Formalising Completeness of AC-Unification

Gabriel Silva
XIV Summer Workshop on Mathematics (Universidade de Brasília)
Funded by a CAPES PhD scholarship
Advisor: Mauricio Ayala-Rincón, Co-Advisor: Maribel Fernández
https://gabriel951.github.io/

April 1, 2022

This work was done in collaboration with:

Figure 1: Mauricio Ayala-Rincón

Figure 2: Maribel Fernández

Figure 3: Daniele Nantes

Overview

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:

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

Related Work

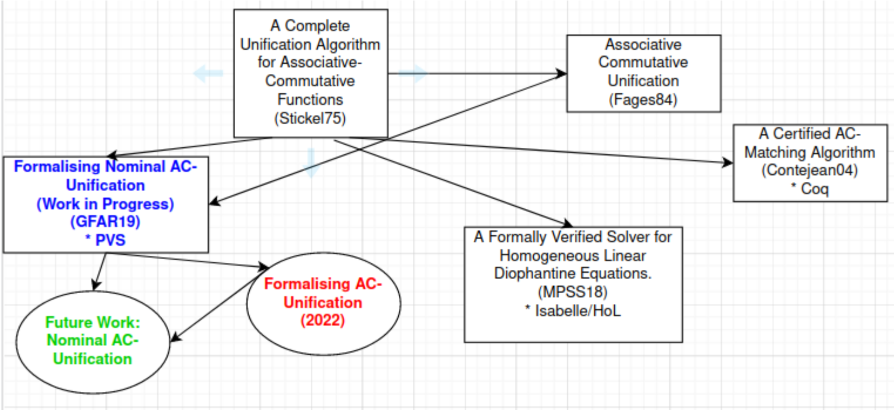

Figure 4: Main Related Work.

What We Have

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

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

In This Talk

First Part:

- Briefly discuss the challenge in AC-unification.
- Exemplify how we solve AC-unification (based on [1]).

Second part:

- Briefly discuss how adding a small hypothesis helped us in proving completeness.
- Argue that the hypothesis was just a technicality and show how we have sharpened the proof of completeness to remove this hypothesis.

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?

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

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:

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

Eliminate Common Arguments

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

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

Introducing a Linear Equation on $\mathbb{N}$

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.

UnB

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:

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

Termination, Soundness and Completeness

- Termination - Hard (use Fages' lexicographic measure)
- Soundness - Easy
- Completeness - Hard

Notation: $\delta \subseteq V$

We say that $\delta \subseteq V$ if $dom(\delta) \subseteq V$ and $Vars(im(\delta)) \subseteq V$.

Example 1

- If $V = \{X, Y\}$ and $\delta = \{X \mapsto f(a, b), Y \mapsto X\}$, then $\delta \subseteq V$.
- If $V = \{X, Y\}$ and $\delta = \{X \mapsto f(a, Z_1), \ Y \mapsto X\}$, then $\delta \not\subseteq V$.

Notation: $\sigma =_V \sigma'$

Given a set of variables V, we say that $\sigma =_V \sigma'$ if for every variable $X \in V$ we have $\sigma X = \sigma' X$.

Parameters of Our Algorithm

Our recursive algorithm receives a triple (P, σ, V) as parameter, where P is the unification problem we have to unify, σ is the substitution we have computed so far and a set of variables V that are/were in use.

The algorithm keeps on calling itself on progressively smaller inputs (according to a lexicographic measure) until it finishes. The initial call in order to unify terms t and s is done with $P = \{t \approx^? s\}$, $\sigma = Id$ and $Vars(t, s) \subseteq V$.

Theorem 2
If δ *unifies* $t \approx^? s$ *and* $\delta \subseteq V$ *and* $Vars(t, s) \subseteq V$*, then there is a substitution* $\gamma \in \mathtt{ACUnif}(\{t \approx^? s\}, Id, V)$ *such that* $\gamma \leq_V \delta$*.*

How bad is it that we have $\delta \subseteq V$?

