Mechanisation of Equational Reasoning[†]

Libraries: https://github.com/nasa/pvslib/nominal and TRS

Mauricio Ayala-Rincón

November 18th, 2025 - Department of Informatics / King's College London

Departamentos de Matemática & Ciência da Computação

Universidade de Brasília

[†] Research supported by the Royal Society, and the Brazilian agencies CAPES, CNPq, and FAPDF

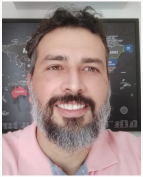
Joint Work With

Ana R. Oliveira

María Júlia D. Lima

Maribel Fernández

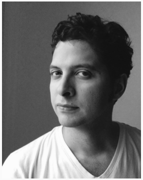
Daniele Nantes



Washington Ribeiro

Gabriela Ferreira

Thaynara de Lima



Mariano Moscato

Gabriel Silva

Marcos Mercandeli

David Cerna

Temur Kutsia

Outline

Equational Reasoning - Unification

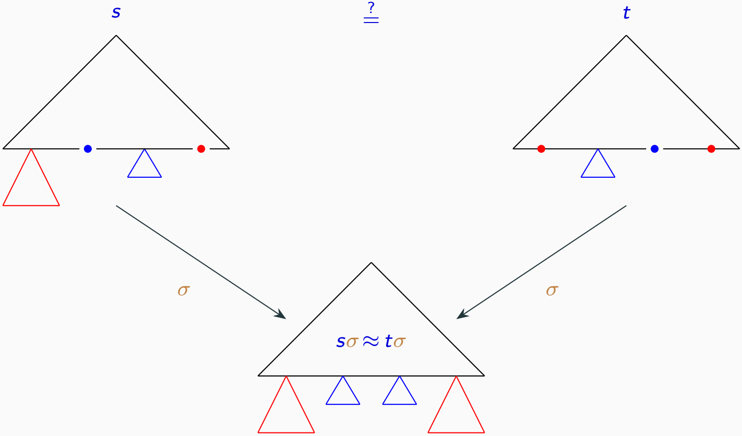
Equational Problems

- **Equality check**: $s = t$?
- **Matching**: There exists σ such that $s\sigma = t$?
- **Unification**: There exists σ such that $s\sigma = t\sigma$?
- **Anti-unification**: There exist r, σ and ρ such that $r\sigma = s$ and $r\rho = t$?

s and t, and u are *terms* in some *signature* and σ and ρ are *substitutions*.

Unification

Goal: find a substitution that identifies two expressions.



Syntactic Unification

- Goal: *to identify* two expressions.
- Method: replace variables with other expressions.

Example: for x and y variables, a and b constants, and f a function symbol,

- *Identify* $f(x, a)$ and $f(b, y)$

Syntactic Unification

- Goal: *to identify* two expressions.
- Method: replace variables with other expressions.

Example: for x and y variables, a and b constants, and f a function symbol,

- *Identify* $f(x, a)$ and $f(b, y)$
- solution $\{x/b, y/a\}$.

Example:

- Solution $\sigma = \{x/b\}$ for $f(x, y) = f(b, y)$ is *more general* than solution $\gamma = \{x/b, y/b\}$.

σ is *more general* than γ:

$$\text{there exists } \delta \text{ such that } \sigma\delta = \gamma;$$

$$\delta = \{y/b\}.$$

Syntactic Unification

Interesting questions:

- Decidability, Unification Type, Correctness and Completeness.
- Complexity.
- With adequate data structures, there are linear solutions (Martelli-Montanari 1976, Petterson-Wegman 1978).

Syntactic unification is of type *unary* and linear.

Unification Modulo

When operators possess equational properties, the problem becomes more complex.

Example: for f *commutative* (C), $f(x,y) \approx f(y,x)$:

- $f(x,y) = f(a,b)$?

The unification problem is of type *finitary*.

Unification Modulo

When operators possess equational properties, the problem becomes more complex.

Example: for f *commutative* (C), $f(x, y) \approx f(y, x)$:

- $f(x, y) = f(a, b)$?
- Solutions: $\{x/a, y/b\}$ and $\{x/b, y/a\}$.

The unification problem is of type *finitary*.

Example: for f *associative* (A), $f(f(x, y), z) \approx f(x, f(y, z))$:

- $f(x, a) = f(a, x)$?

