A Resolution Calculus Based on Eager Second-Order Bounded Unification

Alexander Leitsch Tomer Líbal
Institute of Computer Languages (E185)
Vienna University of Technology
Favoritenstraße 9, 1040 Vienna, Austria
{leitsch,shaolin}@logic.at

The efficiency of the first-order resolution calculus is impaired when lifting it to higher-order logic. The main reason for that is the semi-decidability and infinitary nature of higher-order unification algorithms, which requires the integration of unification within the calculus and results in a non-efficient search for refutations. We present a modification of the constrained resolution calculus (Huet'72) which uses an eager unification algorithm while retaining completeness. The algorithm is complete with regard to bounded unification only, which for many cases, does not pose a problem in practice.

1 Introduction

Full automation of the search for proofs of first-order theorems was first introduced with the resolution method ([6]). A key concept in the method is the unification principle. Unification is the process of finding substitutions for terms which make them syntactically equal. The unification of first-order terms was shown to be unary and terminating and therefore, the search for a unifier could always be done eagerly.

The constrained resolution calculus ([3]) is a direct adaptation of the resolution calculus to higher-order logic. The main weakness of the calculus is the unification procedure. A semi-decidable unification algorithm for higher-order logic is the pre-unification algorithm ([4]).

Due to the semi-decidability of the algorithm, unification cannot be applied eagerly, therefore highly increasing the space requirements.

Bounded higher-order unification ([9]) is an algorithm that decides if higher-order terms have bounded unifiers. Bounded unifiers are substitutions mapping variables to terms, whose number of bounded variables is restricted in advance. Although the unifiability of bounded unification problems is decidable, the algorithm finds only their minimal unifiers with respect to the size of terms ([8]).

As a resolution calculus which is based on minimal unifiers only is incomplete, the bounded unification algorithm cannot replace the pre-unification algorithm. In this paper we propose a refutationally complete resolution calculus that is based on a modified bounded unification algorithm, which can be applied eagerly.

The calculus is based on a unification algorithm that enumerates all pre-unifiers and which can compute minimal unifiers at each point of the refutation. The algorithm ([5]) handles so far second-order languages with unary variables only and it is an on-going work to extend it to the full simply-typed lambda calculus. Some interesting subclasses can already be handled by the algorithm, like second-order arithmetic with quantifier-free comprehension.

Submitted to:

The main advantage of the calculus is its ability to eagerly decide if two terms are unifiable and to get, via back-tracking, the right unifier at each point of the search. Its only weakness with regard to the pre-unification algorithm is its search for bounded unifiers only. We believe that in practice, this restriction is minor and the experiments we have done so far have confirmed this. An implementation of the algorithm is currently under development in the GAPT[1] framework.

The paper is organized as follows: The first section is dedicated for the presentation of the restricted second-order language, the constrained resolution calculus and the the regular unification algorithm, which will be used in our version of the constrained resolution calculus, given in the third section. The last section is devoted to a refutation of a simple second-order arithmetical problem, using the calculus from the third section.

2 Preliminaries

The unification problems dealt with in this paper are called bounded unification problems and differ from the ones presented in [7] by restricting the signature to unary second-order variables only. The restriction to unary second-order variables and even to second-order logic is done in order to simplify the presentation and does not restrict the results.

2.1 Contexts and n-contexts

Let Σ be a signature of function symbols, denoted f,g,h and let ar be a function mapping Σ to natural numbers, which denotes its arity. Symbols with arity 0 are called constant symbols. Let V_1 be the set of first order variables, denoted x,y,z and V_2 be the set of context variables, denoted X,Y,Z such that both are disjoint from each other and from the signature. We assume each function symbol is associated with a type from a set of types, which contains at least the boolean type. We will denote a sequence of n terms (or variables, positions, etc.) using the notation $\overline{t_n}$. Assume $ar(f) = n$ in the rest of this section. *Terms* are formed by the grammar: $t ::= x|f(t_1,..,t_n)|X(t)$. A term of the form $X(t)$ or x is called a *flex term* while $f(\overline{t_n})$ is called *rigid*. A variable not occurring as an argument to another variable in a term is called a *rigid variable occurrence*. Positions of sub-terms are defined as usual. *Contexts* extend the term grammar by $C^1[.] ::= [.]|f(t_1,..,C^1_i[.],..,t_n)|X(C^1[.])$ for $0 < i \leq n$ and denote terms containing exactly one occurrence of the hole ($[.]$), while *n-contexts* extend the term grammar by $C[.] ::= [.]|f(C_1[.],..,C_n[.])|X(C[.])|t$ and denote terms containing at most n occurrences of the hole. The notation $C(t)$ ($C^1(t)$) means replacing all occurrences of $[.]$ in $C[.]$ ($C^1([.])$ with t. The definitions of *(ground) substitutions*, *unifiers*, *most general unifiers pre-unifiers*, *solved forms* and *pre-solved forms* are as in [10]. Given a function λ-bound: $V_2 \to \mathbb{N}$, a *(ground) bounded substitution*, denoted σ, maps first order variables to (ground) terms and context variables X to (ground) n-contexts where $\lambda\text{-bound}(X) = n$. A *bounded unification problem (UBUP)* is a pair of a unification problem and the function λ-bound. The search for unifiers for the problem is restricted to bounded substitutions.

