

A Focused Sequent Calculus for Higher-Order Logic

Fredrik Lindblad

University of Gothenburg / Chalmers University of Technology

Abstract. We present a focused intuitionistic sequent calculus for higher-order logic. It has primitive support for equality and mixes λ -term conversion with equality reasoning. Classical reasoning is enabled by extending the system with rules for *reductio ad absurdum* and the axiom of choice. The resulting system is proved sound with respect to Church's simple type theory. The soundness proof has been formalized in Agda. A theorem prover based on bottom-up search in the calculus has been implemented. It has been tested on the TPTP higher-order problem set with good results. The problems for which the theorem prover performs best require higher-order unification more frequently than the average higher-order TPTP problem. Being strong at higher-order unification, the system may serve as a complement to other theorem provers in the field.

1 Introduction

Benchmarking and development of automated reasoning tools for higher-order logic has been facilitated by the introduction of a new syntax and a dedicated set of higher-order problems at TPTP in 2009 [1, 2]. However, higher-order unification, which is an important component of higher-order reasoning, is still a challenge for several of the established theorem proving systems.

Agda [3] is an intuitionistic higher-order logical framework based on Martin-Löf type theory. As for any interactive theorem prover, Agda users are facilitated by local automation. Agda has a plugin for this, called *Agsy* [4]. In order to be able to evaluate the performance of Agsy, an adaptation to classical higher-order logic has been developed.

This variant of Agsy, called *AgsyHOL*, turned out to be competitive in the context of TPTP. AgsyHOL performs well not least on problems involving higher-order unification. Out of the problems that AgsyHOL either solves in less than 50 milliseconds or at least 100 times faster than any other system on TPTP online (31 problems), roughly 80% of the solutions contain λ -abstractions, i.e. involve higher-order unification. For all problems that AgsyHOL solves (1722 problems) the percentage is 37.

A major idea behind Agsy and AgsyHOL is to take an inference system constructed with proof search in mind and combine it with a simple search mechanism and a small layer of search control heuristics. More precisely, the semantics

is represented by a proof checking algorithm defined over an explicit proof language. Each construct in the language corresponds to one of the inference rules. Proof search is achieved by applying narrowing [5], which lazily instantiates the proof candidate.

The setup results in a clear separation between logic and control. It facilitates reasoning about correctness since it is (an implementation of) the inference rules themselves which drive the search. The implementation of ApsyHOL is also relatively small, approximately 2000 lines of code excluding parsing, printing and the generic narrowing search.

Both Apsy and ApsyHOL are based on focused intuitionistic sequent calculus. Related proof search oriented inference systems include LJ [6], PTSC [7], and focused derivations [8].

In the context of higher-order logic theorem proving proof by refutation is the dominating approach, whereas ApsyHOL produces derivation proofs. Thus it can fill a role when proof readability and integration with interactive systems are of interest.

The inference system which ApsyHOL is based on includes rules for classical reasoning and has been proved correct with respect to Church's simple type theory (STT). The soundness proof has been formalized in Agda, and ApsyHOL produces derivations that can be independently checked by Agda relative to the soundness proof.

Completeness of the inference system has not been formally addressed. Although interesting on the theoretical level, we have instead focused on the performance of the implementation.

Section 2 presents the inference system which forms the basis of ApsyHOL. Section 3 discusses the heuristics which control search on top of the inference rules. Section 4 describes some more technical details of the implementation. Section 5 presents the empirical results of running ApsyHOL on the higher-order problems in TPTP. Finally, section 6 concludes and gives some pointers for future work.

2 Inference System

The system is based on sequent calculus, which is more suitable for proof search than natural deduction. In sequent calculus right and left rules, which correspond to introductions and eliminations, are both guided by formula deconstruction in a backward search. In natural deduction this is only true for introduction rules, whereas elimination rules can be arbitrarily and indefinitely applied backwards.

More precisely, the system is based on the sequent calculus for intuitionistic logic and originates from Gentzen's system LJ. The main reason for using an intuitionistic system in a classical context is that the initial motivation of the work was to evaluate the performance of Apsy, which targets intuitionistic logic.

The original system LJ suffers from significant and unnecessary nondeterminism caused by the proliferation of antecedents and the arbitrary order in which to deconstruct them. One improvement is the concept of *uniform proofs*

[9]. The idea is to identify a subset of the deconstructing rules for which the order of application is unimportant. Such rules are applied in a fixed order, avoiding the corresponding branching of the search tree. However, no restrictions corresponding to uniform proofs are built in to the presented calculus. Instead, corresponding improvements were made on the search control level in the implementation, see section 3.

Another improvement over LJ is *back-chaining* or *focused derivations* [8]. This means that the calculus has two judgment forms, one for introduction and one for elimination. When switching, in the perspective of bottom-up proof construction, from introduction to elimination, one antecedent is selected. It becomes the focus of the derivation. The antecedent is then deconstructed through zero or more eliminations. The result of the elimination steps must at the end be used either to prove the succedent or to justify absurd or disjunctive elimination. This technique is a fundamental part of the presented system.

