HOT: A Concurrent Automated Theorem Prover based on Higher-Order Tableaux

Karsten Konrad, konrad@ags.uni-sb.de

Universität des Saarlandes, Fachbereich Informatik

Abstract. HOT is an automated higher-order theorem prover based on $\mathcal{HTE}$, an extensional higher-order tableaux calculus. The first part of this paper introduces an improved variant of the calculus which closely corresponds to the proof procedure implemented in HOT. The second part discusses HOT's design that can be characterized as a concurrent blackboard architecture. We show the usefulness of the implementation by including benchmark results for over one hundred solved problems from logic and set theory.

1 Introduction

It is a well known result of Gödel's Incompleteness Theorem [Göd31] that completeness for consistent higher-order logics can not be achieved for standard model semantics. On the other hand, complete higher-order calculi can be obtained for weaker notions of semantics such as **Henkin models** [Hen50]. M. Kohlhase's article *Higher-Order Tableaux* [Koh95] presents a free variable tableau calculus for classical higher-order logic which includes substitutivity of equivalence. Kohlhase's $\mathcal{HTE}$ calculus is able to prove for instance tautologies with embedded equivalent formulas like $c(a) \vee \neg c(\neg\neg a)$. The $\mathcal{HTE}$ calculus removes a source of incompleteness that all earlier higher-order machine-oriented calculi exhibited.

However, $\mathcal{HTE}$ in its original form is not Henkin complete. For instance, it can not be used to prove the tautology:

$$(p_{(\alpha\to\beta)\to o}(f_{\alpha\to\beta}) \Rightarrow p_{(\alpha\to\beta)\to o}(g_{\alpha\to\beta})) \Rightarrow f = g$$

which states that two functions f and g must be equal if $p(g)$ follows from $p(f)$ for arbitrary predicates p. This is a direct result of extensionality in Henkin models.

By using two additional inference rules first introduced for the higher-order theorem prover LEO [Ben97b], $\mathcal{HTE}$ becomes complete relative to Henkin models (see section 2.3). The resulting improved $\mathcal{HTE}$ calculus is the theoretical backbone of our higher-order automated theorem prover HOT.

This paper is divided into two parts. In the first part (section 2), we will introduce $\mathcal{ETAB}$, a variant of the extended $\mathcal{HTE}$ calculus which closely corresponds to HOT's actual implementation. In the second part, we will discuss HOT's architecture that can be described as a blackboard system implemented in a concurrent logic programming language (see section 3).

We demonstrate the usefulness of our implementation by including benchmark results for over one hundred solved problems from logic and set theory.

2 Theoretical Background

2.1 Preliminaries

We consider a higher-order logic based on Church's simply typed lambda calculus [Chu40] and choose the set of **basetypes** $\mathcal{BT}$ to consist of the types ι and o, where o denotes the set

of truth values and ι the set of individuals. The set of all **types** $\mathcal{T}$ is inductively defined over $\mathcal{BT}$ and the right-associative type constructor $\rightarrow$. We assume that our signature Σ contains a countably infinite set of variables and constants for every type.

We have the **standard logical connectives** $\neg_{o\rightarrow o}$, $\vee_{o\rightarrow o\rightarrow o}$, $\wedge_{o\rightarrow o\rightarrow o}$, $\Rightarrow_{o\rightarrow o\rightarrow o}$, and $\equiv_{o\rightarrow o\rightarrow o}$ and the **quantifier constants** $\forall_{(\alpha\rightarrow o)\rightarrow o}$ and $\exists_{(\alpha\rightarrow o)\rightarrow o}$. Furthermore, we postulate constants for **unification constraints** $\neq^{?}_{\alpha\rightarrow\alpha\rightarrow o}$ for all types α.

If the type of a symbol is determined by the given context we avoid its explicit mention. To ease readability, we follow the usual conventions for logical expressions and λ-terms, leaving out brackets where the construction of an expression is uniquely determined. Also, we will use infix notation whenever constants denote traditional infix connectives.

We distinguish **bound variables** such as x in $\lambda x.\ x$ from **free** variables. Free variables are written in upper-case letters X, Y, V etc., while constants and bound variables appear as lower-case letters.

Terms and formulas are denoted by bold capital letters like e.g., $\mathbf{A}_\alpha$ or $\mathbf{F}$. We will sometimes write $h\overline{\mathbf{U}^n}$ to abbreviate $(hU^1,\ldots,U^n)$, where function application is considered to be left-associative. We abbreviate formulas of the form $(\forall(\lambda x.\ \mathbf{F}))$ by $\forall x.\ \mathbf{F}$ and $(\exists(\lambda x.\ \mathbf{F}))$ by $\exists x.\ \mathbf{F}$.

The notions of α-, β- and η-**conversion**, **substitutions** and the **application of substitutions** are as usual, see e.g., [Bar84].

We use the **uniform notation** for higher-order inference systems analogous to the notational system presented for first-order logics in [Fit90]. The idea behind uniform notation is to classify formulas as implicitly conjunctive (α), disjunctive (β), existentially quantified (δ) or universally quantified (γ). Using this notation, inference systems can be specified in a compact way regardless of the actual number of logical connectives or quantifiers.

Tableaux calculi usually decompose α- and β-formulas into their **components** while **instantiating** δ- and γ-formulas. Figure 1 shows the components of α- and β-formulas, and figure 2 shows the relation between higher-order γ- and δ-formulas and their instantiations. Note that the notion of α and β-formulas here is neither related to the α- and β-conversion of higher-order terms nor the use of α and β as type variables.

