A Certified Sound Algorithm for AC-unification

Gabriel Silva
Department of Computer Science (Universidade de Brasília)
Funded by a CAPES PhD scholarship
Advisor: Mauricio Ayala-Rincón, Co-Advisor: Maribel Fernández
https://gabriel951.github.io/

October 29, 2021

This work was done in collaboration with:

Figure 1: Mauricio Ayala-Rincón

Figure 2: Maribel Fernández

Figure 3: Daniele Nantes

Overview

Unification is about "finding a way" to make two terms equal:

- $f(a, X)$ and $f(Y, b)$ can be made equal by "sending" X to b and Y to a, as they both become $f(a, b)$.

Unification has a lot of applications: logic programming, theorem proving, type inference and so on.

We consider the problem of AC-unification, i.e., unification in the presence of associative-commutative function symbols.

For instance, if f is an AC function symbol, then:

$$f(a, f(b, c)) \approx f(c, f(a, b)).$$

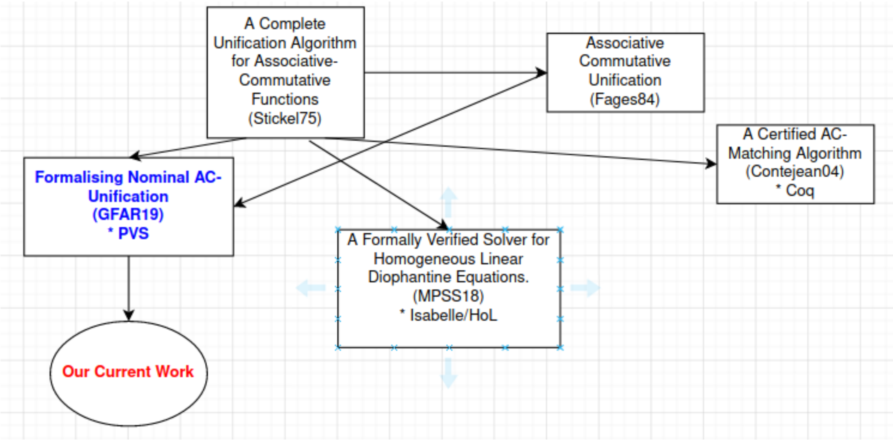

Figure 4: Main Related Work.

What We Currently Have

An AC-unification algorithm, which we have specified in PVS and formalised it to be terminating and sound.

The algorithm is recursive, calling itself on progressively simpler versions of the problem until it finishes.

In This Talk

- Briefly discuss the challenge in AC-unification.
- Present our approach to AC-unification (based on [1]).
- Motivate the lexicographic measure used in the proof of termination.
- Tell about the state of our PVS formalisation.

What is Tricky About AC? An Example

Let f be an AC function symbol.

The solutions that come to mind when unifying:

$$f(X, Y) \approx_? f(a, Z)$$

are: $\{X \to a, Y \to Z\}$ and $\{X \to Z, Y \to a\}$.

Are there other solutions?

Yes!

For instance, $\{X \to f(a, Z_1),\ Y \to Z_2,\ Z \to f(Z_1, Z_2)\}$ and $\{X \to Z_1,\ Y \to f(a, Z_2),\ Z \to f(Z_1, Z_2)\}$.

The AC-Step for AC-Unification

We explain via an example the `AC-Step` for AC-unification.

How do we generate a complete set of unifiers for:

$$f(X, X, Y, a, b, c) \approx_? f(b, b, b, c, Z).$$

1. Eliminate common arguments in the terms we are trying to unify.

Now we must unify $f(X, X, Y, a)$ with $f(b, b, Z)$.

2. According to the number of times each argument appear in the terms, transform the unification problem into a linear equation on $\mathbb{N}$.

After this step, our equation is:

$$2X_1 + X_2 + X_3 = 2Y_1 + Y_2,$$

where variable X_1 corresponds to argument X, variable X_2 corresponds to argument Y and so on.

