EasyChair Preprint

№ 15904

Unification as a Simple Theorem Prover

Murat Sinan Aygün

March 11, 2025

Unification As A Simple Theorem Prover

Murat Sinan Aygün[1]

sinan_aygun@yahoo.com

Abstract. In this paper, unification is considered as a simple theorem prover in which variables to be solved are directly represented by free logic variables. This approach is different from other works which consider unification as the application of a set of rewrite rules. The benefit of the approach this paper considers is naturalness, clearness and uniform framework. The theorem prover is not in full setting. Since programming formulas are fixed and simple a few rules, goal formulas are matched with the left hand-side of rewrite rules. The theorem prover acts like a term rewriting system. But when free logic variables are at the top, they are subject to search. This paper considers the solution of this technical problem.

Keywords: Unification · Theorem prover · Term rewriting · The most general substitution. · Nondeterminism.

1 Introduction

Unification plays a central role in automated theorem proving [1, 2]. In unification, a set of rewrite rules are applied to a unification problem repeatedly until a trivial solution is reached. Solving unification problem is a term rewriting. Variables to be solved are defined in a fixed domain and rewrite rules are defined based on this domain accordingly. Unification is used as a tool theorem prover applies in its inference system to prove theorems. Instead, it is interesting to specify unification as a theorem to be proved not an auxiliary mechanism. Theorem provers are developed so and they are so important, why should they be incapable of doing this task using their logic? So, specifying unification as a theorem proving should be as important as they are.

As a result, unification is given as a query to be proved where variables to be solved are defined as free logic variables and they are not bound by a domain as in term rewriting system. They can take whatever value to take as long as it satisfies the goal.

Definition 1. *The recursive relation* $\Leftrightarrow$ *is defined as follows*

1. $a \Leftrightarrow a$
2. *if* $f(x) \Leftrightarrow f(y)$ $x \Leftrightarrow y$
3. *if* $g(x_1, x_2) \Leftrightarrow g(x_3, x_4)$ $x_1 \Leftrightarrow x_3$ *and* $x_2 \Leftrightarrow x_4$

Definition 1 is given a simple theorem prover. The rules are so simple that goal does not need to be unified with the left hand-side of a rule during inference. Goal matching with the left hand-side of a rule suffices.

Example 1. The rules of Definition 1 are applied to (1)

$$g(g(a,a),f(a)) \Leftrightarrow g(g(a,a),f(a)) \tag{1}$$

- $g(g(a,a),f(a)) \Leftrightarrow g(g(a,a),f(a))$ (using rule 3)
- $g(a,a) \Leftrightarrow g(a,a)$ and $f(a) \Leftrightarrow f(a)$ (using rule 2)
- $g(a,a) \Leftrightarrow g(a,a)$ and $a \Leftrightarrow a$ (using rule 1)
- $g(a,a) \Leftrightarrow g(a,a)$ (using rule 3)
- $a \Leftrightarrow a$ and $a \Leftrightarrow a$ (using rule 1)
- $a \Leftrightarrow a$ (using rule 1)
-

Example 2. Some terms may be variables as in (2)

$$g(x,f(a)) \Leftrightarrow g(f(y),f(a)) \tag{2}$$

- $g(x,f(a)) \Leftrightarrow g(f(y),f(a))$(using rule 3)
- $x \Leftrightarrow f(y)$ and $f(a) \Leftrightarrow f(a)$ (using rule 2)
- $x \Leftrightarrow f(y)$ and $a \Leftrightarrow a$ (using rule 1)
- $x \Leftrightarrow f(y)$ (if x is substituted by $f(x_1)$ and using rule 2)
- $x_1 \Leftrightarrow y$ (if x_1 is substituted by $f(x_2)$ and y by $f(y_1)$, then using rule 2)
- $x_2 \Leftrightarrow y_1$

In Example 2, the free variables take infinite values, each of which is valid and makes the goal provable. The aim of this work is to rule out these infinite values and compute the most general substitution using a technique. Terms are defined in a special format and should be linear.

2 Marking Solvable Variables

Definition 2. *Given that the symbols a, ν and f are respectively 0-ary, 1-ary and n-ary functions, a well defined form is either a or $\nu(x)$. $f(t_1,...,t_n)$ is a well defined form if each t_i is a well defined form.*

Example 3. $g(\nu(x),f(a))$ and $g(f(\nu(y)),f(a))$ are well defined forms. On the other hand, $g(\nu(f(a)),f(a))$ and $g(f(\nu(y)),x)$ are not.