α	α_1	α_2	β	β_1	β_2
$\mathbf{A}\wedge\mathbf{B}$	$\mathbf{A}$	$\mathbf{B}$	$\neg(\mathbf{A}\wedge\mathbf{B})$	$\neg\mathbf{A}$	$\neg\mathbf{B}$
$\neg(\mathbf{A}\vee\mathbf{B})$	$\neg\mathbf{A}$	$\neg\mathbf{B}$	$\mathbf{A}\vee\mathbf{B}$	$\mathbf{A}$	$\mathbf{B}$
$\neg(\mathbf{A}\Rightarrow\mathbf{B})$	$\mathbf{A}$	$\neg\mathbf{B}$	$\mathbf{A}\Rightarrow\mathbf{B}$	$\neg\mathbf{A}$	$\mathbf{B}$
$\mathbf{A}\equiv\mathbf{B}$	$\mathbf{A}\Rightarrow\mathbf{B}$	$\mathbf{B}\Rightarrow\mathbf{A}$	$\neg(\mathbf{A}\equiv\mathbf{B})$	$\neg\mathbf{A}\wedge\mathbf{B}$	$\mathbf{A}\wedge\neg\mathbf{B}$

Fig. 1. α- and β-Formulas and Components

γ	$\gamma(\mathbf{A})$	δ	$\delta(\mathbf{A})$
$\forall x.\ \mathbf{F}$	$[\mathbf{A}/x]\mathbf{F}$	$\neg\forall x.\ \mathbf{F}$	$[\mathbf{A}/x]\neg\mathbf{F}$
$\neg\exists x.\ \mathbf{F}$	$[\mathbf{A}/x]\neg\mathbf{F}$	$\exists x.\ \mathbf{F}$	$[\mathbf{A}/x]\mathbf{F}$

Fig. 2. γ- and δ-Formulas and Instantiations

2.2 A Higher-Order Tableau Calculus

The $\mathcal{ETAB}$ calculus presented in this section is an extended variant of $\mathcal{HTE}$. $\mathcal{ETAB}$ uses the "naive" Skolemization known from first-order calculi. Strictly speaking, this form of Skolemization is not sound for classical higher-order logic: it would permit us to prove an instance of the Axiom of Choice which is known to be independent from higher-order logic [And73]. A solution due to [Mil83] is to associate with each Skolem constant the minimum number of arguments the constant has to be applied to.

We will now follow Kohlhase's approach and first introduce a calculus without extensionality. We begin with a set of rules that decompose the logical structure of the formulas in the tableau:

$$\frac{\beta}{\beta_1 \mid \beta_2}\,beta \qquad \frac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}}\,alpha \qquad \frac{\delta}{\delta((sk^n X_1,\ldots,X_n))}\,delta \qquad \frac{\gamma}{\gamma(V)}\,gamma \qquad \frac{\neg\neg\mathbf{F}}{\mathbf{F}}\,notnot$$

In these rules, V is a new variable and $(sk^n X_1,\ldots,X_n)$ is a Skolem term with a new Skolem function sk^n requiring a minimum of n arguments. We utilize the sound Skolemization method presented in [Mil83]. $X_1,\ldots,X_n$ are all free variables of δ.

The rules above recursively build up the tableau tree by decomposing the logical structure of formulas and adding new nodes and branches. Before we can close a tableau branch, we have to select a **linked pair** of formulas $\mathbf{F}_1$ and $\mathbf{F}_2$ in this branch from which we can construct a contradiction. The two link rules below correspond to the *cut*-rule of $\mathcal{HTE}$. They introduce unification constraints for a linked pair:

$$\frac{\begin{array}{c}\mathbf{A}_o\\ \mathbf{B}_o\end{array}}{\neg\mathbf{A} \neq^? \mathbf{B}}\,link_1 \qquad \frac{\begin{array}{c}\mathbf{A}_o\\ \mathbf{B}_o\end{array}}{\mathbf{A} \neq^? \neg\mathbf{B}}\,link_2$$

The next group of rules solve the unification constraints[1] that are introduced by the *link* rules:

[1] For a general introduction to higher-order unification and especially for the definition of a set of *general bindings* $\mathbf{G}_\alpha$ for a type α and a (head-)constant h, we refer to [GS89].

$$\frac{(\lambda x_\alpha.\ \mathbf{T}_1) \neq^? (\lambda y_\alpha.\ \mathbf{T}_2)}{[(sk^n X_1, \ldots, X_n)/x]\mathbf{T}_1 \neq^? [(sk^n X_1, \ldots, X_n)/y]\mathbf{T}_2}\, lam_1$$

$$\frac{(\lambda x_\alpha.\ \mathbf{T}_1) \neq^? \mathbf{T}_2}{[(sk^n X_1, \ldots, X_n)/x]\mathbf{T}_1 \neq^? \mathbf{T}_2(sk^n X_1, \ldots, X_n)}\, lam_2$$

$$\frac{h\overline{\mathbf{U}^n} \neq^? h\overline{\mathbf{V}^n}}{\mathbf{U}^1 \neq^? \mathbf{V}^1 \;\Big|\; \ldots \;\Big|\; \mathbf{U}^n \neq^? \mathbf{V}^n}\, dec$$

$$\frac{F\overline{\mathbf{U}} \neq^? h\overline{\mathbf{V}}}{F \neq^? \mathbf{G} \;\Big|\; F\overline{\mathbf{U}} \neq^? h\overline{\mathbf{V}}}\, gb$$

In each of these rules, $(sk^n x_1, \ldots, X_n)$ is a Skolem term for the rule's antecedent and $\mathbf{G}$ is a **general binding** that approximates the head h.