The following lemma is straight-forward.

Lemma 1. *Bounded unification problems over second-order logic with unary variables can be described by the language given above.*

[1] http://code.google.com/p/gapt/

2.2 The Constrained Resolution Calculus

In this section we will describe the constrained resolution calculus ([3]) as presented in [2] and [1].

Literals are propositions over the language of n-contexts labeled with an intended truth value. A literal is called a *unification constraint* if it is negative and its head symbol is $\doteq$. We denote literals using C and D and atomic literals using A and B. *Clauses* are disjunctions of literals. We denote the existence of a literal D in a clause C by $D \in C$.

Given a term and a context variable, the *partial bindings* ($\mathtt{PB}(X, f(\overline{s_n}))$) is the set $\{f(w_1,..,w_n)|0 < k \leq n\}$, where the $w_i \in V_1 \cup V_2$ are new variables such that $\Sigma_{0<i\leq n}\lambda\text{-}\mathtt{bound}(w_i) = \lambda\text{-}\mathtt{bound}(X)$. For $x \in V_1$ we always assume $\lambda\text{-}\mathtt{bound}(x) = 0$.

The constrained resolution calculus (CRC) over the language defined in the previous section is given in figures 1, 3, 2 and 4. The rules in Fig. 1 are an adaptation of the pre-unification algorithm ([10]) to clauses and literals.

A *derivation* in CRC is a sequence of rule applications such that the upper parts of a rule must occur before in the sequence. A derivation can also be described by an acyclic directed graph.

$$\frac{C \vee [t \doteq t]^F}{C}\ \mathtt{(Delete)} \qquad \frac{C \vee [f(\overline{s_n}) \doteq f(\overline{t_n})]^F}{C \vee [s_1 \doteq t_1]^F \vee .. \vee [s_n \doteq t_n]^F}\ \mathtt{(Decomp)}$$

$$\frac{C \vee [x \doteq t]^F \qquad x \notin Var(t)}{C\{x \mapsto t\}}\ \mathtt{(Bind}_x\mathtt{)} \qquad \frac{C \vee [X([.]) \doteq t]^F \qquad X \notin Var(t)}{C\{X \mapsto t\}}\ \mathtt{(Bind}_X\mathtt{)}$$

$$\frac{C \qquad [X(s) \doteq t]^F \in C, t \notin Var(C), u \in \mathtt{PB}(X,t)}{C \vee [X([.]) \doteq u]^F}\ \mathtt{(Imitate)}$$

$$\frac{C \qquad [X(s) \doteq t]^F \in C, t \notin Var(C)}{C \vee [X([.]) \doteq [.]]^F}\ \mathtt{(Project)}$$

Figure 1: PUA - Pre-unification Algorithm

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

$$\frac{C \vee [D_1 \vee D_1]^T}{C \vee [D_1]^T \vee [D_2]^T}\ (\vee^T) \qquad \frac{C \vee [D_1 \vee D_2]^F}{C \vee [D_1]^F}\ (\vee^F_l) \qquad \frac{C \vee [D_1 \vee D_2]^F}{C \vee [D_2]^F}\ (\vee^F_r)$$

Figure 2: Simplification Rules

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

$$\frac{[A[t]]^p \vee C \qquad [s_1 = s_2]^T}{[A[s_2]]^p \vee C \vee [t \doteq s_1]^F}\ \mathtt{(Para)}$$

Figure 3: Resolution and Paramodulation Rules

$$\frac{C \vee [Xt]^T}{C \vee [Yt]^T \vee [Zt]^T \vee [Xt \doteq (Yt \vee Zt)]^F}\ (S^T_\vee) \qquad \frac{C \vee [Xt]^p}{C \vee [Yt]^{\neg p} \vee [Xt \doteq \neg Yt)]^F}\ (S^{TF}_\neg)$$

$$\frac{C \vee [Xt]^F}{C \vee [Yt]^F \vee [Xt \doteq (Yt \vee Zt)]^F}\ (S^{Fl}_\vee) \qquad \frac{C \vee [Xt]^F}{C \vee [Zt]^F \vee [Xt \doteq (Yt \vee Zt)]^F}\ (S^{Fr}_\vee)$$

Figure 4: Splitting Rules

CRC was shown to be sound and complete (see [2] for more information about the completeness of the algorithm and its variations). Note that we restrict the splitting rules (and therefore also the simplification rules) to propositional connectives only because of our restriction of the language to unary variables. This restriction of course limits the completeness result but we can find theories, such as second-order arithmetic with quantifier-free comprehension axiom, for which the calculus is complete. The example in the last section will be taken from this theory.

3 Unification by Regular Terms

The algorithm presented in this section differs from the bounded unification algorithm in [7] in several aspects. First. it is based completely on the rules of Huet's algorithm, which makes the algorithm and its correctness proofs more concise. Second, it enumerates all pre-unifiers of a problem.