Definition 3. *Let $\forall x$ $equal(x,x)$ be a binary predicate and used to perform variable substitution.*

Definition 4. *Given that the symbols a, ν and f are respectively 0-ary, 1-ary and n-ary functions, unification is defined as follows*

1. $a \Leftrightarrow a$
2. *if* $f(x_1,..,x_n) \Leftrightarrow f(y_1,..,y_n)$ $x_1 \Leftrightarrow y_1$ *and ...* $x_n \Leftrightarrow y_n$
3. *if* $\nu(x) \Leftrightarrow \nu(y)$ $equal(x,y)$
4. *if* $\nu(x) \Leftrightarrow y$ $equal(x,y)$ *if y is a function a or f*
5. *if* $y \Leftrightarrow \nu(x)$ $equal(x,y)$ *if y is a function a or f*

Example 4.

$$g(\nu(x), f(a)) \Leftrightarrow g(f(\nu(y)), f(a)) \tag{3}$$

- $g(\nu(x), f(a)) \Leftrightarrow g(f(\nu(y)), f(a))$(using rule 2)
- $\nu(x) \Leftrightarrow f(\nu(y))$ and $f(a) \Leftrightarrow f(a)$ (using rule 2)
- $\nu(x) \Leftrightarrow f(\nu(y))$ and $a \Leftrightarrow a$ (using rule 1)
- $\nu(x) \Leftrightarrow f(\nu(y))$ (using rule 4)
- $equal(x, f(\nu(y)))$ (x is substituted by $f(\nu(y))$)
-

Each variable of the terms to be unified must be different. This restriction guarantees that when performing $equal(x, y)$ in unification process, at least one argument x or y is a variable. Otherwise equal predicate should also perform unification not just variable substitution. This contradicts the aim of this paper that considers unification as a theorem to be proved not an auxiliary mechanism we apply during inference. At meta-level only variable substitution is performed.

Example 5. The following is a prolog program

```
1. equal(X,X).
2. eq(a,a).
3. eq(b,b).
4. eq(v(X),v(Y)) :- equal(X,Y).
5. eq(v(X),Y) :- mu(Y), equal(X,Y).
6. eq(Y,v(X)) :- mu(Y), equal(X,Y).
7. eq(f(X),f(Y)) :- eq(X,Y).
8. eq(h(X),h(Y)) :- eq(X,Y).
9. eq(g(X1,X2),g(X3,X4)) :- eq(X1,X3),eq(X2,X4).
10. mu(a).
11. mu(b).
12. mu(f(_)).
13. mu(h(_)).
14. mu(g(_,_)).
```

when the goal (4) is submitted to prolog

$$eq(g(v(X), h(v(Z))), g(f(v(Y)), v(U))) \tag{4}$$

it returns the answer (5)

$$U = h(v(Z)), X = f(v(Y)) \tag{5}$$

On the other hand, this restriction can be solved when we make marking not in the original term but using its copy. We need to check free logic variable occurrences in goal formulas in order to rule out their infinite instantiations by using the copy of the original term. Suppose $z_1, .., z_n$ are free logic variables and $c_1, .., c_n$ are new arbitrary constants not available in the signature. Assume that φ is a one to one mapping form $\{z_1, .., z_n\}$ to $\{c_1, .., c_n\}$, written as $\{z_i \rightarrow c_i\}$ $(1 \leq i \leq n)$. For a given term s, t is the encoding of s if and only if t = s φ, in other

words, t is the resultant term after applying φ to s. We simultaneously apply the transformation rules to s and its encoding t at each step as $s, t \rightarrow ... \rightarrow s_n, t_n$ such that t_n is the encoding of s_n and at each step the mapping is preserved. We update Definition 1 according to this formulation.

Definition 5. *The recursive relation $\Leftrightarrow$ is defined as follows*

1. $a, a \Leftrightarrow a, a$
2. *if* $f(x), f(x') \Leftrightarrow f(y), f(y')$ $x, x' \Leftrightarrow y, y'$
3. *if* $g(x_1, x_2), g(x_1', x_2') \Leftrightarrow g(x_3, x_4), g(x_3', x_4')$
   $x_1, x_1' \Leftrightarrow x_3, x_3'$ *and* $x_2, x_2' \Leftrightarrow x_4, x_4'$

Example 6. Encoded and encoding terms are in the same goal as in (6) where $\varphi = \{x \rightarrow c_1, y \rightarrow c_2\}$

$$g(x, f(a)), g(c_1, f(a)) \Leftrightarrow g(f(y), f(a)), g(f(c_2), f(a)) \tag{6}$$