The unification problem is of type *infinitary*.

Unification Modulo

Example: for f *associative* (A), $f(f(x, y), z) \approx f(x, f(y, z))$:

- $f(x, a) = f(a, x)$?
- Solutions: $\{x/a\}$, $\{x/f(a, a)\}$, $\{x/f(a, f(a, a))\}, \ldots$

The unification problem is of type *infinitary*.

Unification Modulo

Example: for f AC with *unity* (U), $f(x, e) \approx x$:

- $f(x, y) = f(a, b)$?

The unification problem is of type *finitary*.

Unification Modulo

Example: for f AC with *unity* (U), $f(x, e) \approx x$:

- $f(x, y) = f(a, b)$?
- Solutions: $\{x/e, y/f(a, b)\}$, $\{x/f(a, b), y/e\}$, $\{x/a, y/b\}$, and $\{x/b, y/a\}$.

The unification problem is of type *finitary*.

Unification Modulo

Example: for f A, and *idempotent* (I), $f(x,x) \approx x$:

- $f(x, f(y,x)) = f(f(x,z),x)$?

The unification problem is of type *zero* (Schmidt-Schauß 1986, Baader 1986).

Example: for f A, and *idempotent* (I), $f(x,x) \approx x$:

- $f(x, f(y,x)) = f(f(x,z),x))$?
- Solutions: $\{y/f(u, f(x,u)), z/u\}, \ldots$

The unification problem is of type *zero* (Schmidt-Schauß 1986, Baader 1986).

Unification Modulo

Example: for $+$ AC, and h *homomorphism* (h),
$h(x + y) \approx h(x) + h(y)$:

- $h(y) + a = y + z$?

The unification problem is of type *zero* and undecidable (Narendran 1996). The same happens for ACUh (Nutt 1990, Baader 1993).

Example: for $+$ AC, and h *homomorphism* (h),
$h(x+y) \approx h(x) + h(y)$:

- $h(y) + a = y + z$?
- Solutions: $\{y/a, z/h(a)\}, \{y/h(a)+a, z/h^2(a)\}, \ldots,$
  $\{y/h^k(a) + \ldots + h(a) + a, z/h^{k+1}(a)\}, \ldots$

The unification problem is of type *zero* and undecidable (Narendran 1996). The same happens for ACUh (Nutt 1990, Baader 1993).

Synthesis Unification modulo i

		Synthesis Unification modulo			
Theory	Unif. type	Equality-checking	Matching	Unification	Related work
Syntactic	1	$O(n)$	$O(n)$	$O(n)$	R65 MM76 PW78
C	ω	$O(n^2)$	NP-comp.	NP-comp.	BKN87 KN87
A	∞	$O(n)$	NP-comp.	NP-hard	M77 BKN87
AU	∞	$O(n)$	NP-comp.	decidable	M77 KN87
AI	0	$O(n)$	NP-comp.	NP-comp.	Klíma02 SS86 Baader86