Investigations using similar calculi for proof search in higher-order logic include the presentation of a focused sequent calculus for pure type systems by Lengrand et. al. [7], called PTSC. This work is based on system LJ_T [6], a type theoretic variation of system LJ due to Herbelin. In contrast to these systems, the presented one has a full set of logical connectives, including equality, and is extended with rules for classical reasoning.

2.1 Syntax and Judgments

$\gamma, \tau ::= \iota$ (type of individuals)	$A, B, \dots ::= x_\gamma$ (variable)
o (type of truth values)	$A_{\gamma\tau} B_\tau$ (application)
$\gamma\tau$ (function type)	$\lambda x_\tau. A_\gamma$ (abstraction)
	$\iota_{\gamma(o\gamma)}$ (choice operator)
	$\sim A_o$ (negation)
	$A_o \vee B_o$ (disjunction)
	$A_o \wedge B_o$ (conjunction)
	$A_o \supset B_o$ (implication)
	$\forall x_\gamma. A_o$ (universal quantification)
	$\exists x_\gamma. A_o$ (existential quantification)
	$\top$ (truth)
	$\perp$ (falsity)
	$A_\gamma = B_\gamma$ (equality)

Fig. 1. Type and Formula Syntax

The type and formula syntax is shown in figure 1. Type subscripts in formulas will be suppressed whenever the type is clearly resolved by the context. Syntax variables are denoted by γ, τ for types and by plain capital letters for formulas. The inference system will be denoted $FSC_{=}^{ho}$.

As already mentioned, there are two interdependent judgments in intuitionistic sequent calculus with focusing. They will be called the *main judgment* and *focusing judgment*. The main and focusing judgments have the forms $\Gamma \vdash A$ and $\Gamma; B \vdash A$, where Γ represents a set of antecedent formulas, $A_1, A_2, \dots, A_n$. An empty set of antecedents is denoted ϵ . The notation essentially follows Girard [10] and system LJT [6].

The meaning of both judgments is that the proposition represented by A is derivable from the propositions in Γ . In addition, the rules of the focusing judgment restrict the derivations in such a way that the focusing mechanism described above is imposed.

Apart from this there is a conversion judgment which determines when two formulas are equal. Instead of an axiomatic representation of equality reasoning on the level of the main judgment, this is expressed in the conversion judgment. Each step in an equality chain is justified by an antecedent. Therefore the conversion judgment, just as the other judgments, is parameterized by a set of antecedents, Γ . The conversion judgment has the form $\Gamma \vdash A \leftrightarrow_\gamma B$.

Each judgment form has some variants which express search oriented restrictions. These variants are presented along the way in the respective sub-sections.

The inference rules rely on a few standard notions, whose definitions are omitted. Replacing x by A in B is denoted $[A/x]B$. A formula A being well-formed of type γ is denoted $wf_\gamma A$. A formula A being a member of a set Γ is denoted $A \in \Gamma$. Reduction of a formula A to head normal form, A' , is denoted $A \rightarrow_\beta A'$.

2.2 Main Judgment

$$\begin{array}{c}
\frac{A \rightarrow_\beta B \quad \Gamma \vdash_i B}{\Gamma \vdash A} \text{ intro} \quad \frac{\Gamma \vdash \sim \sim A}{\Gamma \vdash A} \text{ RAA} \quad \frac{B \in \Gamma \quad \Gamma; B \vdash^\uparrow A}{\Gamma \vdash A} \text{ focus} \\
\\
\frac{\Gamma; \forall x_{o\gamma}. (\exists y_\gamma. xy) \supset x (\iota_{\gamma(o\gamma)} x) \vdash^\uparrow A}{\Gamma \vdash A} \text{ AC} \quad \frac{\Gamma \vdash A}{\Gamma \vdash_i A \vee B} \vee\text{-I}_1 \\
\\
\frac{\Gamma \vdash B}{\Gamma \vdash_i A \vee B} \vee\text{-I}_r \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash_i A \wedge B} \wedge\text{-I} \quad \frac{}{\Gamma \vdash_i \top} \top\text{-I} \\
\\
\frac{\Gamma, A \vdash B}{\Gamma \vdash_i A \supset B} \supset\text{-I} \quad \frac{\Gamma, A \vdash \perp}{\Gamma \vdash_i \sim A} \sim\text{-I} \quad \frac{\Gamma \vdash A}{\Gamma \vdash_i \forall x_\gamma. A} \forall\text{-I} \\
\\
\frac{wf_\gamma B \quad \Gamma \vdash [B/x]A}{\Gamma \vdash_i \exists x_\gamma. A} \exists\text{-I} \quad \frac{\Gamma \vdash A \leftrightarrow_\gamma B}{\Gamma \vdash_i A_\gamma = B_\gamma} =\text{-I}
\end{array}$$

Fig. 2. Inference rules for $\Gamma \vdash A$

A good principle to adhere to when constructing a calculus intended for proof search in predicate logic is to inspect the structure of a formula in a lazy manner. The rationale is that the structure of the formula may be currently unknown. By waiting as long as possible until splitting search according to the different possible constructs of the formula, as much restricting judgments as possible come into play. Therefore β -reduction of a formula is performed in direct connection to matching against its head form.