- $g(x, f(a)), g(c_1, f(a)) \Leftrightarrow g(f(y), f(a)), g(f(c_2), f(a))$(using rule 3)
- $x, c_1 \Leftrightarrow f(y), f(c_2)$ and $f(a), f(a) \Leftrightarrow f(a), f(a)$ (using rule 2)
- $x, c_1 \Leftrightarrow f(y), f(c_2)$ and $a, a \Leftrightarrow a, a$ (using rule 1)
- $x, c_1 \Leftrightarrow f(y), f(c_2)$

The goal $x, c_1 \Leftrightarrow f(y), f(c_2)$ fails because c_1 fails to match $f(x')$ in rule 2. There is no way the current goal to go using any of these three rules or there is no substitution which makes it provable using these three rules. The new arbitrary constants $c_1, c_2, ..., c_n$, not available in the current signature, fail to match any term and this property is used to stop the search machinery automatically applied to free variables when they are at the top in goal formulas. The search machinery is applied to a free variable together with its marking c_i and fails because c_i fails to match any term.

Definition 6. *Let φ be a one to one substitution (mapping) from a set of free logic variables to a set of new arbitrary constants (not in the signature). A term t is the encoding of a term s if and only if $t = s\ \varphi$. We may call t an encoding term and s an encoded term. We may also call the new arbitrary constants encoding constants.*

Example 7. Given the function symbols f and g, the new arbitrary constants c_1, c_2 and the free logic variables x and y, f(c_1,g(c_1,c_2)) is the encoding of f(x,g(x,y)) where $\varphi = \{x \rightarrow c_1, y \rightarrow c_2\}$

In the following, we use c to denote any encoding constant.

Definition 7. *We denote an ordered list by $[s_1, ..., s_n]$. The symbol [] is used to denote an empty list. We give the following operations on lists.*

- $s_1 :: [s_2, ..., s_n] = [s_1, ..., s_n]$
- $[s_1, ..., s_n]\ \partial\ [t_1, ..., t_n] = [s_1, ..., s_n, t_1, ..., t_n]$
- $L_1\ \partial\ (t :: L_2) = (L_1\ \partial\ L_2) \cup \{t\}$ *for any ordered lists L_1, L_2 possibly empty.*

Definition 8. *Let L denote any ordered list and a,f respectively denote 0-ary and n-ary functions.* $s_1, t_1 \Leftrightarrow s_2, t_2$ *is symmetric, in other words,* $s_1, t_1 \Leftrightarrow s_2, t_2$ *implies* $s_2, t_2 \Leftrightarrow s_1, t_1$*. Unification can be defined by the following rules.*

1. *if* $L \cup \{a, a \Leftrightarrow a, a\}$ L
2. *if* $L \cup \{f(x_1, .., x_n), f(x'_1, .., x'_n) \Leftrightarrow f(y_1, .., y_n), f(y'_1, .., y'_n)\}$ $L \; \partial \; [x_1, x'_1 \Leftrightarrow y_1, y'_1, ..., x_n, x'_n \Leftrightarrow y_n, y'_n]$
3. *if* $L \cup \{x, c \Leftrightarrow y, y'\}$ $equal(x, y)$ *and* $L \; \{c \rightarrow y'\}$

Example 8. The rules of Definition 8 are applied to (7) where
$\varphi = \{x \rightarrow c_1, y \rightarrow c_2, z \rightarrow c_3\}$

$$[g(g(x, x), f(a)), g(g(c_1, c_1), f(a)) \Leftrightarrow g(g(y, z), f(z)), g(g(c_2, c_3), f(c_3))] \quad (7)$$

- $[g(g(x, x), f(a)), g(g(c_1, c_1), f(a)) \Leftrightarrow g(g(y, z), f(z)), g(g(c_2, c_3), f(c_3))]$
- $[(g(x, x), g(c_1, c_1) \Leftrightarrow g(y, z), g(c_2, c_3)), (f(a), f(a) \Leftrightarrow f(z), f(c_3))]$
- $[(x, c_1 \Leftrightarrow y, c_2), (x, c_1 \Leftrightarrow z, c_3), (f(a), f(a) \Leftrightarrow f(z), f(c_3))]$
- $equal(x, y)$ *and* $[(x, c_1 \Leftrightarrow z, c_3), (f(a), f(a) \Leftrightarrow f(z), f(c_3))]$ $\{c_1 \rightarrow c_2\}$
- $[(y, c_2 \Leftrightarrow z, c_3), (f(a), f(a) \Leftrightarrow f(z), f(c_3))]$ (x is substituted by y)
- $equal(y, z)$ *and* $[f(a), f(a) \Leftrightarrow f(z), f(c_3)]\{c_3 \rightarrow c_2\}$
- $[f(a), f(a) \Leftrightarrow f(y), f(c_2)]$ (z is substituted by y)
- $[a, a \Leftrightarrow y, c_2]$
- $equal(a, y)$ *and* $[]\{c_2 \rightarrow a\}$
- [] (y is substituted by a)

