Unranked Nominal Unification*

Besik Dundua[1,3], Temur Kutsia[2], and Mikheil Rukhaia[3]

[1] International Black Sea University, Tbilisi, Georgia
[2] RISC, Johannes Kepler University Linz, Austria
[3] Institute of Applied Mathematics, Tbilisi State University, Georgia

Solving equations between logic terms is a fundamental problem with many important applications in mathematics, computer science, and artificial intelligence. It is needed to perform an inference step in reasoning systems and logic programming, to match a pattern to an expression in rule-based and functional programming, to extract information from a document, to infer types in programming languages, to compute critical pairs while completing a rewrite system, to resolve ellipsis in natural language processing, etc. Unification and matching are well-known techniques used in these tasks.

Unification (as well as matching) is a quite well-studied topic for the case when the equality between function symbols is precisely defined. This is the standard setting. There is quite some number of unification algorithms whose complexities range from exponential [22] to linear [20]. Besides, many extensions and generalizations have been proposed. Those relevant to our interests are equational unification (more precisely, associative unification with unit element) (see, e.g., [3]), word unification [5, 10, 19], and sequence unification [14, 16, 17]. There are some good surveys on unification [4, 6, 11, 12].

Nominal logic [7, 21] extends first-order logic with primitives for renaming via name-swapping, for freshness of names, and for name-binding. Such kind of constructs are important in meta-programming and meta-deduction. Nominal logic provides a simple formalism for reasoning about abstract syntax modulo α-equivalence. A nominal term $a.t$ is an example of abstraction, binding every occurrence of atom a in t. Term equality ($t \approx t'$) in nominal language is considered modulo renaming of bound variables (atoms), i.e., it is α-equivalence, formalized inside the language itself. The α-equivalence is a meta-relation in first-order syntax, but it is formulated on object level in nominal languages. For such formulations it is important to explicitly define which atom can be considered as a new atom for a given term. This relation ($a\#t$), called freshness relation, is also formulated on the object level in nominal languages.

Solving equations between nominal terms needs a special unification algorithm [24], which is first-order, but can be also seen from the higher-order perspective via mapping from/to higher-order pattern unification [18]. The standard nominal language contains fixed-arity symbols and one kind of variable, corresponding to individual variables from first-order syntax. In nominal languages, a unification problem, e.g. $a.x \approx^? b.y$, is solved by a pair $\langle \{b\#x\}, \{y \mapsto (a\,b) \cdot x\} \rangle$. The first component of the solution, the freshness constraint $b\#x$, requires that b should not occur free in every possible instantiation of x. The second component, the substitution, tells us that the solution must replace the variable y with the term $(a\,b) \cdot x$. The latter means that atoms a and b are swapped in every possible instantiation of x.

As we mentioned above, the constructs provided by nominal logic are important for meta-deduction. However, this formalism, as well as many representation formats for formalized mathematics typically do not provide a structural analog for ellipses $(\dots)$ which are commonly used in mathematical texts [8, 9]. In the literature, the latter problem has been addressed by permitting unranked (also known as variadic, flexary, or flexible arity) symbols in the language, introducing sequences in the meta-level, and extending the language with sequence variables, see, e.g., [8, 13, 15].

In this talk we present a combination of these two approaches, extending nominal languages by unranked symbols and studying the fundamental computational mechanism for them: unification. However, unlike the above mentioned unranked languages, where sequences are introduced in the meta-level, nom-

* Supported by Austrian Science Fund, project 28789-N32, and by the Shota Rustaveli National Science Foundation, grant FR17_439.

inal syntax allows us to introduce their analogs in the object level. This is done by generalizing already existing syntactic constructs, pairs, to arbitrary tuples. They should be flat, which is achieved by imposing a special α-equivalence rule for them.

Term pairs, which are a part of nominal syntax in some papers (e.g., [1, 24]) have been extended to term tuples in [2], but our approach differs in that we additionally introduce variables that can be instantiated by tuples (*tuple variables*), and the mentioned notion of flatness.

In our signature we have pairwise disjoint sets of atoms $(a, b, \ldots)$, function symbols $(f, g, \ldots)$, individual variables $(x, y, \ldots)$, tuple variables $(X, Y, \ldots)$, and the tuple constructor $\langle\rangle$. *Unranked nominal terms* $(t, s, \ldots)$ are defined by the grammar:

$$t ::= a \mid a.t \mid \pi \cdot x \mid \pi \cdot X \mid f\langle t_1, \ldots, t_n\rangle \mid \langle t_1, \ldots, t_n\rangle,$$

where π is a *permutation*: a finite (possibly empty) sequence of swappings, which are pairs of atoms $(a\, b)$. We write id for the identity (empty) permutation and $\pi_1 \circ \pi_2$ for concatenating two permutations.

