Computational Verification of Anti-Unification

Mauricio Ayala-Rincón

RISC Forum (RISC/JKU), Hagenberg, 28th April 2025

Departamentos de Matemática & Ciência da Computação

Universidade de Brasília

† Research supported by the Brazilian agencies CAPES and CNPq

Joint Work With

Maria Júlia Dias Lima

Mariano Miguel Moscato

Thaynara Arielly de Lima

Temur Kutsia

Outline

1. Equational Reasoning
2. Unification
3. Anti-unification
4. Syntactic anti-unification
5. PVS Verification
6. Conclusions and Future Work

Equational Reasoning

Equational Problems

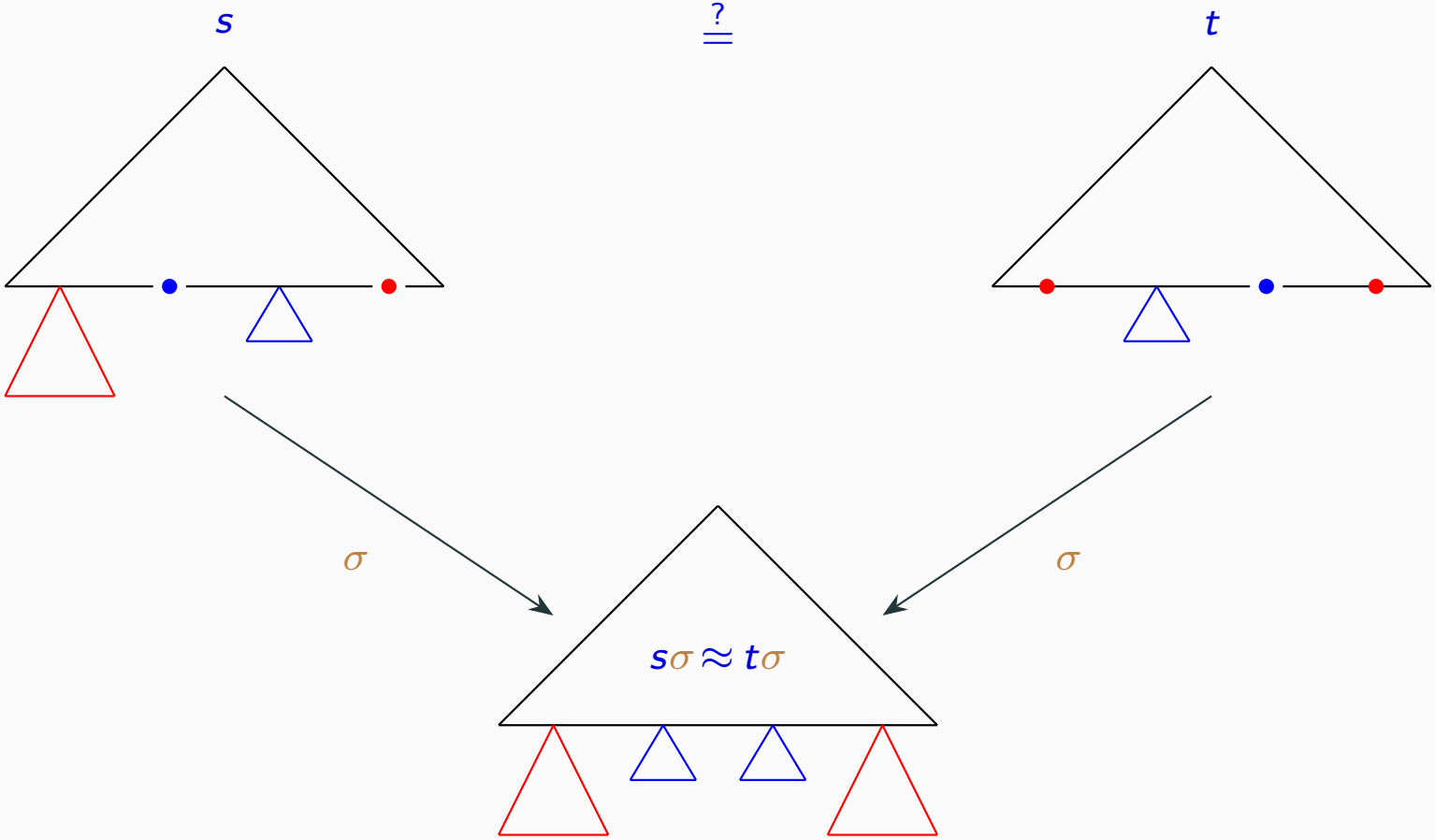
- **Equality check**: $s = t$?
- **Matching**: There exists σ such that $s\sigma = t$?
- **Unification**: There exists σ such that $s\sigma = t\sigma$?
- **Anti-unification**: There exist r, σ and ρ such that $r\sigma = s$ and $r\rho = t$?

