Utilizing Higher-order Unifiability Algorithms in the Resolution Calculus

Tomer Libal

Microsoft Research - Inria Joint Center / Ecole Polytechnique
1 rue Honoré d'Estienne d'Orves, Palaiseau, 91120 France
tomer.libal@inria.fr

Abstract. Unifiability algorithms for higher-order logic are algorithms which decide the unification problem for sub-classes of higher-order logic by providing a witness. They contrast with unification procedures by deciding unification problems of infinitary nature, which might have infinitely many most general unifiers. Unification procedures for these sub-classes return a complete set of these unifiers and do not terminate. The common practice in automated deduction for higher-order logic is to utilize unification procedures and to force their termination by restricting the size of the generated unifiers. The unifiability algorithms, which are complete for certain sub-classes, allow us to have a more semantical approach. In this paper we claim that the standard resolution calculi for higher-order automated deduction do not take full advantage of the strengths of these algorithms and suggest a new calculus. We prove that this calculus can have an exponential speed-up over the traditional calculi.

1 Introduction

When one needs to refute higher-order formulas, the constrained resolution calculus [8] is normally preferred. Since there are possibly infinitely-many most-general unifiers, one can either eagerly compute a finite number or postpone the application of the unification rules. Another problem is the undecidability of the unifiability problem of higher-order terms [6]. The two common solutions to this problem are to either restrict the depth and size of the searched-for unifiers or again, to postpone the application of the unification rules and decrease the chances of non-termination. There also exist algorithms which decide the unifiability question for sub-classes of higher-order logic, such as for free groups [15] and semi-groups [14], monadic second-order [4] and bounded higher-order [20]. The main reasons these algorithms are not used in practice in automated deduction is the NP-hardness of the problems they solve and the absence of specialized calculi which can take advantage of their special feature, which is to decide the unifiability problem by giving a finite set of witnesses. Although the problems they solve are NP-hard, it was shown that at least with regard to most of the above problems, they are in fact in NP [12] which might imply their

usability in automated deduction in practice. Naive uses of the constrained resolution calculus can run into one of the following two problems: If the algorithms are used just for trimming non-unifiable branches by deciding if there exists a unifier, then we face serious efficiency problems as the sets of constraints keep growing. On the other hand, an eager computation of the finite set of witnesses suffers from the "locality" property - the order of choosing clauses to resolve upon, even if all are required for the refutation, greatly affects the efficiency of the search. In this paper we introduce a specialized form of the constrained resolution calculus for utilizing efficiently these unifiability algorithms and for eliminating the "locality" property. Although we were not been able to show that the "locality" property actually harms the completeness of the search for a refutation, we have been able to show an exponential speed-up of the presented resolution calculus.

This paper is organized as follows. In the first section we introduce the abstract notions of unification and unifiability algorithms and the traditional calculus which utilizes higher-order pre-unification, the constrained resolution calculus. The second section is used for the presentation of the hybrid resolution calculus. We prove in this section its relative completeness with regard to the constrained resolution calculus. In the last section we present an infinite sequence of sets of clauses, on which the hybrid resolution calculus finds a refutation exponentially faster than the constrained resolution calculus.

2 Preliminaries

2.1 Higher-order Unification

In this section we will define the general form of unification and unifiability algorithms. We assume our language to be the simply typed lambda calculus [3] whose type set contains at least the type o over a signature containing at least the logical symbols $\mathtt{T}$, $\mathtt{F}$, $\neg$, $\vee$ and Π_α for each type α with types $o, o, o \to o, o \to o \to o \to o$ and $(\alpha \to o) \to o$ respectively and the equality symbol $\doteq_\alpha$ of type $\alpha \to \alpha \to o$ for each type α. The equality symbol $=$ denotes syntactic equality between terms. *variables* are denoted by the symbols x, y, z while *constant symbols* are denoted by the rest of the lowercase Latin letters. Both might occur with sub or superscripts. The notions of free and bound variables are defined as usual. The *head* of a term is its topmost symbol which is not a λ-binder. A term whose head is a variable is called a *flex* term while a term whose head is a constant or a bound variable is called a *rigid* term. A *formula* is any term in the language which is of type o. A *literal* is a formula labeled by an intended truth value and is denoted by $[f]^v$ where f is a formula and $v \in \{\mathtt{T}, \mathtt{F}\}$. A *clause* is a disjunction $L_1 \vee \ldots \vee L_n$ of literals with $\vee$ having the usual properties (associativity, etc.), $[\mathtt{T}]^\mathtt{F}$ and $[\mathtt{F}]^\mathtt{T}$ are the identity elements and $[\mathtt{T}]^\mathtt{T}$ and $[\mathtt{F}]^\mathtt{F}$ are the absorbing elements. Except for the above-mentioned terms, the rest of the terms will be denoted in polish notation. A *substitution* is a mapping σ of variables to terms of the same type such that for some finite set of variables S, $\sigma(y) = y$ for all $y \notin S$. We extend the notion of substitutions to apply also to arbitrary terms as usual.