For both $\mathcal{HTE}$ and $\mathcal{ETAB}$ there is the **tableau substitution rule** *subst* that instantiates the whole tableau with an elementary substitution $[\mathbf{T}/X]$, iff some path ends in a formula of the form $X \neq^? \mathbf{T}$ such that the X is not free in $\mathbf{T}$. $\mathcal{ETAB}$ also inherits from $\mathcal{HTE}$ the **primitive substitution rule** *prim* that instantiates a flexible literal with a general binding that approximates some logical constant in $\{\neg, \vee\} \cup \{\forall_{(\alpha\to o)\to o} | \alpha \in \mathcal{T}\}$.

We call a branch Θ in a higher-order tableau **closed**, iff Θ ends in a flex/flex pair[2] or a formula of the form $\mathbf{A} \neq^? \mathbf{A}$. Note that the *subst* rule immediately closes the branch Θ that ends in a solved pair. A tableau is called **closed**, iff each branch of it is closed. A formula $\mathbf{A}$ is called a $\mathcal{ETAB}$ **theorem** iff there exists a closed higher-order tableau which can be constructed from $\neg\mathbf{A}$.

2.3 Extensionality

The **Leibniz definition** of equality defines two terms to be equal if they have the same properties. We will use $=$ as defined by

$$=_{\alpha\to\alpha\to o} := \lambda x \lambda y.\ \forall p_{\alpha\to o}.\ px \Rightarrow py$$

The following tableau construction rules complement $\mathcal{ETAB}$ with regard to extensionality in Henkin model semantics:

$$\frac{\mathbf{A}_o \neq^? \mathbf{B}_o}{\neg(\mathbf{A} \equiv \mathbf{B})}\, ext_o \qquad \frac{\mathbf{A}_\alpha \neq^? \mathbf{B}_\alpha}{\neg(\mathbf{A} = \mathbf{B})}\, ext_\alpha \qquad \frac{\mathbf{A}_{\alpha\to\beta} \neq^? \mathbf{B}_{\alpha\to\beta}}{\neg\forall p_\alpha.\ (\mathbf{A}p = \mathbf{B}p)}\, ext_{\alpha\to\beta}$$

[2] A **flex/flex** pair is a unification problem with variable heads, e.g., $P_{\iota\to\iota}a \neq^? Q_{\iota\to\iota}b$. A flex/flex pair has infinitely many incomparable solutions, e.g., $P = Q = \lambda x.\ c_\iota$ for all constants c_ι.

The three rules specify that *unification* constraints $\mathbf{A} \neq^? \mathbf{B}$ can be replaced by negated *equality* constraints using either equivalence, Leibniz equality or functional extensionality depending on the type of the constrained terms. Functional extensionality can be formalized as $\lambda x_{\bullet}\ \lambda y_{\bullet}\ \forall p(xp = yp)$. In other words, we consider two functions $\mathbf{A}$ and $\mathbf{B}$ to be equal if they map identical arguments p of their domain to equal values.

Both $ext_{\alpha\to\beta}$ and ext_{α} have no counterpart in the original $\mathcal{HTE}$ specification. The three rules together correspond to the extensionality rules of the higher-order resolution calculus $\mathcal{ERES}$ as implemented in LEO [Ben97b].

2.4 Soundness and Completeness

Soundness and completeness proofs for extensional higher-order calculi can be found in [BK97] and [Koh98]. $\mathcal{ETAB}$-like calculi with a different Skolemization are investigated in [Koh95] and [Koh98].

The soundness of Skolemization as used in $\mathcal{ETAB}$ is discussed in [Mil83]. In [BK97], this Skolemization technique is used in order to form a sound higher-order resolution calculus $\mathcal{ERES}$. Benzmüller and Kohlhase show that $\mathcal{ERES}$ is Henkin complete by using the technique of abstract consistency classes. Considering the strong relationship between $\mathcal{ETAB}$, $\mathcal{HTE}$ and $\mathcal{ERES}$, we conjecture that there is a straightforward proof for soundness and completeness of $\mathcal{ETAB}$ following the same approach.

2.5 Examples

As an example, we discuss the theorem $(\exists p_{o\bullet}\ p) \wedge (\exists p_{o\bullet}\ \neg p)$, i.e., there exists a true and a false statement. The negation of this is equivalent to the formula at the root of the following tableau:

$$
\begin{array}{c|c}
(\forall p_{o\bullet}\ \neg p) \vee (\forall p_{o\bullet}\ p) & \\
\forall p_{o\bullet}\ \neg p & \forall p_{o\bullet}\ p \\
\neg P_1 & P_3 \\
\neg P_2 & P_4 \\
\neg\neg P_1 \neq^? \neg P_2 & *\,[\neg P_3 \neq^?\ P_4] \\
*\,[\neg P_1 \neq^?\ P_2] &
\end{array}
$$

The negated theorem is a disjunction, and the *beta* rule splits the formula into two branches, each one holding a γ-formula. The *gamma* rule instantiates the γ-formula in each branch twice, creating the literals $\neg P_1$ and $\neg P_2$ for the first branch and P_3 and P_4 for the second branch. All newly introduced variables P_i are of type o. The *link* selects linked pairs for both branches by introducing the unification constraints $\neg\neg P_1 \neq^? \neg P_2$ and $\neg P_3 \neq^? P_4$. While the left branch becomes a candidate for decomposition (we remove the leading negation sign), the right branch can directly be closed using the *subst* rule. In the end, P_2 gets bound to $\neg P_1$ and P_4 to $\neg P_3$. Both branches now end in solved pairs and the tableau is closed.