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

Unification

Unification

Goal: find a substitution that identifies two expressions.



Syntactic Unification

- Goal: *to identify* two expressions.
- Method: replace variables by 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 by 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 $\{x/b, y/a\}$.

Syntactic Unification

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:

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

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

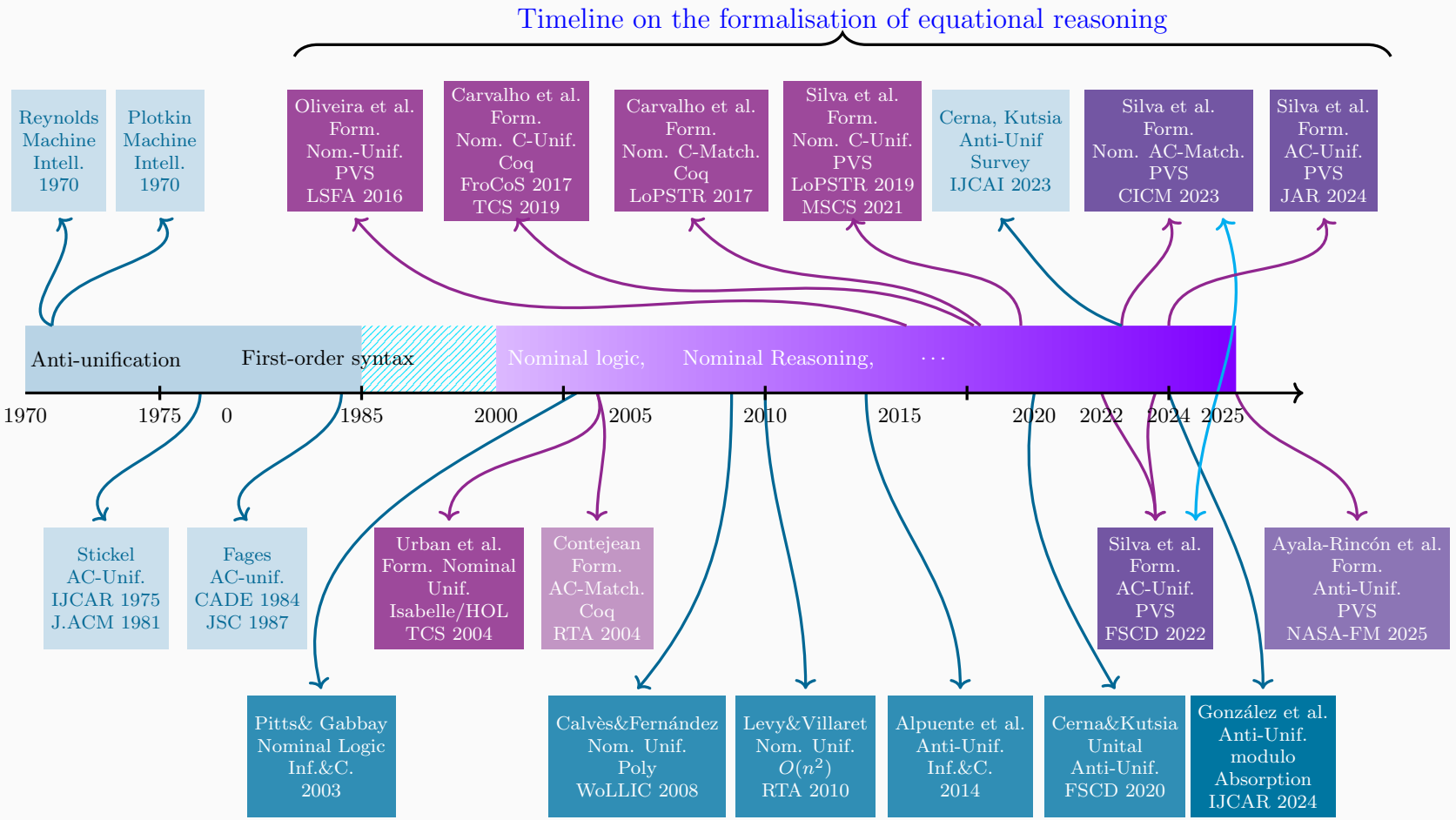
Syntactic Unification

Relevant questions:

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

Syntactic unification is of type *unary* and linear.