How bad is it that we have $\delta \subseteq V$?

The hypothesis $\delta \subseteq V$ is simply a technicality that was put only in order to guarantee that the new variables introduced by the algorithm do not clash with the variables in $dom(\delta)$ or in $im(\delta)$.

Suppose that we call:
$\texttt{ACUnif}(\{f(X, X, Y, a, b, c) \approx^? f(b, b, b, c, Z)\}, Id, \{X, Y, Z\})$, and that
$\delta = \{X \mapsto f(Z_2, a, b), Z \mapsto f(a, Y, Z_2, a, Z_2, a), Z_4 \mapsto c\}$.

Then $V = \{X, Y, Z\}$ and $\delta \not\subseteq V$, but the substitution
$\sigma_4 = \{X \mapsto f(Z_6, b), Z \mapsto f(a, Y, Z_6, Z_6)\}$ that we computed is still more general than δ (restricted to the variables in V).

Indeed, if we take $\delta_1 = \{Z_6 \mapsto f(Z_2, a)\}$ then $\delta =_V \delta_1\sigma_4$.

Let's try removing the hypothesis $\delta \subseteq V$ and simply proving:

Theorem 3

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

What happens when we change V?

Investigate: what happens when we call `ACUnif` with an arbitrary V such that $Vars(t, s) \subseteq V$ versus what happens when we call `ACUnif` with $V = Vars(t, s)$?

Consider two situations, where we are trying to unify $\{f(X, X, Y, a, b, c) \approx^? f(b, b, b, c, Z)\}$:

1. **Situation 1**: We call our algorithm with $V = \{X, Y, Z\}$. In this case, the variables introduced by the algorithm are $Z_1, \ldots, Z_7$ (in this order).
2. **Situation 2**: We call our algorithm with $V' = \{X, Y, Z, Z_1, Z_2\}$. In this case, the variables introduced by the algorithm are $Z_3, \ldots, Z_9$ (in this order).

The variable Z_1 in **Situation 1** will play the same role as the variable Z_3 in **Situation 2** and so on. The renaming $\rho = \{Z_1 \mapsto Z_3, \ldots, Z_7 \mapsto Z_9\}$ should let us go from **Situation 1** to **Situation 2**.

In our example, we would have:

$$\sigma_4 = \{X \mapsto f(Z_6, b), Z \mapsto f(a, Y, Z_6, Z_6)\} \qquad \text{Situation 1}$$
$$\sigma'_4 = \{X \mapsto f(Z_8, b), Z \mapsto f(a, Y, Z_8, Z_8)\} \qquad \text{Situation 2}$$

Notice that we have $\sigma'_4 =_V \rho\sigma_4$.

Removing $\delta \subseteq V$

We have Theorem 2 and want to prove Theorem 3

Theorem 2

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

Theorem 3

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

Our attempt to prove Theorem 3:

Step 1: Let $V = \mathit{Vars}(t, s)$ and $V' = V \cup \mathit{dom}(\delta) \cup \mathit{Vars}(\mathit{im}(\delta))$. By Theorem 2, there is a substitution $\gamma' \in \texttt{ACUnif}(\{t \approx^? s\}, \mathit{Id}, V')$ such that $\gamma' \leq_{V'} \delta$. Therefore, we can write $\delta =_{V'} \delta_1\gamma'$.

Step 2: Find a substitution $\gamma \in \texttt{ACUnif}(\{t \approx^? s\}, \mathit{Id}, V)$ and a renaming ρ such that $\gamma' =_V \rho\gamma$.

Step 3: $\delta =_{V'} \delta_1\gamma' =_V \delta_1\rho\gamma$. So, we can conclude that $\gamma \leq_V \delta$.

How can we prove Step 2?

All that is left is to prove Step 2.