The formula $p_{o\to o}(a_o \wedge b_o) \Rightarrow p(b \wedge a)$ is a theorem that requires extensionality of equivalence. The theorem is proved by the following $\mathcal{ETAB}$ tableau:

$$p(a \wedge b) \wedge \neg p(b \wedge a)$$
$$p(a \wedge b)$$
$$\neg p(b \wedge a)$$
$$\neg p(a \wedge b) \neq^? \neg p(b \wedge a)$$
$$p(a \wedge b) \neq^? p(b \wedge a)$$
$$(a \wedge b) \neq^? (b \wedge a)$$
$$\neg((a \wedge b) \equiv (b \wedge a))$$

$\neg(a \wedge b) \wedge (b \wedge a)$		$(a \wedge b) \wedge \neg(b \wedge a)$	
$\neg a$	$\neg b$	a	
$(b \wedge a)$	$(b \wedge a)$	b	
b	b	$\neg(b \wedge a)$	
a	a	$\neg b$	$\neg a$
$\neg a \neq^? \neg a$	$\neg b \neq^? \neg b$	$\neg b \neq^? \neg b$	$\neg a \neq^? \neg a$
$*[a \neq^? a]$	$*[b \neq^? b]$	$*[b \neq^? b]$	$*[a \neq^? a]$

Here again, we start with the negated theorem at the root of the tableau. We decompose its logical structure using the α-rule and create a linked pair $p(a \wedge b) \neq^? p(b \wedge a)$. We decompose using rule *dec* and obtain the unification problem $a \wedge b \neq^? b \wedge a$. The interesting step here is the transition from this unification problem which is unsolvable to the refutation proof of the equivalence $a \wedge b \equiv b \wedge a$ using the extensionality rule ext_o. From this point on, we have a plain propositional problem, and it becomes a trivial task to close the tableau.

3 The Theorem Prover HOT

Theorem provers can be characterized by many different features, for instance by their underlying logical system, programming language, heuristics, and so forth. The first part of this paper deals with the arguably most important characterization, namely the prover's underlying logic and the inference rules it employs. The $\mathcal{ETAB}$ calculus outlines a proof procedure, but only on a very abstract level. In the following we describe how this abstract proof procedure has been realized as an actual theorem proving system. We will also discuss some important design decisions, especially those that affect completeness.

We conceptualize tableaux implementations as **blackboard systems** [EM88] where tableau agents, equipped with abilities that implement parts of their underlying calculus, manipulate a blackboard-like data structure. The blackboard contains all globally accessible data such as the tableau itself and all variable assignments. The proof search space is defined by each agent's nondeterministic decisions.

Existing tableaux provers, for instance the first-order implementation presented in [Fit90], construct proofs using only one single tableau agent. We propose an alternative approach where this task is distributed among multiple concurrent agents, all working together in order to create a proof on the blackboard (see section 3.2). blackboard systems are considered a basic form of a quasi-parallel system architecture, and we propose this concept as a natural and simple implementation of parallel theorem proving[3] in tableaux calculi. The construction of each branch in a free-variable tableaux is an isolated task except for the global variable substitutions which are derived from choosing unifiers and linked pairs when closing a branch.

[3] For an overview to different approaches to parallel deduction, we refer to [BH94].

Blackboard systems employ advanced concepts to control search, for instance so-called referee agents and ambassadors [LT88] that evaluate actions of other agents if there is any conflict between their decisions. For the current implementation, we have chosen a simple control strategy that favors short branches and simple unifiers. Basically, each agent tries to close its branch as fast as possible. The first agent that computes a linked pair and a unifier decides on the global variable substitution that has to be respected by all other agents, while a complete search strategy, iterative deepening, ensures that all linked pairs and all possible unifiers up to a certain complexity will be considered eventually.

3.1 Basic Design

HOT is basically a first-order theorem prover using extended Higher-Order Unification (HOU) instead of first-order unification. Theorem proving and HOU interact: while HOU is used to close tableau branches in the theorem proving part, extensionality is implemented by calling the theorem prover within unification (see section 2.3).

HOT's theorem proving part has been inspired by the LEANTAP tableau theorem prover [BP95] that implements a complete and sound theorem prover for classical first-order logic.

Figure 3 schematically shows HOT's tableau construction. Theorem proving starts with a pre-processing step (called `read-problem`) that constructs an initial tableau. This initial tableau is the input for the initial tableau agent that tries to extend and contract branches using a search strategy based on iterative deepening.

Pre-processing Problems Proof problems in HOT are defined as a triple $\langle D, A, T\rangle$ where D is a set of higher-order definitions, A is a set of assumptions and T is a conjecture (theorem) to be proved. The pre-processing step expands all defined expressions while simultaneously checking for type errors. Definitions may be polymorph. For instance, the operator *intersection* for sets can be defined as $\lambda x_{\alpha\to o}.\ \lambda y_{\alpha\to o}.\ \lambda z_{\alpha}.\ xz \wedge yz$ with α being an arbitrary type.

Next, the theorem is negated and added to the set of assumptions to form the initial tableau. A simplification procedure removes all double negations and creates normal forms in each part of the tableau. The basic simplified tableau is the input for the initial tableau agent.

Tableau Agents A tableau agent consists of three parts, `prove`, `extend`, and `contract` (see figure 3). The `extend` part performs extension on the examined branch of the tableau and includes extensionality rules, decomposition rules and HOU. The `contract` part tries to close branches by systematically applying the *link* rules to the last member of the branch and all its literal predecessors. `prove` chooses between extending and closing a branch, calculating suitable candidates for `contract` by a simple filter and indexing mechanism.

The rules *alpha*, *beta*, *delta* and *gamma* in $\mathcal{ETAB}$ have direct counterparts in the implementation of `extend`. The *notnot*-rule is replaced by the preprocessing done in the `read-problem` step and a local formula simplification whenever a proof step introduces a new negation. Whenever HOU is applied, the agent chooses between actually unifying or applying an extensionality step *ext* if the selected term pair is not $\alpha\beta\eta$-equal. The number of *ext* applications is restricted for each branch by an extensionality depth limit.