Figure 2 shows the rules of the main judgment, including the sub-form $\Gamma \vdash_i A$, which contains all introduction rules. The *intro* rule has this judgment as premise, but with the succedent replaced by its head normal form.

In the case of equality, $=$, the introduction rule simply has a conversion derivation as premise. In the introduction rule for universal quantification, $\forall$ -I, x implicitly serves as the quantified variable. In the formalized soundness proof as well as in the implementation of AgsyHOL the judgments have an explicit context of universally quantified variables.

The *focus* rule identifies an antecedent and focuses on this in the premise. The upward arrow in the focusing judgment is explained in the next sub-section.

In order to allow a close relationship between a proof and the original problem that it solves, $FSC_{=}^{ho}$ has a rich set of logical connectives. This is also a natural consequence of the fact that the basis of the system is intuitionistic. However, the rules for negation corresponds to the standard representation, i.e. $\sim A \equiv A \supset \perp$, and could be excluded in an intuitionistic setting.

Apart from the rules mentioned so far there are two more, which enable classical reasoning, one for *reductio ad absurdum*, RAA, and one for the axiom of choice, AC. These rules could have been added as axioms except for the inability to quantify over types in the case of AC. However, both are included as rules of the system in order to facilitate search control. The AC rule, just like *focus*, has an focusing judgment as premise. It inserts the axiom of choice as the antecedent in *focus*.

2.3 Focusing Judgment

The focusing judgment has two modes, denoted $\Gamma; B \vdash^\uparrow A$ and $\Gamma; B \vdash^\downarrow A$, respectively. The first mode is used in the main judgment and the second in the conversion judgment, which is presented in next sub-section.

On top of these two modes, there is a sub-form which has the same function as the sub-form of the main judgment, but expresses elimination instead of introduction rules. It is denoted $\Gamma; B \vdash_e A$, and is used to express the restriction of β -reduction for antecedents.

Figure 3 shows the rules for the focusing judgment. The rules in which the turnstile is decorated with neither up nor down arrow apply to both modes of the judgment, with the same direction of the arrow both in the premise and conclusion. This accounts both for the focusing judgment itself and its elimination sub-form.

Derivations of the focusing judgment are sequences of elimination steps. An elimination sequence for the mode $\Gamma; B \vdash^\uparrow A$ terminates either with using the

$$\begin{array}{c}
\frac{A \rightarrow_{\beta} C \quad \Gamma; C \vdash_e B}{\Gamma; A \vdash B} \text{ elim} \qquad \frac{A \rightarrow_{\beta} A' \quad B \rightarrow_{\beta} B' \quad \Gamma \vdash_s B' \leftrightarrow_o A'}{\Gamma; A \vdash^{\uparrow} B} \text{ use} \\
\\
\frac{\Gamma; A \vdash C}{\Gamma; A \wedge B \vdash_e C} \wedge\text{-E}_l \qquad \frac{\Gamma; B \vdash C}{\Gamma; A \wedge B \vdash_e C} \wedge\text{-E}_r \\
\\
\frac{\Gamma \vdash A \quad \Gamma; B \vdash C}{\Gamma; A \supset B \vdash_e C} \supset\text{-E} \qquad \frac{wf_{\gamma} C \quad \Gamma; [C/x]A \vdash B}{\Gamma; \forall x_{\gamma}. A \vdash_e B} \forall\text{-E} \\
\\
\frac{\Gamma; [(\iota_{\gamma(o\gamma)}(\lambda x_{\gamma}. A))/x]A \vdash B}{\Gamma; \exists x_{\gamma}. A \vdash_e B} \exists\text{-E} \qquad \frac{}{\Gamma; \perp \vdash_e^{\uparrow} A} \perp\text{-E} \\
\\
\frac{\Gamma; (A \supset B) \wedge (B \supset A) \vdash C}{\Gamma; A_o = B_o \vdash_e C} =\text{-E}_{\text{bool}} \\
\\
\frac{\Gamma \vdash A \supset C \quad \Gamma \vdash B \supset C}{\Gamma; A \vee B \vdash_e^{\uparrow} C} \vee\text{-E} \qquad \frac{\Gamma \vdash A}{\Gamma; \sim A \vdash_e^{\uparrow} B} \sim\text{-E} \\
\\
\frac{}{\Gamma; A = B \vdash_e^{\downarrow} A = B} \text{ useeq} \qquad \frac{}{\Gamma; A = B \vdash_e^{\downarrow} B = A} \text{ useeq-symm}
\end{array}$$

Fig. 3. Inference rules for $\Gamma; A \vdash B$

antecedent in focus to prove the succedent (the **use** rule), or the elimination of disjunction, absurdity or negation. A sequence for the mode $\Gamma; B \vdash^{\downarrow} A$ ends either with **useeq** or **useeq-symm**, inferring an equality or its symmetric correspondence.