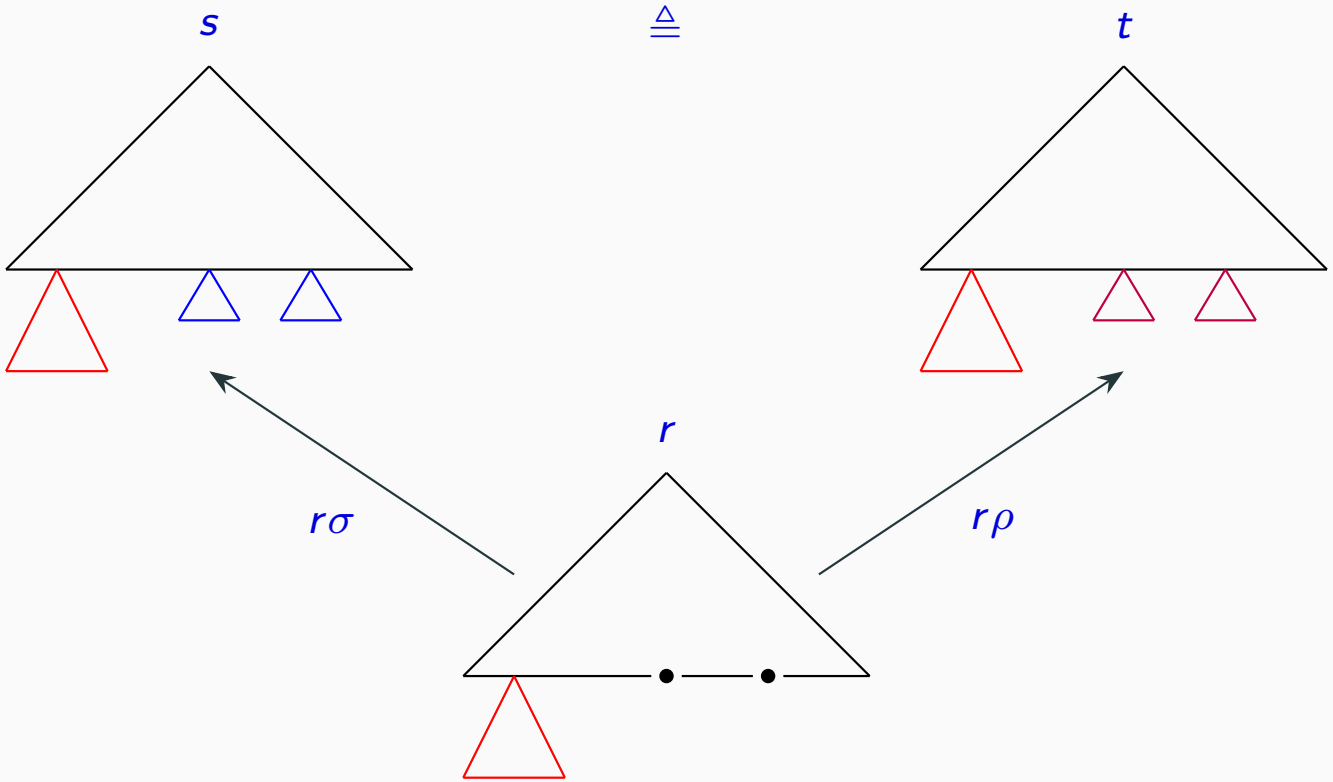
Our Formalizations on Equational Reasoning



