulas quantified by All are written $(\texttt{All}\ \beta)$ or $(\texttt{All}\ \lambda(x : \texttt{expr con})\ .\ \beta\ x)$. The latter is the η-long form with types included explicitly. The other quantifiers are treated similarly.

Note that in theorem statements written as inference rules we write β or δ for formulas (i.e. have type oo), and α for elements of type atm, possibly with subscripts. In the linear form of theorem statements and in proofs we 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.

The type oo is an inductive type, so Coq will automatically generate the induction principle shown in Figure 3.5 as discussed in Section 2.6. We can use this induction principle to prove a statement of the form $\forall(o : \texttt{oo}), P\ o$ for some $P : \texttt{oo} \to \texttt{Prop}$. This proof will have one subcase for each constructor of oo.

A Hybrid SL is defined as an inductive type in Coq to encode a sequent calculus. Each rule of the sequent calculus is represented by a constructor of the inductive type. The constructor name is the rule name and the type arrow is used for implication from premises to conclusion. The context of the sequent is defined to behave as a set of elements of type oo. We write Γ or c for contexts.

Since we explore the SL and proofs of its structural properties in detail later when describing the contributions of this research, we cut short the discussion here. For continuity in this chapter, some notation and the meaning of provability judgments of the SL are all we need now. We write $\Gamma \rhd \beta$ to say that β is provable in context Γ at the level of the SL. The symbol $\rhd$ is used as the SL sequent arrow.

3.5 Example OL Implementation

Now we can see how to encode our example syntax and judgments in Hybrid. Let `tm` represent the type of HOAS terms and `dtm` represent the type of de Bruijn terms. Since these are used to form OL expressions, `tm` and `dtm` are aliases for `expr con`. Before stating the implementation of the rules of the logic, we have to define the OL constants. For HOAS application and abstraction we have `hApp` : `tm` → `tm` → `tm` and `hAbs` : (`tm` → `tm`) → `tm`, respectively. HOAS variables are encoded as meta-level variables. For de Bruijn application, abstraction, and variables we have `dApp` : `dtm` → `dtm` → `dtm`, `dAbs` : `dtm` → `dtm`, and `dVar` : `nat` → `dtm`, respectively.

In Figure 3.6 the constants of the OL are defined in the inductive type `con`. We also have the definitions of OL applications and abstractions for the HOAS and de Bruijn forms of λ-terms in terms of the OL constants and HOAS application and `lambda` operator. Note that in Coq, `fun` is notation for abstractions. When we write Coq code we use this notation but when writing pretty-printed versions of the code we will use λ-calculus abstraction notation. For example, we often write Coq abstractions `fun x => f x` as $\lambda x.f\ x$ because the latter is often more readable in our discussions. In Figure 3.6 we can see the use of HOAS in the definition of `hAbs` where we use the Hybrid `lambda` operator.

The atomic judgment discussed for this example (equivalence between the two representations of lambda terms) is part of the inductive type `atm` defined below.

```
Inductive atm : Set :=
| hodb : tm -> nat -> dtm -> atm.
```

Now the type of SL formulas with all parameters filled in is `oo atm (expr con) X`.

The rules shown in Section 3.1 can now be defined in Hybrid using HOAS and a SL. More specifically, we can now define the inductive type `prog` in the formulas of the SL as shown in Figure 3.7. The Coq notation for $\langle\ a\ \rangle$ is `<<a>>`.

An example theorem for this encoding is to prove that the judgment `hodb` is deterministic in its first and third arguments (and thus the relational definition of the rules represents a function). To do this we want to prove the two theorems below (where = is equality in the ambient logic).

Theorem 3.1 (`hodb_det1`)**.**

$$\forall(\Gamma : \texttt{context})(e : \texttt{tm})(d_1\ d_2 : \texttt{dtm})(n : \texttt{nat}),$$
$$\Gamma \rhd \langle\ \texttt{hodb}\ e\ n\ d_1\ \rangle \rightarrow \Gamma \rhd \langle\ \texttt{hodb}\ e\ n\ d_2\ \rangle \rightarrow d_1 = d_2.$$

Theorem 3.2 (`hodb_det3`)**.**

$$\forall(\Gamma : \texttt{context})(e_1\ e_2 : \texttt{tm})(d : \texttt{dtm})(n : \texttt{nat}),$$
$$\Gamma \rhd \langle\ \texttt{hodb}\ e_1\ n\ d\ \rangle \rightarrow \Gamma \rhd \langle\ \texttt{hodb}\ e_2\ n\ d\ \rangle \rightarrow e_1 = e_2.$$

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

Definition hApp : tm -> tm -> tm :=
  fun (e1 : tm) =>
    fun (e2 : tm) =>
      APP (APP (CON hAPP) e1) e2.
Definition hAbs : (tm -> tm) -> tm :=
  fun (f : tm -> tm) =>
    APP (CON hABS) (lambda f).

Definition dApp : dtm -> dtm -> dtm :=
  fun (d1 : dtm) =>
    fun (d2 : dtm) =>
      APP (APP (CON dAPP) d1) d2.
Definition dAbs : dtm -> dtm :=
  fun (d : dtm) =>
    APP (CON cdABS) d.
Definition dVar : nat -> dtm :=
  fun (n : nat) =>
    (CON (dVAR n)).
```

Figure 3.6: Example OL: Encoding Syntax in Hybrid