The existential elimination rule uses the choice operator applied to the predicate in order to represent the witness. In an intuitionistic setting a variable is introduced for this purpose, or, in the type theoretic case, the first projection of the corresponding Σ -type.

The rule $=\text{-E}_{\text{bool}}$ expresses the converse of Boolean extensionality. This rule eliminates the need to allow equality reasoning prior to head normal reduction in the introduction and elimination rules.

When using natural deduction for proof search, introduction steps must be constructed bottom up and elimination steps top down. In bottom-up construction of elimination steps, any rule is applicable indefinitely. A top-down construction on the other hand guides the search by deconstructing a concrete antecedent incrementally. Focused sequent calculus inverts the structure of eliminations such that a uniform bottom-up search effectively implements that division into an introductions part at the bottom of the derivation and an eliminations part at the top.

$$\begin{array}{c}
\frac{C \in \Gamma \quad \Gamma; C \vdash^\downarrow D_\gamma = E_\gamma \quad A \rightarrow_\beta A' \quad D \rightarrow_\beta D' \quad \Gamma \vdash_s A' \leftrightarrow_\gamma D' \quad \Gamma \vdash E \leftrightarrow_\gamma B}{\Gamma \vdash A \leftrightarrow_\gamma B} \text{step} \\
\\
\frac{A \rightarrow_\beta A' \quad B \rightarrow_\beta B' \quad \Gamma \vdash_s A' \leftrightarrow_\gamma B'}{\Gamma \vdash A \leftrightarrow_\gamma B} \text{simp} \\
\\
\frac{\Gamma \vdash (A \supset B) \wedge (B \supset A)}{\Gamma \vdash A \leftrightarrow_o B} \text{ext}_{\text{bool}} \quad \frac{\Gamma \vdash Ax \leftrightarrow_\tau Bx}{\Gamma \vdash A \leftrightarrow_{\tau\gamma} B} \text{ext}_{\text{fun}} \\
\\
\frac{}{\Gamma \vdash_s x \leftrightarrow_\gamma x} \text{simp}_{\text{var}} \quad \frac{\Gamma \vdash A \leftrightarrow_{\tau\gamma} C \quad \Gamma \vdash B \leftrightarrow_\gamma D}{\Gamma \vdash_s A_{\tau\gamma} B_\gamma \leftrightarrow_\tau C_{\tau\gamma} D_\gamma} \text{simp}_{\text{app}} \\
\\
\frac{\Gamma \vdash A \leftrightarrow_\tau B}{\Gamma \vdash_s \lambda x_\gamma. A_\tau \leftrightarrow_{\tau\gamma} \lambda x_\gamma. B_\tau} \text{simp}_{\text{lam}} \\
\\
\frac{\Gamma \vdash A \leftrightarrow_o C \quad \Gamma \vdash B \leftrightarrow_o D}{\Gamma \vdash_s A_o \vee B_o \leftrightarrow_o C_o \vee D_o} \text{simp}_{\text{disj}} \quad \dots
\end{array}$$

Fig. 4. Inference rules for $\Gamma \vdash A \leftrightarrow_\gamma B$ (incomplete list)

2.4 Conversion Judgment

The conversion judgment also has a sub-form, denoted $\Gamma \vdash_s A \leftrightarrow_\gamma B$, which restricts derivations to top-level simplification of the formulas. Figure 4 shows the rules for the conversion judgment.

In the **step** rule an antecedent, C , justifies, after a sequence of eliminations, rewriting the left-hand side of the conversion. The direction of rewriting is imposed by restricting the conversion of the left-hand side to simplification. In other words, the left-hand side of the succedent and one of the sides of the antecedent must have identical heads, module β -reduction. In the subsequent conversion judgment the opposite side of the antecedent appears as the left-hand side of the succedent.

The **simp** rule terminates an equality chain by requiring the formulas of the conversion to have identical heads. However, the premises in the simplification rules refer back to the full conversion judgment, allowing rewriting of the sub-formulas to take place.

As seen in the previous sub-section the rules for elimination of disjunctions, falsity and negations are not available in the inference mode of the focusing judgment, and therefore not allowed in equality reasoning. This restriction does not compromise completeness and empirical results indicate that enforcing it improves performance.

In the simplification rule for λ -abstractions, **simp_{lam}**, both sides are assumed to bind the same variable, x , and it is implicitly quantified in the premise.

The rule simp_{lam} is redundant as far as completeness is concerned, due to the presence of ext_{fun} , which also expresses η -conversion. However, using ext_{fun} when the left- and right-hand sides are both λ -abstractions introduces unnecessary substitutions in the formulas, which complicate proof search.

The simplification rules for the connectives have the expected form. The figure only includes the rule for disjunction, while the rest are omitted.