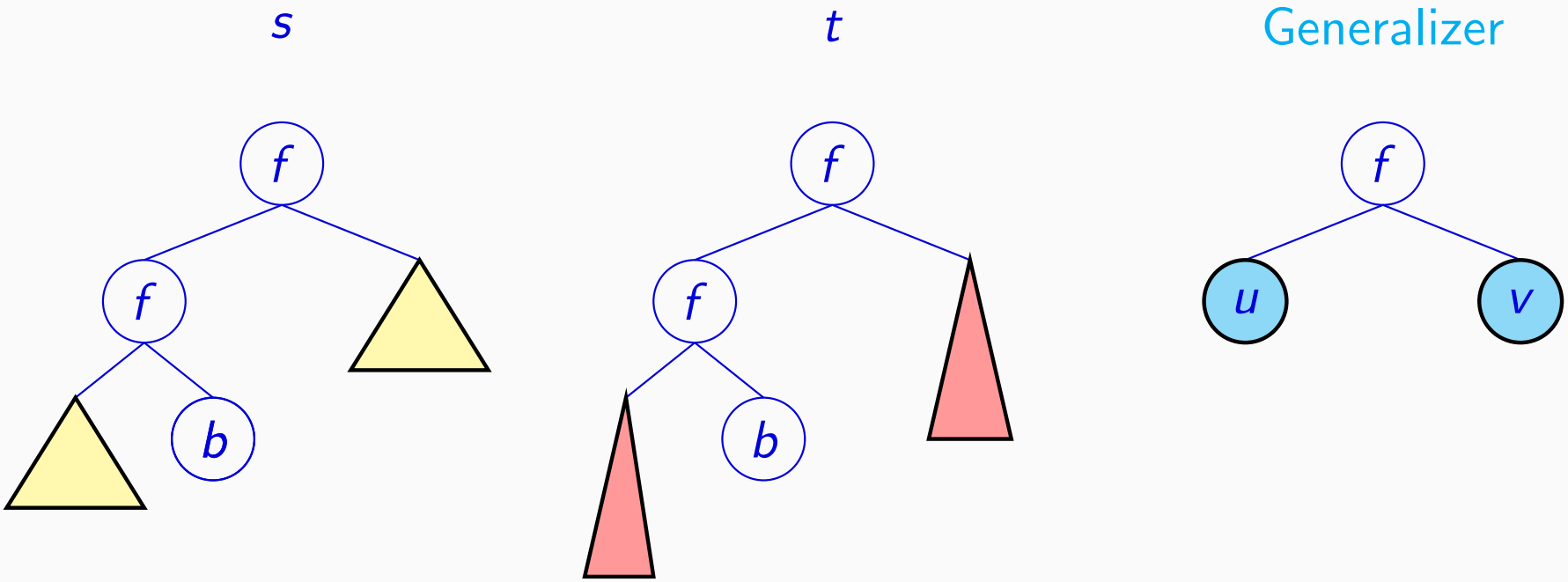
Anti-unification

Anti-unification

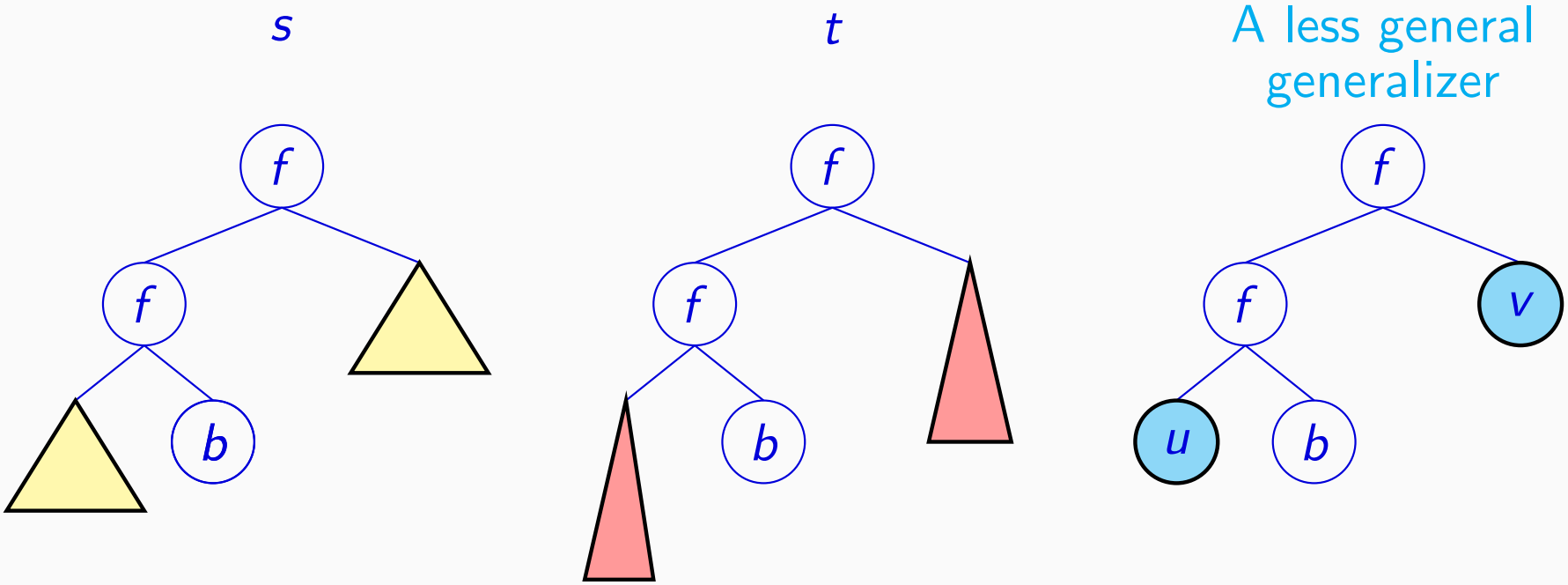
Goal: find the commonalities between two expressions.