The *composition* of substitutions is defined as usual and is denoted by $\circ$. The substitutions mentioned in this paper are all normalized [22]. We assume all terms to be β-normalized and in η-expanded form unless otherwise stated (see for example [21]).

Definition 1 (Unification constraints). *A unification constraint is a literal of the form* $[t \doteq_\alpha s]^F$ *where* t *and* s *are terms of type* α.

Since the type α of the symbol $\doteq_\alpha$ can be derived from the types of its arguments, we will omit the subscript α from this symbol in the rest of the paper.

Definition 2 (Unification problems). *A unification problem is a disjunction of unification constraints. Given any clause, the unification problem associated with it is the disjunction of unification constraints in this clause.*

Example 1. The unification problem associated with the clause $[x(b)]^{\mathtt{T}} \vee [xa \doteq fgb]^{\mathtt{F}} \vee [a \doteq y]^{\mathtt{F}}$ is $[xa \doteq fgb]^{\mathtt{F}} \vee [a \doteq y]^{\mathtt{F}}$.

Definition 3 (Solved forms). *A unification constraint in* η*-normal form* $[x \doteq t]^F$ *is in solved form in a unification problem* S *if* x *does not occur elsewhere in* S *or in* t. *A unification problem* P *is in solved form if it contains only solved unification constraints. For a unification problem* P *in solved form, we denote by* σ_P *the substitution* $[t/x \mid [x \doteq t]^F \in P]$. *A unification problem is in pre-solved form if it contains only solved constraints or constraints of the form* $t \doteq s$ *where both* t *and* s *are flex terms. The substitution* σ_P *in this case is* $\sigma_{P'} \circ \eta_P$ *where* P' *is the sub-problem containing all solved constraints and* η_P *is a fixed substitution mapping all variables in the problem to fixed terms according to the types of the variables [9].*

Example 2. The problem $[x \doteq fa]^{\mathtt{F}} \vee [y(ga) \doteq zb]^{\mathtt{F}}$ is in pre-solved form while $[x \doteq fa]^{\mathtt{F}}$ is in solved form.

Definition 4 (Unifiers). *Given a unification constraint* $[t \doteq s]^F$, *a substitution* σ *is called a unifier for it if* $\sigma(t) = \sigma(s)$. *Let the relation* $=_v$ *extends* $=$ *such that* $t =_v s$ *if either* $t = s$ *or both* t *and* s *are flex terms, then a substitution* σ *is called a pre-unifier of the unification constraint if* $\sigma(t) =_v \sigma(s)$. *A substitution is called a (pre-)unifier of a unification problem if it (pre-)unifies all the constraints in it.*

Definition 5 (Most general unifiers). *A substitution* σ *is more general than a substitution* θ, *denoted* $\sigma \leq \theta$ *if there is a substitution* δ, *such that* $\sigma \circ \delta = \theta$. *A unifier for a unification problem is called most general if there is no other unifier of the problem, up to renaming of free variables, which is more general.*

Example 3. The substitution $[fy/x]$ is a most general unifier of the problem $[gxa \doteq g(fy)a]^{\mathtt{F}}$. Another, less general unifier, is $[a/y, fa/x]$.

Definition 6 (Complete sets of unifiers). *Given a unification problem P, we denote by $\texttt{unifiers}(P)$ the set of all its unifiers. The set Q is called a complete set of unifiers for P if $Q \subseteq \texttt{unifiers}(P)$ and for every substitution $\sigma \in \texttt{unifiers}(P)$, there exists a substitution $\theta \in Q$ such that $\theta \leq \sigma$.*

Definition 7 (Unification transformations). *A unification transformation is a rule of the form*

$$\frac{C \vee D}{\sigma(C \vee D')}$$

where D and D' are unification problems and C is a clause without unification constraints, σ is a substitution such that $\texttt{unifiers}(\sigma(C \vee D')) \subseteq \texttt{unifiers}(C \vee D)$.

Definition 8 (Unification procedures). *A unification procedure for a class of problems S is any set of unification transformations T such that for every unification problem $P \in S$ and unifier $\sigma \in \texttt{unifiers}(P)$, there is a sequence of transformations from T on P resulting in a solved problem P' such that $\sigma_{P'} \leq \sigma$. A pre-unification procedure is defined similarly where P' is a problem in pre-solved form.*

The most famous higher-order pre-unification procedure is Huet's [9]. In general Higher-order unification procedures do not terminate. Nevertheless, there are procedures for restricted classes, such as for problems with unifiers of restricted depth, which terminate.