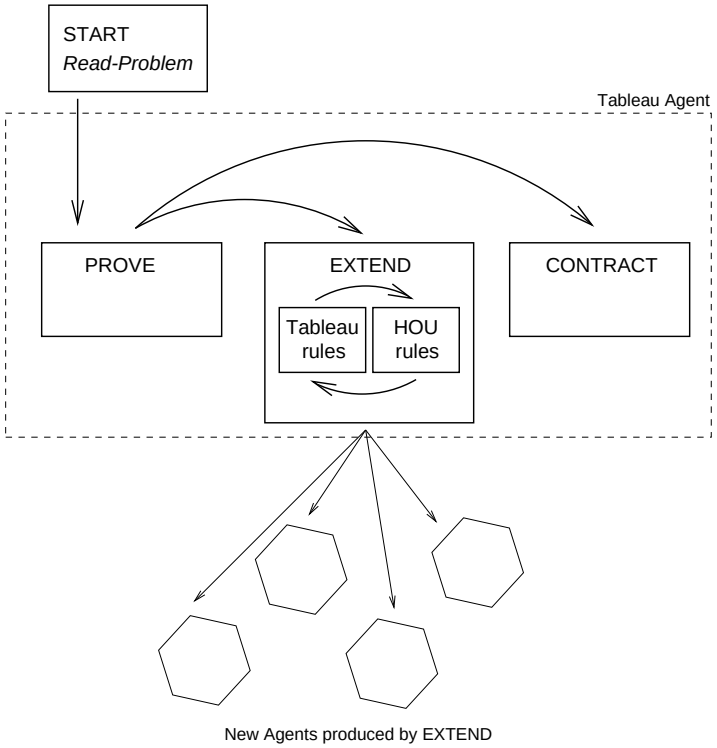

Fig. 3. HOT's schematized proof procedure.

Proof Search Tableau expansion is possibly infinite even for refutable conjectures. Hence, a naive depth-first strategy for tableau expansion would result in an incomplete search. In order to circumvent this, HOT performs **iterative deepening** depending on the **γ-depth** of branches: it first searches for all proofs which can be found with only one application of the γ-rule per branch, then for all proofs with two applications and so forth.

In the first-order case, for instance when proving theorems with LEANTAP, this search strategy is sound and complete as long as the choice of γ-formulas is fair. We can only have finitely many first-order tableau proofs for a given γ-depth, and when we make sure that eventually each γ-formulas can be used as often as needed, each possible tableau proof can be constructed. A fair choice of γ-formulas can be realized for instance by keeping all γ-formulas in a queue.

A complete proof search is harder to obtain in the higher-order case. In contrast to first-order unification, HOU is undecidable, so we can not simply use it as a procedure to decide unifiability. Instead, HOT restricts the number of general bindings for each unification attempt. This leads to a finite HOU search space. HOU is not unitary, i.e., a given unification problem may yield several solutions. HOT must consider all pre-unifiers that can be found within the unification depth limit. By gradually increasing both the unification limit and the γ-depth for each iteration, we can make sure that all unifiers will be used eventually[4].

[4] This has not been implemented yet. Section 3.3 comments on this and other possible sources of incompleteness in the current implementation.

3.2 Concurrency and the Blackboard Architecture

HOT has been implemented in OZ [Saa98], a constraint programming language based on a new computation model providing a uniform foundation for higher-order functional programming, constraint logic programming, and concurrent objects with multiple inheritance [Smo95]. OZ is a concurrent programming language, i.e., a procedure may start sub-processes, called **threads** which are executed concurrently in a fair way. HOT makes use of this feature when descending a branching tableau and when solving flex/flex pairs.

Extending Disjunctive Branches A generally useful heuristic for first-order tableaux is to extend those branches first that have a simpler, less disjunctive structure. Otherwise, a tableau agent may construct a large tree before it can detect that the variable substitutions computed so far do not allow to close the simpler parts of the tableau. In this case, the proof search must backtrack, and most of the previous work may be lost. In order to circumvent this problem, LEANTAP for instance orders formulas in a pre-processing step while moving sub-formulas in front which have a smaller, less branching structure.

For higher-order tableaux, this pre-processing can not be fully performed. Flexible heads may be instantiated either by primitive substitution or unification and change their propositional structure during proof search. HOT implements an alternative optimization of branch extension using concurrent tableau agents.

A tableau agent that encounters a disjunctive formula β will start a new thread that extends the β_1 component while the original agent continues to extend the β_2 branch (see figure 4). Also, we split expansion between distinct agents when decomposing unification problems. Instead of a single agent that has to decide on the order of branches to visit, we analyze separate branches by separate agents, each one working autonomously on its own part of the tableau. Agents communicate with each other by manipulating the global variable assignments that are part of the blackboard. The first agent which is able to close its branch by applying a substitution decides on the important choice of the next unifier to explore. An agent closing a flat, less branching branch will hopefully inhibit unnecessary unification attempts in other, more complex parts of the tableau. The search strategy backtracks and considers other possibilities to close a branch if a unifier found in this way can not be used to construct a refutation.

As long as the concurrent execution of the agents is fair in a small enough time segmentation, this technique implements a weak form of breadth-first tableau expansion. It is not unusual for a proof search to create several hundred tableau agents.

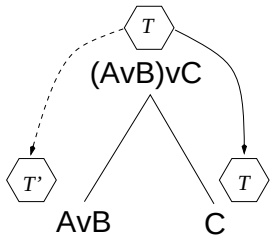

Fig. 4. An Agent T splitting up into a concurrent copy T' and an original T when analyzing a disjunction.

