A Certified Algorithm for AC-Unification

Mauricio Ayala-Rincón (Universidade de Brasília)
Maribel Fernández (King's College London)
Gabriel Ferreira Silva (Universidade de Brasília)
Daniele Nantes-Sobrinho (Universidade de Brasília and Imperial College London)
https://github.com/gabriel951/first_order_ac_unification

August 2, 2022

Authors

Figure: Mauricio Ayala-Rincón

Figure: Maribel Fernández

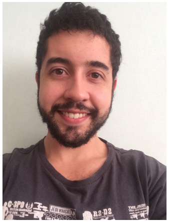


Figure: Gabriel Ferreira Silva

Figure: Daniele Nantes

Outline

Unification

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.

Unification Modulo AC

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

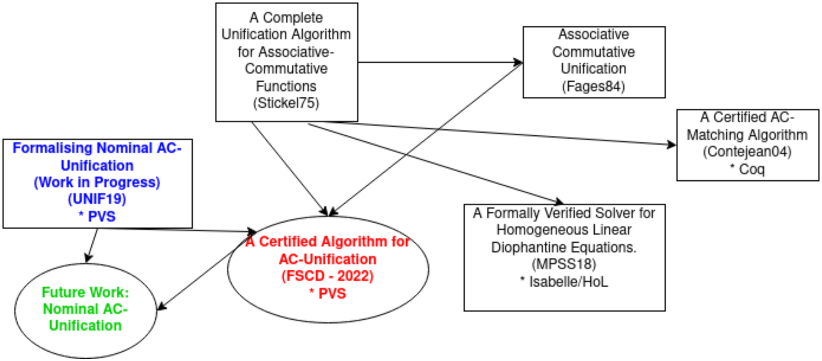
For instance, if f is an AC function symbol, then:

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

Our Work in a Nutshell

We modified Stickel's seminal AC-unification algorithm to avoid mutual recursion and formalised it in the PVS proof assistant. We proved the adjusted algorithm's termination, soundness and completeness.

Main Related Work


A Complete Unification Algorithm for Associative-Commutative Functions (Stickel75)
Associative Commutative Unification (Fages84)
A Certified AC-Matching Algorithm (Contejean04) * Coq
Formalising Nominal AC-Unification (Work in Progress) (UNIF19) * PVS
A Certified Algorithm for AC-Unification (FSCD - 2022) * PVS
A Formally Verified Solver for Homogeneous Linear Diophantine Equations. (MPSS18) * Isabelle/HoL
Future Work: Nominal AC-Unification

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

Are there other solutions?

What is Tricky About AC? An Example

Yes!

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

Example of AC-Unification

How do we generate a complete set of unifiers for

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

Eliminate Common Arguments

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

The problem remains:

$$f(a, X) \approx^? f(b, Y).$$

The Connection with Linear Equations

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:

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

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

Basis of Solutions and New Variables

Generate a basis of solutions to the linear equation and associate a new variable with each solution.

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

X_1	X_2	Y_1	Y_2	$X_1 + X_2$	$Y_1 + Y_2$	**New Variables**
0	1	0	1	1	1	Z_1
0	1	1	0	1	1	Z_2
1	0	0	1	1	1	Z_3
1	0	1	0	1	1	Z_4

Relating Old and New Variables

Observing Table 1, relate the "old" variables and the "new" ones.

After this step, we obtain:

$$
\begin{aligned}
X_1 &\approx^? Z_3 + Z_4 \\
X_2 &\approx^? Z_1 + Z_2 \\
Y_1 &\approx^? Z_2 + Z_4 \\
Y_2 &\approx^? Z_1 + Z_3
\end{aligned}
$$

All Possible Cases

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 $2^4 = 16$ possibilities of including/excluding the variables $Z_1, \dots, Z_4$, but after observing that X_1, X_2, Y_1, Y_2 cannot be set to zero, we have 7 branches.