Definition 9 (Unifiability algorithms). *A unifiability algorithm for a class of problems S is any set of unification transformations T together with a function Π from problems in S to well-founded measures such that for every unification problems $P, P' \in S$ such that P' is obtained from P using a rule in T, $\Pi(P') < \Pi(P)$ and such that if P is unifiable, we can obtain a problem P' in solved form. We will refer to this function, when the unifiability algorithm is given, just as Π.*

Definition 10 (Measure's bound). *Given a function Π as above, we define its bound for a given problem P as the maximal number of steps which can be taken before the measure $\Pi(P)$ reaches its minimal element. We will denote this value by $\texttt{bound}(\Pi(P))$.*

Note that a unifiability algorithm effectively decides the unifiability of a unification problem in its class.

The most well-known unifiability algorithm is for string unification [14]. Other algorithms are for monadic second-order unification [4] and several algorithms for context [5, 19], distributive [18], linear [11] and bounded higher-order unification [20, 13].

2.2 Huet's Constrained Resolution Calculus

In this section we will introduce the constrained resolution calculus [8].

An important aspect of clause normalization is Skolemization. We will use the Skolem terms defined in [17]

Definition 11 (Skolemization). *Given a clause C, let $x_1^{\alpha_1}, .., x_n^{\alpha_n}$ be the set of all free variables occurring in C where α_i is the type of variable x_i for $0 < i \leq n$, then a Skolem term of type α for C, which will be denoted by s_α is the term $f(x_1, .., x_n)$ for f a new function symbol of type $\alpha_1 \to .. \to \alpha_n \to \alpha$.*

The constrained resolution calculus is based on literals and clauses. Therefore, it is necessary to have rules for the normalization of terms into clauses.

Definition 12 (Simplification rules). *The set of simplification rules, which are used for normalizing terms into clauses, is given in Fig. 1.*

$$\frac{C \vee [\neg D]^{\mathtt{T}}}{C \vee [D]^{\mathtt{F}}}\ (\neg^T) \qquad \frac{C \vee [\neg D]^{\mathtt{F}}}{C \vee [D]^{\mathtt{T}}}\ (\neg^F)$$

$$\frac{C \vee [D_1 \vee D_2]^{\mathtt{T}}}{C \vee [D_1]^{\mathtt{T}} \vee [D_2]^{\mathtt{T}}}\ (\vee^T) \quad \frac{C \vee [D_1 \vee D_2]^{\mathtt{F}}}{C \vee [D_1]^{\mathtt{F}}}\ (\vee_l^F) \quad \frac{C \vee [D_1 \vee D_2]^{\mathtt{F}}}{C \vee [D_2]^{\mathtt{F}}}\ (\vee_r^F)$$

$$\frac{C \vee [\Pi_\alpha A]^{\mathtt{T}}}{C \vee [Ax^\alpha]^{\mathtt{T}}}\ (\Pi^T)^1 \qquad \frac{C \vee [\Pi_\alpha A]^{\mathtt{F}}}{C \vee [As_\alpha]^{\mathtt{T}}}\ (\Pi^F)^2$$

1. x is a new variable not occurring in A or C
2. s_α is a new Skolem term of type α

Fig. 1. Simplification Rules

The resolution and factorization rules, given next, correspond to cuts and contractions over terms which are not syntactically equal and their correctness is based on the unifiability of the added unification constraint.

Definition 13 (Resolution and factorization rules). *The resolution and factorization rules are given in Fig. 2.*

$$\frac{[A]^p \vee C \qquad [B]^{\neg p} \vee D}{C \vee D \vee [A \doteq B]^{\mathtt{F}}}\ \texttt{(Resolve)} \qquad \frac{[A]^p \vee [B]^p \vee C}{[A]^p \vee C \vee [A \doteq B]^{\mathtt{F}}}\ \texttt{(Factor)}$$

Fig. 2. Resolution and factorization rules

Since the simplification rules eliminate logical constants, such symbols cannot occur inside unification constraints. Therefore, a search for unifiers containing

logical symbols will always fail. Huet's solution to the problem was to add splittings rules which try to instantiate set variables with different terms that contain logical symbols.

Definition 14 (Splitting rules). *The set of splitting rules is given in Fig. 3 where Y, Z and z are new variables and s_α a new Skolem term.*

$$\frac{C \vee [X(\overline{t_n})]^{\mathtt{T}}}{C \vee [Y]^{\mathtt{T}} \vee [Z]^{\mathtt{T}} \vee [X(\overline{t_n}) \doteq (Y \vee Z)]^{\mathtt{F}}}\ (S^T_\vee) \qquad \frac{C \vee [X(\overline{t_n})^p}{C \vee [Y]^{\neg p} \vee [X(\overline{t_n}) \doteq \neg Y)]^{\mathtt{F}}}\ (S^{TF}_\neg)$$