Equality reasoning is part of conversion instead of having rules for this on the level of the main judgment, which would be the case with an axiomatic representation. By interleaving simplification and rewriting, equality chains appear as locally as possible. This allows the inference system itself to guide the proof search in terms of in what parts of formula rewriting should take place. The setup also avoids the choice of in which order to do the rewriting steps in the case of independent rewriting of multiple sub-formulas.

Example 1. Let $\mathbf{h}$ define the axiom $x = z \wedge w = y$ and the conjecture be $f x y = f z w$. The solution reported by AgsyHOL is

```
=-I (simp-app (step <<h>> (And-El use-eq) simp-all simp-all)
      (step <<h>> (And-Er use-eq-sym) simp-all simp-all))
```

On the top level is an equality introduction, followed by a simplification of the application. For the first argument of f the left conjunct of $\mathbf{h}$ is used, and for the second argument the symmetrical counterpart of the right conjunct is used. The term simp-all is a shorthand for a conversion which contains no equality steps. Thanks to the representation of equality reasoning, the positions in the proof at which to rewrite the two sub-formulas are fixed. There is not one proof corresponding to the equality chain $f x y = f z y = f z w$ and another one corresponding to $f x y = f x w = f z w$.

2.5 Soundness

$\sim A_o \equiv \mathbf{N}_{oo} A_o$	$A, B ::= x_\gamma$	(variable)
$A_o \vee B_o \equiv \mathbf{A}_{ooo} A_o B_o$	$\mid \mathbf{N}_{oo}$	(negation operator)
$A_o \wedge B_o \equiv \sim (\sim A_o \vee \sim B_o)$	$\mid \mathbf{A}_{ooo}$	(disjunction operator)
$A_o \supset B_o \equiv \sim A_o \vee B_o$	$\mid \Pi_{o(o\gamma)}$	(quantification operator)
$\forall x_\gamma. A_o \equiv \Pi_{o(o\gamma)} (\lambda x_\gamma. A_o)$	$\mid \iota_{\gamma(o\gamma)}$	(choice operator)
$\exists x_\gamma. A_o \equiv \sim \forall x_\gamma. \sim A_o$	$\mid A_{\gamma\tau} B_\tau$	(application)
$\top \equiv \forall x_o. (x_o \supset x_o)$	$\mid \lambda x_\tau. A_\gamma$	(abstraction)
$\perp \equiv \forall x_o. x_o$		
$A_\gamma = B_\gamma \equiv (\lambda x_\gamma. \lambda y_\gamma. \forall z_{o\gamma}. (z_{o\gamma} x_\gamma \supset z_{o\gamma} y_\gamma)) A_\gamma B_\gamma$		

Fig. 5. STT terms and interpretation of logical connectives

The reference logic is STT [11]. Figure 5 shows the syntax of STT and how the logical connectives of $FSC_{=}^{ho}$ are interpreted as STT formulas.

Proposition Assume A is a well formed Boolean formula in $FSC_{\equiv}^{ho}$. If $\epsilon \vdash A_o$ is derivable in $FSC_{\equiv}^{ho}$ then the interpretation of a A is derivable in STT.

Proof The proof is straightforward and conducted by induction on the structure of proof derivations. The proof has been formalized in Agda [3] and the code can be found at [12]. The proof uses de Bruijn indices and explicit quantification of variables. Consequently, the inference rule for α -conversion in STT is dropped while a new rule for adding a fresh variable is required.

3 Proof Search

System $FSC_{\equiv}^{ho}$ is based on focused sequent calculus, which already facilitates proof search compared to e.g. natural deduction. It also expresses some proof search oriented details, e.g. the restrictions of where to perform β -reduction and how equality reasoning and simplification are mixed.

Another such detail is the order of B and A in the premise $\Gamma \vdash_s B \leftrightarrow_o A$ in the `use` rule. Empirical tests indicate that replacing the judgment with $\Gamma \vdash_s A \leftrightarrow_o B$ gives inferior performance. This outcome depends on the exact configuration of the conversion judgment rules. The conversion rules are devised in such a way that backward application constructs equality chains from left to right. The direction of unfolding equality chains can have an impact on performance whenever parts of the formulas in the conversion are unknown. In the case of the `use` rule the succedent is to the left and the antecedent to the right. It is reasonable to believe that if one side of an equality is more frequently uninstantiated than the other, the search is more restricted if rewriting from the more instantiated side than vice versa. Using this argument to explain the empirical difference, the succedent is presumably on average more known than the antecedent.

Although $FSC_{\equiv}^{ho}$ itself is devised with proof search in mind, some additional restrictions and search control heuristics are needed in order to achieve an efficient theorem prover. This section presents the search control mechanisms which are part of the AgsyHOL implementation. Proof search is based on applying the inference rules of $FSC_{\equiv}^{ho}$ backwards and constructing a complete derivation step by step. The search state consists of a derivation tree where some of the sub-derivations are incomplete. When no unknown parts of the derivation tree remain, a proof has been found and the given formula is valid.