Solving Flex/Flex Constraints HOT's concurrent implementation gives us an efficient treatment of flex/flex pairs. $\mathcal{ETAB}$ considers all branches ending in flex/flex pairs as closed, but an instantiation of one of the flexible heads will open the branch again. Each unification problem is part of the tableau, and therefore a unique agent deals with it. In the case of a branch ending in a flex/flex pair, the agent related to the branch simply suspends and waits for one of the flexible heads to become determined. An instantiation will reactivate the agent, and the extension/contraction cycle of the branch continues.

The blackboard design leads to a nondeterministic behavior of the whole system. Since we do not synchronize agents in any way, a proof search is not guaranteed to follow the same route every time. An instantiation of a flexible head can reactivate several agents at the same time, and all of these will try to apply general bindings immediately. For a few proof problems with flexible heads, for instance Cantor's theorem (see appendix A), it is a matter of luck for the right agent to "win the race". The difference between a good and a bad choice for a unifier results in a difference of more than two orders of magnitude in the case of Cantor's theorem.

3.3 Completeness of the Implementation

Implementations of automated theorem provers always feature some trade-offs between the theoretical concept of completeness and the intended problem solving power. For instance, completeness relative to a calculus can not be maintained if the rules of a proof procedure span a search space which is too large for practical purposes. In the following, we will discuss some design decisions that affect HOT's completeness relative to $\mathcal{ETAB}$ and Henkin model semantics.

Primitive Substitutions In the case of higher-order theorem proving, the *prim* rule is a case where a single inference rule creates a much larger search space without producing more solutions except for some few examples where primitive substitutions are clearly necessary. The only flexible heads that are quite common are those introduced by Leibniz equality when unifying individual constants. As a rule, these flexible heads are better treated in a goal-oriented way by literal links than by primitive substitutions. Therefore, we have left out an implementation of *prim* in HOT.

Literal Links The *link* rules of the calculus allow to link arbitrary formulas, as long as one is the negation of the other. Experiments have shown that this feature, together with the extensionality rule ext_o, creates an overwhelming number of unification problems. We therefore restricted the *link* rules to literals, which produces deeper proofs on the one hand and less unification attempts on the other. This approach alone does not lead to incompleteness [Koh98], but as a result of the missing *prim* rule, more proofs may become unobtainable. In some way, linking non-literal formulas can have an identical effect as a sequence of *prim* applications. Instead of guessing the right substitutions for a flexible head in order to create a linked pair, the HOU will directly instantiate the flexible head with the appropriate substitution.

Extensionality The number of applications of the extensionality rule in each tableau branch as well as the number of general bindings for each unification attempt is restricted because both are (a) a potential source of infinite loops and (b) can not be simply linked to the

increasing γ-depth. Especially extensionality is critical since most proofs are unobtainable if the extensionality depth limit is too high. Note that HOT can apply ext whenever unification is attempted. An extensionality depth of 5 or higher usually is devastating.

HOT furthermore implements a possibly incomplete heuristic choice of the extensionality rule. Basically, we will apply ext_α exclusively to unification constraints of the form $\mathbf{A}_\iota \neq^? \mathbf{B}_\iota$. Leibniz equality is the potentially most expensive form of equality since it introduces new flexible heads that can be freely instantiated by unification. We therefore restrict Leibniz equality to individuals and treat boolean values and functions only by ext_o and $ext_{\alpha\to\beta}$. Following the same idea, the equality constant $=$ will be substituted by equivalence or functional extensionality whenever possible.

Indexing Indexing is a source of incompleteness because the `prove` procedure will not use terms $\mathbf{F}_1$ and $\neg\mathbf{F}_2$ for contraction if $\mathbf{F}_1$ and $\mathbf{F}_2$ have incompatible constant heads. Such pairs can not be solved alone by unification, but extensionality may nevertheless lead to a proof. On the other hand, removing indexing increases the number of unification attempts even more. With the help of the extensionality rule, this again makes proof search hopeless for all except simple examples. The question whether indexing can be implemented at all as a useful and complete heuristic for extensional higher-order proof procedures such as higher-order resolution or higher-order tableaux remains open.

4 Conclusion and Future Work

We have presented a calculus and a concurrent implementation for an automated theorem prover based on an extensional higher-order tableaux calculus. While the theorem prover's design is still quite simple, we can demonstrate that extensional higher-order tableaux is a worthwhile contribution to machine-oriented reasoning (see appendix). Some implementation-related questions that are raised in this paper are usually neglected in purely theoretical research. For instance, the problematic interaction of indexing and extensionality discussed in section 3.3 is important to all automated higher-order theorem proving systems that are based on full extensionality. HOT helped us to identify these problems and allowed us to experiment with possible solutions. For instance, HOT's concurrent architecture evolved from the observation that many proof problems can not be solved using a standard depth-first expansion of the tableaux.

HOT's intended application is the construction of natural language semantics. [KK98] describes how HOT tableaux can be used to analyze a certain class of natural language utterances (corrections). So far, there exists no theorem prover that is optimized for inferences in natural language processing. Full-scale automated theorem proving systems like TPS [ABI+96] are optimized for mathematical theorems that may require long and deeply nested proofs. Inferences in natural language processing tend to be shallow, but require answer-complete or even abductive reasoning techniques that are not as common in mathematical theorem proving. The author is especially interested in proof techniques and heuristics for semantics construction. One of these techniques is higher-order **coloured** unification (HOCU) [HK95] which can be used to guide the search for unifiers in natural language semantics [GKK97]. HOT's concurrent HOU module developed by Martin Müller and the author already includes the constraint propagation needed for computing coloured unifiers.

The author would like to link HOT to the ΩMEGA proof development system [BCF+97]. ΩMEGA features a database of mathematical knowledge (e.g., definitions in higher-order for-

malization), a large selection of examples, proof-checking and human-readable proof representation. HOT itself only uses a simple pre-processing mechanism for definition expansion and does not produce compact and easily verifiable proofs.