$$\frac{C \vee [X(\overline{t_n})]^{\mathtt{F}}}{C \vee [Y]^{\mathtt{F}} \vee [X(\overline{t_n}) \doteq (Y \vee Z)]^{\mathtt{F}}}\ (S^{F_l}_\vee) \qquad \frac{C \vee [X(\overline{t_n})]^{\mathtt{F}}}{C \vee [Z]^{\mathtt{F}} \vee [X(\overline{t_n}) \doteq (Y \vee Z)]^{\mathtt{F}}}\ (S^{F_r}_\vee)$$

$$\frac{C \vee [X(\overline{t_n})]^{\mathtt{T}}}{C \vee [Y z^\alpha]^{\mathtt{T}} \vee [X(\overline{t_n}) \doteq \Pi_\alpha Y]^{\mathtt{F}}}\ (S^T_\Pi) \qquad \frac{C \vee [X(\overline{t_n})]^{\mathtt{F}}}{C \vee [Y s_\alpha]^{\mathtt{F}} \vee [X(\overline{t_n}) \doteq \Pi_\alpha Y]^{\mathtt{F}}}\ (S^F_\Pi)$$

Fig. 3. Splitting Rules

Definition 15 (Variants). *Let C be a clause, V the set of all free variables in C and σ a substitution mapping each variable in V to a new variable, then $C\sigma$ is a variant of C.*

Definition 16 (Constrained resolution calculus). *The constrained resolution calculus contains the rules given in figures 1, 2 and 3 as well as the rules of a unification or a unifiability algorithm.*

Definition 17 (Search strategies). *Given a set of clauses to choose from and a set of rules to apply, a search strategy chooses one rule and one or more clauses such that the rule can be applied to the chosen clauses.*

Definition 18 (Derivations). *Given a set of initial clauses and a search strategy, a derivation in the constrained resolution calculus is a sequence of clauses such that each clause is obtained, using a rule from the calculus, from variants of clauses occurring earlier in the sequence or from variants of initial clauses while respecting the search strategy at each step.*

Example 4. Fig. 4 shows a derivation of the empty clause from the following clause set using a standard first-order unification algorithm.

$$\{[Pa]^{\mathtt{T}}, [Px]^{\mathtt{F}} \vee [Pfx]^{\mathtt{T}}, [Pffa]^{\mathtt{F}}\} \tag{1}$$

The following lemma will be used later in the paper.

Lemma 1. *If a clause set S is refutable using the constrained resolution calculus, then we can obtain a refutation of S containing a clause C, such that above it we have only rule applications from figures 1, 2 and 3 and below it we have only unification rules.*

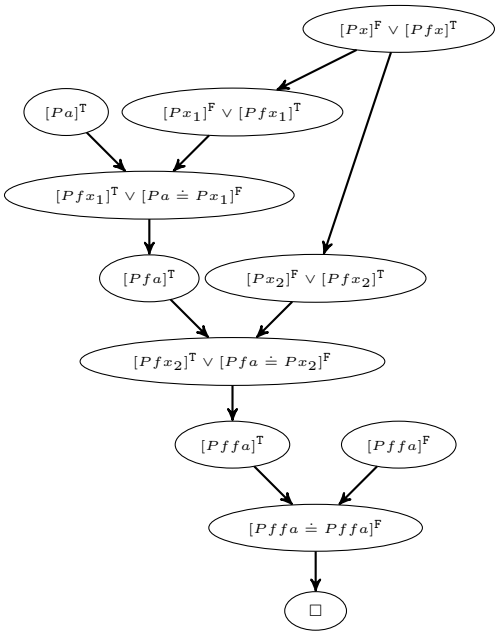

Fig. 4. A derivation of the empty clause for clause set 1

Proof. We need to show that no rule among the ones from figures 1, 2 and 3 depends on a substitution generated by the unification rules. For the splitting rules it is clear as if they are applicable after a substitution is applied they are also applicable before. For the simplification rules, we might generate a literal by applying a substitution and then apply simplification rules. But, in this case we can apply splitting on the same variable and allow unification (later) to decompose the term. With regard to `(Resolve)` and `(Factor)`, if they can be applied before they can always be applied (using splittings if necessary) also afterwards.

Remark 1. The constrained resolution calculus can be applied in a fully lazy mode by postponing the application of the unification rules. The above lemma shows that when applied in this mode, the simplification rules are not required for completeness.

Theorem 1 ([8]). *A finite set of formulas is unsatisfiable with regard to Henkin's semantics [7] if and only if it is refutable by the constrained resolution calculus using a pre-unification procedure for the simply typed lambda calculus.*

Remark 2. For the above theorem to be correct, it was shown [2] that infinitely-many extensionality initial clauses must be added.