```
Inductive prog : atm -> oo atm (expr con) X -> Prop :=
| hobd_app : forall (e1 e2 : tm) (n : nat) (d1 d2 : dtm),
   prog (hodb (hApp e1 e2) n (dApp d1 d2))
    (<<hodb e1 n d1>> & <<hodb e2 n d2>>)
| hodb_abs : forall (f : tm -> tm) (n : nat) (d : dtm),
   abstr f ->
   prog (hodb (hAbs f) n (dAbs d))
    (All (fun (x : tm) =>
     (Allx (fun (k : X) => <<hodb x (n + k) (dVar k)>>)) --->
       <<hodb (f x) (n + 1) d>>)).
```

Figure 3.7: Example OL: Encoding OL Inference Rules in Hybrid

3.6 Comparison to Other Architectures

> *Our approach to growing flowers is not the only solution. One alternative is to build a factory specializing in the production of flowers. This would give us full control over lighting, water, and soil composition; but the startup costs are high and modifications can be prohibitively expensive.*

Other systems use HOAS for encoding and reasoning about OLs with binders but different choices are made in the implementation of these systems. We will briefly look at the features of the two most closely related systems, Abella [9] and Beluga [18], and compare these systems to Hybrid. These three systems, along with Twelf [19], are compared in detail using benchmark problems in [8].

One feature that sets Hybrid apart from these systems is that Hybrid is a library in an existing theorem proving system while Abella, Beluga, and Twelf are special-purpose theorem proving systems built for reasoning about OLs using HOAS. Using an existing theorem proving system means we can trust the proofs without having to develop extra infrastructure. These proofs can be independently checked because a proof term is a λ-term; a proof check is a type check in the Calculus of Constructions, a trusted and well studied theoretical foundation for our work. The trade-off is less control over the reasoning logic of Hybrid and more levels of encoding.

Abella

- $\mathcal{G}$ as reasoning logic
- two-level logical framework
- tactic-based interactive theorem prover
- does not have proof terms on completion of proof, so we have to trust the system
- has ∇-quantifier, a new specialized quantifier providing better direct reasoning about OLs, some of which cannot be done in Hybrid until we implement ∇

Beluga

- contextual LF as reasoning logic
- a type theory instead of a logic, specialized for ease of reasoning with HOAS
- supports reasoning over contexts
- some metatheory about contexts is implicit, meaning built-in to the implementation instead of admissible as rules

Part II

Contributions

Chapter 4

Specification Logic

The first stage of the contributions outlined in this thesis is representing a specification logic to increase the reasoning power of Hybrid. The new specification logic for Hybrid is based on hereditary Harrop formulas. 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 (see Figure 3.4). Types are built from the primitive types and the function arrow $\to$ as usual. Logical connectives and quantifiers are introduced as constants with their corresponding types as in [4]. For example, conjunction has type $\texttt{oo} \to \texttt{oo} \to \texttt{oo}$ and the quantifiers have type $(\tau \to \texttt{oo}) \to \texttt{oo}$, with some restrictions on τ described below. Predicates are function symbols whose target type is `oo`. Following [12], 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 $\tau_1 \to \tau_2$ where both τ_1 and τ_2 are primitive types. We note that unlike in all previous SLs for Hybrid there is no restriction on the implicational complexity (see [6]).

The specification logic presented in this chapter is a sequent calculus implemented as an inductive type in Coq. Section 4.1 describes how contexts are defined for this SL. Section 4.2 presents the Coq implementation of the SL based on hereditary Harrop formulas and we see how to prove properties of this SL by structural induction in Section 4.3.

4.1 Contexts in Coq

The type `context` represents contexts of assumptions in sequents and is defined using the Coq `ensemble` library as `ensemble oo` since we want contexts to behave as sets with elements of type `oo`. In proofs of some context lemmas stated below we use the `ensemble` extensional equality axiom:

$$\texttt{Extensionality\_Ensembles} : \forall(E_1\ E_2 : \texttt{ensemble}), (\texttt{Same\_set}\ E_1\ E_2) \rightarrow E_1 = E_2$$

where `Same_set` is defined in the `Ensemble` library. We use $o \in c$ as notation for `elem` o c which means formula o is an element of context c. Context subset, written $\Gamma_1 \subseteq \Gamma_2$, is defined as $\forall(o : \texttt{oo}), o \in \Gamma_1 \rightarrow o \in \Gamma_2$.

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 context lemmas below are proven as part of this work and are used in later proofs in this thesis. Note that all variables are externally quantified and each occurrence of β and Γ, possibly with subscripts, has type `oo` and `context`, respectively.

Lemma 4.1 (`elem_inv`)**.**

$$\frac{\beta_1 \in (\Gamma, \beta_2)}{(\beta_1 \in \Gamma) \vee (\beta_1 = \beta_2)}$$

Lemma 4.2 (`elem_sub`)**.**

$$\frac{\beta_1 \in \Gamma}{\beta_1 \in (\Gamma, \beta_2)}$$

Lemma 4.3 (`elem_self`)**.**

$$\frac{}{\beta \in (\Gamma, \beta)}$$

Lemma 4.4 (`elem_rep`)**.**

$$\frac{\beta_1 \in (\Gamma, \beta_2, \beta_2)}{\beta_1 \in (\Gamma, \beta_2)}$$

Lemma 4.5 (`context_swap`)**.**

$$\frac{}{(\Gamma, \beta_1, \beta_2) = (\Gamma, \beta_2, \beta_1)}$$

Lemma 4.6 (`context_sub_sup`)**.**

$$\frac{\Gamma_1 \subseteq \Gamma_2}{(\Gamma_1, \beta) \subseteq (\Gamma_2, \beta)}$$

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

$$\frac{\Gamma \rhd G_1 \quad \Gamma \rhd G_2}{\Gamma \rhd G_1 \,\&\, G_2}\ g\_and \qquad \frac{\Gamma\,,\, D \rhd G}{\Gamma \rhd D \longrightarrow G}\ g\_imp \qquad \frac{\mathtt{proper}\ E \quad \Gamma \rhd G\, E}{\Gamma \rhd \mathtt{Some}\ G}\ g\_some$$

$$\frac{\forall (E : \mathtt{expr\ \ con}), (\mathtt{proper}\ E \rightarrow \Gamma \rhd G\, E)}{\Gamma \rhd \mathtt{All}\ G}\ g\_all \qquad \frac{\forall (E : \mathtt{X}), (\Gamma \rhd G\, E)}{\Gamma \rhd \mathtt{Allx}\ G}\ g\_allx$$

Figure 4.1: Goal-Reduction Rules, $\mathtt{grseq} : \mathtt{context} \rightarrow \mathtt{oo} \rightarrow \mathtt{Prop}$

4.2 Hereditary Harrop Specification Logic in Coq

We follow the description in [21] of a specification logic for Abella based on hereditary Harrop formulas. The inference rules of HH are defined 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.

Figures 4.1 and 4.2 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 inductive types `grseq` and `bcseq`. Recall that 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 $\forall (E : \mathtt{expr\ con})(H : \mathtt{proper}\ E), (\Gamma \rhd G\, E)$.

Goal-reduction sequents have type $\mathtt{grseq} : \mathtt{context} \rightarrow \mathtt{oo} \rightarrow \mathtt{Prop}$, and we write $\Gamma \rhd \beta$ as notation for $\mathtt{grseq}\ \Gamma\ \beta$. Backchaining sequents have type $\mathtt{bcseq} : \mathtt{context} \rightarrow \mathtt{oo} \rightarrow \mathtt{atm} \rightarrow \mathtt{Prop}$ and we write $\Gamma, [\beta] \rhd \alpha$ as notation for $\mathtt{bcseq}\ \Gamma\ \beta\ \alpha$, 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

$$\frac{}{\Gamma, [\langle\ A\ \rangle] \rhd A}\ b\_match \qquad \frac{\Gamma, [D_1] \rhd A}{\Gamma, [D_1 \,\&\, D_2] \rhd A}\ b\_and_1 \qquad \frac{\Gamma, [D_2] \rhd A}{\Gamma, [D_1 \,\&\, D_2] \rhd A}\ b\_and_2$$

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

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

Figure 4.2: Backchaining Rules, $\texttt{bcseq} : \texttt{context} \to \texttt{oo} \to \texttt{atm} \to \texttt{Prop}$

from Figure 4.1 with all quantifiers explicit is

$$\forall (\Gamma : \texttt{context})(D : \texttt{oo})(A : \texttt{atm}), D \in \Gamma \to \Gamma, [D] \rhd A \to \Gamma \rhd \langle\ A\ \rangle$$

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 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* in Figure 4.1 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 $\texttt{atm} \to \texttt{oo} \to \texttt{Prop}$, and represent the inference rules of the OL (this is discussed further in Section 3.4). The rule *g_prog* 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 $(\texttt{prog}\ A\ G)$ 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 of Figure 4.2 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.

4.3 Mutual Structural Induction

The theorem statements in this thesis all have the form

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

where P_1 : $\texttt{context} \to \texttt{oo} \to \texttt{Prop}$ and P_2 : $\texttt{context} \to \texttt{oo} \to \texttt{atm} \to \texttt{Prop}$ are predicates extracted from the statement to be proven. The `Scheme` command provides an induction principle to allow us to prove statements of the above form by mutual structural induction.

To prove such a statement 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 the sequent mutual induction principle in Figure 4.3, 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 technique. After backchaining over the induction principle, the 15 subcases are generated. As an aside, externally quantified variables in each antecedent can be introduced to the context of assumptions of the proof state and are then considered *signature variables*. For example, the subcase for *g_prog* will have signature variables c : `context`, o : `oo`, and a : `atm`.

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. For example, $o \in c$ 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 $(\forall(x_1 : T_1) \cdots (x_n : T_n), P_1\ \Gamma\ \beta)$. For example, $\forall(e : \texttt{expr con}), \texttt{proper}\ e \to c \rhd o\ e$ and $\forall(e : \texttt{expr con}), \texttt{proper}\ e \to P_1\ c\ (o\ e)$ 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 $(\forall(x_1 : T_1) \cdots (x_n : T_n), P_2\ \Gamma\ \beta\ \alpha)$. For example, $c, [o_2] \rhd a$ and $(P_2\ c\ o_2\ a)$ from the *b_imp* rule.