We first describe the concept of regular terms, which denote infinite sets of contexts. *Regular contexts* extend the context grammar by using the Kleene star $C_r[.] ::= C[.]|C(C_r[.])|C^*(C_r[.])$. *Regular variables* denote variables which can be mapped to members described by regular contexts only and their set is disjoint from the sets of first and context variables. We assume the existence of a unique global mapping between regular variables and regular contexts and therefore denote regular variables by $[D]_X$ where D is a regular context. We denote by $\mathtt{occ}([D]_X)$ the set of all occurrences of Kleene stars in D.

A *cycle* c in a UBUP Γ, is a sequence $\overline{X_n \doteq t_n}$ of equations of Γ such that for all $1 \leq i \leq n$, $X_{i \bmod n+1}$ occurs rigidly in t_i and there is $1 \leq j \leq n$, such that t_j is a rigid term. A cycle is called a *standard cycle* if there is exactly one such j. The variables $\overline{X_n}$ are called *cyclic variables* of c and are also denoted by $X_i \in c$. The number $m = n-1$, is called the *size of the cycle*. A standard cycle is called *path-unique* if t_j contains exactly one rigid occurrence of $X_{j \bmod n+1}$. It is called ambigious otherwise. Let $t_j = C[X_{j \bmod n+1}]$ in a path-unique standard cycle s for a non trivial context C generated by the grammar C^1 from the first section, then C is called the *cycle context* and is denoted by $\mathtt{cc}(c)$. The set of all path-unique standard cycles in a unification problem Γ is denoted by $\mathtt{scy}(\Gamma)$. An important lemma in proving the completeness of the algorithm states that cycle contexts can always be generated by the grammar C^1.

We define the *first order bound* of a UBUP Γ (denoted $b^1(\Gamma)$) as $(k+1) * l$, where k is the number of variables and l is the maximum rigid depth of a term in Γ. Another key concept is the computation of regular contexts from standard cycles, which will be done by the rec function. A main part of the completeness proof of the algorithm is devoted to proving that a variable in a standard cycle must be mapped to a context described by the right regular context.

Definition 2 (The function $\mathtt{rec}^1$)**.** *Let c be a standard cycle with cycle context C and size m and let $X_i \in c$. Then, $\mathtt{rec}^1(c,X_i) = [\mathtt{rec}_0(m,C)]_{X_i}$ and*

- *if $m = 0$ and let $S = \{C^*C' | \exists C''\ C'C'' = C\}$, then $\mathtt{rec}_0(0,C) = \{s' | s' \in\}$*

- *if* $m > 0$ *then* $\mathtt{rec}_0(m,C) = \{C^*(C'f(v_1,..W_k,..,v_n)) \mid$
  $W_k \in \mathtt{rec}_0(m-1,D_k(C'f(v_1,..,[.]_k,..,v_n))), \exists C''\ C'C'' = C, C'' = f(t_1,..D_k,..,t_n), n > 1, I \subset \{i | 0 < i \leq n, i \neq k\}\}$ *where* $v_i = x_i$ *if* $i \in I$ *and* $v_i = t_i$ *otherwise. The* x_i *are new first order variables.*

We denote by $C \in [D]_X$ *the fact that a context* C *is a member of the regular language defined by* D*.*

Example 3. *Given a signature consisting of constants* a *and* b*, unary function symbols* e *and* h *and binary functions symbols* f *and* g*:*

- $\mathtt{rec}_0(2, ef(ha, hg([.],b)), X) = [(ef(ha, hg([.],b)))^*(ef(x_1, \mathtt{rec}_0(1, hg(ef(x_1,[.]),b))))]_X$ *for* $C' = e[.]$, $C'' = f(ha, hg([.],b))$ *and* $I = \{1\}$.
- $\mathtt{rec}_0(1, hg(ef(x_1,[.]),b)) = (hg(ef(x_1,[.]),b))^* hg(\mathtt{rec}_0(0, ef(x_1, hg([.],x_2))), x_2)$ *for* $C' = h[.]$, $C'' = g(ef(x_1,[.]),b)$ *and* $I = \{2\}$.
- $\mathtt{rec}_0(0, ef(x_1, hg([.],x_2))) = ef(x_1, hg([.],x_2))^* ef(x_1, h[.])$ *for* $C' = ef(x_1, h[.])$.

The following definitions are used in the completeness proof and describes the extension of the $\mathtt{rec}^1$function to n-contexts.

Definition 4 (Transforming contexts to n-contexts)**.** *Let* C *be a context and* X *a variable such that* $\lambda\text{-}\mathtt{bound}(X) = n$*, then* $\mathtt{bext}(C,X)$ *is the set of all contexts* C' *which are equivalent to* C *but contain up to* $n-1$ *holes in arbitrary positions. The hole in* C *is called the principle hole in* C'*. The notation* $C' *_p t$ *for such a* n*-context* C' *means replacing the principle hole in* C' *with* t*.*

Definition 5 (The function rec)**.** *Let* c *be a standard cycle with cycle context* C *and let* $X_i \in c$*, then* $\mathtt{rec}(C,m,X_i) = \{C^0 | [D]_{X_i} \in \mathtt{rec}_0(m,C), C' \in D, C^0 \in \mathtt{bext}(C', X_i)\}$

We assume the existence of a mapping imt-bound from context variables to natural numbers, which denotes the maximal depth of terms which can be generated by applying the (Imitate) rule. Note that imt-bound differs from the function λ-bound, which denotes the maximal number of holes ([.]) allowed. Let L be a context, the unification algorithm in Fig. 5 is based on the one in Fig. 1.