3 The Hybrid Resolution Calculus

Our main goal in this paper is to describe a resolution calculs which utilizes unifiability algorithms instead of unification procedures. The motivation for that is clear: unifiability algorithms terminate on much larger and more interesting classes of problems than unification procedures. A trivial example is the following string unification problem [10].

$$x_1 b x_2 b \dots b x_n = x_2 x_2 x_2 b x_3 x_3 x_3 b \dots b x_n x_n x_n baaa \tag{2}$$

which has a unique unifier σ such that $\sigma(x_1) = a^{3^n}$. This problem is clearly not included in any unification class with a terminating algorithm - the depth of terms we need to search for cannot be smaller than 3^n, but the size of the problem is only $6n - 2$. But, on the other hand, a unifiability algorithm for this problem exists [14].

The hybrid resolution calculus described in this section will use two different unification approaches in parallel. The first will be to apply the unifiability algorithms eagerly in order to find a witness for the unifiability of the *current* set of constraints. The second will be to keep track of the *global* search for a refutation.

This will allow us to backtrack, once the witness we have found does not suffice, to the same unification problem again but this time compute a witness based, not on the local Π of the problem, but on the Π of the problem that was not unifiable by the original witness. In this way we are assured that our unifiability algorithms always compute the "correct" witnesses required for a refutation.

In order to keep track of the search, we will use search graphs.

Definition 19 (Search graphs). *Given a clause set S, a search graph for S is a directed graph, with a one-to-one labeling function* `lbl` *from nodes to clauses such that:*

- *for all clauses in S, there are nodes bearing them as labels.*
- *if there is an edge from node v_1 to node v_2 then the clause $\texttt{lbl}(v_2)$ can be obtained from clause $\texttt{lbl}(v_1)$ using one rule from the sets defined in definitions 12, 13 and 14.*

A full search graph is a search graph that in addition satisfies:

- *if there are nodes v and v_1 and $\texttt{lbl}(v)$ is obtained from $\texttt{lbl}(v_1)$ and some clause c by a binary rule, then there is a node v_2 such that $\texttt{lbl}(v_2) = c$ and there is an edge from v_2 to v. We will consider only full search graphs from now on.*

Note that since we might have many ways to derive each clause using the allowed rules, we might also have many edges coming into each node in the search graph.

Example 5. Fig. 5 shows a possible search graph for the search from Ex. 4. As can be seen, the main role of the graphs is to factor out the unification rules.

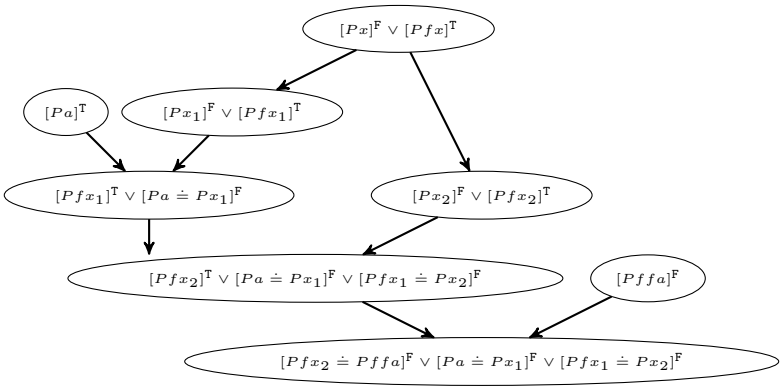

Fig. 5. Some search graph for the refutation in Ex. 4.

We will also define the following function on nodes in search graphs.

Definition 20 (Maximal descendant). *Given a node v, its maximal descendant is the node whose label has the maximal Π-value for the associated unification problem of all nodes which are descendants of v.*

We can now define the hybrid calculus.

Definition 21 (Dynamic Π). *Given a function Π, a clause C and an ongoing search for a refutation denoted by the search graph G and let n be the node corresponding to C, then the dynamic Π, denoted by Π_G, computes for C the value $\Pi(P')$ where P' is the unification problem in the clause labeling the maximal descendant of n in G.*

Definition 22 (Hybrid resolution calculus). *The hybrid resolution calculus is identical to the constrained resolution calculus but utlizes the dynamic Π_G when applying the unification rules where G is the current search graph.*

The correctness of the calculus will be proved with regard to the constrained resolution calculus. In order to prove the above theorem, we need some more technical terms.

Definition 23 (Skeletons). *A skeleton of a derivation D in the constrained resolution calculus together with a unifiability algorithm is a sequence of clauses $\mathtt{skeleton}(D)$ created recursively on D as follows:*

- *if D is an initial clause then $\mathtt{skeleton}(D) = D$.*
- *if D is a clause obtained using a rule ρ applied to previous clauses $D_1, .., D_n$:*
  - *if ρ is from figures 12, 13 and 14, then $\mathtt{skeleton}(D)$ is obtained from $\mathtt{skeleton}(D_1), .., \mathtt{skeleton}(D_n)$ using ρ.*
  - *else ρ must be a unifiability rule and therefore $n = 1$. In this case we take $\mathtt{skeleton}(D) = \mathtt{skeleton}(D_1)$.*

Example 6. The skeleton of the refutation in Ex. 4 is identical (when denoted as an acyclic graph) to the graph in Fig. 5. Note that in general the search graphs might be very complex and we chose a simple search graph for the matter of demonstration only.