$$
\begin{array}{rl}
\texttt{seq\_mutind} : \forall (P_1 : \texttt{context} \to \texttt{oo} \to \texttt{Prop})(P_2 : \texttt{context} \to \texttt{oo} \to \texttt{atm} \to \texttt{Prop}), & \\
(*g\_prog*) & (\forall (c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), \\
 & \quad a :\!\!- \; o \to c \rhd o \to P_1\ c\ o \to P_1\ c\ \langle\ a\ \rangle) \to \\
(*g\_dyn*) & (\forall (c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), \\
 & \quad o \in c \to c, [o] \rhd a \to P_2\ c\ o\ a \to P_1\ c\ \langle\ a\ \rangle) \to \\
(*g\_tt*) & (\forall (c : \texttt{context}), P_1\ c\ \texttt{T}) \to \\
(*g\_and*) & (\forall (c : \texttt{context})(o_1\ o_2 : \texttt{oo}), \\
 & \quad c \rhd o_1 \to P_1\ c\ o_1 \to c \rhd o_2 \to P_1\ c\ o_2 \to P_1\ c\ (o_1 \& o_2)) \to \\
(*g\_imp*) & (\forall (c : \texttt{context})(o_1\ o_2 : \texttt{oo}), \\
 & \quad c, o_1 \rhd o_2 \to P_1\ (c, o_1)\ o_2 \to P_1\ c\ (o_1 \longrightarrow o_2)) \to \\
(*g\_all*) & (\forall (c : \texttt{context})(o : \texttt{expr con} \to \texttt{oo}), \\
 & \quad (\forall (e : \texttt{expr con}), \texttt{proper}\ e \to c \rhd o\ e) \to \\
 & \quad (\forall (e : \texttt{expr con}), \texttt{proper}\ e \to P_1\ c\ (o\ e)) \to P_1\ c\ (\texttt{All}\ o)) \to \\
(*g\_allx*) & (\forall (c : \texttt{context})(o : \texttt{X} \to \texttt{oo}), \\
 & \quad (\forall (e : \texttt{X}), c \rhd o\ e) \to (\forall (e : \texttt{X}), P_1\ c\ (o\ e)) \to P_1\ c\ (\texttt{Allx}\ o)) \to \\
(*g\_some*) & (\forall (c : \texttt{context})(o : \texttt{expr con} \to \texttt{oo})(e : \texttt{expr con}), \\
 & \quad \texttt{proper}\ e \to c \rhd o\ e \to P_1\ c\ (o\ e) \to P_1\ c\ (\texttt{Some}\ o)) \to \\
(*b\_match*) & (\forall (c : \texttt{context})(a : \texttt{atm})), c, [\langle\ a\ \rangle] \rhd a) \\
(*b\_and_1*) & (\forall (c : \texttt{context})(o_1\ o_2 : \texttt{oo})(a : \texttt{atm}), \\
 & \quad c, [o_1] \rhd a \to P_2\ c\ o_1\ a \to P_2\ c\ (o_1 \& o_2)\ a) \to \\
(*b\_and_2*) & (\forall (c : \texttt{context})(o_1\ o_2 : \texttt{oo})(a : \texttt{atm}), \\
 & \quad c, [o_2] \rhd a \to P_2\ c\ o_2\ a \to P_2\ c\ (o_1 \& o_2)\ a) \to \\
(*b\_imp*) & (\forall (c : \texttt{context})(o_1\ o_2 : \texttt{oo})(a : \texttt{atm}), \\
 & \quad c \rhd o_1 \to P_1\ c\ o_1 \to c, [o_2] \rhd a \to P_2\ c\ o_2\ a \to \\
 & \quad P_2\ c\ (o_1 \longrightarrow o_2)\ a) \to \\
(*b\_all*) & (\forall (c : \texttt{context})(o : \texttt{expr con} \to \texttt{oo})(a : \texttt{atm})(e : \texttt{expr con}), \\
 & \quad \texttt{proper}\ e \to c, [o\ e] \rhd a \to P_2\ c\ (o\ e)\ a \to P_2\ c\ (\texttt{All}\ o)\ a) \to \\
(*b\_allx*) & (\forall (c : \texttt{context})(o : \texttt{X} \to \texttt{oo})(a : \texttt{atm})(e : \texttt{X}), \\
 & \quad c, [o\ e] \rhd a) \to P_2\ c\ (o\ e)\ a \to P_2\ c\ (\texttt{Allx}\ o)\ a \to \\
(*b\_some*) & (\forall (c : \texttt{context})(o : \texttt{expr con} \to \texttt{oo})(a : \texttt{atm}), \\
 & \quad (\forall (e : \texttt{expr con}), \texttt{proper}\ e \to c, [o\ e] \rhd a) \to \\
 & \quad (\forall (e : \texttt{expr con}), \texttt{proper}\ e \to P_2\ c\ (o\ e)\ a) \to P_2\ c\ (\texttt{Some}\ o)\ a) \to \\
 & (\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{array}
$$

Figure 4.3: SL Sequent Mutual Induction Principle

- If the rule conclusion is a goal-reduction sequent of the form $\Gamma \rhd \beta$, then the subcase goal is $P_1\ \Gamma\ \beta$. For example, $(P_1\ c\ \langle\ a\ \rangle)$ 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 $P_2\ \Gamma\ \beta\ \alpha$. For example, $(P_2\ c\ (o_1 \longrightarrow o_2)\ a)$ from the *b_imp* rule.

Assumptions in the second and third bullets above that contain P_1 or P_2 are induction hypotheses. Implicit in the 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). As a trivial example, if $P_1\ \Gamma\ \beta$ is $\forall(\delta : \texttt{oo}), \beta \in \Gamma \rightarrow \beta \in (\Gamma, \delta)$, then we can introduce δ into the context as a new signature variable and $\beta \in \Gamma$ as a new assumption. 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.

In describing proofs, we will follow the Coq style as introduced in Section 2.5.2 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 requires a proof of

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

After introductions, the proof state will have the following form:

$$\begin{array}{r} c : \texttt{context} \\ o : \texttt{oo} \\ a : \texttt{atm} \\ H_1 : o \in c \\ H_2 : c, [o] \rhd a \\ IH : P_2\ c\ o\ a \\ \hline P_1\ c\ \langle\ a\ \rangle \end{array}$$

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 $c : \texttt{context}$, $o : \texttt{oo}$, and $a : \texttt{atm}$, when they can be easily inferred from context.

Chapter 5

Specification Logic Metatheory

Proving admissibility of structural rules of a specification logic (SL) frees us from defining them as axiomatic and having to make external justifications for such axioms. We prove admissibility of the structural rules of contraction, weakening, exchange, and cut for both goal-reduction and backchaining sequents. Once proven at the specification level, they can be reused for any OL using this SL. Cut admissibility is particularly useful and considerably more challenging to prove than the other structural rules. It 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 can prove properties of this logic using the mutual structural induction principle over the rules of the SL from Figure 4.3 when the theorem (or goal statement) is the same form as the conclusion of the induction principle. Backchaining over the induction principle, we will have fifteen subcases; one subcase corresponding to each rule of the SL. Many of these cases have similar proofs. We will look at a few cases that are interesting for the following reasons:

g_dyn
: This rule has a goal-reduction sequent conclusion, a non-sequent premise depending on the context of the conclusion and a backchaining sequent premise.

g_imp
: This rule has a goal-reduction sequent conclusion and a sequent premise with a context different from that of the conclusion.

b_imp
: This rule has a backchaining sequent conclusion and both a goal-reduction and backchaining sequent premise.

5.1 Structural Rules

For our intuitionistic SL we prove the standard structural rules of weakening, contraction, and exchange for both goal-reduction and backchaining sequents:

Theorem 5.1 (`gr_weakening`).

$$\frac{\Gamma \rhd \beta_2}{\Gamma\,, \beta_1 \rhd \beta_2}$$

Theorem 5.2 (`bc_weakening`).

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

Theorem 5.3 (`gr_contraction`).

$$\frac{\Gamma\,,\ \beta_1\,, \beta_1 \rhd \beta_2}{\Gamma\,, \beta_1 \rhd \beta_2}$$

Theorem 5.4 (`bc_contraction`).

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

Theorem 5.5 (`gr_exchange`).

$$\frac{\Gamma\,,\ \beta_2\,,\ \beta_1 \rhd \beta_3}{\Gamma\,, \beta_1\,,\ \beta_2 \rhd \beta_3}$$

Theorem 5.6 (`bc_exchange`).

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

These are all corollaries of a general theorem:

Theorem 5.7 (`monotone`).

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

Proof:

Theorem 5.7 is proven by mutual structural induction over the premises $\Gamma \rhd \beta$ and $\Gamma, [\beta] \rhd \alpha$. Defining P_1 and P_2 as

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

we are proving

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

which has the form discussed in Section 4.3, so the mutual structural induction principle may be used. Here we will show the cases for the rules *g_dyn*, *g_imp*, and *b_imp*. The antecedent of the induction principle for each subcase gives the initial subgoals.

Case $\dfrac{D \in \Gamma \quad \Gamma, [D] \rhd A}{\Gamma \rhd \langle\ A\ \rangle}\ \textit{g\_dyn}$**:**

This rule has one non-sequent premise and one backchaining sequent premise. So there will be one induction hypothesis from the backchaining sequent premise. From the induction principle we need to prove

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

After introductions the proof state is

$$\frac{\begin{aligned} H_1 &: o \in c \\ Hb_1 &: c, [o] \rhd a \\ IHb_1 &: P_2\ c\ o\ a \end{aligned}}{P_1\ c\ \langle\ a\ \rangle}$$

Unfolding P_1 and P_2 as defined for this theorem, we have

$$\begin{array}{r} H_1 : o \in c \\ Hb_1 : c, [o] \rhd a \\ IHb_1 : \forall\ (\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma', [o] \rhd a \end{array}$$
$$\hline$$
$$\forall\ (\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma' \rhd \langle\ a\ \rangle$$

Next we make introductions from the goal.

$$\begin{array}{r} H_1 : o \in c \\ Hb_1 : c, [o] \rhd a \\ IHb_1 : \forall\ (\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma', [o] \rhd a \\ \Gamma' : \texttt{context} \\ P_1 : c \subseteq \Gamma' \end{array}$$
$$\hline$$
$$\Gamma' \rhd \langle\ a\ \rangle$$

Now the goal is a goal-reduction sequent with an atomic formula. We can backchain with the rule *g_dyn* and will get two new subgoals from the premises of this rule.

$$\begin{array}{r} H_1 : o \in c \\ Hb_1 : c, [o] \rhd a \\ IHb_1 : \forall\ (\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma', [o] \rhd a \\ \Gamma' : \texttt{context} \\ P_1 : c \subseteq \Gamma' \end{array}$$
$$\hline$$
$$(o \in \Gamma'), (\Gamma', [o] \rhd a)$$

To prove the second subgoal we use induction hypothesis IHb_1 to get the new subgoal $c \subseteq \Gamma'$ which is provable by induction assumption P_1. To prove the first, we need to unfold the definition of subset in P_1.

$$\begin{array}{r} H_1 : o \in c \\ Hb_1 : c, [o] \rhd a \\ IHb_1 : \forall\ (\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma', [o] \rhd a \\ \Gamma' : \texttt{context} \\ P_1 : \forall (o : \texttt{oo}), o \in c \to o \in \Gamma' \end{array}$$