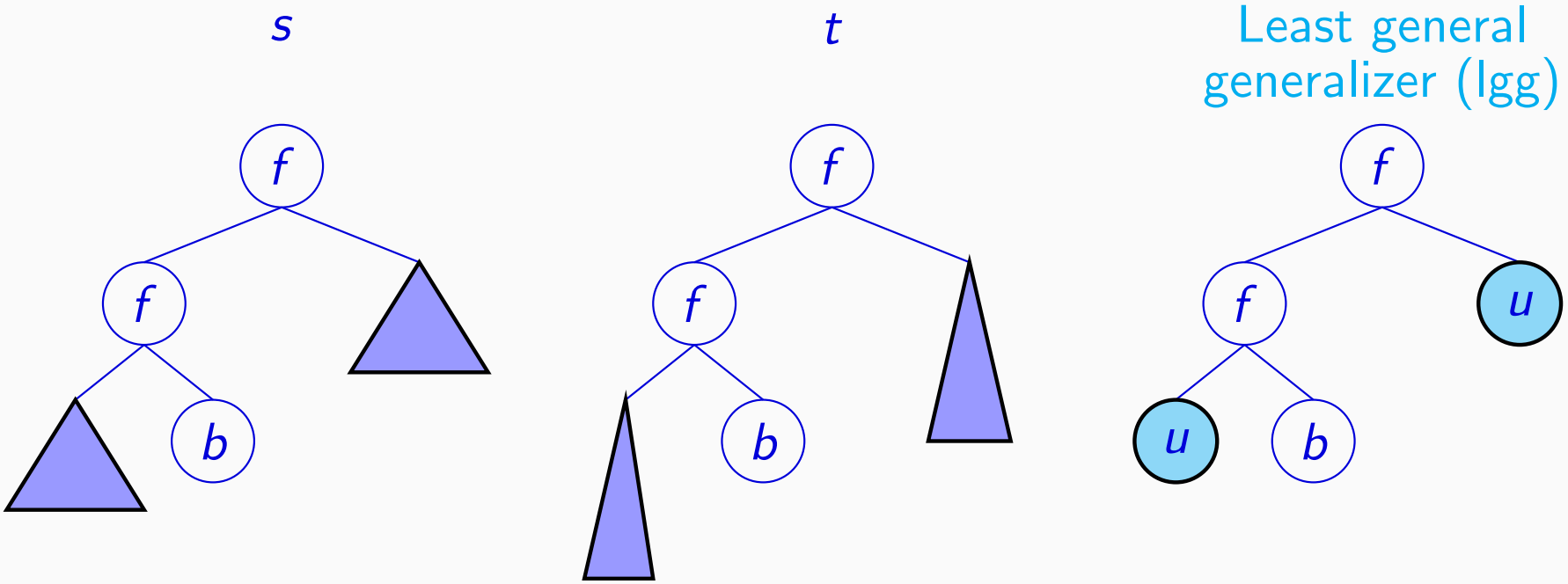
Anti-Unification



Anti-Unification



Anti-Unification



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 generalizations 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].

Syntactic anti-unification

Formal verification - Syntactical case

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

Configurations:

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

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 $dom(\sigma)$.