Eliminating Some Cases

The seven branches:

$\{X_1 \approx^? Z_4, X_2 \approx^? Z_1, Y_1 \approx^? Z_4, Y_2 \approx^? Z_1\}$

$\{X_1 \approx^? Z_3, X_2 \approx^? Z_2, Y_1 \approx^? Z_2, Y_2 \approx^? Z_3\}$

$\{X_1 \approx^? f(Z_3, Z_4), X_2 \approx^? Z_2, Y_1 \approx^? f(Z_2, Z_4), Y_2 \approx^? Z_3\}$

$\{X_1 \approx^? f(Z_3, Z_4), X_2 \approx^? Z_1, Y_1 \approx^? Z_4, Y_2 \approx^? f(Z_1, Z_3)\}$

$\{X_1 \approx^? Z_4, X_2 \approx^? f(Z_1, Z_2), Y_1 \approx^? f(Z_2, Z_4), Y_2 \approx^? Z_1\}$

$\{X_1 \approx^? Z_3, X_2 \approx^? f(Z_1, Z_2), Y_1 \approx^? Z_2, Y_2 \approx^? f(Z_1, Z_3)\}$

$\{X_1 \approx^? f(Z_3, Z_4), X_2 \approx^? f(Z_1, Z_2), Y_1 \approx^? f(Z_2, Z_4), Y_2 \approx^? f(Z_1, Z_3)\}$

Dropping More Impossible Cases I

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:

$$\{X_1 \approx^? f(Z_3, Z_4), X_2 \approx^? f(Z_1, Z_2), Y_1 \approx^? f(Z_2, Z_4), Y_2 \approx^? f(Z_1, Z_3)\}$$

should be discarded as the variable X_1, which represents the constant a, cannot unify with $f(Z_3, Z_4)$.

Dropping More Impossible Cases II

Three branches remain:

$$\{X_1 \approx^? Z_4, X_2 \approx^? Z_1, Y_1 \approx^? Z_4, Y_2 \approx^? Z_1\}$$
$$\{X_1 \approx^? Z_3, X_2 \approx^? Z_2, Y_1 \approx^? Z_2, Y_2 \approx^? Z_3\}$$
$$\{X_1 \approx^? Z_3, X_2 \approx^? f(Z_1, Z_2), Y_1 \approx^? Z_2, Y_2 \approx^? f(Z_1, Z_3)\}$$

Replacing Variables And Proceeding

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

Replacing Variables And Proceeding

The three branches become:

$$\{a \approx^? Z_4, X \approx^? Z_1, b \approx^? Z_4, Y \approx^? Z_1\}$$
$$\{a \approx^? Z_3, X \approx^? Z_2, b \approx^? Z_2, Y \approx^? Z_3\}$$
$$\{a \approx^? Z_3, X \approx^? f(Z_1, Z_2), b \approx^? Z_2, Y \approx^? f(Z_1, Z_3)\}$$

Solutions For The Example

The solutions will be:

$$\left\{ \begin{array}{c} \sigma_1 = \{Z_3 \rightarrow a, X \rightarrow b, Y \rightarrow a\}, \\ \sigma_2 = \{Z_3 \rightarrow a, X \rightarrow f(b, Z_1), Y \rightarrow f(a, Z_1)\} \end{array} \right\}$$

which, since Z_3 is not part of the original problem, can be simplified to:

$$\left\{ \begin{array}{c} \sigma_1 = \{X \rightarrow b, Y \rightarrow a\}, \\ \sigma_2 = \{X \rightarrow f(b, Z_1), Y \rightarrow f(a, Z_1)\} \end{array} \right\}$$

Input and Output of Algorithm

The algorithm `ACUnif` is recursive and receives as input a triple (P, σ, V), where P is the current unification problem, σ is the substitution computed so far and V are the variables that are/were in the problem.

To unify two terms t and s the initial call is with $P = \{t \approx^? s\}$, $\sigma = Id$ and $V = Vars(t, s)$.