$$\hline$$
$$o \in \Gamma'$$

Backchaining over P_1 we get the new subgoal $o \in c$ which is provable by assumption H_1. The proof for this case is complete.

Case $\dfrac{\Gamma\,,\, D \rhd G}{\Gamma \rhd D \longrightarrow G}$ *g_imp*:

This rule has one goal-reduction sequent premise which gives one induction hypothesis. From the induction principle the goal is

$$\forall(c : \texttt{context})(o_1\ o_2 : \texttt{oo}), c, o_1 \rhd o_2 \to P_1\ (c, o_1)\ o_2 \to P_1\ c\ (o_1 \longrightarrow o_2)$$

After introductions we are proving

$$\frac{\begin{array}{r} Hg_1 : c, o_1 \rhd o_2 \\ IHg_1 : P_1\ (c, o_1)\ o_2 \end{array}}{P_1\ c\ (o_1 \longrightarrow o_2)}$$

Unfolding P_1 as defined for this theorem, we have

$$\frac{\begin{array}{r} Hg_1 : c, o_1 \rhd o_2 \\ IHg_1 : \forall(\Gamma' : \texttt{context}), (c, o_1) \subseteq \Gamma' \to \Gamma' \rhd o_2 \end{array}}{\forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma' \rhd o_1 \longrightarrow o_2}$$

Next we make introductions from the goal.

$$\frac{\begin{array}{r} Hg_1 : c, o_1 \rhd o_2 \\ IHg_1 : \forall(\Gamma' : \texttt{context}), (c, o_1) \subseteq \Gamma' \to \Gamma' \rhd o_2 \\ \Gamma' : \texttt{context} \\ P_1 : c \subseteq \Gamma' \end{array}}{\Gamma' \rhd o_1 \longrightarrow o_2}$$

The rule *g_imp* is the only rule of the SL that we can backchain over with the current goal.

$$\frac{\begin{array}{r} Hg_1 : c, o_1 \rhd o_2 \\ IHg_1 : \forall(\Gamma' : \texttt{context}), (c, o_1) \subseteq \Gamma' \to \Gamma' \rhd o_2 \\ \Gamma' : \texttt{context} \\ P_1 : c \subseteq \Gamma' \end{array}}{\Gamma', o_1 \rhd o_2}$$

Now we use the induction hypothesis IHg_1. This step of backward reasoning gives the new subgoal $c, o_1 \subseteq \Gamma', o_1$. Next backchain with the context lemma `context_sub_sup` (Lemma 4.6) and we have to prove $c \subseteq \Gamma'$ which is provable by the induction assumption P_1. The proof for this case is complete.

Case $\dfrac{\Gamma \rhd G \quad \Gamma, [D] \rhd A}{\Gamma, [G \longrightarrow D] \rhd A}$ ***b_imp*:**

This rule has one goal-reduction sequent premise and one backchaining sequent premise. So there will be one induction hypothesis from each sequent premise. From the induction principle we need to prove

$$\begin{array}{l}\forall(c : \texttt{context})(o_1\ o_2 : \texttt{oo})(a : \texttt{atm}),\\ \quad c \rhd o_1 \to P_1\ c\ o_1 \to c, [o_2] \rhd a \to P_2\ c\ o_2\ a \to P_2\ c\ (o_1 \longrightarrow o_2)\ a\end{array}$$

After introductions the proof state is

$$\begin{array}{r}Hg_1 : c \rhd o_1\\ IHg_1 : P_1\ c\ o_1\\ Hb_1 : c, [o_2] \rhd a\\ IHb_1 : P_2\ c\ o_2\ a\\ \hline P_2\ c\ (o_1 \longrightarrow o_2)\ a\end{array}$$

We unfold uses of P_1 and P_2.

$$\begin{array}{r}Hg_1 : c \rhd o_1\\ IHg_1 : \forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma' \rhd o_1\\ Hb_1 : c, [o_2] \rhd a\\ IHb_1 : \forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma', [o_2] \rhd a\\ \hline \forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to c, [o_1 \longrightarrow o_2] \rhd a\end{array}$$

Next we can make introductions from the goal.

$$\begin{array}{r}Hg_1 : c \rhd o_1\\ IHg_1 : \forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma' \rhd o_1\\ Hb_1 : c, [o_2] \rhd a\\ IHb_1 : \forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma', [o_2] \rhd a\\ IP_1 : c \subseteq \Gamma'\\ \hline \Gamma', [o_1 \longrightarrow o_2] \rhd a\end{array}$$

The only SL rule whose conclusion matches the goal is *b_imp* so we backchain with this rule to get two new subgoals.

$$\begin{array}{l} Hg_1 : c \rhd o_1 \\ IHg_1 : \forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma' \rhd o_1 \\ Hb_1 : c, [o_2] \rhd a \\ IHb_1 : \forall(\Gamma' : \texttt{context}), c \subseteq \Gamma' \to \Gamma', [o_2] \rhd a \\ IP_1 : c \subseteq \Gamma' \\ \hline (\Gamma' \rhd o_1), (\Gamma', [o_2] \rhd a) \end{array}$$

We backchain over the appropriate induction hypothesis for each of these subgoals, and in both cases get the subgoal $c \subseteq \Gamma'$, provable by induction assumption IP_1. The proof of this subcase is complete.

5.2 Cut Admissibility

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

Theorem 5.8 (`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.

Proof: (**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.

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

$$\frac{}{\boxed{\text{goal sequent}}}\ \texttt{constructor}$$

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).

$$\dfrac{\dfrac{\dfrac{}{\{\boxed{\text{IH antecedents}}\}}\,\texttt{assumption}}{\{\boxed{\text{rule sequent premise(s)}}\}}\,\texttt{apply}\ IH}{\boxed{\text{goal sequent}}}\,\texttt{constructor}$$

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, `gr_weakening` (Theorem 5.1), 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).

$$\dfrac{\dfrac{}{\boxed{\text{non-sequent premise}}}\,\texttt{assumption} \qquad \dfrac{\dfrac{}{\boxed{\text{IH antecedents}}}\,\texttt{assumption}}{\boxed{\text{sequent premise}}}\,\texttt{apply}\ IH}{\boxed{\text{goal sequent}}}\,\texttt{constructor}$$

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.

So 98 of 105 subcases are proven following this outline.

(end outline)

∎

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

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

Proof:

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

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

where

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

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.

In the proof presentation here we will only look at cases for the rule *g_dyn*. Later we will see a generalization of the SL and a proof that captures the remaining 98 cases, as well as the proof of `monotone` (Theorem 5.7) seen above. Since in the proof of `monotone` we have already seen how to prove a few concrete cases in detail using the mutual structural induction principle, it would be tedious to continue to work through more subcases in the same way.

5.2.1 Subcase for *g_dyn*: Alternate Proof Attempt

Before proving this subcase for the nested induction, suppose that rather than an outer induction over the cut formula δ we had simply introduced this variable into the context of the proof state and begun the proof as a mutual structural induction

over the sequent premises with δ in their context. Then we can wait until it is necessary to have an induction over the cut formula.

The subcase of the induction principle for *g_dyn* requires a proof of

$$\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), o \in c \to c, [o] \rhd a \to P_2\ c\ o\ a \to P_1\ c\ \langle\ a\ \rangle$$

After introductions and unfolding P_1 and P_2 as defined for this theorem, the proof state is

$$\begin{array}{r} H_1 : o \in c \\ Hb_1 : c, [o] \rhd a \\ IHb_1 : \forall(\Gamma' : \texttt{context}), c = (\Gamma', \delta) \to \Gamma' \rhd \delta \to \Gamma', [o] \rhd a \\ \hline \forall(\Gamma' : \texttt{context}), c = (\Gamma', \delta) \to \Gamma' \rhd \delta \to \Gamma' \rhd \langle\ a\ \rangle \end{array}$$

Next we make introductions from the goal.

$$\begin{array}{r} H_1 : o \in c \\ Hb_1 : c, [o] \rhd a \\ IHb_1 : \forall(\Gamma' : \texttt{context}), c = (\Gamma', \delta) \to \Gamma' \rhd \delta \to \Gamma', [o] \rhd a \\ \Gamma' : \texttt{context} \\ IP_1 : c = (\Gamma', \delta) \\ IP_2 : \Gamma' \rhd \delta \\ \hline \Gamma' \rhd \langle\ a\ \rangle \end{array}$$

Next we substitute (Γ', δ) for c using IP_1 and rename Γ' to Γ_0 in IHb_1 to distinguish the bound variable from the free variable Γ'. Now ignore IP_1.

$$\begin{array}{r} H_1 : o \in \Gamma', \delta \\ Hb_1 : \Gamma', \delta, [o] \rhd a \\ IHb_1 : \forall(\Gamma_0 : \texttt{context}), (\Gamma', \delta) = (\Gamma_0, \delta) \to \Gamma_0 \rhd \delta \to \Gamma_0, [o] \rhd a \\ \Gamma' : \texttt{context} \\ IP_2 : \Gamma' \rhd \delta \\ \hline \Gamma' \rhd \langle\ a\ \rangle \end{array}$$

We can get a new premise $P_3 : \Gamma', [o] \rhd a$ by specializing IHb_1 with Γ', a reflexivity lemma and IP_2. Now ignore IHb_1 which is no longer needed and Hb_1 which we can get from `bc_weakening` (Theorem 5.2) and P_3.

$$\frac{\begin{array}{r} H_1 : o \in \Gamma', \delta \\ \Gamma' : \texttt{context} \\ IP_2 : \Gamma' \rhd \delta \\ P_3 : \Gamma', [o] \rhd a \end{array}}{\Gamma' \rhd \langle\ a\ \rangle}$$

We can apply the context lemma `elem_inv` to H_1 to get the premise $(o \in \Gamma') \vee (o = \delta)$. Applying `inversion` to this, we have two new subgoals with diverging sets of assumptions. In the second we substitute δ for o by H_1 in that proof state.

$$\frac{\begin{array}{r} H_1 : o \in \Gamma' \\ \Gamma' : \texttt{context} \\ IP_2 : \Gamma' \rhd \delta \\ P_3 : \Gamma', [o] \rhd a \end{array}}{\Gamma' \rhd \langle\ a\ \rangle} \qquad \frac{\begin{array}{r} H_1 : o = \delta \\ \Gamma' : \texttt{context} \\ IP_2 : \Gamma' \rhd \delta \\ P_3 : \Gamma', [\delta] \rhd a \end{array}}{\Gamma' \rhd \langle\ a\ \rangle}$$

The left subgoal is provable by first applying *g_dyn* to get subgoals $o \in \Gamma'$ and $\Gamma', [o] \rhd a$, both proven by assumption.

The proof on the right will be continued with an induction over δ. The property to prove is

$$\begin{array}{l} P_0 : \texttt{oo} \to \texttt{Prop} := \lambda(\delta : \texttt{oo})\ . \\ \qquad \forall(\Gamma' : \texttt{context})(a : \texttt{atm}), \\ \qquad\qquad \Gamma' \rhd \delta \to \Gamma', [\delta] \rhd a \to \Gamma' \rhd \langle\ a\ \rangle \end{array}$$

We will now look at a specific subcase of this induction.

Subcase $\delta = o_1 \longrightarrow o_2$:

In this case we prove the appropriate antecedent of the induction principle for induction over δ (see Figure 3.5), shown below.