3. Generate a basis of solutions to the linear equation.

Table 1: Solutions for the Equation $2X_1 + X_2 + X_3 = 2Y_1 + Y_2$

X_1	X_2	X_3	Y_1	Y_2	$2X_1 + X_2 + X_3$	$2Y_1 + Y_2$
0	0	1	0	1	1	1
0	1	0	0	1	1	1
0	0	2	1	0	2	2
0	1	1	1	0	2	2
0	2	0	1	0	2	2
1	0	0	0	2	2	2
1	0	0	1	0	2	2

4. Associate new variables with each solution.

Table 2: Solutions for the Equation $2X_1 + X_2 + X_3 = 2Y_1 + Y_2$

X_1	X_2	X_3	Y_1	Y_2	$2X_1 + X_2 + X_3$	$2Y_1 + Y_2$	**New Variables**
0	0	1	0	1	1	1	Z_1
0	1	0	0	1	1	1	Z_2
0	0	2	1	0	2	2	Z_3
0	1	1	1	0	2	2	Z_4
0	2	0	1	0	2	2	Z_5
1	0	0	0	2	2	2	Z_6
1	0	0	1	0	2	2	Z_7

5. Observing Table 2, relate the "old" variables and the "new" ones.

After this step, we obtain:

$$
\begin{aligned}
X_1 &\approx_? Z_6 + Z_7 \\
X_2 &\approx_? Z_2 + Z_4 + 2Z_5 \\
X_3 &\approx_? Z_1 + 2Z_3 + Z_4 \\
Y_1 &\approx_? Z_3 + Z_4 + Z_5 + Z_7 \\
Y_2 &\approx_? Z_1 + Z_2 + 2Z_6
\end{aligned}
$$

6. Decide whether we will include (set to 1) or not (set to 0) every "new" variable. Observe that every "old" variable must be different than zero.

In our example, we have $2^7 = 128$ possibilities of including/excluding the variables $Z_1, \ldots, Z_7$, but after observing that X_1, X_2, X_3, Y_1, Y_2 cannot be set to zero, we have 69 cases.

7. Drop the cases where the variables that in fact represent constants or subterms headed by a different AC function symbol are assigned to more than one of the "new" variables.

For instance, the potential new unification problem

$$\{X_1 \approx_? Z_6, X_2 \approx_? Z_4, X_3 \approx_? f(Z_1, Z_4), \\ Y_1 \approx_? Z_4, Y_2 \approx_? f(Z_1, Z_6, Z_6)\}$$

should be discarded as the variable X_3, which represents the constant a, cannot unify with $f(Z_1, Z_4)$.

8. Replace "old" variables by the original terms they substituted and proceed with the unification.

Some new unification problems may be unsolvable and **will be discarded later**. For instance:

$$\{X \approx_? Z_6, Y \approx_? Z_4, a \approx_? Z_4, b \approx_? Z_4,\ Z \approx_? f(Z_6, Z_6)\}$$

Solutions For The Example

In our example, the solutions will be:

$$\left\{ \begin{array}{c} \sigma_1 = \{Y \to f(b,b), Z \to f(a,X,X)\} \\ \sigma_2 = \{Y \to f(Z_2,b,b), Z \to f(a,Z_2,X,X)\} \\ \sigma_3 = \{X \to b, Z \to f(a,Y)\} \\ \sigma_4 = \{X \to f(Z_6,b), Z \to f(a,Y,Z_6,Z_6)\} \end{array} \right\}$$

The Structure of the Problem After AC-Step

Suppose that $P = \{t \approx^? s\}$, where t and s are AC-functions, headed by a symbol f. Let $t_1, \ldots, t_m$ be the different arguments of t and let $s_1, \ldots, s_n$ be the different arguments of s, after we eliminate common arguments.

An arbitrary unification problem P_1 after the AC-Step is of the form $P_1 = \{t_1 \approx^? t'_1, \ldots, t_m \approx^? t'_m, s_1 \approx^? s'_1, \ldots, s_n \approx^? s'_n\}$, where the terms in the right-hand side are either new variables Z_is or AC-functions headed by f whose arguments are all new variables Z_is.