The following lemma is immediate.

Lemma 6. *Given a derivation of a clause* L*, such that the derivation contains an application of* (Rec) *for cycle context* C*, cycle size* m *and cycle variable* X *and* L *does not contain a regular variable* $[D]_X$ *for some* D *and let* σ *be the derived substitution, then* $\sigma(X) \in \mathtt{rec}(C,m,X)$*.*

Proof. Following the definitions of $\mathtt{rec}^1$, bext and rec. □

The main results in [5] are with regard to the correctness and termination of CUA, whose main difference over BUA is that the number of bounded variables allowed (λ-bound) is at most one. In the next two sections, we will extend the results for UBUPs.

3.1 Correctness of the Algorithm

The proof of the soundness of BUA follows directly from the proof of the soundness of the algorithm CUA.

Lemma 7 (Soundness)**.** *Every substitution obtained from a pre-solved form of a* BUA *execution on a* UBUP P *is a pre-unifier of* P*.*

In order to prove the completeness of CUA we have proven ([5]) that:

- the depth of terms mapped to acyclic variables is bounded by b^1 as long as there are no application of (Project) or (Rec)(Lemma 17 in [5]).

$$\frac{C \vee [t \doteq t]^F}{C}\ \texttt{(Delete)} \qquad \frac{C \vee [f(\overline{s_n}) \doteq f(\overline{t_n})]^F}{C \vee [s_1 \doteq t_1]^F \vee .. \vee [s_n \doteq t_n]^F}\ \texttt{(Decomp)}$$

$$\frac{C \vee [x \doteq t]^F \qquad x \notin Var(t)}{C\{x \mapsto t\}}\ \texttt{(Bind}_x\texttt{)} \qquad \frac{C \vee [X([.]) \doteq t]^F \qquad X \notin Var(t)}{C\{X \mapsto t\}}\ \texttt{(Bind}_X\texttt{)}$$

$$\frac{C \qquad [X(s) \doteq t]^F \in C, t \notin Var(C), u \in \texttt{PB}(X,t), \texttt{imt-bound}(X) > 0}{C \vee [X([.]) \doteq u]^F}\ \texttt{(Imitate)}^3$$

$$\frac{C \qquad [X(s) \doteq t]^F \in C, t \notin Var(C)}{C \vee [X([.]) \doteq [.]]^F}\ \texttt{(Project)}^4 \qquad \frac{C \qquad c \in \texttt{scy}(C), X \in c, C' \in \texttt{rec}^1(c,X)}{C \vee [X[.] \doteq C']^F}\ \texttt{(Rec)}$$

$$\frac{C \qquad [[L_0^*(L[.])]_X \doteq t]^F \in C, L \in C_C}{C\{[L_0^*(L[.])]_X \mapsto L'[.]\}}\ \texttt{(Rec}_0^1\texttt{)}^{4,5} \qquad \frac{C \qquad [[L^*(L_2[.])]_X \doteq t]^F \in C, L_2 \in C_R}{C\{[L^*(L_2[.])]_X \mapsto [L_2[.]]_X\}}\ \texttt{(Rec}_0^2\texttt{)}^4$$

$$\frac{C \qquad [[L^*(L_2[.])]_X \doteq t]^F \in C}{C\{[L^*(L_2[.])]_X \mapsto L' *_p ([L^*(L_2[.])]_X\}}\ \texttt{(Rec}^*\texttt{)}^{4,6}$$

1. for all variables Y, we initialize $\texttt{imt-bound}(Y) = b^1(C)$.
2. each call of `(Imitate)`, `(Project)` and `(Rec)` is followed by $\texttt{(Bind}_X\texttt{)}$.
3. all fresh variables $w_i \in V_2$ introduced in u have $\texttt{imt-bound}(w_i) = \texttt{imt-bound}(X) - 1$.
4. for all variables $Y \in Var_2(C)$, $\texttt{imt-bound}(Y) = b^1(C)$.
5. $L' \in \texttt{bext}(L,X)$, assuming L' has m holes, then $\lambda\texttt{-bound}(X)$ is decreased by m.
6. $L' \in \texttt{bext}(L,X)$, assuming L' has m holes, then $\lambda\texttt{-bound}(X)$ is decreased by $m-1$.

Figure 5: BUA - Unification Rules[1,2]

- given cycles, we can obtain standard cycles without using `(Project)` or `(Rec)`and without mapping variables to terms whose depth is bigger than b^1 (Lemma 23 in [5]).
- given a standard cycle, then for every pre-unifier of the problem, there is a variable in the cycle which is mapped to a term that can be generated by the `rec` function (Lemma 24 in [5]).

The fact that we allow for arbirary values in the range of $\lambda\texttt{-bound}$ does not affect the proof of Lemma 17 ([5]). However, the proof of Lemma 24 ([5]) does not hold any longer because the assumptionm that there might be at most one hole in terms mapped to cyclic variables, plays a crucial role. In order to simplify the adaptation of the proof to n-contexts, we first require that our standard cycles are path unique.