An unknown part of a derivation may be a formula. Formula and derivation refinements are handled uniformly by defining a proof language with one construct for each inference rule. There is no separate mode of the search that performs unification. Unification effectively takes place when applying the rules for the conversion judgment backwards.

Unknown parts of formulas and proofs are represented by *meta-variables*. They only exist on the level of the narrowing algorithm and search control. Meta-variables that act as placeholders for formulas, e.g. B in the $\exists$ -I rule, have the role of existential variables.

Since the set of possible partial derivations is in general infinite, search depth must be limited. In AgsyHOL search is limited by the size of the derivation tree. Instead using the maximum depth of the derivation tree as the limit is inferior according to our experience.

In section 2 it was mentioned that the concept of uniform proofs is not adopted in AgsyHOL. The reason is that even though uniform proofs preserve completeness, it does not necessarily improve the overall performance of proof search. AgsyHOL uses proof size as the notion of search depth in order to stratify the search. Finding the first solution at a small depth is often more important in practice than keeping the number of solutions at a minimum. Most notably, the adoption of a uniform application of the right rule for implication is not preferable in our experience. The rule extends the set of antecedents which increases the number of choices for elimination, so it should be invoked conservatively.

The following sub-sections present the important aspects of search control in AgsyHOL. The spectrum of configurations for the parameters involved has not been explored sufficiently in order for the exact numbers to be relevant. Therefore the presentation is not very specific, but intended to give a rough view of the amount of and nature of search control we found necessary to include.

3.1 Customized Order of Refining Sub-derivations

When choosing which incomplete part of the derivation to refine, a system of priorities is used. Different priorities are assigned to the judgments of the calculus. At each step in the search an incomplete sub-derivation or formula, which has an unresolved judgment with the highest priority, is chosen. The principle which has been followed is to assign a higher priority to judgments which typically lead to a smaller degree of branching. The judgments of FSC_{-}^{ho} are listed below in decreasing order of priority in AgsyHOL.

- Simplification judgment, $\Gamma \vdash_s A \leftrightarrow_{\gamma} B$, flexible-rigid situation
- Conversion judgment, $\Gamma \vdash A \leftrightarrow_{\gamma} B$
- Focusing judgment, $\Gamma; B \vdash A$
- Main judgment, $\Gamma \vdash A$
- Simplification judgment, flexible-flexible situation

The list shows that resolving unification has top priority. Moreover, deciding whether to proceed with or end an elimination sequence has higher priority than the main judgment, so that how to end a sequence is explored before attacking any of its sub-problems.

3.2 Weighted Refinement of the Derivation Tree

The proof constructs are associated with individual weights. The size of a proof is taken to be the sum of the weights of the constructs it contains. The weights have been chosen by trial and error aiming to maximize performance. The following list summarizes the weights used in AgsyHOL.

- Introduction rules have a small weight, except for implication which has a larger weight.
- The use of a hypothesis has a small weight if it has a low degree of generality and large weight when it is highly general. The generality of a hypothesis is measured by the portion of the statement which is constituted by variables. This is an ad hoc mechanism but improves success rate on problems which have a large amount of hypotheses.
- The rules **RAA** and **AC** have a large weight.
- The rules ext_{bool} , ext_{fun} have a large weight.
- The rule **step** has a large weight.
- Simplification rules have a small weight.
- Refining a formula in the presence of a conversion constraint without pending substitutions has a small weight. In such situations unification becomes first-order and there is no real search taking place. When there are postponed substitutions, refining the formula has a slightly larger weight.

The reader may refer to the source code [12] for the exact relation between weights.

3.3 Unification

Applying the rules of conversion in $FSC_{\equiv}^{ho}$ effectively implements full higher-order unification. General higher-order unification is undecidable. However, in practice this is often, including for AgsyHOL, not very problematic. Therefore restricting unification to some decidable fragment, like pattern unification, has not been investigated.

Formulas which are unknown are constructed incrementally during search, just as the derivation tree constructs. Since formulas are higher-order, substitutions must be postponed at meta-variables and performed once they are instantiated. This is handled by using explicit substitutions in the implementation.

How to deal with occurs checks in higher-order logic is not straightforward. One observation is that a meta-variable may occur in a pending substitution and that it is not in general clear whether such an occurrence will cause circularity. A second observation is that since the formulas are constructed step by step, a circularity check the way it is done in first-order unification will not catch circularity caused by the interplay between several equality constraints.

AgsyHOL has the following way to deal with occurs check. For conversion judgments the systems keeps track of which formula heads surround the current sub-formulas. Instantiating one of the formulas to a head which is already among the surrounding heads is discouraged by associating it with a large weight. The following simple example should clarify this mechanism.