Permutation action on terms is defined as follows:

- $id \cdot a = a$ and $((a_1\, a_2) \circ \pi) \cdot a = \begin{cases} a_1, & \text{if } \pi \cdot a = a_2, \\ a_2, & \text{if } \pi \cdot a = a_1, \\ \pi \cdot a, & \text{otherwise.} \end{cases}$
- $\pi \cdot (a.t) = (\pi \cdot a).(\pi \cdot t)$.
- $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$ and $\pi \cdot (\pi' \cdot X) = (\pi \circ \pi') \cdot X$.
- $\pi \cdot (f\langle t_1, \ldots, t_n\rangle) = f(\pi \cdot \langle t_1, \ldots, t_n\rangle)$ and $\pi \cdot \langle t_1, \ldots, t_n\rangle = \langle \pi \cdot t_1, \ldots, \pi \cdot t_n\rangle$.

The terms $\pi \cdot x$ and $\pi \cdot X$ are called suspensions. We skip π if $\pi = id$.

A freshness environment (denoted by ∇) is a list of freshness constraints $a\#x$ and $a\#X$, meaning that the instantiations of x and X cannot contain free occurrences of a. The flatness property of tuples is formalized by the axiom (where $n \geq 0, k \geq 0, m \geq n$)

$$\frac{}{\nabla \vdash \langle t_1, \ldots, t_n, \langle t'_1, \ldots, t'_k\rangle, t_{n+1}, \ldots, t_m\rangle \approx \langle t_1, \ldots, t_n, t'_1, \ldots, t'_k, t_{n+1}, \ldots, t_m\rangle}$$

Substitution is a mapping from individual variables to terms (which are neither tuples not tuple variables) and from tuple variables to tuples such that all but finitely many individual variables are mapped to themselves, and all but finitely many tuple variables are mapped to singleton tuples consisting of that variable only (i.e., mapping X to $\langle X\rangle$). They are usually written as finite sets, e.g., $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n, X_1 \mapsto \langle t_{11}, \ldots, t_{1n_1}\rangle, \ldots, X_m \mapsto \langle t_{m1}, \ldots, t_{mn_m}\rangle]$. *Application* of a substitution σ to a term t is defined as follows:

- $a\sigma = a,\ (a.t)\sigma = a.t\sigma,\ (f\langle t_1, \ldots, t_n\rangle)\sigma = f\langle t_1, \ldots, t_n\rangle\sigma$.
- $\langle t_1, \ldots, t_n\rangle\sigma = \langle t_1\sigma, \ldots, t_n\sigma\rangle$, where nested tuples are flattened.
- $(\pi \cdot x)\sigma = \pi \cdot \sigma(x),\ \ (\pi \cdot X)\sigma = \pi \cdot \sigma(X)$, where π acts on $\sigma(x)$ and $\sigma(X)$ as permutation action.

For instance, we have

$$a.f\langle (a\, b) \cdot x,\ (a\, b) \cdot X,\ (a\, b) \cdot Y\rangle[x \mapsto f\langle b.b\rangle,\ X \mapsto \langle a, f\langle c.c, (a\, c) \cdot Z\rangle\rangle,\ Y \mapsto \langle\rangle] =$$
$$a.f\langle f\langle a.a\rangle,\ b,\ f\langle c.c, (a\, b)(a\, c) \cdot Z\rangle\rangle.$$

An unranked nominal unification problem P is a finite set of equational $t \approx^? t'$ or freshness problems $a\#^? t$. Tuples occurring in the unification problem are always flattened. A solution for P is a pair (∇, σ) such that for all problems in P we have $\nabla \vdash \sigma(t) \approx \sigma(t')$ and $\nabla \vdash a\#\sigma(t)$.

To describe the unification algorithm we use so called labeled transformation of unification problems: $P \stackrel{\sigma}{\Longrightarrow} P'$ and $P \stackrel{\nabla}{\Longrightarrow} P'$ (due to space limitation, we cannot list the rules here). The algorithm is divided into two phases: first apply as many $\overset{\sigma}{\Longrightarrow}$ transformations as possible. It might cause branching due to tuple variables. On some branches, there might be no equational problems left. We expand them by $\overset{\nabla}{\Longrightarrow}$ transformations as long as possible. If we do not end up with the empty problem, then halt with failure, otherwise from the sequence of transformations $P \overset{\sigma_1}{\Longrightarrow} \cdots \overset{\sigma_n}{\Longrightarrow} P' \overset{\nabla_1}{\Longrightarrow} \cdots \overset{\nabla_m}{\Longrightarrow} \emptyset$ construct the solution $(\nabla_1 \cup \cdots \cup \nabla_m, \sigma_n \circ \cdots \circ \sigma_1)$. Some branches might directly lead to failure after application of $\overset{\sigma}{\Longrightarrow}$ rules. Some branches might cause more and more branching, leading to infinite set of solutions. Employing some fair strategy of search tree development, we can have a complete method to enumerate them.