The output is a list of substitutions, where each substitution δ in this list is an AC-unifier of P.

Mutual Recursion

When specifying Stickel's algorithm we tried to follow closely the pseudocode from Fages' paper (see [Fag84]) [1].

In [Fag84] there are two functions:

- `uniAC` - unifies two terms t and s
- `unicompound` - unifies a list of terms $(t_1, \ldots, t_n)$ with a list of terms $(s_1, \ldots, s_n)$.

Those function are mutually recursive, something not allowed in PVS. We have adapted the algorithm to use only one main function.

[1] Stickel's paper [Sti75] and [Sti81] give a higher-level description

Termination

Almost a decade after Stickel's paper on AC-unification, Fages discovered and fixed a flaw in the proof of termination. Our proof of termination is based on Fages' proof.

Nice Inputs

While the recursive calls of `ACUnif` may change P, σ and V, some nice relations between them are preserved. Those are captured in the definition of **nice input**:

Definition 1

We say that (P, σ, V) is a nice input if:

- σ is idempotent.
- $Vars(P) \cap dom(\sigma) = \emptyset$.
- $Vars(P) \cup dom(\sigma) \subseteq V$

Soundness

We cannot prove Corollary 3 directly by induction, since the parameters of `ACUnif` may change between recursive calls. We prove Theorem 2 by induction and then use it to prove Corollary 3.

Theorem 2 (Soundness for nice inputs)

Let (P, σ, V) be a nice input, and $\delta \in \mathtt{ACUnif}(P, \sigma, V)$. Then, δ unifies P.

Corollary 3 (Soundness of `ACUnif`)

If $\delta \in \mathtt{ACUnif}(\{t \approx^? s\}, Id, Vars(t, s))$ then δ unifies $t \approx^? s$.

Notation: $\delta \subseteq V$

Let V be a set of variables and δ a substitution. We say that $\delta \subseteq V$ if $\mathit{dom}(\delta) \subseteq V$ and $\mathit{Vars}(\mathit{im}(\delta)) \subseteq V$.

Completeness

To prove Corollary 5 we first prove Theorem 4.

Theorem 4 (Completeness for nice inputs)

Let (P, σ, V) be a nice input, δ unifies P, $\sigma \leq \delta$, and $\delta \subseteq V$. Then, there is a substitution $\gamma \in \mathtt{ACUnif}(P, \sigma, V)$ such that $\gamma \leq_V \delta$.

Corollary 5 (Completeness of `ACUnif` with $\delta \subseteq V$)

Let V be a set of variables such that $\delta \subseteq V$ and $Vars(t, s) \subseteq V$. If δ unifies $t \approx^? s$, then there is a substitution $\gamma \in \mathtt{ACUnif}(\{t \approx^? s\}, Id, V)$ such that $\gamma \leq_V \delta$.

$\delta \subseteq V$

The hypothesis $\delta \subseteq V$ in the theorems of completeness guarantees that the new variables introduced by `ACUnif` do not clash with the variables in $\mathit{dom}(\delta)$ or the variables in the terms in $\mathit{im}(\delta)$.

New - Removing $\delta \subseteq V$

Recently, we were able to remove hypothesis $\delta \subseteq V$ and prove:

Theorem 6 (Completeness of `ACUnif`)

If δ unifies $t \approx^? s$, then there is a substitution $\gamma \in \mathit{ACUnif}(\{t \approx^? s\}, Id, Vars(t, s))$ such that $\gamma \leq_{Vars(t,s)} \delta$.

High-level View of How to Remove $\delta \subseteq V$ I

The key step to prove Theorem 6 is to prove that the substitutions computed when we call `ACUnif` with input (P, σ, V) “differ only by a renaming” from the substitutions computed when we call `ACUnif` with input (P, σ, V'), where $\delta \subseteq V'$.