Lemma 8. *Given a cycle $\overline{e_m = X_m \doteq t_m}$ in a (Z) CUP P, we can derive a unique-path standard cycle using the BUA rules (Delete), (Decomp), (Bind) and (Imitate).*

Proof. We follow the same proof as in the proof of Lemma 23 ([5]) for obtaining a standard cycle. If the cycle is path-unique, then we are done. Otherwise, the obtained ambigious standard cycle has a cycle context of maximal depth of $m * l$, where l is the maximal depth of a term t_i. The sizes of the positions of all occurrences of the variable X_1 in the context cannot be more than m as they are on rigid positions. Assume the size of one of them is $m_0 \leq m$, then another m_0 calls to the `(Imitate)` rule will produce

a unique-path standard cycle (with cycle context in equation $m-1$) such that all the original variables of the cycle are mapped to contexts of depth smaller than the first-order bound. This is so because after applying the transformations in the proof of Lemma 23 ([5]), each variable is mapped to a context whose depth is bounded by $m*l$. The extra `(Imitate)` rule applications may increase it by at most m. Therefore, the variables are indeed mapped to contexts whose depth is bounded by $(m+1)*l \leq b^1(P)$ for P a problem containing the cycle. □

We can now give the version of Lemma 24 ([5]) for UBUPs.

Lemma 9. *Let P be a UBUP with a path-unique standard cycle c (of length m and context C), then for any pre-unifier σ of P, there is a variable X in the cycle such that $\sigma(X) \in \mathtt{rec}(C,m,X)$.*

Proof. Let $\overline{X_n}$ ($n = m+1$) be the context variables in the cycle and let p_i^j ($j \leq \lambda\text{-}\mathtt{bound}(X_i)$) be the positions of all holes in $\sigma(X_i)$ and p be the position of the hole in C^*. Define the position p_0 to be the greatest common prefix of some member of $\{p_i^j\}$ for each i and of p. Let $p = p_0.r.q$ for some position q and let $(\Sigma(C))^l C'|_{p_0} = f(t_1,..,C''_r([.]),..,t_n)$ for some l and $\sigma(C) = C'f(v_1,..,C''_r([.]),..,v_n)$ and let $A = (\Sigma(C))^l C'$. We prove by induction on m that there is $0 < i \leq n$ such that $\sigma(X_i) \in \mathtt{rec}(C,X_i)$.

- for $m = 0$ we first show that p_0 is a position of a hole in $\sigma(X_1)$. Assume otherwise, then none of the holes in $\sigma(X_i)$ has a prefix $p_0.r$. The cycle, after applying σ, is of the following form:

  $$Af(v'_1,..,v'_r,..,v'_n) = \sigma(C)(A(f(v''_1,..,v''_r,..,v''_n))),$$

  After applying `(Decomp)`s up to position p_0 we get:

  $$f(v'_1,..,v'_r,..,v'_n) = f(v_1,..,C''(C'(f(v''_1,..,v''_r,..,v''_n)))_r,..,v_n),$$

  Because none of the holes in $\sigma(X_i)$ has a prefix $p_0.r$, we have that $v'_r = v''_r$ and after one `(Decomp)` we get

  $$v'_r \doteq C''C'(f(v''_1,..,v'_r,..,v''_n))$$

  which is a contradiction to the unifiability of the cycle. We now see that since we have at most $m = \lambda\text{-}\mathtt{bound}(X_1)$ holes in $\sigma(X_1)$ and one hole is at position p_0, $\sigma(X_1) \in \mathtt{rec}(C,0,X_1)$.

- for $m > 0$, if there is $0 < i \leq n$ such that $\sigma(X_i)$ has a hole at position p_0, then $\sigma(X_i) \in \mathtt{rec}(C,X_i)$. Assume otherwise, then p_0 is a proper prefix and therefore there is $0 < i \leq n$ such that none of the holes in $\sigma(X_i)$ has a prefix $p_0.r$. Let I be the set of all indices of variables X_i with a hole with a prefix $p_0.r$ and $J = \overline{I}$. Clearly, $size(I) < n$. We consider the cycle after applying `(Imitate)`s and `(Decomp)`s up to position p_0:

  $$X'_1t_1 \doteq X'_2s_1, \,..\, , X'_nt_n \doteq f(v_1,..,C''(C'(X'_1s_n))_r,..,v_n)$$

  Since $\sigma(X'_j)$ for all $j \in J$ does not contain a hole with prefix $1.r$ we can apply `(Imitate)`s with values in PB placing a first-order variable at that position and after applying `(Decomp)`s and `(Bind`$_x$`)`s we get (among other equations):

  $$X''_{I_1}t'_1 \doteq X''_{I_2}s'_1, \,..\, , X''_{I_{size(I)}}t'_{size(I)} \doteq C''(C'(f(v'_1,..,X''_{I_1}s''_{size(I)},..,v'_n)))$$

  where $\overline{t'_{size(I)}}$ and $\overline{s'_{size(I)}}$ are permutations of subsets of $\overline{t_n}$ and $\overline{s_n}$. which is a smaller path-unique standard cycle and we can apply the induction hypothesis to obtain that there is $i \in I$ such that $\sigma(X''_i) \in \mathtt{rec}(D,m-1,X''_i)$ for $D = C''(C'(f(v'_1,..,[.]_r,..,v'_n)))$. According to the definition of `rec`, $\sigma(X'_i) \in \mathtt{rec}(C,m,X'_i)$.