Let $V = \mathit{Vars}(t, s)$ and V' be a set of variables such that $V \subseteq V'$. How can we prove that if $\gamma' \in \mathtt{ACUnif}(\{t \approx^? s\}, \mathit{Id}, V')$ then there is a substitution $\gamma \in \mathtt{ACUnif}(\{t \approx^? s\}, \mathit{Id}, V)$ and a renaming ρ such that $\gamma' =_V \rho\gamma$?

We could try a proof by induction on the algorithm. Something like:

If $\gamma' \in \mathtt{ACUnif}(P, \sigma, V')$ then there is a substitution $\gamma \in \mathtt{ACUnif}(P, \sigma, V)$ and a renaming ρ such that $\gamma' =_V \rho\gamma$.

Would that work?

No! Although initially the two inputs (P, σ, V) and (P, σ, V') only differ in the third component (V vs V'), as we introduce new variables, the first component of the two inputs will differ. Moreover, if we instantiate the new variables, the second component will also differ.

How can we prove Step 2? (cont.)

How about we try proving this:

If $\gamma' \in \mathtt{ACUnif}(P', \sigma', V')$ then there is a substitution $\gamma \in \mathtt{ACUnif}(P, \sigma, V)$ and a renaming ρ such that $\gamma' =_V \rho\gamma$.

Would that work?

How can we prove Step 2? (cont.)

Not quite. We have lost the link between the input (P, σ, V) and the input (P', σ', V')! To make this relation, we introduced the definition of **renamed inputs** (shown in the next slide). The theorem we have to prove becomes:

Theorem 4 (Theorem of Renamed Inputs)

Suppose that (P', σ', V') is a renamed input of (P, σ, V) fixing $Vars(t, s)$. If $\gamma' \in \mathtt{ACUnif}(P', \sigma', V')$ then there is a substitution $\gamma \in \mathtt{ACUnif}(P, \sigma, V)$ and a renaming ρ such that $\gamma' =_V \rho\gamma$.

We say that (P', σ', V') is a renamed input of (P, σ, V) fixing a set of variables χ if there exists a renaming ρ such that the following conditions are met:

1. $\sigma' =_{\chi} \rho\sigma$.
2. $P' = \rho P$,
3. $\texttt{greatest}(V) \leq \texttt{greatest}(V')$.
4. $\chi \subseteq V$.
5. $dom(\rho) \subseteq V$
6. $Vars(img(\rho)) \subseteq V'$.
7. If $X \in im(\rho)$ and $X \notin dom(\rho)$ then $X \notin V$

Proving the Theorem of Renamed Inputs

We do a proof by induction using the lexicographic measure that we have used to prove termination of `ACUnif`.

Base of Induction

When `ACUnif` has finished unifying, the renamed inputs will be $(\emptyset, \sigma', V')$ and $(\emptyset, \sigma, V)$ and the algorithm returns as output $\gamma' = \sigma'$ (for the input $(\emptyset, \sigma', V')$) and $\gamma = \sigma$ (for the input $(\emptyset, \sigma, V)$). Due to Item 1 (i.e. $\sigma' =_{Vars(t,s)} \rho\sigma$), we guarantee the thesis of our theorem.

This is the motivation for Item 1 of the definition of renamed inputs.

Item 2 is subtly being used here to guarantee that when $P' = \emptyset$ we have $P = \emptyset$.

For the inductive step, let (P', σ', V') be a renamed input of (P, σ, V) fixing $Vars(t, s)$. Suppose that $\texttt{ACUnif}(P, \sigma, V)$ calls itself recursively with input (P_1, σ_1, V_1) and that $\texttt{ACUnif}(P', \sigma', V')$ calls itself recursively with input (P'_1, σ'_1, V'_1). To apply the inductive hypothesis, we must show that (P'_1, σ'_1, V'_1) is a renamed input of (P_1, σ_1, V_1) fixing $Vars(t, s)$.

The hardest cases are when `ACUnif` instantiates a variable or when `ACUnif` applies the `AC-Step` for AC-unification.