High-level View of How to Remove $\delta \subseteq V$ II

This cannot be proven by induction directly because if V and V' differ and `ACUnif` enters the AC-part, the new variables introduced for each input may "differ only by a renaming", i.e. the first component of the two inputs, will also "differ only by a renaming".

High-level View of How to Remove $\delta \subseteq V$ III

Once `ACUnif` instantiates variables, it may happen that the substitutions computed so far, i.e. the second component of the two inputs, will also “differ only by a renaming”.

High-level View of How to Remove $\delta \subseteq V$ IV

The solution is to prove by induction the more general statement that if the inputs (P, σ, V) and (P', σ', V') “differ only by a renaming” then the substitutions computed when we call `ACUnif` with (P, σ, V) “differ only by a renaming” from the substitutions computed when we call `ACUnif` with (P', σ', V').

New - Removing $\delta \subseteq V$

The formalisation size increased by approximately 10% after we removed hypothesis $\delta \subseteq V$.

Amount of Theorems and TCCs Proved

Table: Number of theorems and TCCs in each file.

File	Theorems	TCCs	Total	New
unification_alg.pvs	10	19	29	2
termination_alg.pvs	80	35	115	0
rename_input.pvs	21	23	44	44
apply_ac_step.pvs	29	12	41	6
aux_unification.pvs	204	58	262	29
diophantine.pvs	73	44	117	0
unification.pvs	86	14	100	11
substitution.pvs	144	22	166	42
equality.pvs	67	18	85	0
terms.pvs	131	48	179	3
list.pvs	256	110	366	6
Total	1101	403	1504	143

Conclusion

In 2004, Evelyn Contejean formalised AC-matching, leaving as future work to formalise AC-unification. It's now 2022 and we have finished the first formalisation of AC-unification.

Future Work

- **Currently Working** - Extend the algorithm to the nominal setting, which let us treat free and bound variables in a natural manner.
- **Not Immediate Work** - Use our formalisation as a basis to formalise more efficient algorithms.

Thank You

Thank you! Any comments/suggestions/doubts?

References I

- François Fages, *Associative-Commutative Unification*, 7th International Conference on Automated Deduction, Napa, LNCS, vol. 170, Springer, 1984, pp. 194–208.
- Mark E. Stickel, *A Complete Unification Algorithm for Associative-Commutative Functions*, Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, 1975, pp. 71–76.
- Mark E. Stickel, *A Unification Algorithm for Associative-Commutative Functions*, J. of the ACM **28** (1981), no. 3, 423–434.

`solveAC` and `instantiateStep`

In our algorithm, when solving AC-unification problems, there are two important functions:

- `solveAC` - deals with the eliminating common arguments, generating a basis of solutions to the correspondent diophantine equation, relating “old” and “new” variables.
- `instantiateStep` - instantiates the variables that it can.

In the problem $f(a, X) \approx^? f(b, Y)$, we only need one call to `solveAC` followed by one call to `instantiateStep`. This, however, is not always the case.

Let f and g be AC function symbols and consider the problem:

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

Solving $f(X, Y, g(a, Z)) \approx^? f(W, Z, g(U, V))$

After we call `solveAC`, one branch generated is:

$$\{X \approx^? f(Z_1, Z_2), Y \approx^? f(Z_3, Z_4), g(a, Z) \approx^? Z_5$$
$$W \approx^? f(Z_1, Z_3), Z \approx^? f(Z_2, Z_4), g(U, V) \approx^? Z_5\}$$

and after we call `instantiateStep` we have:

$$\sigma = \{X \mapsto f(Z_1, Z_2), Y \mapsto f(Z_3, Z_4), W \mapsto f(Z_1, Z_3), Z \mapsto f(Z_2, Z_4)\}$$
$$P_1 = \{g(a, f(Z_2, Z_4)) \approx^? g(U, V)\}$$

We will need another call to `solveAC` (which will involve generating another basis of solution to a Diophantine equation) and to `instantiateStep`!