□

The completeness of BUA can now be shown.

Lemma 10. *The algorithm BUA is complete with respect to PUA.*

Proof. Following lemmas 6, 9 and 26 ([5]). □

3.1.1 Termination and Minimal Unifiers

If we are interested in minimal unifiers only, then we can restrict the iterations over the Kleene star used in the regular contexts, i.e. we can bound the size of $\mathtt{occ}([D]_X)$ for all context variables $[D]_X$. The restriction is based on the exponent of periodicity, while the completeness proof is based on the periodicity lemma ([8]). let e be a function from occurrences of Kleene stars in regular contexts to natural numbers. Let C be a clause, we initialize the value of each Kleene star occurrence to $\mathtt{exp}(C)$, the exponent of periodicity of C and we decrease the value by 1 every time the $(\mathtt{Rec}^*)$ rule is applied to the occurrence. Since the exponent of periodicity restricts the number of repetition of identical contexts and we add holes into the generated contexts in $(\mathtt{Rec}^*)$ and $(\mathtt{Rec}_0^1)$, the initial bound value should be multiplied by $\lambda\text{-}\mathtt{bound}(X)$ for regular variables $[D]_X$. This holds as the number of possible holes is bounded by $\lambda\text{-}\mathtt{bound}(X)$ and the rest of the contexts are indeed identical. The restricted BUA (RBUA) is the unification algorithm where the $(\mathtt{Rec}^*)$ rule can be applied on a Kleene star's occurrence only if this value is greater than 0. Termination of the algorithm is proved similarly to the proof given in [5] by using the measure $\mu =< m_1, m_2, m_3, m_4, m_5, m_6 >$ where:

- $m_1 = \Sigma_{X \in Var_2(C)} \lambda\text{-}\mathtt{bound}(X) + \Sigma_{[D]_X \in Var_r(C)} \lambda\text{-}\mathtt{bound}(X)$,
- $m_2 = size(Var_2(C))$
- $m_3 = \Sigma_{[D]_X \in Var_r(C), o \in \mathtt{occ}([D]_X)} \mathtt{e}(o)$,
- $m_4 = \Sigma_{X \in Var_2(C)} \mathtt{imt\text{-}bound}(X)$,
- m_5 is the number of first order variables and
- m_6 is the number of symbols other than $\doteq$ in the problem.

The measure is decreased in a similar way to that in Thm. 34 ([5]). The only exception is that now $(\mathtt{Bind}_X)$might not decrease m_1 in the case it replaces a context variable by a regular variable $((\mathtt{Rec}))$ or a regular variable by a regular variable $((\mathtt{Rec}^*)$ and $(\mathtt{Rec}_0^2))$. In the first case, m_2 is decreased and in the second case m_3 is decreased.

4 The Resolution Calculus

In this section we will describe a resolution calculus which is based on CRC. Our calculus differs from CRC in the choice of the unification algorithm and in its use of global values.

The following corollary follows the soundness and completeness of the BUA.

Corollary 11. *The constrained resolution calculus given in figures 5, 3, 2 and 4 is sound and complete with regard to the calculus given in Sec. 2.2.*

We will use our version of CRC in the rest of the paper.

Trying to apply unification rules on unification constraints eagerly may result in non-termination even if the clause set is refutable. This is because the pre-unification algorithm is semi-decidable only. In the rest of this section we will show how the new algorithm can be used in such an eager strategy.

We face two problems, on the one hand the set of pre-unifiers is infinite and therefore any attempt to obtain it eagerly might not terminate. On the other hand, when using the RBUA and using the exponents of periodicity, we obtain minimal unifiers only.

Our solution to the problem is to follow both approaches simultaneously. We search for minimal unifiers only but the minimal unifiers will be with regard to an increasing set of constraints, which at the end, will encompass all the constraints used in the refutation.

Definition 12 (Derivable clauses)**.** *Given a clause c, the set of all its derivable clauses* $\mathtt{deriv}(c)$ *of c is the set containing all clauses, which can be derived from a set of clauses containing c using CRC without applications of the unification rules from Fig. 5.*

Definition 13 (Dynamic Exponents)**.** *The dynamic exponent* de *of the clause c is chosen non-deterministically from the infinite set* $\{\mathtt{exp}(d) | d \in deriv(c)\}$.

Now we can redefine the RBUA from the previous section to use a function e whose initial values are based, not on the exponent of periodicity of the current clause but on that of a clause derivable from it.

The following definition tries to normalize sets of unification constraints by ignoring applications of substitutions. We need first the following lemma, which is similar to the mid-sequent theorem of sequent calculus. For the proof of the lemma we will consider CRC without the (Para) rule. Since this rule is redundant in the presence of the Leibniz equality, which is expressible in our language, this modification should not have any theoretical impact. We hope in the future to establish the results of this section in the presence of the (Para) rule as well.