When going from (P, σ, V) to (P_1, σ_1, V_1) and from (P', σ', V') to (P'_1, σ'_1, V'_1) we need to guarantee that (P'_1, σ'_1, V'_1) is a renamed input of (P_1, σ_1, V_1) fixing $Vars(t, s)$. In order to guarantee that, we had to enlarge the definition of renamed inputs with Items 3-7.

Theorem 5

Suppose that $X \notin Vars(t)$ and that (P', σ', V') is a renamed input of (P, σ, V) fixing χ (with renaming ρ). Let $P = \{X \approx^? t\} \cup P_1$ and $P' = \{\rho X \approx^? \rho t\} \cup \rho P_1$. Let $\sigma_1 = \{X \mapsto t\}$ and $\sigma'_1 = \{\rho X \mapsto \rho t\}$. Then, $(\sigma'_1 P'_1, \sigma'_1 \sigma', V')$ is a renamed input of $(\sigma_1 P_1, \sigma_1 \sigma, V)$ fixing χ (with renaming ρ).

In this lemma, to guarantee that Items 1 and 2 of the definition of renamed inputs hold (for $(\sigma'_1 P'_1, \sigma'_1 \sigma', V')$ and $(\sigma_1 P_1, \sigma_1 \sigma, V)$), we had to use Items 4 and 7 of of the definition of renamed inputs (for (P', σ', V') and (P, σ, V)).

Dependency Between the PVS Files

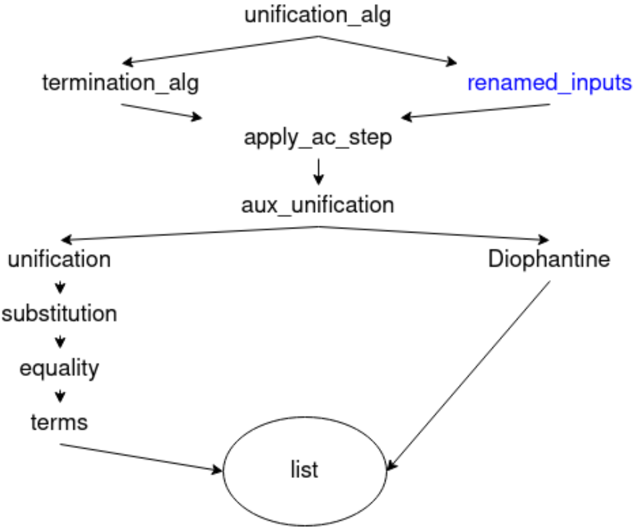

Figure 5: PVS Files Dependency Diagram

Amount of Theorems and TCCs Proved

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

File	Theorems	TCCs	Total	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

Size By File

Table 4: Size of .pvs and .prf files

File	.pvs	.prf	Percentage
unification_alg	6kB	2.2MB	5 %
termination_alg	22kB	11MB	26 %
rename_input	10kB	2.6MB	6 %
apply_ac_step	13kB	9.7MB	23 %
aux_unification	58kB	8.2MB	19 %
diophantine	23kB	1.1MB	3 %
unification	20kB	1MB	2 %
substitution	26kB	2.4MB	6 %
equality	12kB	1.1MB	3 %
terms	27kB	1MB	2 %
list	54kB	2MB	5 %
Total	271kB	42.3MB	**100%**

Conclusion

- We specified Stickel's AC-unification algorithm in the proof assistant PVS and proved it terminating, sound and complete.
- 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 pointed how we can improve the proof of completeness by removing $\delta \subseteq V$ and passing $Vars(t, s)$ as the initial parameter.

Future Work

We envision three different paths for future work:

1. Coming back to our initial goal: adapting the algorithm to the nominal setting, which would give the first nominal AC-unification algorithm.
2. Use the formalisation as a basis to formalise more efficient first-order AC-unification algorithms (for instance the one in [2]).
3. Use the formalisation to extract verified code and test AC-unification implementations (for instance in Maude, see [3]) for correctness/completeness.