Example 9. The following is a prolog program that is used for the unification given Definition 8.

```
1. m_a_r(X,[X|L],L).
2. m_a_r(X,[Y|K],[Y|L]) :- m_a_r(X,K,L).
3. append([],K,K).
4. append([X|XS],YS,[X|ZS]) :- append(XS,YS,ZS).
5. cp(I,O,I,O,_).
6. cp(X,X,_,_,U) :- m_a_r(X,U,_).
7. cp(a,a,_,_,_).
8. cp(f(X1),f(Y1),I,O,U) :- cp(X1,Y1,I,O,U).
9. cp(h(X1),h(Y1),I,O,U) :- cp(X1,Y1,I,O,U).
10. cp(g(X1,X2),g(Y1,Y2),I,O,U) :- cp(X1,Y1,I,O,U),cp(X2,Y2,I,O,U).
11. subst([],[],_,_,_).
12. subst([eqq(X1,X2,X3,X4)|L1], [eqq(X1,Y2,X3,Y4)|L2],I,O,U) :-
        cp(X2,Y2,I,O,U), cp(X4,Y4,I,O,U), subst(L1,L2,I,O,U).
13. equal(X,X).
14. unf([],_).
15. unf(L,U) :- m_a_r(eqq(a,a,a,a),L,L1),unf(L1,U).
16. unf(L,U) :- m_a_r(eqq(f(X1),f(X2),f(X3),f(X4)),L,L1),
    unf([eqq(X1,X2,X3,X4)|L1],U).
17. unf(L,U) :- m_a_r(eqq(h(X1),h(X2),h(X3),h(X4)),L,L1),
    unf([eqq(X1,X2,X3,X4)|L1],U).
```

```
18. unf(L,U) :-
    m_a_r(eqq(g(X11,X12),g(X21,X22),g(X31,X32),g(X41,X42)),L,L1),
    append([eqq(X11,X21,X31,X41),eqq(X12,X22,X32,X42)],L1,L2),unf(L2,U).
19. unf(L,U) :-
    (m_a_r(eqq(X,X1,Y,Y1),L,L1);m_a_r(eqq(Y,Y1,X,X1),L,L1)),
    equal(X,Y),m_a_r(X1,U,U1),subst(L1,L2,X1,Y1,U1),unf(L2,U).
```

When the goal

$$unf([eqq(g(g(X,X),f(a)),g(g(c_1,c_1),f(a)),\\ g(g(Y,Z),f(Z)),g(g(c_2,c_3),f(c_3)))],[c_1,c_2,c_3])$$

is submitted to prolog, it returns the following answer

$$\mathrm{X} = \mathrm{Y},\ \mathrm{Y} = \mathrm{Z},\ \mathrm{Z} = \mathrm{a}$$

3 Conclusion

Unification is considered as term rewriting in [1, 2] where variables are bound by a domain and repeated application of rewrite rules until a trivial solution is reached. On the other hand, this paper considers unification as a theorem to be proved where variables are free. Variables are subject to search and this leads to inefficiency. This problem is solved in this paper. The benefit of the approach we use gives us naturalness and clearness. Theorem provers use unification as auxiliary machinery when goal needs to be unified with the left-hand side of a rule. Unification, the most important part of a theorem prover should not be used as an external tool. Instead, it should be defined inside a theorem prover. In our strategy, we start with a simple theorem prover whose rules are so simple that it acts like a term writing system. Goal is matched (not unified) with the left-hand side of rules and this satisfies our needs. When free variables are at the top in goal formulas, by default, automatic search machinery is applied. The variables of the terms to be unified can take whatever values to take as long as this substitution makes them equal. So, by definition, infinitely many solutions may exist and each may make the goal provable. By using marking technique, we restrict the search space. We are only interested in most general one.

Acknowledgments. The author thanks to referees for helpful comments on previous drafts.

References

1. Chang, C., Char, R.-Lee, T.: Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York, San Francisco, London (1973)
2. Jouannaud, J.P., Kirchner, C.: Solving Equations in Abstract Algebras: a Rule-Based Survey of Unification. Computational Logic. Essay in honor of Alan Robinson. The MIT Press, pages 257-321, Cambridge, 1991
3. Easy Chair Publications, https://easychair.org/publications/preprint/l7Vp
4. Easy Chair Publications, https://easychair.org/publications/preprint/JCqv