Related Work

Our formalisation is based on the works of Stickel ([1]) and Fages ([2]).

1. Stickel, in 1975, presents the first AC-unification algorithm
2. Fages, in 1984, discovered an error in Stickel's proof of termination and presented a fix for it.

Let f be an AC-function symbol. Suppose we want to solve

$$P = \{f(X,Y) \approx^? f(U,V), X \approx^? Y, U \approx^? V\}$$

and we decide to solve the first equation. We obtain as one of the branches the unification problem

$$\begin{aligned}&\{X \approx^? f(X_1,X_2), Y \approx^? f(X_3,X_4),\\ &\ U \approx^? f(X_1,X_3), V \approx^? f(X_2,X_4), X \approx^? Y, U \approx^? V\}.\end{aligned}$$

We then instantiate the variables that we can, obtaining:

$$\{f(X_1,X_2) \approx^? f(X_3,X_4), f(X_1,X_3) \approx^? f(X_2,X_4)\}.$$

If we then solve the first equation, one of the branches get us:

$$P' = \{f(X_1,X_3) \approx^? f(X_2,X_4), X_1 \approx^? X_3, X_2 \approx^? X_4\}.$$

which is essentially the same unification problem we started with.

Avoiding Infinite Loops

How did we avoid looping forever?

Instantiate as early as possible, leave the AC-part last.

Termination - Notation

To explain the ideas to prove termination, we will consider the restricted case where $P = \{t \approx^? s\}$, and $t \equiv f(t_1, \ldots, t_m)$ and $s \equiv f(s_1, \ldots, s_n)$.

After we apply `AC-Step`, we will denote an arbitrary unification problem obtained as $P_1 = \{t_1 \approx^? t'_1, \ldots, t_m \approx^? t'_m, s_1 \approx^? s'_1, \ldots, s_n \approx^? s'_n\}$. We will denote by P_2 the unification problem obtained from P_1 after you do the necessary instantiations.

What we need for termination

All the terms in the right-hand side of P_1 are new terms. After introducing all these new terms and possible making some instantiations, can we still find a lexicographic measure lex such that $lex(P_2) < lex(P)$?

Termination - First Idea

Idea: Define a set of admissible subterms (AS) of a term in a way that every term t'_i in the right-hand side of P_1 has $AS(t'_i) = \emptyset$.

Admissible Subterms - Definition

We say that s is an admissible subterms of t if s is a proper subterm of t and s is not a variable.

The set of admissible subterms of a unification problem P is defined as:

$$AS(P) = \bigcup_{t \in P} AS(t).$$

Let $P = \{a \approx^? f(Z_1, Z_2), b \approx^? Z_3, g(h(c), Z) \approx^? Z_4\}$. Then $AS(P) = \{h(c), c\}$.

If at least one of the terms in the left-hand side of P_1 is not a variable, then $|AS(P_1)| < |AS(P)|$.

Example

In the previous example, the unification problem before the `AC-Step` was:

$$P = \{f(X, X, Y, a) \approx^? f(b, b, Z)\}\}$$

and we had $AS(P) = \{a, b\}$. After the `AC-Step`, one of the unification problems that is generated is:

$$P_1 = \{X \approx^? Z_6, Y \approx^? f(Z_5, Z_5), a \approx^? Z_1, b \approx^? Z_5, Z \approx^? f(Z_1, Z_6, Z_6)\},$$

where $AS(P_1) = \emptyset$.

When All The Arguments Are Variables

But what happens if all the arguments of t and s are variables?

Then, after the `AC-Step` we would instantiate all of them and the problem would be solved.

Can Instantiation Mess Up with $|AS|$?

All that is left in this simplified example where $P = \{t \approx^? s\}$ is to make sure that when we instantiate the variables in the unification problem P_1 and obtain as output a unification problem P_2 we maintain $|AS(P_2)| \leq |AS(P_1)|$.