$$\forall(o_1\ o_2 : \texttt{oo}), P_0\ o_1 \to P_0\ o_2 \to P_0\ (o_1 \longrightarrow o_2)$$

The expanded proof state after premise introductions is:

$$\begin{array}{l} IH_1 : \forall(\Gamma' : \texttt{context})(a : \texttt{atm}), \Gamma' \rhd o_1 \to \Gamma', [o_1] \rhd a \to \Gamma' \rhd \langle\, a \,\rangle \\ IH_2 : \forall(\Gamma' : \texttt{context})(a : \texttt{atm}), \Gamma' \rhd o_2 \to \Gamma', [o_2] \rhd a \to \Gamma' \rhd \langle\, a \,\rangle \\ \Gamma' : \texttt{context} \\ IP_2 : \Gamma' \rhd (o_1 \longrightarrow o_2) \\ P_3 : \Gamma', [o_1 \longrightarrow o_2] \rhd a \\ \hline \Gamma' \rhd \langle\, a \,\rangle \end{array}$$

We can apply `inversion` to the premises IP_2 and P_3 to get new assumptions in the context.

$$\begin{array}{l} IH_1 : \forall(\Gamma' : \texttt{context})(a : \texttt{atm}), \Gamma' \rhd o_1 \to \Gamma', [o_1] \rhd a \to \Gamma' \rhd \langle\, a \,\rangle \\ IH_2 : \forall(\Gamma' : \texttt{context})(a : \texttt{atm}), \Gamma' \rhd o_2 \to \Gamma', [o_2] \rhd a \to \Gamma' \rhd \langle\, a \,\rangle \\ \Gamma' : \texttt{context} \\ IP_2 : \Gamma', o_1 \rhd o_2 \\ P_{3_1} : \Gamma', [o_2] \rhd a \\ P_{3_2} : \Gamma' \rhd o_1 \\ \hline \Gamma' \rhd \langle\, a \,\rangle \end{array}$$

IH_1 is not useful here, since we have no way to prove sequents with o_1 focused. Applying IH_2 and ignoring induction hypotheses, we have:

$$\begin{array}{l} \Gamma' : \texttt{context} \\ IP_2 : \Gamma', o_1 \rhd o_2 \\ P_{3_1} : \Gamma', [o_2] \rhd a \\ P_{3_2} : \Gamma' \rhd o_1 \\ \hline (\Gamma', o_2, [o_2] \rhd a), (\Gamma' \rhd o_2), (\Gamma', [o_2] \rhd a) \end{array}$$

The first subgoal is proven using `bc_weakening` (Theorem 5.2) and assumption P_{3_1}, and the third subgoal by P_{3_1}.

On trying to prove the second subgoal, we should reflect on two things. First, proving $\Gamma' \rhd o_2$ from the assumptions IP_2 and P_{3_2} would be a use of the goal-reduction cut rule. Second, we are proving the subcase corresponding to the *g_dyn* rule and the only sequent premise of this rule is a backchaining sequent; we only get the backchaining part of the cut rule in the induction hypothesis. To illustrate this,

recall that for this subcase we have $c, [o] \rhd a$ and the induction hypothesis $P_2\ c\ o\ a$ in the context of assumptions. The induction hypothesis expands to

$$\forall(\Gamma' : \texttt{context}), c = (\Gamma', \delta) \to \Gamma' \rhd \delta \to \Gamma', [o] \rhd a$$

Combining these assumptions we have

$$\Gamma', \delta, [o] \rhd a \to \Gamma' \rhd \delta \to \Gamma', [o] \rhd a$$

which is the conjunct of the cut rule for backchaining sequents. Combining the above observations, we see that this branch cannot be continued any further.

5.2.2 Subcase for *g_dyn*: Original Proof Structure

Convinced of the necessity of our original proof structure, now we will move on with our proof of the cut rule by nested inductions, first on the cut formula δ then over the sequent premises with δ in the context. 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 $\delta = o_1 \longrightarrow o_2$ **:**

From Figure 3.5, 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

$$\begin{array}{l} IH_1 : P\ o_1 \\ IH_2 : P\ o_2 \\ \hline \begin{array}{l} (\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \to \forall(\Gamma' : \texttt{context}), \\ \qquad c = (\Gamma', (o_1 \longrightarrow o_2)) \to \Gamma' \rhd (o_1 \longrightarrow o_2) \to \Gamma' \rhd o) \wedge \\ (\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \to \forall(\Gamma' : \texttt{context}), \\ \qquad c = (\Gamma', (o_1 \longrightarrow o_2)) \to \Gamma' \rhd (o_1 \longrightarrow o_2) \to \Gamma', [o] \rhd a) \end{array} \end{array}$$

Next we have the mutual induction over sequents. As stated above, we will only show the subcase for the *g_dyn* rule.

Subcase $\dfrac{D \in \Gamma \quad \Gamma, [D] \rhd A}{\Gamma \rhd \langle\ A\ \rangle}$ *g_dyn* **:**

The goal for this subcase is

$$\forall (c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), o \in c \to c, [o] \rhd a \to P_2\ c\ o\ a \to P_1\ c\ \langle\ a\ \rangle$$

After introductions, the proof state is

$$\begin{aligned}
IH_1 &: P\ o_1 \\
IH_2 &: P\ o_2 \\
H_1 &: o \in c \\
Hb_1 &: c, [o] \rhd a \\
IHb_1 &: \forall (\Gamma' : \texttt{context}), c = (\Gamma', o_1 \longrightarrow o_2) \to \Gamma' \rhd (o_1 \longrightarrow o_2) \to \Gamma', [o] \rhd a \\
\Gamma' &: \texttt{context} \\
IP_1 &: c = \Gamma', o_1 \longrightarrow o_2 \\
IP_2 &: \Gamma' \rhd o_1 \longrightarrow o_2 \\
\hline
&\Gamma' \rhd \langle\ a\ \rangle
\end{aligned}$$

Next substitute $(\Gamma', o_1 \longrightarrow o_2)$ for c using IP_1 and rename Γ' to Γ_0 in IHb_1 to distinguish the bound variable from the free variable Γ'. Now ignore IP_1.

$$\begin{aligned}
IH_1 &: P\ o_1 \\
IH_2 &: P\ o_2 \\
H_1 &: o \in (\Gamma', o_1 \longrightarrow o_2) \\
Hb_1 &: \Gamma', o_1 \longrightarrow o_2, [o] \rhd a \\
IHb_1 &: \forall (\Gamma_0 : \texttt{context}), \\
&\qquad (\Gamma', o_1 \longrightarrow o_2) = (\Gamma_0, o_1 \longrightarrow o_2) \to \Gamma_0 \rhd (o_1 \longrightarrow o_2) \to \Gamma_0, [o] \rhd a \\
\Gamma' &: \texttt{context} \\
IP_2 &: \Gamma' \rhd o_1 \longrightarrow o_2 \\
\hline
&\Gamma' \rhd \langle\ a\ \rangle
\end{aligned}$$

We can specialize IHb_1 with Γ', a reflexivity lemma and IP_2 to get the new premise $P_3 : \Gamma', [o] \rhd a$ and apply `elem_inv` (Lemma 4.1) to H_1 to get $(o \in \Gamma') \vee (o = o_1 \longrightarrow o_2)$. Now ignore IHb_1 and Hb_1 (we can get the latter from assumption P_3 and `bc_weakening`, Theorem 5.2).

$$\begin{array}{l} IH_1 : P\ o_1 \\ IH_2 : P\ o_2 \\ H_1 : (o \in \Gamma') \vee (o = o_1 \longrightarrow o_2) \\ IP_2 : \Gamma' \rhd o_1 \longrightarrow o_2 \\ P_3 : \Gamma', [o] \rhd a \\ \hline \Gamma' \rhd \langle\ a\ \rangle \end{array}$$

Inverting H_1, we get two new subgoals with different sets of assumptions. In the second we substitute $o_1 \longrightarrow o_2$ for o using H_1 in that proof state.

$$\begin{array}{l} IH_1 : P\ o_1 \\ IH_2 : P\ o_2 \\ H_1 : o \in \Gamma' \\ IP_2 : \Gamma' \rhd o_1 \longrightarrow o_2 \\ P_3 : \Gamma', [o] \rhd a \\ \hline \Gamma' \rhd \langle\ a\ \rangle \end{array} \qquad \begin{array}{l} IH_1 : P\ o_1 \\ IH_2 : P\ o_2 \\ H_1 : o = o_1 \longrightarrow o_2 \\ IP_2 : \Gamma' \rhd o_1 \longrightarrow o_2 \\ P_3 : \Gamma', [o_1 \longrightarrow o_2] \rhd a \\ \hline \Gamma' \rhd \langle\ a\ \rangle \end{array}$$

To prove the first, we apply *g_dyn* to the goal, then need to prove $o \in \Gamma'$ and $\Gamma', [o] \rhd a$ 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 IP_2 and P_3, and unfolding P, we have:

$$\begin{array}{l} IH_1 : (\forall (c : \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow \\ \qquad \forall (\Gamma' : \texttt{context}), c = (\Gamma', o_1) \rightarrow \Gamma' \rhd o_1 \rightarrow \Gamma' \rhd o) \wedge \\ \quad (\forall (c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow \\ \qquad \forall (\Gamma' : \texttt{context}), c = (\Gamma', o_1) \rightarrow \Gamma' \rhd o_1 \rightarrow \Gamma', [o] \rhd a) \\ IH_2 : (\forall (c : \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow \\ \qquad \forall (\Gamma' : \texttt{context}), c = (\Gamma', o_2) \rightarrow \Gamma' \rhd o_2 \rightarrow \Gamma' \rhd o) \wedge \\ \quad (\forall (c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow \\ \qquad \forall (\Gamma' : \texttt{context}), c = (\Gamma', o_2) \rightarrow \Gamma' \rhd o_2 \rightarrow \Gamma', [o] \rhd a) \\ IP_2 : \Gamma', o_1 \rhd o_2 \\ P_{3_1} : \Gamma' \rhd o_1 \\ P_{3_2} : \Gamma', [o_2] \rhd a \\ \hline \end{array}$$

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

Backchaining on the first conjunct of IH_2, instantiating c with (Γ', o_2), gives three new subgoals.

$$\begin{array}{l}
IH_1 : (\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_1) \rightarrow \Gamma' \rhd o_1 \rightarrow \Gamma' \rhd o) \wedge \\
\quad (\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_1) \rightarrow \Gamma' \rhd o_1 \rightarrow \Gamma', [o] \rhd a) \\
IH_2 : (\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_2) \rightarrow \Gamma' \rhd o_2 \rightarrow \Gamma' \rhd o) \wedge \\
\quad (\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_2) \rightarrow \Gamma' \rhd o_2 \rightarrow \Gamma', [o] \rhd a) \\
P_{3_1} : \Gamma' \rhd o_1 \\
P_{3_2} : \Gamma', [o_2] \rhd a \\
IP_2 : \Gamma', o_1 \rhd o_2 \\
\hline
(\Gamma', o_2 \rhd \langle\ a\ \rangle), (\Gamma', o_2 = \Gamma', o_2), (\Gamma' \rhd o_2)
\end{array}$$

For the first, apply *g_dyn*, then we need to prove $o_2 \in (\Gamma', o_2)$ (proven by `elem_self`, Lemma 4.3) and $\Gamma', o_2, [o_2] \rhd a$ (proven by `bc_weakening`, Theorem 5.2, and assumption P_{3_2}). The second is proven by `reflexivity`. For the third, we backchain on the first conjunct of IH_1, instantiating c with (Γ', o_1), and get three new subgoals.

$$\begin{array}{l}
IH_1 : (\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_1) \rightarrow \Gamma' \rhd o_1 \rightarrow \Gamma' \rhd o) \wedge \\
\quad (\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_1) \rightarrow \Gamma' \rhd o_1 \rightarrow \Gamma', [o] \rhd a) \\
IH_2 : (\forall(c : \texttt{context})(o : \texttt{oo}), c \rhd o \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_2) \rightarrow \Gamma' \rhd o_2 \rightarrow \Gamma' \rhd o) \wedge \\
\quad (\forall(c : \texttt{context})(o : \texttt{oo})(a : \texttt{atm}), c, [o] \rhd a \rightarrow \\
\qquad \forall(\Gamma' : \texttt{context}), c = (\Gamma', o_2) \rightarrow \Gamma' \rhd o_2 \rightarrow \Gamma', [o] \rhd a) \\
P_{3_1} : \Gamma' \rhd o_1 \\
P_{3_2} : \Gamma', [o_2] \rhd a \\
IP_2 : \Gamma', o_1 \rhd o_2 \\
\hline
(\Gamma', o_1 \rhd o_2), (\Gamma', o_1 = \Gamma', o_1), (\Gamma' \rhd o_1)
\end{array}$$

The sequent subgoals are proven by `assumption` and the other by `reflexivity`.