5 Acknowledgments

The work reported in this paper was funded by the Deutsche Forschungsgemeinschaft (DFG) in Sonderforschungsbereich SFB–378, Project C2 (LISA). The results reported here owe to stimulating and clarifying discussions with Christoph Benzmüller, Volker Sorge, Martin Müller, Michael Kohlhase and Jörg Siekmann. I would like to thank Martin Müller for the original concurrent HOU implementation and Christian Schulte for his technical assistance regarding search engines and timer routines. Furthermore, I appreciate the various helpful comments and suggestions made by the anonymous referees.

A Performance

Figure 5 shows the performance of HOT for some selected problems. The runtime has been measured on a Pentium Pro 200 using Oz 3.0.2 [Saa98].

Nr.	Theorem	msec
1	*Cantor*	1100
2	$\exists p_o.\ p$	10
3	*Counting*	660
4	$s_1 \subseteq s_2 \Rightarrow \neg\exists x.\ x \in s_1 \wedge x \notin s_2$	20
5	$a \in \wp a$	20
6	$s_1 \subseteq s_2 \Rightarrow \wp s_1 \subseteq \wp s_2$	340
7	$c \in \wp(a \backslash b) \Rightarrow c \subseteq a \wedge c \cap b = \emptyset$	130
8	$c \in \wp(a \backslash b) \equiv c \subseteq a \wedge c \cap b = \emptyset$	460
9	$\emptyset \in \wp a$	10
10	$\{x\} \subseteq \wp a \equiv x \in a$	620
11	$(a \cap b) \cap (a \backslash b) = \emptyset$	20
12	$p(a \wedge b) \Rightarrow p(b \wedge a)$	70
13	$p(a) \wedge p(b) \Rightarrow p(a \wedge b)$	260
14	$p(c \vee (a \wedge b)) \Rightarrow p((b \wedge a) \vee c)$	120
15	$(c_1 \equiv c_2) \Rightarrow p(c_1) \vee \neg p(c_2)$	150
16	$(c_1 \equiv c_2) \wedge (f = g) \Rightarrow p(fc_1) \vee \neg p(gc_2)$	490
17	$(c_1 = c_2) \Rightarrow (pc_1) \vee \neg(pc_2)$	1870
18	$(f = g) \wedge (c_1 = c_2) \Rightarrow p(fc_1) \vee \neg p(gc_2)$	15170
19	$\neg\forall x_o.\ \forall y_o.\ x = y$	20
20	$(f = g) \Rightarrow (g = f)$	160
21	$f = f$	0
22	$(f = g) \Rightarrow (fc = gc)$	2780
23	$(\forall c.\ fc = gc) \Rightarrow (pf = pg)$	3020
24	$(f = g) \wedge (g = h) \Rightarrow (f = h)$	12100
25	*Santa*	10780

Fig. 5. Benchmark results for some selected problems

The first theorem is a variant of Cantor's theorem. We prove that the set of functions $f_{\alpha\to\beta}$ is not enumerable if there exists at least one fix-point free function.

A restricted version of Cantor's theorem states that the set of functions $f_{\alpha\to o}$ is not enumerable. In this case, we do not need an additional axiom for the existence of fix-point free functions, because the existence of such functions can be inferred from the properties of type o. This version of Cantor's theorem is one of the few examples where nondeterministic proof search is really unstable: it is possible to prove the theorem very quickly – in about 70msecs – when HOT's tableau agents hit upon a favorable choice of unifiers and linked pairs. In the worst case, it can take over 25 seconds!

Theorem (3) is a plain benchmark that basically makes the theorem prover count to 6 using a unary encoding of numbers. We prove that $(f(f(f(f(f(fc))))))$ holds if both c_o and $f_{o\to o}(x) \Rightarrow f(f(x))$ hold.

Theorems (5)–(10) state some properties of power-sets. Theorem (11) is proposition (111) from [TS89]. This problem is trivial for higher-order theorem provers like LEO, TPS or HOT, while prominent first-order theorem provers are not able to solve it in less than 15 seconds (see appendix B).

All theorems so far do not require extensionality and all proofs were found with an extensionality depth limit of 0. Theorems (12) to (25) have been proved using extensionality with a depth limit of 4. The last theorem is a complicated variant of (13). Its formulation is

$$\begin{array}{l} believes(peter, \exists x.\ santa(x)) \wedge \\ believes(peter, \exists x.\ toothfairy(x)) \Rightarrow \\ believes(peter, (\exists x.\ santa(x)) \wedge (\exists x.\ toothfairy(x))) \end{array}$$

This proposition is quite hard to prove without concurrent branch expansion. An extensionality depth of 4 for this proposition will not lead to a solution because of the high branching factor of the proof search, while an extensionality limit of less than 3 makes it impossible to close some branches.

B Boolean Properties of Sets

Figure 6 shows benchmark results for propositions from [TS89], again measured on a Pentium Pro 200. For comparing these results with those achieved by the extensional higher-order resolution theorem prover LEO, we refer to [Ben97a].

Each entry documents the plain runtime (Run), the time spend for copying data (Copy) and the total time for finding the proof (Total), including garbage collection. All values are given in msec. The table is divided into three parts. The first part are those theorems which can be found with a γ-depth ranging from 2 to 5. The rest are "harder" theorems that require a γ-depth of at least 5 (problems 50, 59, 99, 100, 110, 115, 119 and 120) and those that require a γ-depth of at least 8 (51, 55, 114 and 121) in order to be solvable in less than 15 seconds. All proofs where found with an extensionality depth of 0.