Thank You

Thank you! Any comments/suggestions/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] M. Adi and C. Kirchner, "Ac-unification race: The system solving approach, implementation and benchmarks," *Journal of Symbolic Computation*, vol. 14, no. 1, pp. 51–70, 1992.

[3] M. Clavel, F. Durán, S. Eker, *et al.*, "Maude: Specification and programming in rewriting logic," *Theoretical Computer Science*, vol. 285, no. 2, pp. 187–243, 2002.

[4] L. Lamport, "How to write a 21st century proof," *Journal of Fixed Point Theory and Applications*, vol. 11, no. 1, pp. 43–63, 2012.

Pseudocode for the Algorithm

```
procedure ACUnif(P, σ, V)
    if nil?(P) then
        return cons(σ, nil)
    else
        ((t, s), P1) = choose(P)
        if (s matches X) and (X not in t) then
            σ1 = {X → t}
            σ′ = append(σ1, σ)
            P′ = σ1 P1
            return ACUnif(P′, σ′, V)
        else
            if t matches a then
                if s matches a then
                    return ACUnif(P1, σ, V)
                else
                    return nil
                end if
```

Pseudocode for the Algorithm (cont.)

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

Pseudocode for the Algorithm (cont.)

```
        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
        else
            if s matches f^AC s_1 then
                InputLst = applyACStep(P, σ, V)
                LstResults = map(ACUnif, InputLst)
                return flatten (LstResults)
            else
                return nil
            end if
        end if
    end if
end if
```

Pseudocode for the Algorithm (cont.)

end procedure

We will give a proof of:

Theorem 5

Suppose that $X \notin Vars(t)$ and that (P', σ', V') is a renamed input of (P, σ, V) fixing χ (with renaming ρ). Let $P = \{X \approx^{?} t\} \cup P_1$ and $P' = \{\rho X \approx^{?} \rho t\} \cup \rho P_1$. Let $\sigma_1 = \{X \mapsto t\}$ and $\sigma_1' = \{\rho X \mapsto \rho t\}$. Then, $(\sigma_1' P_1', \sigma_1' \sigma', V')$ is a renamed input of $(\sigma_1 P_1, \sigma_1 \sigma, V)$ fixing χ (with renaming ρ).

Renaming

We say that ρ is a renaming if two conditions are met:

- For every variable X, ρX is a variable.
- If $X \in dom(\rho)$ and $Y \in dom(\rho)$ and $\rho X = \rho Y$ then $X = Y$.

Nice Input

In our proof of completeness, we notice that every input (P, σ, V) that the algorithm receives satisfy these conditions:

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

When (P, σ, V) satisfies these conditions, we say that (P, σ, V) is a nice input.

We present a **structured proof** of the lemma, where some steps decompose in substeps and so on, as described by Leslie Lamport in [4].

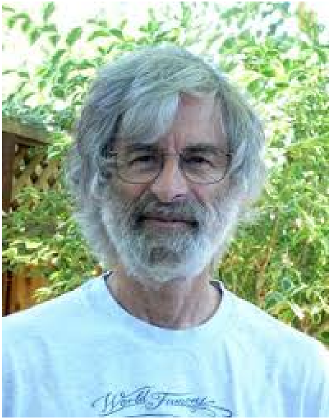

Figure 6: Leslie Lamport.

Proof:

$\langle 1 \rangle 1$. We first establish that: $\sigma'_1 \rho =_V \rho \sigma_1$.

Proof:

$\langle 2 \rangle 1$. It suffices to prove that for every variable $Z \in V$ we have $\sigma'_1 \rho Z = \rho \sigma_1 Z$. This is the same as proving that $[\rho X \mapsto \rho t] \rho Z = \rho [X \mapsto t] Z$.

$\langle 2 \rangle 2$. **Case:** $Z = X$. Then both sides are equal to ρt.