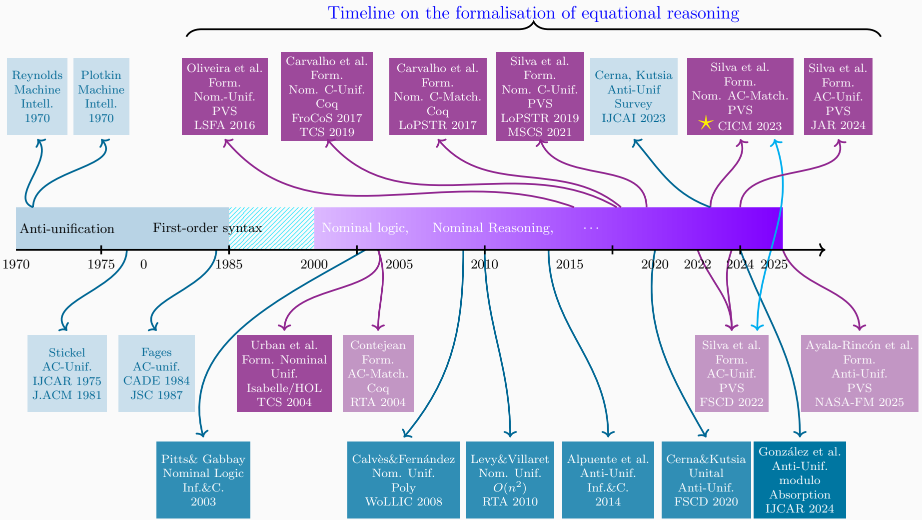
Synthesis Unification modulo

		Synthesis Unification modulo			
Theory	Unif. type	Equality-checking	Matching	Unification	Related work
AC	ω	$O(n^3)$	NP-comp.	NP-comp.	BKN87 KN87 KN92
ACU	ω	$O(n^3)$	NP-comp.	NP-comp.	KN92
AC(U)I	ω	-	-	NP-comp.	KN92 BMMO20
D	ω	-	NP-hard	NP-hard	TA87
ACh	0	-	-	undecidable	B93 N96 EL18
ACUh	0	-	-	undecidable	B93 N96

Results

		Synthesis Unification Nominal Modulo			
Theory	Unif. type	Equality-checking	Matching	Unification	Related work
$\approx_\alpha$	1	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	UPG04 LV10 CF08 CF10 LSFA2015
C	∞	$O(n^2 \log n)$	NP-comp.	NP-comp.	LOPSTR2017 FroCoS2017 TCS2019 LOPSTR2019 MSCS2021
A	∞	$O(n \log n)$	NP-comp.	NP-hard	LSFA2016 TCS2019
AC	ω	$O(n^3 \log n)$	NP-comp.	NP-comp.	LSFA2016 TCS2019 CICM2023 JAR2024

Synthesis on Nominal Equational Modulo



Anti-unification

Joint Work With

Maria Júlia Dias Lima
CC / U. Brasília

Marcos Mercandeli
Math / U. Brasília

NASA Formal Methods 2025

Thaynara Arielly de Lima
U. F. Goiâs

Mariano Miguel Moscato
AMA - NASA LaRC

Temur Kutsia
RISC/JKU Linz

Anti-unification - History

- Introduced by Gordon Plotkin [Plo70] and John Reynolds [Rey70]
- First-order: syntactic [Baa91]; C, A, and AC [AEEM14]; idempotent [CK20b], unital [CK20c], semirings [Cer20], absorptive [ACBK24]
- Higher-Order: patterns [BKLV17], top maximal and shallow generalisations variants [CK20a], equational patterns [CK19], modulo [CK20a]
- See david Cerna and Temur Kutsia survey [CK23].

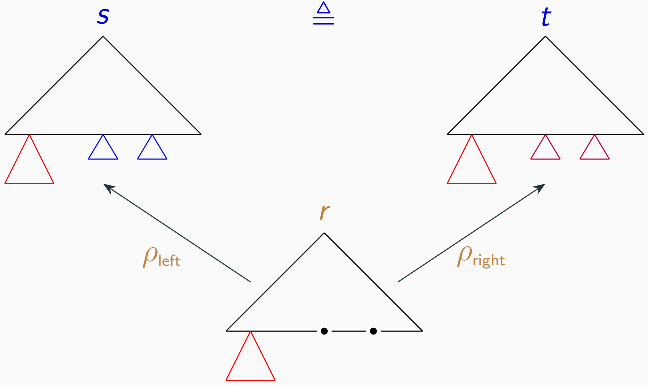
Applications

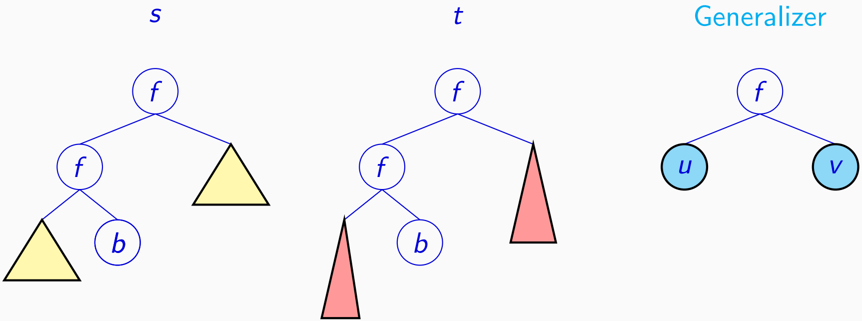
Applications of anti-unification include:

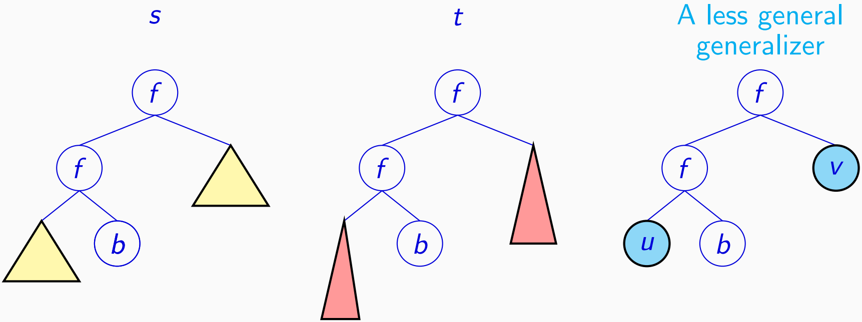
- searching a large hypothesis space in inductive logic programming (ILP) for logic-based machine learning [CDEM22];
- preventing bugs and misconfigurations in software [MBK+20];
- detecting code clones [VY19];
- searching recursion schemes for efficient parallel compilation [BBH18].

Anti-unification

Goal: find the commonalities between two expressions.

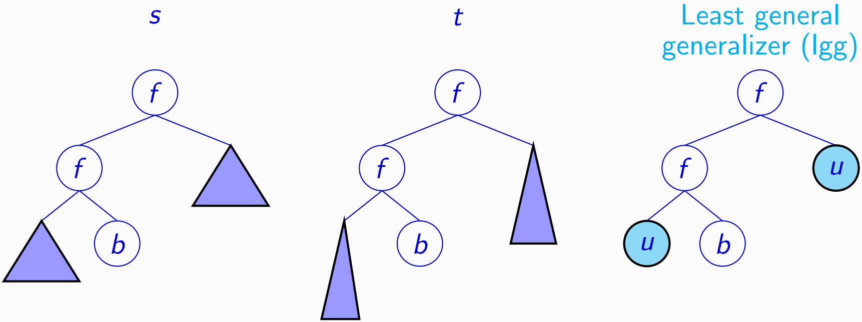



s
f
f
b
t
f
f
b
Generalizer
f
u
v



s
f
f
b
t
f
f
b
A less general generalizer
f
f
v
u
b

Anti-Unification



Syntactic anti-unification

- terms: $t ::= x \mid \langle\rangle \mid \langle t, t\rangle \mid f\ t$
- Labelled equations (AUTs): $E = \{s_i \underset{x_i}{\triangleq} t_i \mid i \leq n\}$

Configurations: $$\left\langle \underbrace{E_U}_{\text{Unsolved equations}} \quad \underbrace{E_S}_{\text{Solved equations}} \quad \underbrace{\sigma}_{\text{Substitution}} \right\rangle$$

Configuration constraints

- All labels in $E_U \cup E_S$ are different,
- no *repeated equations* appear in E_S, and
- no label in $E_U \cup E_S$ belongs to $Domain(\sigma)$.

Solved equations have left- and right-hand sides not headed by the same symbols.

Inference Rules

$$\text{(Decompose Function)}\ \frac{\langle \{f\ s \underset{x}{\triangleq} f\ t\} \cup E, S, \sigma \rangle}{\langle \{s \underset{y}{\triangleq} t\} \cup E, S, \{x \mapsto f\ y\} \circ \sigma \rangle}$$

$$\text{(Decompose Pair)}\ \frac{\langle \{\langle s, u \rangle \underset{x}{\triangleq} \langle t, v \rangle\} \cup E, S, \sigma \rangle}{\langle \{s \underset{y}{\triangleq} t, u \underset{z}{\triangleq} v\} \cup E, S, \{x \mapsto \langle y, z \rangle\} \circ \sigma \rangle}$$

$$\text{(Solve Repeated)}\ \frac{\langle \{s \underset{x}{\triangleq} t\} \cup E, S, \sigma \rangle}{\langle E, S, \{x \mapsto x'\} \circ \sigma \rangle}\ \text{if } s \underset{x}{\triangleq} t \text{ solved and } \exists x' \text{ with } s \underset{x'}{\triangleq} t \in S$$

$$\text{(Solve Non-Repeated)}\ \frac{\langle \{s \underset{x}{\triangleq} t\} \cup E, S, \sigma \rangle}{\langle E, \{s \underset{x}{\triangleq} t\} \cup S, \sigma \rangle}\ \text{if if } s \underset{x}{\triangleq} t \text{ solved and there is no } s \underset{x'}{\triangleq} t \in S$$

$$\text{(Syntactic)}\ \frac{\langle \{s \underset{x}{\triangleq} s\} \cup E, S, \sigma \rangle}{\langle E, S, \{x \mapsto s\} \circ \sigma \rangle}\ \text{if } s \underset{x}{\triangleq} s \text{ neither decomposable nor solvable}$$