Inference Rules

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

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

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

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

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

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle\{f\langle f\langle c,b\rangle,c\rangle \underset{x}{\triangleq} f\langle f\langle d,b\rangle,d\rangle\},\emptyset,id\rangle}{\langle\{\langle f\langle c,b\rangle,c\rangle \underset{y}{\triangleq} \langle f\langle d,b\rangle,d\rangle\},\emptyset,\{x\mapsto f\ y\}\rangle}\,(\text{DecF})}{\langle\{f\langle c,b\rangle \underset{z_1}{\triangleq} f\langle d,b\rangle, c\underset{z_2}{\triangleq} d\},\emptyset,\{x\mapsto f\ \langle z_1,z_2\rangle\}\rangle}\,(\text{DecP})}{\langle\{\langle c,b\rangle \underset{z_3}{\triangleq} \langle d,b\rangle, c\underset{z_2}{\triangleq} d\},\emptyset,\{x\mapsto f\ \langle f\ z_3,z_2\rangle\}\rangle}\,(\text{DecF})}{\langle\{c\underset{z}{\triangleq} d, b\underset{z_4}{\triangleq} b, c\underset{z_2}{\triangleq} d\},\emptyset,\{x\mapsto f\ \langle f\ \langle z,z_4\rangle,z_2\rangle\}\rangle}\,(\text{DecP})}{\langle\{b\underset{z_4}{\triangleq} b, c\underset{z_2}{\triangleq} d\},\{c\underset{z}{\triangleq} d\},\{x\mapsto f\ \langle f\ \langle z,z_4\rangle,z_2\rangle\}\rangle}\,(\text{SolNR})}{\langle\{c\underset{z_2}{\triangleq} d\},\{c\underset{z}{\triangleq} d\},\{x\mapsto f\ \langle f\ \langle z,b\rangle,z_2\rangle\}\rangle}\,(\text{Synt})}{\emptyset,\{c\underset{z}{\triangleq} d\},\{x\mapsto f\ \langle f\ \langle z,b\rangle,z\rangle\}\rangle}\,(\text{SolR})$$

PVS Verification

The type `Configuration` and the predicate `validConfiguration?` state the notions expressed in the Definition of configuration.

Solved and unsolved lists represent solved and unsolved equations of a configuration (`list[AUT]`). This allows the deterministic classification of the derivability type of a configuration based on the classification of its first unsolved AUT: `match_DecF?`, `match_DecP?`, `match_Synt?`, and `match_Sol?`.

Configurations have 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 rules (DecF), (DecP) and (Synt) were specified as function declarations `DecF(c)`, `DecP(c)`, and `Synt(c)`, respectively. The solve rules (SolR) and (SolNR) were integrated into a unique rule `Solve(c)`

To automatize 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.

E.g., consider the type of the function `DecF(c)`.

- Its input type is defined as `(match_DecF_conf?)`.
- Its output type denotes those valid configurations cp, with the required properties to automatize proofs.

Antiunification Algorith Specification

The `Antiunify` algorithm is defined in PVS as a recursive function of type `[(validConfiguration?) -> (validConfiguration?)]`.

By restricting the types for the functions specifying the inference rules (DecF), (DecP), (Solve), and (Synt), PVS automatically proves `Antiunify`'s termination and that every output of the `Antiunify` algorithm fulfills the `validConfiguration?` predicate.

Anti-unification Algorithm Verification

Three auxiliary lemmas about invariants and configuration preservation are highlighted.

1. `antiunify_sub_preserves_terms` states that if a `t`$\in$ range(`c'subs`) and vars(`t`)$\cap$labels(`c'unsolved`) $= \emptyset$ then `(Antiunify(c)'subs)(t) = (c'subs)(t)`.
   Applied twice to (2) and once to (SolveR).
2. `antiunify_dom_sub_preserves_vars_unsolved` states that domain(`Antiunify(c)'subs`)$\cap$ vars(`c'unsolved`) $= \emptyset$.
   Applied once to (Synt).
3. `antiunify_solved_labels_preserve_vars_unsolved` states that labels(`Antiunify(c)'solved`) $\cap$ vars(`c'unsolved`) $= \emptyset$.
   Applied twice to (Synt).

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

Anti-unification Verification

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

Surprising the resulting higher complexity of the formal analysis of the "simpler" analytical cases of the rules (Syntactic) and (SolveR).

Quantitative Data

Table 1: Formalization 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

Anti-unification modulo types

Theory	Anti-unification type	References
Syntactic	1	[Plo70, Rey70]
A	ω	[AEEM14]
C	ω	[AEEM14]
† (U)^{1}	ω	[CK20c]
(U)$^{\geq 2}$	0	[CK20c]
‡ $\mathfrak{a}$	∞	[ACBK24]
$\mathfrak{a}$(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

- 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 have been shown to be challenging from theoretical and practical perspectives [ACBK24].

Thank you for your attention!

Vielen Dank für Ihre Aufmerksamkeit!

References i

- 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.

References ii

- 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.

References iii

- 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.

References iv

- 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

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.