Can we prove this?

Unfortunately not.

Example

Let f and g be AC-function symbols. If we instantiate the variables in

$$P_1 = \{X \approx^? f(Z_1, Z_2), g(X, W) \approx^? g(a, c)\}$$

we would obtain

$$P_2 = \{g(f(Z_1, Z_2), W) \approx^? g(a, c)\}.$$

In this case we have $AS(P_1) = \{a, c\}$ while $AS(P_2) = \{f(Z_1, Z_2), a, c\}$ and therefore $|AS(P_2)| > |AS(P_1)|$.

 Instantiation Does Not Always Mess Up With $|AS|$

If we changed the previous example to make it so that X only appears as argument of AC-functions headed by f, then instantiating X to an AC-function headed by f would not increase $|AS|$:

Example

If

$$P_1' = \{X \approx^? f(Z_1, Z_2), f(X, W) \approx^? g(a, c)\}$$

and we instantiate the variables we would obtain:

$$P_2' = \{f(Z_1, Z_2, W) \approx^? g(a, c)\},$$

where $AS(P_1') = AS(P_2') = \{a, c\}$.

Suppose that X is a variable in the left-hand side of P_1 and is instantiated to an AC-function headed by f. X would only contribute in increasing $|AS(P_2)|$ in relation to $|AS(P_1)|$ if X also occurred as an argument of a function term t^* headed by a different symbol than f.

Also, if X is in the left-hand side of P_1, then it is an argument of either t or s, both of which are functions headed by f.

Termination - Second Idea

Idea: X only contributes in increasing $|AS(P_2)|$ in relation to $|AS(P_1)|$ if X were “an argument to two different function symbols” in P. Since X was instantiated it is not “an argument to two different function symbols” in P_2.

To capture the idea of a variable being "an argument to two different function symbols" in P we define $V_{>1}(P)$.

Definition

We denote by $V_{>1}(P)$ the set of variables that are arguments of (at least) two terms t and s in $Subterms(P)$ such that t and s are headed by different function symbols.

$V_{>1}(P)$ - Example

Let f be an AC-function symbol and let g be a standard function symbol. Let

$$P = \{X \approx^? a, g(X) \approx^? h(Y), f(Y, W, h(Z)) \approx^? f(c, W)\}.$$

In this case $V_{>1}(P) = \{Y\}$.

$V_{>1}(P)$ in a Nutshell

In the cases where $|AS(P_2)|$ may be greater than $|AS(P_1)|$, we necessarily have $|V_{>1}(P_2)| < |V_{>1}(P)|$.

Lexicographic Measure for the non-associative part

In syntactic unification, given a unification problem P a usual lexicographic measure for termination $(|Vars(P)|, size(P))$.

We needed to change $Vars(P)$ to $V_{NAC}(P)$, the variables that occur in the problem P excluding those that only occur as arguments of AC-function symbols.

$V_{NAC}(P)$ - Example

Let f be an AC-function symbol and let g be a standard function symbol. Let

$$P = \{X \approx^? a, f(X, Y, W, g(Y)) \approx^? Z\}.$$

Then $V_{NAC}(P) = \{X, Y, Z\}$.

Motivation for changing $Vars(P)$ to $V_{NAC}(P)$

The AC-Step introduces new variables. By replacing $Vars(P)$ with $V_{NAC}(P)$, we exclude the new variables that only occur as arguments of AC-function symbols.

But in a problem like:

$$P_1 = \{X \approx^? Z_6, Y \approx^? f(Z_5, Z_5), a \approx^? Z_1, b \approx^? Z_5, Z \approx^? f(Z_1, Z_6, Z_6)\},$$

the new variable Z_1 does not occur only as an argument of AC-function symbols. Can variables like Z_1 potentially cause $|V_{NAC}(P_2)| > |V_{NAC}(P)|$?

Motivation for changing $Vars(P)$ to $V_{NAC}(P)$ (cont.)