Example

$$
\begin{array}{ll}
 & \langle\{f\langle f\langle c,b\rangle,c\rangle \triangleq_{x} f\langle f\langle d,b\rangle,d\rangle\},\emptyset,id\rangle \\
\text{(DecF)} & \overline{\langle\{\langle f\langle c,b\rangle,c\rangle \triangleq_{y} \langle f\langle d,b\rangle,d\rangle\},\emptyset,\{x\mapsto f\ y\}\rangle} \\
\text{(DecP)} & \overline{\langle\{f\langle c,b\rangle \triangleq_{z_1} f\langle d,b\rangle, c \triangleq_{z_2} d\},\emptyset,\{x\mapsto f\ \langle z_1,z_2\rangle\}\rangle} \\
\text{(DecF)} & \overline{\langle\{\langle c,b\rangle \triangleq_{z_3} \langle d,b\rangle, c \triangleq_{z_2} d\},\emptyset,\{x\mapsto f\ \langle f\ z_3,z_2\rangle\}\rangle} \\
\text{(DecP)} & \overline{\langle\{c \triangleq_{z} d, b \triangleq_{z_4} b, c \triangleq_{z_2} d\},\emptyset,\{x\mapsto f\ \langle f\ \langle z,z_4\rangle,z_2\rangle\}\rangle} \\
\text{(SolNR)} & \overline{\langle\{b \triangleq_{z_4} b, c \triangleq_{z_2} d\},\{c \triangleq_{z} d\},\{x\mapsto f\ \langle f\ \langle z,z_4\rangle,z_2\rangle\}\rangle} \\
\text{(Synt)} & \overline{\langle\{c \triangleq_{z_2} d\},\{c \triangleq_{z} d\},\{x\mapsto f\ \langle f\ \langle z,b\rangle,z_2\rangle\}\rangle} \\
\text{(SolR)} & \overline{\langle\emptyset,\{c \triangleq_{z} d\},\{x\mapsto f\ \langle f\ \langle z,b\rangle,z\rangle\}\rangle}
\end{array}
$$

* Generalizer: $r = f\langle f\langle z,b\rangle,z\rangle$, $\rho_{\text{left}} = \{z\mapsto c\}$, and $\rho_{\text{right}} = \{z\mapsto d\}$.

PVS Verification

Verification Basics

The type `Configuration` is represented as $[unsolved, solved : list[AUT], substitution : nice?]$ and the predicate $validConfiguration?$ states the constraints expressed in the Definition of configuration.

Let $\langle Eu, Es, \sigma \rangle$ be a $Configuration$. It has type $validConfiguration?$ if for $eqs = Eu \cup Es$:

$$\begin{array}{ll}
disjoint?(Vars(eqs), labels(eqs)) & \wedge \\
card(labels(eqs)) = length(eqs) & \wedge \\
disjoint?(Domain(\sigma), labels(eqs) \cup Vars(eqs)) & \wedge \\
Solved?(Es) & \wedge \\
NotRepeated?(Es) &
\end{array}$$

Verification Basics

unsolved and *solved* lists of AUTs ($list[AUT]$) represent solved and unsolved equations of a configuration.

Configurations are deterministically classified according to their *derivability type* based on the *type* of their head unsolved AUT: $match\_DecF?$, $match\_DecP?$, $match\_Synt?$, and $match\_Sol?$.

Let $s \underset{x}{\triangleq} t$ be the head of Eu in a configuration $\langle Eu, Es, \sigma \rangle$.

$\langle Eu, Es, \sigma \rangle$ has type $match\_DecF\_conf?$ if $s \underset{x}{\triangleq} t$ has type $match\_DecF?$, specified as:

$$app?(s) \wedge app?(t) \wedge fun\_Symbol(s) = fun\_Symbol(t)$$

Verification Basics

Then, configurations have the type:

- *match_DecF_conf?*,
- *match_DecP_conf?*,
- *match_Synt_conf?*,
- *match_Sol_conf?*

or, when the unsolved part is empty,

- *normal_configuration?*.