Lemma 14 (Mid-clause lemma)**.** *If there is a derivation in CRC of a clause C, then there is a derivation of a clause* C' *in CRC without the application of rules from Fig. 5 such that C can be derived from* C' *by applications of rules from Fig. 5 only.*

Proof. We need to show that no rule among the ones from figures 3, 2 and 4 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. The simplification rules deal with logical connectives only, which cannot be generated by unification. (Resolve) and (Factor) are applied to atomic literals only and as unifiers can replace atomic literals by atomic literals, we can also apply them before. □

Definition 15 (Normalized sets of unification constraints)**.** *Let* φ *be a derivation in CRC of the clause C and let* ψ *be the derivation of the clause* C' *according to Lemma 14, then the normalized set of unification constraints of C is the set of constraints of* C'.

Definition 16 (CRC with dynamic exponents)**.** *The CRC with dynamic exponents is CRC with the algorithm RBUA instead of BUA where for every derivation* ψ *of C, we update the initial values of* de *of all clauses in* ψ *to the exponent of periodicity of the the normalized set of unification constraints of C.*

Theorem 17. *If a clause set* Γ *is refutable, then there is a derivation of the empty clause from* Γ *using CRC with dynamic exponents.*

Proof. By induction on the derivation ψ obtained by CRC and using Lemma 14 we can obtain for each derived clause, a minimal unifier which unifies a set of constraints which, for every unification constraint generated in ψ contains a more general constraint. The exponents generated in such ways are enough to unify all unification constraints in ψ and therefore, we can obtain exactly the same derivation ψ using the dynamic exponents. □

5 An Example

In this section we will give a refutation of correctness of the Gauss sum formula using the second-order induction axiom. Despite the fact that the refutation does not contain cyclic unification constraints and therefore also no regular variables, the example can demonstrate the application of the unification algorithm eagerly by the use of the pre-computed bounds.

The set of clauses we would like to refute is:

$[X_0(0)]^F \vee [X_0(f(\neg X_0([.])))]^T \vee [X_0(y_0)]^T$	(I_1)	induction axiom
$[X_0(0)]^F \vee [X_0(f(\neg X_0([.]))+1)]^F \vee [X_0(y_0)]^T$	(I_2)	induction axiom
$[2 \cdot \Sigma(s) = s \cdot (s+1)]^F$	(T_1)	negation of the Gauss sum
$[\Sigma(x_0+1) = \Sigma(x_0) + (x_0+1)]^T$	(S_1)	summation definition
$[\Sigma(0) = 0]^T$	(S_2)	summation definition

We also use the following Peano arithmetic axioms:

$[x \cdot 0 = 0]^T$	(PAX_1)
$[x \cdot (y+z) = x \cdot y + x \cdot z]^T$	(PAX_2)
$[x \cdot (x+1) + 2 \cdot (x+1) = (x+1) \cdot ((x+1)+1)]^T$	(PAX_3)

In the refutation we label all context variables by a tuple $Y^{m,n}$ where $m = \lambda\texttt{-bound}(Y)$ and $n = \texttt{imt-bound}(Y)$. We assume $\lambda\texttt{-bound}(X_0) = 3$, i.e. that we look for contexts with at most 3 holes. In Fig. 6, we resolve each of the two induction clauses on the literal $[X_0(0)]^F$ with a clause obtained from (PAX_1) and (S_2) getting the substitution $\{X_0 \mapsto x_1 \cdot \Sigma([.]) = [.] \cdot X_5[.]\}$.

Let Q be defined as $\neg(x_1 \cdot \Sigma([.]) = [.] \cdot X_5([.]))$, then the resulting clauses (I'_1) and (I'_2) are:

$[x_1 \cdot \Sigma(f(Q)) = f(Q) \cdot X_1(f(Q))]^T \vee [x_1 \cdot \Sigma(y_0) = y_0 \cdot X_1(y_0)]^T$	(I'_1)
$[x_1 \cdot \Sigma(f(Q)+1) = (f(Q)+1) \cdot X_1(f(Q)+1)]^F \vee [x_1 \cdot \Sigma(y_0) = y_0 \cdot X_1(y_0)]^T$	(I'_2)

In Fig. 6, the meta-variable C denotes the remaining parts of (I_1) and (I_2), while the meta-variables C_i denotes C after the application of the respective substitutions. The meta-variable C_6 denotes (I'_1) and (I'_2).

In Fig. 7, we resolve the two clauses (I'_1) and (I'_2) with (T_1) using again the meta-variable C in order to get the substitution $\{X_5 \mapsto [.]+1, x_1 \mapsto 2\}$.