Assume $X_1, X_2, \dots$ are meta-variables. Given the equality

$$X_1 = f (g X_1)$$

X_1 will first be refined to $f X_2$. The equality is then simplified to

$$X_2 = g (f X_2)$$

keeping in mind that the head f has been traversed. Then X_2 is refined to $g X_3$ and simplification yields

$$X_3 = f (g X_3)$$

where both f and g now have been traversed. Refining X_3 to $f X_4$ in this situation is assigned a large weight since f is already among the traversed heads.

3.4 β -Reduction

Since the search state may include partially instantiated formulas β -reduction may take place in formulas which are not completely type checked. There is therefore a risk of non-termination. There are various ways to overcome this problem. Some are theoretically appealing, such as the one based on linearity/non-linearity discrimination, presented in e.g. Lengrand [13].

However, AgsyHOL has a very straightforward mechanism to deal with non-termination of reduction. It simply limits the number of consecutive reduction steps and lets this limit increase with the search depth.

4 Implementation

The source code of AgsyHOL is available at [12] and the system can be invoked on Systems on TPTP. It is written in Haskell, and is, as already mentioned, composed of three parts, namely a generic search algorithm, semantics and search control. The search algorithm is an extension of lazy narrowing [5].

Lazy narrowing is a search procedure which, given a function, f , and a fixed output, y , exploits non-strictness in f in order to efficiently find inputs, X , such that $f(X) = y$. Lazy narrowing refines the input step-wise and has meta-variables which represent yet unknown parts of it.

The semantics of $FSC_{\equiv}^{ho}$ is represented by a proof checking algorithm unaware of meta-variables. The proof checker takes a boolean formula and a proof, and decides whether the proof is a valid derivation of the formula. Proof search is achieved by applying lazy narrowing on the proof checking function with a given formula and an unknown proof, i.e. the proof given to the proof checker is initially a fully uninstantiated meta-variable.

The narrowing algorithm is first-order. In order to be able use it to construct higher-order formulas, the formulas have a first-order representation. In AgsyHOL de Bruijn-indices represented by integers are used. The search control added on top of the narrowing algorithm keeps track of the size of the variable scope.

The extensions of lazy narrowing used in AgsyHOL are concurrent conjunction [14], and customizable depth weights and priorities for instantiation. Concurrent conjunction enables inference rules with multiple premises to result in a collection of simultaneously active constraints. This improves the pruning accomplished by the narrowing algorithm. It is also a prerequisite for the mechanism of priorities, since otherwise there would always be only one blocking meta-variable.

The search control part of the theorem prover consists of annotations in the proof checker which specify the weights and priorities associated with each branching point at which execution may stop because of the presence of a meta-variable.

The lazy narrowing mechanism is implemented as an embedded library in Haskell. It uses depth-first search with back-tracking and iterated deepening. There are functional logic programming languages, such as Curry [15], which natively implement lazy narrowing. These languages however do not fit the needs regarding the extensions for expressing priorities and costs used in AgsyHOL.

The proof checker in AgsyHOL uses a sequent calculus style, not only of eliminations, but also applications in formulas. This means that instead of binary application, applications are represented by a head, which is a variable, and a list of arguments. Having this representation of application simplifies the interplay between conversion and head normal reduction in the way that only the top level construct has to be inspected in order to determine whether a formula is head normal.

AgsyHOL reads the THF0 syntax [2], the basic higher-order syntax in TPTP. When translating from THF0 to the internal representation of formulas definitions are inlined. In STT there is only one set of individuals. THF permits multiple sets and this is reflected in the implementation of $FSC_{\equiv}^{ho}$ in AgsyHOL. The representation of applications in AgsyHOL syntactically disallows β -redexes, which are therefore reduced away in the translation.

The use of RAA is problematic since it is universally applicable and difficult to know when needed. The weight of this rule is therefore high in AgsyHOL. This has the effect that proofs that require classical reasoning can be time consuming to find. In order to improve this, each problem is transformed using double negation elimination and the de Morgan laws. The transformation done in AgsyHOL aims to limit the need for classical reasoning by minimizing the number of negations. This makes it solve around 10% more problems in TPTP (within the standard 300 seconds time limit) compared to having no transformation. On the other hand, bringing the formulas to negation normal form does not improve the performance on the problem set at all. Another alternative would be to use double-negation transformation in order to avoid the need of the RAA rule. This however substantially lowers the success rate of AgsyHOL.

5 Empirical Result

AgsyHOL has been tested on the higher-order problems in the TPTP version 6.0.0, January 2014. There are 3025 problems in total. Figure 6 shows a comparison between AgsyHOL and the other higher-order theorem provers available on Systems on TPTP. All theorem provers had 300 seconds to solve each problem on a standard desktop computer. The table shows the number of confirmed theorems after timeout and how many unique solutions each system has.

AgsyHOL positions itself below the two top performing systems, Satallax and Isabelle, and gets a similar result as LEO-II. There is some distance to the best

system	solutions	unique
Satallax—2.7	2080	73
Isabelle—2013	1883	3
Isabelle-HOT—2013	1876	2
AgsyHOL—1.0	1722	17
LEO-II—1.6.0	1700	20
TPS—3.120601S1b	1469	25
cocATP—0.1.8	577	0

Fig. 6. AgsyHOL compared to other theorem provers in TPTP.

performing system. We however consider the performance of AgsyHOL to be surprisingly good, given its straightforward approach of searching for derivation proofs.