No. Variables like Z_1 would be instantiated and we will always have $|V_{NAC}(P_2)| \leq |V_{NAC}(P)|$.

Lexicographic Measure

The lexicographic measure for termination is:

$$lex = (|V_{NAC}(P)|,\ |V_{>1}(P)|,\ |AS(P)|,\ size(P)),$$

We always have $lex(P_2) < lex(P)$.

When specifying the algorithm, we tried to follow closely the pseudocode of Fages. However, in Fages work, there are two functions:

1. `uniAC` - used to unify terms t and s
2. `unicompound` - used to unify a list of terms $(t_1, \ldots, t_n)$ with $(s_1, \ldots, s_n)$

which are mutually recursive, something not allowed in PVS.

Our Approach

We adapted the algorithm to use only one function, which works in a unification problem P and operates (with the exception of the AC-part of the algorithm) by simplifying one of the equations $\{t \approx^? s\}$ of P.

Why is this a big deal?

The lexicographic measure we use would not diminish if in the AC-part of the algorithm we simplified only one equation $\{t \approx^? s\}$ of P.

Choose an equation $t \approx^? s \in P$ that we will simplify. Heuristic: leave AC-equations last.

If $t \approx^? s$ is not an AC-equation, proceed as in syntactic unification.

If all that remains are AC-equations, pick the first AC-equation, apply `AC-Step` and instantiate the variables. Go to the second AC-equation, apply `AC-Step` and instantiate the variables. Proceed in this way until the last one.

The State of our Formalisation

1. Termination - Formalised (hard).
2. Soundness - Formalised (easy).
3. Completeness - Working on.

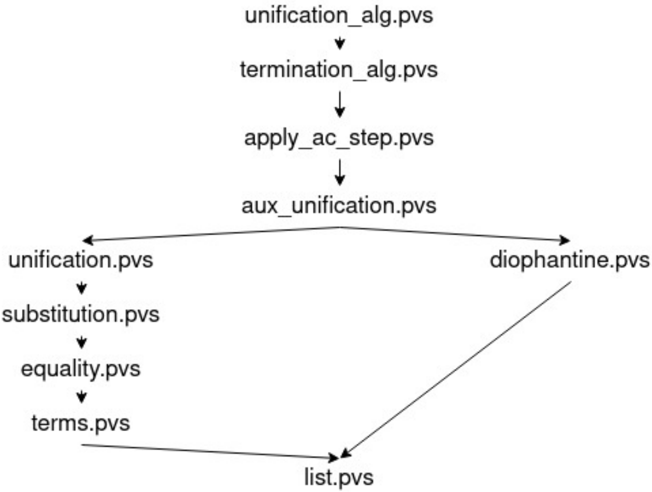

Figure 5: PVS Files Dependency Diagram

Amount of Theorems and TCCs Proved

Table 3: Number of theorems and TCCs in each file.

File	Theorems	TCCs	Total
unification_alg.pvs	6	8	14
termination_alg.pvs	84	35	119
apply_ac_step.pvs	16	10	26
aux_unification.pvs	134	44	178
diophantine.pvs	20	25	45
unification.pvs	43	11	54
substitution.pvs	76	16	92
equality.pvs	36	12	48
terms.pvs	115	44	159
list.pvs	161	84	245
Total	691	289	**980**

Size By File

Table 4: Size of .pvs and .prf files

File	.pvs	.prf	Percentage
`unification_alg`	4,4 kB	3,3 MB	12,5 %
`termination_alg`	22 kB	11,4 MB	43,2 %
`apply_ac_step`	8,7 kB	2,8 MB	10,6 %
`aux_unification`	40,5 kB	5,1 MB	19,3 %
`diophantine`	9,7 kB	193,1 kB	0,7 %
`unification`	13,5 kB	234,7 kB	0,9 %
`substitution`	14,7 kB	1,1 MB	4,2 %
`equality`	8,5 kB	901,0 kB	3,4 %
`terms`	23,8 kB	642,7 kB	2,4 %
`list`	37,5 kB	748,6 kB	2,8 %
Total	183,3 kB	26,4 MB	**100%**

A Comparison With Another Formalisation

In our paper "A Certified Functional Nominal C-unification Algorithm", there were 168 lemmas. In our current formalisation there are 691.