Let P be defined as $\neg(2 \cdot \Sigma([.]) = [.] \cdot ([.]+1))$, then the resulting clauses are:

$[2 \cdot \Sigma(f(P)) = f(P) \cdot (f(P)+1)]^T$	(I''_1)
$[2 \cdot \Sigma(f(P)+1) = (f(P)+1) \cdot ((f(P)+1)+1)]^F$	(I''_2)

The rest of the refutation can be seen in Fig. 8 where all unification constraints are eliminated using first-order only steps.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(PAX_1) \qquad (S_2)}{[x_1 \cdot \Sigma(0) = 0]^T}\,\texttt{(Para)} \qquad (PAX_1)}{[x_1 \cdot \Sigma(0) = 0 \cdot x_2]^T}\,\texttt{(Para)} \qquad [X_0(0)]^F \vee C}{[x_1 \cdot \Sigma(0) = 0 \cdot x_2 \doteq X_0^{3,12}(0)]^F \vee C}\,\texttt{(Resolve)}}{[x_1 \cdot \Sigma(0) \doteq X_1^{1,11}(0)]^F, [0 \cdot x_2 \doteq X_2^{2,11}(0)]^F \vee C_1}\,\texttt{(Imitate)}, \texttt{(Bind}_X\texttt{)}, \texttt{(Decomp)} X_0 \mapsto X_1([.]) = X_2([.])}{[y_1 \doteq x_1]^F, [\Sigma(0) \doteq X_3^{1,10}(0)]^F, [0 \cdot x_2 \doteq X_2^{2,11}(0)]^F \vee C_2}\,\texttt{(Imitate)}, \texttt{(Bind}_X\texttt{)}, \texttt{(Decomp)} X_1 \mapsto y_1 \cdot X_3([.])}{[y_1 \doteq x_1]^F, [0 \cdot x_2 \doteq X_2^{2,11}(0)]^F \vee C_3}\,\texttt{(Imitate)}, \texttt{(Bind}_X\texttt{)}, \texttt{(Delete)} X_3 \mapsto \Sigma([.])}{[0 \cdot x_2 \doteq X_2^{2,11}(0)]^F \vee C_4}\,\texttt{(Bind}_x\texttt{)}}{[0 \doteq X_4^{1,10}(0)]^F, [x_2 \doteq X_5^{1,10}(0)]^F \vee C_5}\,\texttt{(Imitate)}, \texttt{(Bind}_X\texttt{)}, \texttt{(Decomp)} X_2 \mapsto X_4([.]) \cdot X_5([.])}{[x_2 \doteq X_5^{1,10}(0)]^F \vee C_6}\,\texttt{(Project)}, \texttt{(Bind}_X\texttt{)}, \texttt{(Delete)} X_4 \mapsto [.]$$

Figure 6: Obtaining (I_1') and (I_2')

$$\dfrac{\dfrac{\dfrac{\dfrac{[x_1 \cdot \Sigma(y_0) = y_0 \cdot X_5(y_0)]^T \vee C \qquad (T_1)}{[x_1 \cdot \Sigma(y_0) = y_0 \cdot X_5^{1,12}(y_0) \doteq 2 \cdot \Sigma(s) = s \cdot (s+1)]^F \vee C}\,\texttt{(Resolve)}}{[x_1 \doteq 2]^F \vee [y_0 \doteq s]^F \vee [X_5^{1,12}(y_0) \doteq (s+1)]^F \vee C}\,4 \times \texttt{(Decomp)}, \texttt{(Factor)}}{[X_6^{1,11}(s) \doteq s]^F \vee [x_3 \doteq 1]^F \vee C_1}\,2 \times \texttt{(Bind}_x\texttt{)}, \texttt{(Imitate)}, \texttt{(Bind}_X\texttt{)}, \texttt{(Decomp)} X_5 \mapsto X_6([.]) + x_3}{C_2}\,\texttt{(Project)}, \texttt{(Bind}_X\texttt{)}, \texttt{(Bind}_x\texttt{)} X_6 \mapsto [.]$$

Figure 7: Obtaining (I_1'') and (I_2'')

$$\dfrac{\dfrac{\dfrac{\dfrac{(I_2'') \qquad (S_1)}{[2 \cdot (\Sigma(f(P)) + (f(P)+1)) = (f(P)+1) \cdot ((f(P)+1)+1)]^F}\,\texttt{(Para)} \qquad (PAX_2)}{[2 \cdot \Sigma(f(P)) + 2 \cdot (f(P)+1)) = (f(P)+1) \cdot ((f(P)+1)+1)]^F}\,\texttt{(Para)} \qquad (I_1'')}{[f(P) \cdot (f(P)+1) + 2 \cdot (f(P)+1)) = (f(P)+1) \cdot ((f(P)+1)+1)]^F}\,\texttt{(Para)} \qquad (PAX_3)}{}\,\texttt{(Resolve)}$$

Figure 8: Obtaining the empty clause

References

[1] Christoph Benzmüller. Extensional higher-order paramodulation and rue-resolution. In *CADE*, pages 399–413, 1999.

[2] Benzmüller C. Comparing approaches to resolution based higher-order theorem proving. *Synthese*, 133(1-2):203–335, 2002.

[3] Gérard P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic*. PhD thesis, Case Western Reserve University, 1972.

[4] Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.

[5] Tomer Libal. Solving context related unification problems using regular languages. Technical report, Institute of Computer Languages, Vienna University of Technology, 2012. Submitted.

[6] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[7] Manfred Schmidt-Schauß. Decidability of bounded second order unification. *Inf. Comput.*, 188:143–178, January 2004.

[8] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equation. In *RTA*, pages 61–75, 1998.

[9] Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher-order unification. *J. Symb. Comput.*, 40(2):905–954, August 2005.

[10] Wayne Snyder and Jean H. Gallier. Higher-order unification revisited: Complete sets of transformations. *J. Symb. Comput.*, 8(1/2):101–140, 1989.