Example 2. The TPTP problem NUM636~2 declares `one` and `succ`, intended to represent the natural numbers. It then defines the axioms `one_is_first`, `succ_injective` and `induction`, and conjectures that for all x , $\text{succ } x \neq x$. The solution produced by AgsyHOL is the following:

```

1 Forall-I (\#0.(elim <<induction>>
2   (Forall-E (\#1:$i.(Not (Eq $i (<<succ>> (#1)) (#1))))
3   (Implies-E (And-I
4     (Not-I (\#1.(elim <<one_is_first>> (Forall-E (<<one>>
5       (Not-E (=-I (step #1 use-eq simp-all simp-all))))))
6     (Forall-I (\#1.(Implies-I (\#2.(Not-I (\#3.
7       (elim #2 (Not-E (=-I (step <<succ_injective>>
8         (Forall-E (<<succ>> (#1)) (Forall-E (#1)
9           (Implies-E (=-I (step #3 use-eq simp-all simp-all)) use-eq)))
10        simp-all simp-all))))))))))
11 ) (Forall-E (#0) (use simp-all))))))

```

Although reading the details of the solution requires knowledge about the exact formulation of the axioms and conjecture, the top-level structure of the proof can readily be identified. On line 1 the proof starts by using `induction`. Line 2 contains the generated λ -term which states the induction hypothesis. Lines 4 and 5 prove the base case using `one_is_first`. Lines 6–10 prove the induction step using the induction hypothesis, `#2`, and `succ_injective`.

6 Conclusions and Future Work

One weakness of AgsyHOL is how it deals with classical reasoning, the RAA and AC rules. These rules are axiomatic and universally applicable. AgsyHOL tends to be weak on problems that require classical reasoning. One direction for future work would be to improve this situation, possibly by switching to classical sequent calculus.

The empirical result shows that a calculus like the presented one, which defines a language of derivation proofs, can be the basis of a competitive theorem prover. The approach can appear naive and it is interesting that it works so well in practice.

As stated in the introduction, AgsyHOL is strong on problems involving higher-order unification. By dealing with derivation steps and formulas uniformly, unification becomes a fully integrated part of the proof search. Being good at dealing with higher-order unification the technique could have an impact on the field of automated higher-order theorem proving.

Producing derivation proofs, it could also be valuable in the area of automation of interactive theorem provers.

References

1. Sutcliffe, G.: The TPTP problem library and associated infrastructure. *J. Autom. Reason.* **43**(4) (2009) 337–362
2. Sutcliffe, G., Benzmüller, C.: Automated reasoning in higher-order logic using the TPTP THF infrastructure. *Journal of Formalized Reasoning* **3**(1) (2010) 1–27
3. Bove, A., Dybjer, P., Norell, U.: A brief overview of Agda — a functional language with dependent types. In: *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics. TPHOLs '09*, Berlin, Heidelberg, Springer-Verlag (2009) 73–78
4. Lindblad, F.: Higher-order proof construction based on first-order narrowing. *Electron. Notes Theor. Comput. Sci.* **196** (January 2008) 69–84
5. Middeldorp, A., Okui, S., Ida, T.: Lazy narrowing: Strong completeness and eager variable elimination. *Theoretical Computer Science* **167** (1995) 95–130
6. Herbelin, H.: A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In: *Selected Papers from the 8th International Workshop on Computer Science Logic*, London, UK, Springer-Verlag (1995) 61–75
7. Lengrand, S., Dyckhoff, R., McKinna, J.: A focused sequent calculus framework for proof search in pure type systems. *Logical Methods in Computer Science* **7**(1) (2011)
8. Andreoli, J.: Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation* **2** (1992) 297–347
9. Miller, D., Nadathur, G., Pfenning, F., Scedrov, A.: Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic* **51**(12) (1991) 125 – 157
10. Girard, J.Y.: A new constructive logic: Classical logic. *Mathematical Structures in Computer Science* **1**(3) (1991) 255–296
11. Henkin, L.: Completeness in the theory of types. *J. Symb. Log.* **15**(2) (1950) 81–91
12. Lindblad, F.: AgsyHOL source code and Agda formalization. Available online at <https://github.com/frelindb/agsyHOL> (2012)
13. Lengrand, S.: Normalisation & Equivalence in Proof Theory & Type Theory. PhD thesis, Université Paris 7 & University of St Andrews (2006)
14. Lindblad, F.: Property directed generation of first-order test data. In: *Trends in Functional Programming. Volume 8.*, Intellect (2008) 105–123
15. Hanus, M.: Curry: An integrated functional logic language. Language report, available online at <http://www.informatik.uni-kiel.de/~curry/report.html> (March 2006)