Antiunification Algorithm Specification

The inference rules (DecF), (DecP), and (Synt) were specified as function declarations *DecF*, *DecP*, and *Synt*, with parameter configurations of types *match_DecF_conf?*, *match_DecP_conf?*, and *match_Synt_conf?*, respectively.

The solve rules (SolR) and (SolNR) were integrated into a unique rule *Solve* with parameter configuration of type *match_Sol_conf?*.

To automate the proofs of *termination*, *configuration validity*, and *preservation of niceness* of the *Antiunify* algorithm, these properties were encoded in the types of the functions representing the rules.

Antiunification Algorithm Specification

E.g., consider the type of the function *DecF*.

Let $c = \langle Eu, Es, \sigma \rangle$ and $c' = \langle Eu', Es', \sigma' \rangle$ be the input and output configurations, respectively.

- Its input type is a configuration of type *match_DecF_conf?*.
- Its output type is specified by the *dependent type* predicate:

$$
\begin{aligned}
&tail(Eu') = tail(Eu) && \wedge \\
&size(Eu') < size(Eu) && \wedge \\
&\underbrace{(label(Eu[0]))}_{\text{head label}}\,\sigma' = \underbrace{fun\_Symbol(Eu[0])}_{\text{head function symbol}}\ \underbrace{(label(Eu'[0]))}_{\text{fresh label}}
\end{aligned}
$$

The *Antiunify* algorithm is defined as a recursive function of type *validConfiguration*? $\to$ *validConfiguration*?. Its *measure* is the size of the unsolved part of the initial configuration.

Antiunify recursively checks the type of the head of the unsolved part to apply the respective inference rule.

By restricting the types of the functions specifying the inference rules (DecF), (DecP), (Solve), and (Synt), PVS automatically proves that *Antiunify* *terminates* and that every output of the *Antiunify* algorithm fulfils the *validConfiguration*? predicate.

Anti-unification Algorithm Verification

Let $\mathit{Antiunify}(\langle \mathit{Eu}, \mathit{Es}, \sigma\rangle) = \langle \emptyset, \mathit{Es}', \sigma'\rangle$. Three auxiliary lemmas about invariants and configuration preservation are highlighted.

1. *antiunify_sub_preserves_terms* states that
   if $t \in \mathit{Range}(\sigma)$ and $\mathit{disjoint?}(\mathit{Vars}(t), \mathit{labels}(\mathit{Eu}))$ then
   $t\sigma' = t\sigma$.

   - It is applied twice to (2) and once to (SolveR).
2. *antiunify_dom_sub_preserves_vars_unsolved* states that
   $\mathit{disjoint?}(\mathit{Domain}(\sigma'), \mathit{Vars}(\mathit{Eu}))$.

   - It is applied once to (Synt).
3. *antiunify_solved_labels_preserve_vars_unsolved* states that
   $\mathit{disjoint?}(\mathit{labels}(\mathit{Es}'), \mathit{Vars}(\mathit{Eu}))$.

   - It is applied twice to (Synt).

No preservation lemma was required by (DecF) and (DecP), and (SolveNR) depends on simple preservation lemmas.

The proof of the *soundness* theorem, *antiunif_is_sound*, stated below, follows by induction on the size of configurations and case analysis.

Let $c = \langle Eu, Es, \sigma \rangle$ be any input valid configuration, and let $\langle Eu', Es', \sigma' \rangle$ be the final configuration computed by $Antiunify(c)$. Then

$$generalizer?(Eu, \sigma', \rho_{\text{left}}(Es'), \rho_{\text{right}}(Es'))$$

i.e.,

$$\text{for any AUT } s \underset{x}{\triangleq} t \in Eu :$$

$$x\sigma'\rho_{\text{left}}(Es') = s, \text{ and } x\sigma'\rho_{\text{right}}(Es') = t$$

Above, ρ_{left} and ρ_{right} are substitutions mapping each label in a solved list of AUTs to the left and right terms of the AUT, respectively.