```
Proof.
Hint Resolve context_sub_sup.
eapply seq_mutind; intros;
try (econstructor; eauto; eassumption).
Qed.
```

Figure 5.1: Coq proof of `monotone` (Theorem 5.7)

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

Figure 5.2: Coq proof of 98/105 cases of `cut_admissible` (Theorem 5.8)

The *g_dyn* subcases for the remaining six constructors of `oo` follow a similar argument requiring `inversion` on hypotheses and induction hypothesis specialization.

From this presentation we can see that working through the details for every case can be a tedious and repetitive task. We later see a generalization that helps us to understand what subcases have the same structure and separate out the challenging cases. This understanding leads us to a condensed automated Coq proof for `monotone` (Theorem 5.7, see Figure 5.1) and proofs of 98 of 105 subcases in the proof of `cut_admissible` (Theorem 5.8, see Figure 5.2 where `delta` is the cut formula in the implementation, in place of δ).

Chapter 6

Generalized Specification Logic

All non-axiomatic 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.

Here we present generalized specification logic rules to reduce the number of induction cases and allow us to partition cases of proofs about 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 have one of the following forms:

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

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

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, $0 \leq m \leq 1$, $0 \leq n \leq 2$, and $0 \leq p \leq 1$.

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 (as will be seen in Section 6.1). 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 in instances of terms where it occurs. 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 $\gamma_1\langle\!\langle D \longrightarrow G : \texttt{oo}\rangle\!\rangle : \texttt{context}$, 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, $Q_i\langle\!\langle c : \texttt{context}, o : \texttt{oo}\rangle\!\rangle : \texttt{Prop}$
- 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, $\gamma_j\langle\!\langle o : \texttt{oo}\rangle\!\rangle : \texttt{context}$ and $F_j\langle\!\langle o : \texttt{oo}, x_{j,1} : R_{j,1}, \ldots, x_{j,s_j} : R_{j,s_j}\rangle\!\rangle : \texttt{oo}$
- 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 $\gamma'_k\langle\!\langle o : \texttt{oo}\rangle\!\rangle : \texttt{context}$ and $F'_k\langle\!\langle o : \texttt{oo}, y_{k,1} : S_{k,1}, \ldots, y_{k,t_k} : S_{k,t_k}\rangle\!\rangle : \texttt{oo}$

6.1 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, $x_{j,1}$, …, x_{j,s_j}, and $R_{j,1}$, …, R_{j,s_j}. For $k = 1, \ldots, p$, we specify γ'_k, F'_k, $y_{k,1}$, …, y_{k,t_k}, and $S_{k,1}$, …, S_{k,t_k}. Below are instantiations for all rules of the SL.

Rule	m	n	p	c	o
$\dfrac{A :- G \quad \Gamma \rhd G}{\Gamma \rhd \langle\ A\ \rangle}\ g\_prog$	1	1	0	Γ	$\langle\ A\ \rangle$
$s_1 \coloneqq 0$ $Q_1(\!\|\Gamma, \langle\ A\ \rangle\|\!) \coloneqq A :- G \quad \gamma_1(\!\|\langle\ A\ \rangle\|\!) \coloneqq \emptyset \quad F_1(\!\|\langle\ A\ \rangle\|\!) \coloneqq G$					
$\dfrac{D \in \Gamma \quad \Gamma, [D] \rhd A}{\Gamma \rhd \langle\ A\ \rangle}\ g\_dyn$	1	0	1	Γ	$\langle\ A\ \rangle$
$t_1 \coloneqq 0 \quad a_1 \coloneqq A$ $Q_1(\!\|\Gamma, \langle\ A\ \rangle\|\!) \coloneqq D \in \Gamma \quad \gamma'_1(\!\|\langle\ A\ \rangle\|\!) \coloneqq \emptyset \quad F'_1(\!\|\langle\ A\ \rangle\|\!) \coloneqq D$					
$\dfrac{}{\Gamma \rhd \mathtt{T}}\ g\_tt$	0	0	0	Γ	$\mathtt{T}$
$\dfrac{\Gamma \rhd G_1 \quad \Gamma \rhd G_2}{\Gamma \rhd G_1 \,\&\, G_2}\ g\_and$	0	2	0	Γ	$G_1 \& G_2$
$s_1 \coloneqq 0 \quad s_2 \coloneqq 0$ $\gamma_1(\!\|G_1 \& G_2\|\!) \coloneqq \emptyset \quad F_1(\!\|G_1 \& G_2\|\!) \coloneqq G_1$ $\gamma_2(\!\|G_1 \& G_2\|\!) \coloneqq \emptyset \quad F_2(\!\|G_1 \& G_2\|\!) \coloneqq G_2$					
$\dfrac{\Gamma\,,\, D \rhd G}{\Gamma \rhd D \longrightarrow G}\ g\_imp$	0	1	0	Γ	$D \longrightarrow G$
$s_1 \coloneqq 0 \quad \gamma_1(\!\|D \longrightarrow G\|\!) \coloneqq \{D\} \quad F_1(\!\|D \longrightarrow G\|\!) \coloneqq G$					
$\dfrac{\forall (E : \mathtt{expr\ \ con}), (\mathtt{proper}\ E \to \Gamma \rhd G\ E)}{\Gamma \rhd \mathtt{All}\ G}\ g\_all$	0	1	0	Γ	$\mathtt{All}\ G$
$s_1 \coloneqq 2 \quad x_{1,1} \coloneqq E \quad R_{1,1} \coloneqq \mathtt{expr\ \ con} \quad x_{1,2} \coloneqq H \quad R_{1,2} \coloneqq \mathtt{proper}\ E$ $\gamma_1(\!\|\mathtt{All}\ G\|\!) \coloneqq \emptyset \quad F_1(\!\|\mathtt{All}\ G, E, H\|\!) \coloneqq G\ E$					
$\dfrac{\forall (E : \mathtt{X}), (\Gamma \rhd G\ E)}{\Gamma \rhd \mathtt{Allx}\ G}\ g\_allx$	0	1	0	Γ	$\mathtt{Allx}\ G$
$s_1 \coloneqq 1 \quad x_{1,1} \coloneqq E \quad R_{1,1} \coloneqq \mathtt{X}$ $\gamma_1(\!\|\mathtt{Allx}\ G\|\!) \coloneqq \emptyset \quad F_1(\!\|\mathtt{Allx}\ G, E, H\|\!) \coloneqq G\ E$					
$\dfrac{\mathtt{proper}\ E \quad \Gamma \rhd G\ E}{\Gamma \rhd \mathtt{Some}\ G}\ g\_some$	1	1	0	Γ	$\mathtt{Some}\ G$
$s_1 \coloneqq 0$ $Q_1(\!\|\Gamma, \mathtt{Some}\ G\|\!) \coloneqq \mathtt{proper}\ E \quad \gamma_1(\!\|\mathtt{Some}\ G\|\!) \coloneqq \emptyset \quad F_1(\!\|\mathtt{Some}\ G\|\!) \coloneqq G\ E$					

Rule	m	n	p	c	o
$\dfrac{}{\Gamma, [\langle\ A\ \rangle] \rhd A}\ b\_match$	0	0	0	Γ	$\langle\ A\ \rangle$
$\dfrac{\Gamma, [D_1] \rhd A}{\Gamma, [D_1\ \&\ D_2] \rhd A}\ b\_and_1$	0	0	1	Γ	$D_1 \& D_2$
$t_1 \coloneqq 0 \quad a_1 \coloneqq A \quad \gamma_1'(\!\langle D_1 \& D_2 \rangle\!) \coloneqq \emptyset \quad F_1'(\!\langle D_1 \& D_2 \rangle\!) \coloneqq D_1$					
$\dfrac{\Gamma, [D_2] \rhd A}{\Gamma, [D_1\ \&\ D_2] \rhd A}\ b\_and_2$	0	0	1	Γ	$D_1 \& D_2$
$t_1 \coloneqq 0 \quad a_1 \coloneqq A \quad \gamma_1'(\!\langle D_1 \& D_2 \rangle\!) \coloneqq \emptyset \quad F_1'(\!\langle D_1 \& D_2 \rangle\!) \coloneqq D_2$					
$\dfrac{\Gamma \rhd G \quad \Gamma, [D] \rhd A}{\Gamma, [G \longrightarrow D] \rhd A}\ b\_imp$	0	1	1	Γ	$G \longrightarrow D$
$s_1 \coloneqq 0 \quad t_1 \coloneqq 0 \quad a_1 \coloneqq A$ $\gamma_1(\!\langle G \longrightarrow D \rangle\!) \coloneqq \emptyset \quad F_1(\!\langle G \longrightarrow D \rangle\!) \coloneqq G$ $\gamma_1'(\!\langle G \longrightarrow D \rangle\!) \coloneqq \emptyset \quad F_1'(\!\langle G \longrightarrow D \rangle\!) \coloneqq D$					
$\dfrac{\texttt{proper}\ E \quad \Gamma, [D\ E] \rhd A}{\Gamma, [\texttt{All}\ D] \rhd A}\ b\_all$	1	0	1	Γ	$\texttt{All}\ D$
$t_1 \coloneqq 0 \quad a_1 \coloneqq A$ $Q_1(\!\langle \Gamma, \texttt{All}\ D \rangle\!) \coloneqq \texttt{proper}\ E \quad \gamma_1'(\!\langle \texttt{All}\ D \rangle\!) \coloneqq \emptyset \quad F_1'(\!\langle \texttt{All}\ D \rangle\!) \coloneqq D\ E$					
$\dfrac{\Gamma, [D\ E] \rhd A}{\Gamma, [\texttt{Allx}\ D] \rhd A}\ b\_allx$	0	0	1	Γ	$\texttt{All}\ D$
$t_1 \coloneqq 0 \quad a_1 \coloneqq A$ $\gamma_1'(\!\langle \texttt{Allx}\ D \rangle\!) \coloneqq \emptyset \quad F_1'(\!\langle \texttt{Allx}\ D \rangle\!) \coloneqq D\ E$					
$\dfrac{\forall (E : \texttt{expr con}), (\texttt{proper}\ E \to \Gamma, [D\ x] \rhd A)}{\Gamma, [\texttt{Some}\ D] \rhd A}\ b\_some$	0	0	1	Γ	$\texttt{Some}\ D$
$t_1 \coloneqq 2 \quad y_{1,1} \coloneqq E \quad S_{1,1} \coloneqq \texttt{expr con} \quad y_{1,2} \coloneqq H \quad S_{1,2} \coloneqq \texttt{proper}\ E \quad a_1 \coloneqq A$ $\gamma_1'(\!\langle \texttt{Some}\ D, E, H \rangle\!) \coloneqq \emptyset \quad F_1'(\!\langle \texttt{Some}\ D, E, H \rangle\!) \coloneqq D\ E$					

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 $(\!\langle \cdot \rangle\!)$ 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 (these are the signature variables for induction subcases corresponding to these rules).

Chapter 7

Generalized Specification Logic Metatheory

7.1 GSL Induction Part I: A Restricted Theorem

Recall we were concerned with proving statements of the form

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

In the GSL we now have two rules instead of the 15 of the SL. To prove this statement by mutual structural induction over the GSL we will have two subcases; one for each of the rules *gr_rule* and *bc_rule*. From the rule *gr_rule* (resp. *bc_rule*) the induction subcase has n induction hypotheses for the n goal-reduction sequent premises and p induction hypotheses for the p backchaining sequent premises of the rule. We also assume the m non-sequent premises. After introductions, the proof state is:

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

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

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

Each IA_i is a predicate that we call an *induction assumption*. P_1 and P_2 can be instantiated to specific statements about the GSL by defining these IA_i. Notice that this is a generalization of all induction predicates we have seen so far. 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).

First we unfold uses of P_1 and P_2 in the proof state.

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

$$\begin{aligned}
&\forall(\Gamma' : \texttt{context}), IA_1\langle\!\langle c, \Gamma'\rangle\!\rangle \to \cdots \to IA_w\langle\!\langle c, \Gamma'\rangle\!\rangle \to \underline{\Gamma' \rhd o}\\