Note that both LEO and HOT outperform prominent high-speed first-order theorem provers on this class of examples [Dah97]. Like LEO, HOT can not solve the problems (56) and (57) that still have a complex first-order structure after definition expansion.

Nr.	Run	Copy	Total	Nr.	Run	Copy	Total
8	30	30	80	9	40	30	70
10	40	30	70	12	30	0	30
13	10	10	20	15	20	0	20
17	20	10	40	18	90	410	700
19	180	450	810	20	150	410	780
23	110	310	640	24	30	0	30
25	800	4620	8210	27	10	0	10
28	1200	1410	3080	29	210	160	370
30	20	10	30	31	10	0	10
32	410	540	1130	33	40	30	80
34	590	620	1440	35	70	80	150
37	10	0	10	38	20	0	20
39	260	260	610	40	50	20	70
41	310	260	660	42	40	120	160
44	130	400	710	45	80	70	150
46	50	20	70	47	40	30	70
48	350	340	790	49	20	0	20
52	100	150	250	53	230	590	980
54	90	90	180	58	30	0	30
60	30	0	30	61	10	10	20
62	30	20	50	64	80	40	120
65	20	10	40	67	100	70	170
68	40	20	60	69	40	40	80
70	100	170	450	71	70	120	190
72	290	1320	2380	73	20	0	20
74	20	10	30	75	20	0	20
76	20	20	40	77	60	50	110
78	40	30	70	79	50	50	100
80	60	100	170	81	70	220	480
82	50	50	100	83	70	80	150
84	70	220	480	85	110	90	200
86	110	160	460	87	100	200	480
88	110	50	170	89	150	200	550
90	640	1810	2960	91	110	200	500
92	90	90	190	93	30	60	100
95	100	120	240	96	110	240	530
97	320	760	1890	98	250	670	1090
101	30	0	30	102	40	0	40
104	10	0	10	111	10	0	10
112	30	0	30	113	30	0	30
116	60	110	170	117	90	130	220
118	70	80	150				
50	50	20	70	59	130	140	480
99	390	2270	3440	100	310	1110	2120
110	130	240	520	115	330	1380	2410
119	160	410	750	120	130	160	420
51	180	250	590	55	200	270	630
114	160	290	640	121	130	110	250

Fig. 6. Benchmark results for set theoretical problems from [TS89]

References

[ABI+96] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[And73] Peter B. Andrews, 1973. letter to Roger Hindley dated January 22, 1973.

[Bar84] H. P. Barendregt. *The Lambda Calculus*. North Holland, 1984.

[BCF+97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In William McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

[Ben97a] Christoph Benzmüller, 1997. LEO benchmark report for Boolean Properties of Sets. `http://www.ags.uni-sb.de/projects/deduktion/projects/hot/mizar/bool-prop-sets/`.

[Ben97b] Christoph Benzmüller. A Calculus and a System Architecture for Extensional Higher-Order Resolution. Research Report 97-198, Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh,USA, June 1997.

[BH94] Maria Paola Bonacina and Jieh Hsiang. Parallelization of Deduction Strategies: An Analytical Study. *Journal of Automated Reasoning*, (13):1–33, 1994.

[BK97] Christoph Benzmüller and Michael Kohlhase. Resolution for henkin models. SEKI-Report SR-97-10, Universität des Saarlandes, 1997.

[BP95] Bernhard Beckert and Joachim Posegga. Lean, Tableau-based Deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Dah97] Ingo Dahn, 1997. Prover Statistics for Proof Problems from the Mizar Library. `http://www-irm.mathematik.hu-berlin.de/~ilf/miz2atp/mizstat.html`.

[EM88] R. Engelmore and T. Morgan, editors. *Blackboard Systems*. Addison-Wesley, 1988.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, 1990.

[GKK97] Claire Gardent, Michael Kohlhase, and Karsten Konrad. Higher-order coloured unification: a linguistic application. Submitted for publication, 1997.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte der Mathematischen Physik*, 38:173–198, 1931. English Version in [vH67].

[GS89] Jean H. Gallier and Wayne Snyder. Complete sets of transformations for general E-unification. *Theoretical Computer Science*, 1(67):203–260, 1989.

[Hen50] Leon Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2):81–91, 1950.

[HK95] Dieter Hutter and Michael Kohlhase. A coloured version of the λ-calculus. SEKI-Report SR-95-05, Universität des Saarlandes, 1995.

[KK98] Michael Kohlhase and Karsten Konrad. Higher-order automated theorem proving for natural language semantics. Seki Report SR-98-04, Fachbereich Informatik, Universität Saarbrücken, 1998.

[Koh94] Michael Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Universität des Saarlandes, 1994.

[Koh95] Michael Kohlhase. Higher-order tableaux. In *Proceedings of the Tableau Workshop*, pages 294–309, Koblenz, Germany, 1995.

[Koh98] Michael Kohlhase. Higher-order automated theorem proving. In Wolfgang Bibel and Peter Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume 2. Kluwer, 1998. forthcoming.

[LT88] Luiz V. Leao and Sarosh N. Talukdar. COPS: A System for Constructing Multiple Blackboards. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, page 547ff. Morgan Kaufmann, 1988.

[Mil83] Dale Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, 1983.

[Saa98] Programming Systems Lab Saarbrücken, 1998. The Oz Webpage: `http://www.ps.uni-sb.de/oz/`.

[Smo95] Gert Smolka. The oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer Verlag, 1995.

[TS89] Z. Trybulec and H. Swieczkowska. Boolean properties of sets. *Journal of Formalized Mathematics*, 1, 1989.

[vH67] Jean van Heijenoort, editor. *From Frege to Gödel A Source Book in Mathematical Logic, 1879-1931*. Source Books in the History of the Sciences. Harvard University Press, 1967.