Quantitative Data

Table 1: Formalisation in numbers

PVS theory	Formulas	TCCs	Inference Rule	Proof size (# lines)	Dependencies (# lines)
Terms	119	37	-	-	-
Substitution	115	18			
Anti-unification	116	41	(DecF)	64	-
			(DecP)	140	-
			(Synt)	269	1624
			(SolveR)	245	663
			(SolveNR)	63	111

Linear Anti-unification and Anti-unification modulo

Linear Anti-unification and Anti-unification modulo

- Variants of anti-unification such as the linear case, give rise to surprising results. Linear anti-unification is the restriction to linear solutions.
- Interest in the formalisation of anti-unification for theories with Commutative, Associative, and Absorptive symbols: C-, A-, and $\mathfrak{a}$-symbols.
- Related $\mathfrak{a}$-symbols are a pair of a function and a constant symbol, (f, ε_f), satisfying the axioms

$$\{f(\varepsilon_f, x) = \varepsilon_f,\ f(x, \varepsilon_f) = \varepsilon_f\}$$

Example

[Communicated by T. Kutsia] Unification type ∞ and *unary* for the linear and the unrestricted case, respectively.

Equational axioms:

$$\{f(g(x,x)) = g(x,x)\}$$

Problem:

$$g(a,a) \triangleq g(b,b)$$

Linear solutions: $\{g(x,y), f(g(x,y)), f(f(g(x,y))), \ldots\}$.

Unrestricted solution: $g(x,x)$.

Example

[Communicated by T. Kutsia] Unification type *zero* and ∞ for the linear and the unrestricted case, respectively.

Equational axioms:

$$E = f(f(x,a),a) = f(x,a), f(f(x,b),b) = f(x,b)]$$

Problem:

$$f(a,a) \triangleq f(b,b)$$

Linear solutions: $f(x,y), f(f(x,y),z), f(f(f(x,y),z),u), \ldots$. It is a descending chain of less general generalisers.

Unrestricted solutions:
$\{f(x,x), f(f(x,x),x), f(f(f(x,x),x),x), \ldots\}$.

Example

[Communicated by T. Kutsia] Unification type *zero* and *unary* for the linear and the unrestricted case, respectively.

Equational axiom:

$$E = f(f(x, s(x)), s(x)) = f(x, x)]$$

Problem:

$$f(a, s(a)) \triangleq f(b, s(b))$$

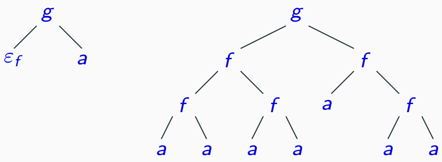
Linear solutions:

$f(x, s(y)), f(f(x, s(y)), s(z)), f(f(f(x, s(y)), s(z)), s(u)), \ldots$. It is a descending chain of less general generalisers.

Unrestricted solution: $\{f(x, s(x))\}$.

Anti-unification in ($\mathfrak{a}$)(A)(C)($\mathfrak{a}$A)($\mathfrak{a}$C)-theories

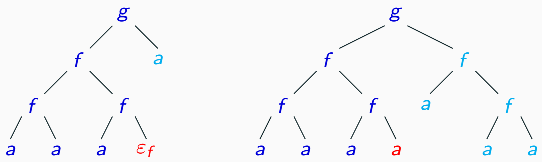
Example



An $\mathfrak{a}$-generalisation and an $\mathfrak{a}$A-generalisation will be illustrated.

Anti-unification in (𝔞)(A)(C)(𝔞A)(𝔞C)-theories

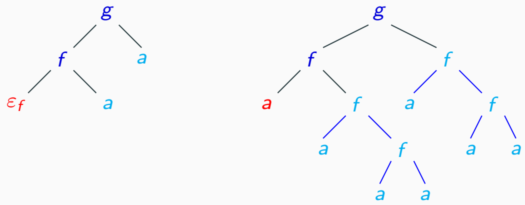
By *expanding* ε_f in $g(\varepsilon_f, a)$, one obtains:



Notice that $g(f(f(a, a), f(a, x)), y)$ is an 𝔞-generalisation.

Anti-unification in ($\mathfrak{a}$)(A)(C)($\mathfrak{a}$A)($\mathfrak{a}$C)-theories

Considering the same terms modulo $\mathfrak{a}$A, and by *expanding* ε_f in $g(\varepsilon_f, a)$, one has:



$g(f(x, y), y)$ is an $\mathfrak{a}$A-generalisation but not an $\mathfrak{a}$-generalisation.