(&\textit{resp.}\ \forall(\Gamma' : \texttt{context}), IA_1\langle\!\langle c, \Gamma'\rangle\!\rangle \to \cdots \to IA_w\langle\!\langle c, \Gamma'\rangle\!\rangle \to \underline{\Gamma', [o] \rhd a})
\end{aligned}$$

Next we 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. Γ' is a new signature variable. See Figure 7.1 for this proof state.

7.1.1 Sequent Subgoals

To prove the last $(n + p)$ subgoals (the "second" and "third" subgoals in Figure 7.1) we first introduce any locally quantified variables as signature variables.

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

Figure 7.1: Proof state of GSL induction after rule application

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

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

$$
\begin{array}{rl}
\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}), \\
& \quad IA_1\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle, \Gamma' \rangle\!\rangle \rightarrow \cdots \rightarrow IA_w\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle, \Gamma' \rangle\!\rangle \rightarrow \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}), \\
& \quad IA_1\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle, \Gamma' \rangle\!\rangle \rightarrow \cdots \rightarrow IA_w\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle, \Gamma' \rangle\!\rangle \rightarrow \Gamma', [\overline{F'_p}\langle\!\langle o, \overline{y_{p,t_p}} \rangle\!\rangle] \rhd \overline{a_p} \\
\Gamma' : & \texttt{context} \\
\overline{IP_w} : & \overline{IA_w}\langle\!\langle c, \Gamma' \rangle\!\rangle \\
\overline{x_{n,s_n}} : & \overline{R_{n,s_n}} \ (\textit{resp.}\ \overline{y_{p,t_p}} : \overline{S_{p,t_p}}) \\
\hline
& \overline{IA_w}\langle\!\langle c \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle, \Gamma' \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle \rangle\!\rangle \ \ (\textit{resp.}\ \overline{IA_w}\langle\!\langle c \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle, \Gamma' \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle \rangle\!\rangle)
\end{array}
$$

Figure 7.2: Incomplete proof branches for sequent premises

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

The proof state in Figure 7.2 will be continued for specific theorem statements which will have the induction assumptions defined.

7.1.2 Non-Sequent Subgoals

The proof of the first m subgoals in Figure 7.1 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 instantiate the variables as we choose when backchaining over the generalized rule. We will illustrate this by considering each rule with non-sequent premises, starting from the proof state in Figure 7.1 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.

There are four rules of the SL with non-sequent premises: *g_prog*, *g_dyn*, *g_some*, and *b_all*.

Case $\dfrac{A :\!- G \quad \Gamma \rhd G}{\Gamma \rhd \langle\ A\ \rangle}$ *g_prog* **:**

This rule has one non-sequent premise and one goal-reduction sequent premise with no local quantification, so $m = n = 1$, $p = 0$, $s_1 = 0$, $o = \langle\ A\ \rangle$, and $c = \Gamma$. Then we are proving the following:

$$\begin{array}{rl} H_1 : & Q_1(\!|\Gamma, \langle\ A\ \rangle|\!) \\ Hg_1 : & \Gamma \cup \gamma_1(\!|\langle\ A\ \rangle|\!) \rhd F_1(\!|\langle\ A\ \rangle|\!) \\ IHg_1 : & \forall(\Gamma' : \texttt{context}), IA_1(\!|\Gamma, \Gamma'|\!) \to \cdots \to IA_w(\!|\Gamma, \Gamma'|\!) \to \Gamma' \rhd F_1(\!|\langle\ A\ \rangle|\!) \\ & \Gamma' : \texttt{context} \\ & \overline{IP_w} : \overline{IA_w}(\!|\Gamma, \Gamma'|\!) \\ \hline & Q_1(\!|\Gamma', \langle\ A\ \rangle|\!) \end{array}$$

Define $Q_1(\!|\_, \langle\ A\ \rangle|\!) \coloneqq A :- G$, $\gamma_1(\!|\langle\ A\ \rangle|\!) \coloneqq \emptyset$, and $F_1(\!|\langle\ A\ \rangle|\!) \coloneqq G$, where $G : \texttt{oo}$ is a signature variable. Now the proof state is

$$\begin{array}{rl} H_1 : & A :- G \\ Hg_1 : & \Gamma \rhd G \\ IHg_1 : & \forall(\Gamma' : \texttt{context}), IA_1(\!|\Gamma, \Gamma'|\!) \to \cdots \to IA_w(\!|\Gamma, \Gamma'|\!) \to \Gamma' \rhd G \\ & \Gamma' : \texttt{context} \\ & \overline{IP_w} : \overline{IA_w}(\!|\Gamma, \Gamma'|\!) \\ \hline & A :- G \end{array}$$

which is completed by assumption H_1.

Case $\dfrac{D \in \Gamma \quad \Gamma, [D] \rhd A}{\Gamma \rhd \langle\ A\ \rangle}$ *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 $Q_1(\!|\Gamma, \langle\ A\ \rangle|\!) \coloneqq D \in \Gamma$, $\gamma'_1(\!|\langle\ A\ \rangle|\!) \coloneqq \emptyset$, and $F'_1(\!|\langle\ A\ \rangle|\!) \coloneqq D$, where $D : \texttt{oo}$ is a signature variable. Then we need to prove what is displayed in Figure 7.3. 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. We will refer to Figure 7.3 later when proving specific theorem statements (i.e. each IA_i defined).

Case $\dfrac{\texttt{proper}\ E \quad \Gamma \rhd G\ E}{\Gamma \rhd \texttt{Some}\ G}$ *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

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

$$D \in \Gamma'$$

Figure 7.3: Incomplete proof branch (*g_dyn* case)

$Q_1\langle\!\langle \Gamma, \texttt{Some}\ G \rangle\!\rangle \coloneqq \texttt{proper}\ E$, $\gamma_1\langle\!\langle \texttt{Some}\ G \rangle\!\rangle \coloneqq \emptyset$, and $F_1\langle\!\langle \texttt{Some}\ G \rangle\!\rangle \coloneqq G\ E$ where $E : \texttt{expr con}$ is a signature variable. Then we are proving the following:

$$\begin{array}{r} H_1 : \texttt{proper}\ E \\ Hg_1 : \Gamma \rhd G\ E \\ IHg_1 : \forall(\Gamma' : \texttt{context}), IA_1\langle\!\langle \Gamma, \Gamma' \rangle\!\rangle \to \cdots \to IA_w\langle\!\langle \Gamma, \Gamma' \rangle\!\rangle \to \Gamma' \rhd G\ E \\ \Gamma' : \texttt{context} \\ \overline{IP_w} : \overline{IA_w}\langle\!\langle \Gamma, \Gamma' \rangle\!\rangle \end{array}$$

$$\texttt{proper}\ E$$

which is completed by assumption H_1.

Case $\dfrac{\texttt{proper}\ E \quad \Gamma, [D\ E] \rhd A}{\Gamma, [\texttt{All}\ D] \rhd A}\ \textit{b\_all}$ **:**

This case is proven as above but with $m = p = 1$, $n = 0$, $c = \Gamma$, and $o = \texttt{All}\ D$. Define $Q_1\langle\!\langle \Gamma, \texttt{All}\ D \rangle\!\rangle \coloneqq \texttt{proper}\ E$, $\gamma_1'\langle\!\langle \texttt{All}\ D \rangle\!\rangle \coloneqq \emptyset$, and $F_1'\langle\!\langle \texttt{All}\ D \rangle\!\rangle \coloneqq D\ E$ where $E : \texttt{expr con}$ is a signature variable. Then we are proving:

$$\begin{array}{r} H_1 : \texttt{proper}\ E \\ Hb_1 : \Gamma, [D\ E] \rhd a_1 \\ IHb_1 : \forall(\Gamma' : \texttt{context}), IA_1\langle\!\langle \Gamma, \Gamma' \rangle\!\rangle \to \cdots \to IA_w\langle\!\langle \Gamma, \Gamma' \rangle\!\rangle \to \Gamma', [D\ E] \rhd a_1 \\ \Gamma' : \texttt{context} \\ \overline{IP_w} : \overline{IA_w}\langle\!\langle \Gamma, \Gamma' \rangle\!\rangle \end{array}$$

$$\texttt{proper}\ E$$

The goal `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 7.2 and 7.3 (the incomplete branches) will be roots of these explanations.

7.2 GSL Induction Part II: The Structural Rules Hold

Recall from Section 5.1 we prove the standard rules of weakening, contraction and exchange for both the goal-reduction and backchaining sequents as corollaries of `monotone` (Theorem 5.7) which states

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

where P_1 and P_2 are defined as

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

We build on the inductive proof in Section 7.1 over the GSL to prove `monotone` for this new logic. 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 7.2 and 7.3). 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 $IA_1(\!|c, \Gamma'|\!) \coloneqq c \subseteq \Gamma'$.

7.2.1 Sequent Subgoals

First we will prove the subgoals coming from the sequent premises, building on Figure 7.2 and using IA_1 as defined above. The proof state for goal-reduction (resp. backchaining) premises is

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

The goal is provable by `context_sub_sup` (Lemma 4.6) and assumption IP_1.

7.2.2 Non-Sequent Subgoals

Still to be proven are the subgoals for non-sequent premises. As seen in Section 7.1.2, the only rule of the SL whose corresponding subcase still needs to be proven is *g_dyn*. From Figure 7.3 and using P_1 and P_2 as defined here, we are proving

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

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 subgoal $D \in \Gamma$, provable by assumption H_1.

In Section 7.1, 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 7.2 and 7.3. In Section 6.1 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. ∎

7.3 GSL Induction Part III: Cut Rule Proven Admissible

Recall from Section 5.2 we are proving $\forall(\delta : \texttt{oo}), P\ \delta$ with P defined as

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

where

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

As in the GSL proof of `monotone` (Theorem 5.7), we build on the inductive proof in Chapter 7, 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 $IA_1(\!|c, \Gamma'|\!) := (c = (\Gamma', \delta))$ and $IA_2(\!|c, \Gamma'|\!) := \Gamma' \rhd \delta$, where δ is the cut formula in the cut rule.

7.3.1 Sequent Subgoals

First we will prove the subgoals coming from the sequent premises, building on Figure 7.2 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