Lemma 2. *For any derivation D of a clause C which is obtained using the constrained resolution calculus without any unifiability rules, if the unification problem associated with C belongs to the class S of unification problems and is unifiable, then there is a unifier σ of C such that we can obtain a derivation of the clause $C\sigma$ using the hybrid resolution calculus and a unifiability algorithm for class S.*

Proof. Let G be the current search graph, we prove by induction on the structure of $\texttt{skeleton}(D)$. Let Sk be the last clause in $\texttt{skeleton}(D)$.

- if Sk is an initial clause, we can derive it also using the hybrid calculus.
- if Sk is obtained by a rule application which does not introduce new unification constraints then we can apply the same rule using the hybrid calculus.
- otherwise, Sk is obtained by a rule ρ which introduces unification constraints. Since we do not apply substitutions in D, these unification constraints occur also in C. Let P be the unification problem associated with C. Since P is contained in the class S of unification problems, it is also unifiable by some unifier σ which can be obtained by applying the unifiability algorithm on P using the measure $\Pi(P)$. Let $C_1, \ldots, C_n$ be the clauses generating Sk. Clearly, the unification problems associated with them are subsets of P and since C is either the maximal descendant of $C_1, \ldots, C_n$ or has the same Π-value as the maximal descendant, there are substitutions $\sigma_1, \ldots, \sigma_n$, which unify them respectively (using Π_G) such that there is a substitution θ such that $\sigma = \sigma_1 \circ \ldots \circ \sigma_n \circ \theta$. Therefore, according to the induction hypthesis (and note that G does not change), there are derivations of $C_1\sigma_1, \ldots, C_n\sigma_n$ using the hybrid calculus and we can apply ρ and unification rules of the unifiability algorithm in order to obtain $C\sigma$.

Theorem 2 (Relative-completeness). *Given a finite set of formulas S, if S is refutable using the constraint resolution calculus with a unifiability algorithm A and function Π, then it is refutable using the hybrid resolution calculus with A and Π.*

Proof. Since S is refutable using the constraint resolution calculus, we can apply Lemma 1 and obtain a derivation of a clause C containing no unification rule and which is unifiable by σ. Since σ can be computed by A using Π, we can use Lemma 2 in order to obtain a derivation of $C\sigma$ in the hybrid resoluton calculus. But, since C contains only unification constraints, $C\sigma$ is the empty clause and we have obtained a refutation.

4 A Speed-up Result

In order to demonstrate how the hybrid calculus takes advantage of the special attributes of the unifiability algorithms, we will define in this section a scheme of clause sets on which the hybrid resolution calculus performs better.

We first need to define a search strategy to be used by both calculi in order to choose the next clauses and literals to process as well as an evaluation function which can be applied to each calculus and be used in order to compare their performances.

Definition 24 (The search strategy). *Given a set of clauses to choose from and a set of rules to apply, the search strategy chooses clauses and the next rule to apply as follows:*

- *choose shortest clauses according to the number of characters.*
- *choose shallowest literals, where the depth of a literal is the maximal depth of a term in it.*
- *compute first unifiers mapping variables to terms of minimal depth.*
- *choose a unification transformation first if possible, otherwise, choose a transformation respecting the previous rules.*

The first two rules in the above strategy are normally used in the search for refutations as they increase the probability for a shorter refutation and simpler unification. The third rule is a consequence of most unification and unifiability procedures which apply unification rules on one symbol at a time and therefore compute minimal unifiers first.

Here is the place to discuss why we force both calculi to apply unification transformations before other transformations. The first reason is, of course, that if we postpone the application of the unification rules to the end, the hybrid calculus will perform in an identical way to the constrained resolution calculus. In fact, this is exactly the meaning of the word hybrid in the calculus name, namely, to combine the lazy and eager approaches. The reason why we would like to apply unification eagerly is clear: with the lazy approach we might traverse paths in the search which cannot be unifiable. Therefore, the search space grows much faster. In general, the size of the search space is a major bottle neck for the efficiency of the search [1].