Example 1. We give examples of some unification problems and their solutions:

- Problem: $f\langle a.\langle X, x, Y\rangle\rangle \approx^? f\langle b.\langle f\langle X\rangle, x, b, c\rangle\rangle$. Solution: $\langle\emptyset, \{X \mapsto \langle\rangle, x \mapsto f\langle\rangle, Y \mapsto \langle f\langle\rangle, a, c\rangle\}\rangle$.
- Problem: $a.b.f\langle X, b\rangle \approx^? b.a.f\langle a, X\rangle$. Solution: $\langle\emptyset, \{X \mapsto \langle\rangle\}\rangle$.
- Problem: $f\langle X, a\rangle \approx^? f\langle a, Y\rangle$. Solutions: $\langle\emptyset, \{X \mapsto \langle\rangle, Y \mapsto \langle\rangle\}\rangle$ and $\langle\emptyset, \{X \mapsto \langle a, Z\rangle, Y \mapsto \langle Z, a\rangle\}\rangle$. If instead of Y we had X, then there would be infinitely many solutions: $\langle\emptyset, \{X \mapsto \langle\rangle\}\rangle$, $\langle\emptyset, \{X \mapsto \langle a\rangle\}\rangle$, $\langle\emptyset, \{X \mapsto \langle a, a\rangle\}\rangle, \ldots$.
- Problem: $a.f\langle X, a\rangle \approx^? b.f\langle b, X\rangle$. Solution: $\langle\emptyset, \{X \mapsto \langle\rangle, Y \mapsto \langle\rangle\}\rangle$.
- Problem: $a.f\langle X, a\rangle \approx^? b.f\langle b, Y\rangle$. Solutions: $\langle\emptyset, \{X \mapsto \langle\rangle, Y \mapsto \langle\rangle\}\rangle$ and $\langle\{b\#Z\}, \{X \mapsto \langle a, Z\rangle, Y \mapsto \langle(a\, b) \cdot Z, b\rangle\}\rangle$.
- Problem: $a.f\langle X, c\rangle \approx^? b.f\langle c, Y\rangle$. Solutions: $\langle\emptyset, \{X \mapsto \langle\rangle, Y \mapsto \langle\rangle\}\rangle$ and $\langle\{b\#Z\}, \{X \mapsto \langle c, Z\rangle, Y \mapsto \langle(a\, b) \cdot Z, c\rangle\}\rangle$.

References

1. M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, and D. Nantes-Sobrinho. A formalisation of nominal α-equivalence with A and AC function symbols. *Electr. Notes Theor. Comput. Sci.*, 332:21–38, 2017.
2. M. Ayala-Rincón, M. Fernández, and D. Nantes-Sobrinho. Fixed-point constraints for nominal equational unification. In H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
3. F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
4. F. Baader and W. Snyder. Unification theory. In Robinson and Voronkov [23], pages 445–532.
5. V. Diekert. Makanin's algorithm. *Algebraic combinatorics on words*, 90:387–442, 2002.
6. G. Dowek. Higher-order unification and matching. In Robinson and Voronkov [23], pages 1009–1062.
7. M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.
8. F. Horozal, F. Rabe, and M. Kohlhase. Flexary operators for formalized mathematics. In S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2014.
9. F. F. Horozal. *A Framework for Defining Declarative Languages*. PhD thesis, Jacobs University, 2014.
10. J. Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, 1990.
11. J. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J. Lassez and G. D. Plotkin, editors, *Computational Logic - Essays in Honor of Alan Robinson*, pages 257–321. The MIT Press, 1991.
12. K. Knight. Unification: A multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124, 1989.
13. T. Kutsia. Solving equations involving sequence variables and sequence functions. In B. Buchberger and J. A. Campbell, editors, *AISC 2004, Proceedings*, volume 3249 of *Lecture Notes in Computer Science*, pages 157–170. Springer, 2004.
14. T. Kutsia. Solving equations with sequence variables and sequence functions. *J. Symb. Comput.*, 42(3):352–388, 2007.
15. T. Kutsia and B. Buchberger. Predicate logic with sequence variables and sequence function symbols. In A. Asperti, G. Bancerek, and A. Trybulec, editors, *MKM 2004, Proceedings*, volume 3119 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 2004.
16. T. Kutsia, J. Levy, and M. Villaret. Sequence unification through currying. In F. Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007*, volume 4533 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2007.
17. T. Kutsia, J. Levy, and M. Villaret. On the relation between context and sequence unification. *J. Symb. Comput.*, 45(1):74–95, 2010.
18. J. Levy and M. Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10:1–10:31, 2012.
19. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 145(2):147–236, 1977.
20. M. Paterson and M. N. Wegman. Linear unification. In A. K. Chandra, D. Wotschke, E. P. Friedman, and M. A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, pages 181–186. ACM, 1976.
21. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Inf. Comput.*, 186(2):165–193, 2003.
22. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
23. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
24. C. Urban, A. M. Pitts, and M. Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.