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

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

To prove the sequent subgoal $\Gamma' \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle \rhd \eta$ (resp. $\Gamma' \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle \rhd \eta$), first apply weakening and the new subgoal is $\Gamma' \rhd \eta$ (resp. $\Gamma' \rhd \eta$), provable by assumption IP_2.

The subgoals concerning context equality are proven by context lemmas and assumption IP_1. That is, we rewrite $((\Gamma' \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle), \eta)$ to $(\Gamma', \eta) \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle$ (resp. $(\Gamma' \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle), \eta$ to $(\Gamma', \eta) \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle$). The new subgoal is $c \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle = (\Gamma', \eta) \cup \overline{\gamma_n}\langle\!\langle o \rangle\!\rangle$ (resp. $(c \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle = (\Gamma', \eta) \cup \overline{\gamma'_p}\langle\!\langle o \rangle\!\rangle)$. Apply `context_sub_sup` (Lemma 4.6) to get assumption IP_1.

7.3.2 Non-Sequent Subgoals

In Section 7.1.2 we saw that the only rule of the SL whose corresponding subcase still needs to be proven is *g_dyn*. For the non-sequent subgoals we were able to complete the proof while the cut formula $\delta$was represented as a parameter (and thus could have any formula structure). In the remaining non-sequent proof branch we need to make use of the nested structure of this induction. The proof of this subcase is shown in detail in Section 5.2.2.

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.

∎

Using the generalized proof presented in this chapter and instantiating the GSL to the SL as in Section 6.1, we have found condensed proofs of `monotone` (Theorem 5.7) and `cut_admissible` (Theorem 5.8).

Chapter 8

Conclusion

In this thesis we have seen how the Coq implementation of Hybrid has been extended by the addition of a new specification logic (SL) based on hereditary Harrop formulas. This extension increases the class of object logics that Hybrid can reason about efficiently. The metatheory of this SL is formalized in Coq with proofs by mutual structural induction over the structure of sequent types. We saw the proofs of some specific subcases and the later insight that many of the cases are proven in a similar way. This led to the development of a generalized SL and form of metatheory statement that we could use to better understand the proofs of the SL metatheory.

8.1 Related Work

Throughout this thesis we have seen some mention of related work. Hybrid is a system implementing HOAS and as seen in Section 3.6 there are other systems with the same goal that also use this technique. As previously discussed, Hybrid is the only known system implementing HOAS in an existing trusted general-purpose theorem prover. See [7] and [8] for a more in-depth comparison of these systems on benchmarks defined there.

Although this work is contributing to the area of mechanizing programming language metatheory, the majority of the research presented here is applicable to the more general field of proof theory. We have seen proofs of the admissibility of structural rules of a specific sequent calculus, as well as a generalized sequent calculus which we tried to make only as general as necessary to encapsulate the specification logic presented earlier. Typically these kinds of proofs are by an induction on the height of derivations, but here we have proofs by mutual structural induction over dependent sequent types; the structural proofs in this thesis follow the style of Pfenning in [15]. The sequents in our logic do not have a natural number to represent the height of the derivation. So our presentation of this sequent calculus is perhaps more "pure" in some sense, but we may have lost a way to reason about some object logics.

It is not yet clear if building proof height into the definition sequents is necessary for studying some object logics. Overall, a better understanding of the relationship between proofs of the metatheory of sequent calculi by induction on the height of derivations versus over the structure of sequents is desirable.

8.2 Future Work

The highest priority future task is to show the utility of the new specification logic in Hybrid. This will be done by presenting an object logic that makes use of the higher-order nature (in the sense of unrestricted implicational complexity) of the new specification logic. Object logics that we plan to represent include:

- correspondence between HOAS and de Bruijn encodings of untyped λ-terms; this is our example OL of Chapter 3 but we have not yet proven Theorems 3.1 and 3.2 (see [21])
- structural characterization of reductions on untyped λ-terms (see [21])
- algorithmic specification of bounded subtype polymorphism in System F (see [17]); this comes from the POPLmark challenge [1]

We would also like to add automation to proofs containing object logic judgments so that the user of Hybrid will not need to be an expert user of proof assistants to be able to use the system.

The encoding of the new Hybrid SL follows the development of the specification logic of Abella as presented in [21], but it seems that the proofs of the admissibility of the structural rules differ between these systems. These proofs in Abella are not fully explained in [21] so some work will need to be done to compare the different proofs. Also, the proof of cut admissibility for this specification logic in Abella requires a third conjunct that we did not need for our proof:

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

Our understanding so far is that these proofs in Abella are over the height of derivations, which is an implicit parameter; it is not by structural induction over sequents in the fashion of the proofs founding this thesis.

The End.

Bibliography

[1] Brian 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.

[2] Chelsea Battell and Amy Felty. The logic of hereditary harrop formulas as a specification logic for hybrid. In *11th International ACM SIGPLAN International Workshop on Logical Frameworks and Metalanguages: Theory and Practice*, ACM Digital Library, 2016.

[3] Yves Bertot and Pierre Casteran. *Interactive Theorem Proving and Program Development*. Springer, 2004.

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

[5] Thierry Coquand and Gerard Huet. The calculus of constructions. *Information and Computation*, 1988.

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

[7] Amy P. Felty, Alberto Momigliano, and Brigitte Pientka. The next 700 challenge problems for reasoning with higher-order abstract syntax representations: Part 1—a common infrastructure for benchmarks. *CoRR*, abs/1503.06095, 2015.

[8] Amy P. Felty, Alberto Momigliano, and Brigitte 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.

[9] Andrew Gacek. The Abella interactive theorem prover (system description). In *Fourth International Joint Conference on Automated Reasoning*, volume 5195 of *LNCS*, pages 154–161. Springer, 2008.

[10] William A. Howard. The formulae-as-type notion of construction, 1969. In *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980.

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

[12] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.

[13] Dale Miller and Catuscia Palamidessi. Foundational aspects of syntax. *ACM Computing Surveys*, 31(3es):1–6, 1999. Article No. 11.

[14] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

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

[16] Frank Pfenning and Conal Elliot. Higher-order abstract syntax. In *Proceedings of the ACM-SIGPLAN Conference on Programming Languages Design and Implementation*, pages 199–208. ACM Press, 1988.

[17] Brigitte 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.

[18] Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Fifth International Joint Conference on Automated Reasoning (IJCAR)*, volume 6173 of *LNCS*, pages 15–21. Springer, 2010.

[19] Carsten Schürmann. The Twelf proof assistant. In *Twenty-Second International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 79–83. Springer, 2009.

[20] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. TypiCal Project, April 2014. Version 8.4pl4.

[21] Yuting Wang, Kaustuv Chaudhuri, Andrew Gacek, and Gopalan 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.

Index

λ-tree syntax, 2

Abella, 25
ambient logic, 19

backchaining, 10
Beluga, 25

calculus of constructions, 5
calculus of inductive constructions, 5
context of assumptions, 11
Coq, 1, 5

dependent product, 8
dependent type, 7

higher-order abstract syntax, 2, 17, 19
Hybrid, 1, 17

induction assumption, 33, 56
induction hypothesis, 13
induction principle, 13
induction property, 13
inductive type, 12

mutually inductive type, 14

object logic, 2, 18

polymorphism, 8
POPLmark, 2
proof state, 11, 33

reasoning logic, 19

simply typed λ-calculs, 7
specification logic, 2, 20
subgoal, 11

tactic, 9
tactical, 9
two-level logical framework, 2
type operator, 9