Definition 25 (Evaluation function for derivations). *Given a resolution calculus R, a unifiability algorithm A , a search strategy S and a clause set C, the evaluation function Ψ for R, A, S and C is computed as follows: let D be the refutation obtained using R, A and S on C and let $\sigma_1, \ldots, \sigma_n$ be all the substitutions computed in D, then $\Psi(R, A, S, C) = \Sigma_{i=1}^{n} \Sigma_{x \in V_i} \mathtt{d}(\sigma_i(x))$ where V_i contains all the higher-order variables in the domain of substitution σ_i for $0 < i \leq n$ and $\mathtt{d}$ computes the depth of a term. If there is no refutation obtainable then $\Psi(R, A, S, C)$ is undefined.*

The motivation for this measure is that we will need only unification rules and rules from Def. 13 in order to refute the clause set below and the number

of applications of the rules from Def. 13 will be much smaller than the number of application of unification rules. We ignore the size of the terms mapped to first-order variables in the measure as their computation requires normally one step and does not depend on the depth of the terms. With regard to the application rules themselves, although we have abstracted over the concrete rules in this paper, in all the unifiability algorithms mentioned, the size of the terms mapped to higher-order variables indeed determines the overall complexity of the algorithms.

Our choice of the clause set will be based on the search strategy defined above. When a search for a refutation is applied to the chosen clause set, the unifiers computed using the non-dynamic Π will not suffice for the resolution of later clauses and we will have to choose different clauses. The dynamic Π will allows us to proceed with the search without regard to the clauses chosen and therefore to have a significant speedup.

Definition 26 (The clause set). *Let $n > 0, m > 0$ and for a given unifiability algorithm and a function Π let:*

- $\Gamma(n) = [P_1(zc)]^T \overset{m}{\vee} .. \vee [P_n(zc)]^T \vee [Q(zc)]^T$
- $\Delta(i,m) = [P_i(\overbrace{a..a}^{v+1} y_i)]^F$ *for all* $0 < i \leq n$
- $\Lambda(m) = [Q(\overbrace{a..a}^{m} x)]^F$ *where* $v = \mathtt{bound}(\Pi(P_1(zc) \doteq P_1(\overbrace{a..a}^{m} y_1)))$

where the types of the predicates P_i and Q is $\iota \to o$ for $0 < i \leq n$, z and a are of type $\iota \to \iota$ and the rest of the terms are of type ι. The clause set $\Xi(n,m)$ is defined to be:

$$\Xi(n,m) = \{\Gamma(n), \Delta(1,m), \ldots, \Delta(n,m), \Lambda(m)\} \tag{3}$$

We abbreviate the constrained resolution calculus as `CRC`, the hybrid resolution calculus as `HRC` and the search strategy as `STG`.

Lemma 3. *Given a unifiability algorithm A with a function Π, the constrained resolution calculus is evaluated, when running on $\Xi(n,m)$, to* $\Psi(\mathtt{CRC}, A, \mathtt{STG}, \Xi(n,m)) = 2^n(\Sigma_{i=m}^{v}\Sigma_{j=1}^{i} i) + v + 1$.

Proof. The only possible resolution step can take place when starting with the clause $\Gamma(n)$. We resolve it with one of the Δs in order, after the elimination of all unification constraints, to obtain a unifier mapping z to $\lambda u.\overbrace{a..a}^{m} u$. We keep resolving the rest of the Δs until we are left with the Λ clause only. Clearly the substitution found so far does not suffice to allow us to add Λ to the derivation and we backtrack to the first step in order to compute a unifier mapping z to $\lambda u.\overbrace{a..a}^{m+1} u$. We continue in this way until reaching a unifier mapping z to $\lambda u.\overbrace{a..a}^{v} u$ which is the largest unifier which can be computed by A (see Def. 10). Since the size of the term in the Λ clause is defined to be larger than the depth

of v (see Def. 10, 26), we cannot find a substitution that will allow us to resolve Γ and Λ and must choose another derivation. We note that each derivation which will start by resolving Γ with one of the Δs will not suffice and will add to the measure $\Sigma_{i=m}^{v}\Sigma_{j=1}^{i}i$. We note as well that we have 2^n possibilities to choose a subset of the Δs until the empty subset will be chosen according to our search strategy and as the last run which completes the refutation chooses the clause Λ first and the substitution computed adds $v+1$ to the total evaluation, $\Psi(\texttt{CRC}, A, \texttt{STG}, \Xi(n,m)) = 2^n(\Sigma_{i=m}^{v}\Sigma_{j=1}^{i}i) + v + 1$.

Lemma 4. *Given a unifiability algorithm and a function Π, the hybrid resolution calculus is evaluated, when running on the clause set,*
to $\Psi(\mathit{HRC}, A, \mathit{STG}, \Xi(n.m)) = \Sigma_{i=m}^{v+1}\Sigma_{j=1}^{i}i$.

Proof. We have a similar execution but since we are using a dynamic Π, upon reaching the clause Λ and backtracking, we can compute the right substitution. Therefore, no attempt to choose different clauses is made. The evaluation is computed by taking into account the backtracking only and is
$\Psi(\texttt{CRC}, A, \texttt{STG}, \Xi(n,m)) = \Sigma_{i=m}^{v+1}\Sigma_{j=1}^{i}i$.