$\langle 2 \rangle 3$. **Case:** $Z \neq X$.

Proof:

$\langle 3 \rangle 1$. The right-hand side is equal to ρZ. It suffices to prove that the left-hand side is also equal to ρZ. To do that, it suffices to prove that $\rho Z \neq \rho X$. Suppose by contradiction that $\rho Z = \rho X$.

Proof (cont.)

$\langle 3 \rangle 2$. **Case:** $X \in dom(\rho)$ and $Z \in dom(\rho)$. By the definition of renaming, we must have $X = Z$.

$\langle 3 \rangle 3$. **Case:** $X \notin dom(\rho)$ and $Z \in dom(\rho)$. We have $\rho Z = X$, which means that $X \in Vars(img(\rho))$. Since we also have that $X \notin dom(\rho)$, by **Item 7** of the definition of renamed inputs we get that $X \notin V$. This however, contradicts the fact that $X \in P$ and $Vars(P) \subseteq V$.

$\langle 3 \rangle 4$. **Case:** $X \in dom(\rho)$ and $Z \notin dom(\rho)$. Similar to the previous case, exchanging the roles of X and Z.

$\langle 3 \rangle 5$. **Case:** $X \notin dom(\rho)$ and $Z \notin dom(\rho)$. Since $\rho X = \rho Z$ we get $X = Z$. This contradicts our hypothesis that $Z \neq X$.

Proof (cont.)

$\langle 1 \rangle 2$. The first item of the definition of renamed inputs is satisfied: $\sigma_1' P_1' = \rho\sigma_1 P_1$.

$\langle 2 \rangle 1$. Let t_i be an arbitrary term in P_1 and let t_i' be the correspondent in $P_1' = \rho P_1$. Then $t_i' = \rho t_i$, and it suffices to prove that $\sigma_1' \rho t_i = \rho\sigma_1 t_i$.

$\langle 2 \rangle 2$. It suffices to prove that for every variable $Z \in Vars(t_i)$ we have $\sigma_1' \rho Z = \rho\sigma_1 Z$. This follows from $\sigma_1' \rho =_V \rho\sigma_1$, since $P_1 \subseteq P$ and $Vars(P) \subseteq V$ (this last "$\subseteq$" is from the fact that (P, σ, V) is a nice input and from the definition of nice input.).

Proof (cont.)

$\langle 1 \rangle 3$. The second item in the definition of renamed inputs is satisfied: $\sigma_1'\sigma' =_\chi \rho\sigma_1\sigma$.

$\langle 2 \rangle 1$. Since (P', σ', V') is a renamed input of (P, σ, V), by **Item 2** of the definition we have $\sigma' =_\chi \rho\sigma$.Therefore $\sigma_1'\sigma' =_\chi \sigma_1'\rho\sigma$ and it suffices to prove that $\sigma_1'\rho\sigma =_\chi \rho\sigma_1\sigma$

$\langle 2 \rangle 2$. By Step $\langle 1 \rangle 1$ we have $\sigma_1'\rho =_V \rho\sigma_1$. Since $\sigma \subseteq V$ we get $\sigma_1'\rho\sigma =_V \rho\sigma_1\sigma$.

$\langle 2 \rangle 3$. Since (P', σ', V') is a renamed input of (P, σ, V), by **Item 4** of the definition we have $\chi \subseteq V$. Along with the last Step, this let us conclude that $\sigma_1'\rho\sigma =_\chi \rho\sigma_1\sigma$.

Proof (cont.)

$\langle 1 \rangle 4$. The Items 3-7 to prove that $(\sigma_1' P_1', \sigma_1' \sigma', V')$ is a renamed input of $(\sigma_1 P_1, \sigma_1 \sigma, V)$ are the same Items 3-7 of our hypothesis that (P', σ', V') is a renamed input of (P, σ, V), since there is no change in the renaming ρ or in the set of variables V.