Anti-unification modulo types

Theory	Anti-unification type	References
Syntactic	1	[Plo70, Rey70]
A	ω	[AEEM14]
C	ω	[AEEM14]
† $(\mathrm{U})^{1}$	ω	[CK20c]
$(\mathrm{U})^{\geq 2}$	nullary	[CK20c]
‡ $\mathfrak{a}$	∞	[ACBK24]
$\mathfrak{a}(\mathrm{C})$	∞	[ACBK24]

($\dagger$)Unital: $\{f(\iota_f, x) = x,\ f(x, \iota_f) = x\}$

($\ddagger$)Absorptive: $\{f(\varepsilon_f, x) = \varepsilon_f,\ f(x, \varepsilon_f) = \varepsilon_f\}$

Conclusions and Future Work

Conclusions

- Formal certification of nominal equational reasoning procedures is a target of the cooperation UnB/KCL.
- Although anti-unification has become of increasing interest, formal certification of anti-unification algorithms has not been explored except for the simplest syntactic case [ARdLK+25].
- The development of procedures to solve anti-unification modulo theories is crucial.
- Only recently, anti-unification modulo $\mathfrak{a}$-, C-, and ($\mathfrak{a}$C)-symbols have been addressed. Procedures combining such properties are challenging from theoretical and practical perspectives [ACBK24].

Thank you for your attention!

Thank you for your attention!

- Mauricio Ayala-Rincón, David M. Cerna, Andrés Felipe González Barragán, and Temur Kutsia, *Equational Anti-unification over Absorption Theories*, IJCAR, 2024.
- María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer, *A modular order-sorted equational generalization algorithm*, Information and Computation **235** (2014), 98–136.
- Mauricio Ayala-Rincón, Thaynara Arielly de Lima, Temur Kutsia, Mariano Moscato, and Maria Julia Dias, *Verification of an Anti-Unification Algorithm in PVS*, 17th Int. Symposium NASA Formal Methods NFM, Lecture Notes in Computer Science, vol. In press, Springer, 2025.

- Franz Baader, *Unification, weak unification, upper bound, lower bound, and generalization problems*, RTA, 1991.
- Adam D. Barwell, Christopher Brown, and Kevin Hammond, *Finding parallel functional pearls: Automatic parallel recursion scheme detection in haskell functions via anti-unification*, Future Gener. Comput. Syst. **79** (2018), 669–686.
- Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret, *Higher-order pattern anti-unification in linear time*, J. Autom. Reason. **58** (2017), no. 2, 293–310.
- Andrew Cropper, Sebastijan Dumancic, Richard Evans, and Stephen H. Muggleton, *Inductive logic programming at 30*, Mach. Learn. **111** (2022), no. 1, 147–172.

- David M. Cerna, *Anti-unification and the theory of semirings*, Theo. Com. Sci. **848** (2020), 133–139.
- David M. Cerna and Temur Kutsia, *A generic framework for higher-order generalizations*, FSCD, 2019.
- ———, *Higher-order pattern generalization modulo equational theories*, Math. Struct. Comput. Sci. **30** (2020), no. 6, 627–663.
- ———, *Idempotent anti-unification*, ACM Trans. Comput. Log. **21** (2020), no. 2, 10:1–10:32.
- ———, *Unital anti-unification: type algorithms*, 2020.
- ———, *Anti-unification and generalization: A survey*, IJCAI, 2023.

- Sonu Mehta, Ranjita Bhagwan, Rahul Kumar, Chetan Bansal, Chandra Maddila, B. Ashok, Sumit Asthana, Christian Bird, and Aditya Kumar, *Rex: Preventing bugs and misconfiguration in large services using correlated change analysis*, 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2020, pp. 435–448.
- Gordon D. Plotkin, *A note on inductive generalization*, Machine Intelligence **5** (1970), 153–163.
- John C. Reynolds, *Transformational system and the algebric structure of atomic formulas*, Machine Intelligence **5** (1970), 135–151.

References v

Wim Vanhoof and Gonzague Yernaux, *Generalization-driven semantic clone detection in CLP*, 29th Int. Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR, LNCS, vol. 12042, 2019, pp. 228–242.