Corollary 1. *There exists a search strategy and an infinite sequence of clause sets such that refuting them is exponentially faster when using the hybrid resolution calculus over the constrained resolution calculus.*

Proof. Since v does not depend on n and since $\Sigma_{i=m}^{v}\Sigma_{j=1}^{i}i < (v-m)(\frac{v(v+1)}{2}) < v^3$, we get that by using our clause set, starting with $n \geq v$, there is an exponential speed-up by using the hybrid calculus.

Remark 3. When comparing the running time of the two calculi, denoted by our measure Ψ, we can also encounter examples where the hybrid calculus will perform worse. Such cases will happen when we expand a path in the search which is not unifiable at some point as the dynamic Π will allow us to try larger substitutions than the non-dynamic one and therefore to have a decrease in performance.

5 Conclusion

There are only a few examples where interesting arithmetical problems could be proved using a fully-automated theorem prover [16]. The main reason for that is that arithmetical problems are normally better denoted in second-order logic. Higher-order automated deduction is not so practical due to the added complexity of higher-order unification and the undecidability of the unification problem. Syntactical restrictions on unifiers, such as restricting the depth of terms, are commonly used in order to get around this problem. These restrictions perform well on some examples but poorly on others. The power of more semantical unification procedures, which are complete with regard to specific unification problems, did not reach, as far as the author is aware, the automated deduction

field. A first step towards their integration is to design a calculus which can take advantage of their benefits. Such a calculus was introduced in this paper and we have shown that this calculus can perform exponentially better when dealing with unifiability algorithms. We intend in the future to investigate the completeness of the two calculi with regard to eager applications of unifiability algorithms as we believe the "locality" property might harm the completeness of the constrained resolution calculus. This open problem can be phrased in the following way: is there a unifiability algorithm for an interesting class S and a refutable (with respect to S) set of clauses such that no refutation of this set exists when using the constrained resolution calculus with the search strategy defined in the previous section? From the relative completeness result in Thm. 2, we know this is not the case with the hybrid resolution calculus.

Another extension of the results in this paper is to test this calculus in practice. The relationship between some unifiability algorithms and arithmetics hints that such a calculus might indeed be of use in practice. An example for this relationship is the bounded higher-order case, where the bound corresponds to set operations.

References

1. Eli Ben-Sasson. Size space tradeoffs for resolution. In *Symp. Theor. Comput.*, pages 457–464, 2002.
2. Benzmüller C. Comparing approaches to resolution based higher-order theorem proving. *Synthese*, 133(1-2):203–335, 2002.
3. Alonzo Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940.
4. William M. Farmer. A unification algorithm for second-order monadic terms. *Ann. Pure Appl. Logic*, 39(2):131–174, 1988.
5. Adria Gascón, Guillem Godoy, Manfred Schmidt-Schauß, and Ashish Tiwari. Context unification with one context variable. *J. Symb. Comput.*, 45(2):173–193, 2010.
6. Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981.
7. Leon Henkin. Completeness in the theory of types. *J. Symb. Log.*, 15(2):pp. 81–91, 1950.
8. Gérard P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic.* PhD thesis, Case Western Reserve University, 1972.
9. Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.
10. A. Koscielski and L. Pacholski. Complexity of unification in free groups and free semi-groups. In *Symp. Found. Comput. Sci.*, pages 824–829 vol.2, Washington, DC, USA, 1990. IEEE Computer Society.
11. Jordi Levy. Linear second-order unification. volume 1103 of *Lect. Not. Comput. Sci.*, 1996.
12. Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. Bounded second-order unification is np-complete. In *RTA*, pages 400–414, 2006.
13. Tomer Libal. Bounded higher-order unification using regular terms. In *EPiC*, 2013. To appear.

14. G S Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR-Sbornik*, 32(2):129, 1977.
15. G. S. Makanin. On the decidability of the theory of free groups (in russian). In *Fund. Comput. Theor.*, pages 279–284, 1985.
16. William Mccune. Solution of the robbins problem. *J. Auto. Reaso.*, 19:263–276, 1997.
17. Dale A Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.
18. Manfred Schmidt-Schauß. A decision algorithm for distributive unification. *Theor. Comput. Sci.*, 208:111–148, 1998.
19. Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *J. Log. Comput.*, 12(6):929–953, 2002.
20. Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher-order unification. *J. Symb. Comput.*, 40(2):905–954, August 2005.
21. Wayne Snyder and Jean H. Gallier. Higher-order unification revisited: Complete sets of transformations. *J. Symb. Comput.*, 8(1/2):101–140, 1989.
22. Wayne S. Snyder. *Complete sets of transformations for general unification.* PhD thesis, Philadelphia, PA, USA, 1988. AAI8824793.