Conclusion

- We specified Stickel's AC-unification algorithm in the proof assistant PVS and proved it sound and terminating.
- We discussed how to solve equations of the form $t \approx^? s$ when t and s are AC-functions headed by the same symbol and the connection between this problem and solving Diophantine linear equations.
- We provided a motivation for the lexicographic measure used to prove termination of the algorithm.

Future Work

1. The immediate future work is proving completeness of this algorithm.
2. After that, a possible future work is to extend the algorithm to the nominal setting, which would give the first nominal AC-unification algorithm.

Thank You

Thank you! Any comments/doubts?

Bibliography

[1] M. E. Stickel, "A unification algorithm for associative-commutative functions," *Journal of the ACM (JACM)*, vol. 28, no. 3, pp. 423–434, 1981.

[2] F. Fages, "Associative-commutative unification," *Journal of Symbolic Computation*, vol. 3, no. 3, pp. 257–275, 1987.

The algorithm is recursive and keeps track of the current unification problem P, the substitution σ computed so far and the variables V that are/were in the problem. The output of the algorithm is a list of substitutions, where each substitution δ in this list is an AC-unifier of P.

Initial Call

The first call to the algorithm, in order to unify terms t and s is done with $P = cons((t, s), nil)$, $\sigma = nil$ and $V = Vars((t, s))$.

Pseudocode for the Algorithm

procedure ACUnif(P, σ, V)
  if nil?(P) **then**
    return $cons(\sigma, \texttt{nil})$
  else
    $((t, s), P_1) = \text{choose}(P)$
    if $(s$ matches $X)$ and $(X$ not in $t)$ **then**
      $\sigma_1 = \{X \to t\}$
      $\sigma' = \text{append}(\sigma_1, \sigma)$
      $P' = \sigma_1 P_1$
      return $\texttt{ACUnif}(P', \sigma', V)$
    else
      if t matches a **then**
        if s matches a **then**
          return $\texttt{ACUnif}(P_1, \sigma, V)$
        else
          return $\texttt{nil}$
        end if

Pseudocode for the Algorithm (cont.)

else if t matches X **then**
    if X not in s **then**
        $\sigma_1 = \{X \to s\}$
        $\sigma' = \texttt{append}(\sigma_1, \sigma)$
        $P' = \sigma_1 P_1$
        return $\texttt{ACUnif}(P', \sigma', V)$
    else if s matches X **then**
        return $\texttt{ACUnif}(P_1, \sigma, V)$
    else
        return `nil`
    end if
else if t matches $\langle\rangle$ **then**
    if s matches $\langle\rangle$ **then**
        return $\texttt{ACUnif}(P_1, \sigma, V)$
    else
        return `nil`
    end if

Pseudocode for the Algorithm (cont.)

```
else if t matches ⟨t_1, t_2⟩ then
    if s matches ⟨s_1, s_2⟩ then
        P' = cons((t_1, s_1), cons((t_2, s_2), P_1))
        return ACUnif(P', σ, V)
    else
        return nil
    end if
else if t matches f t_1 then
    if s matches f s_1 then
        P' = cons((t_1, s_1), P_1)
        return ACUnif(P', σ, V)
    else
        return nil
    end if
```

Pseudocode for the Algorithm (cont.)

            else
                if s matches f^{AC} s_1 **then**
                    $P' = \texttt{simplify}\ (P)$
                    $\mathit{InputLst} = \texttt{applyACStep}(P', \sigma, V)$
                    $\mathit{LstResults} = \texttt{map}(\texttt{ACUnif}, \mathit{InputLst})$
                    return $\texttt{flatten}\ (\mathit{LstResults})$
                else
                    return `nil`
                end if
            end if
        end if
    end if
end procedure