Rank-2 Intersection Type Inference Revisited

Pedro Ângelo
pedro.angelo@fc.up.pt
LIACC, Departamento de Ciência de Computadores,
Faculdade de Ciências, Universidade do Porto,
rua do Campo Alegre s/n, 4169 - 007
Porto, Portugal

Mário Florido
amflorid@fc.up.pt
LIACC, Departamento de Ciência de Computadores,
Faculdade de Ciências, Universidade do Porto,
rua do Campo Alegre s/n, 4169 - 007
Porto, Portugal

KEYWORDS

intersection types, E-unification, type inference

ACM Reference Format:
Pedro Ângelo and Mário Florido. 2023. Rank-2 Intersection Type Inference Revisited. In *Proceedings of Workshop on the Implementation of Type Systems (WITS '23).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

There has always been a strong link between type inference and unification. Examples go from the seminal works on principal types for combinatory logic [16, 21] and for the ML programming language [17, 28], relying on Robinson unification [31], to general frameworks such as HM(X) [29]. Here we propose a general framework for type inference for a decidable fragment of intersection types, relying on set, multiset and sequence unification, dealing with different algebraic properties of the intersection operator.

Intersection types [8, 13–15] allow the study of normalization properties of λ-terms under β-reduction. These systems extend the Curry type assignment system, allowing multiple types to be assigned to the same λ-term. This is achieved by the introduction of the intersection type constructor $\cap$. Restricting intersection types to rank-2, i.e. no $\cap$ occurs to the left of two or more $\rightarrow$, is a sensible decision: systems retain an acceptable level of expressiveness while also making type inference decidable and simpler.

Several algebraic properties can be conferred to the $\cap$ constructor: Associativity (A), i.e. $\tau_1 \cap (\tau_2 \cap \tau_3) \equiv (\tau_1 \cap \tau_2) \cap \tau_3$, Commutativity (C), i.e. $\tau \cap \sigma \equiv \sigma \cap \tau$, Idempotence (I), i.e. $\tau \cap \tau \equiv \tau$, or any combination of the three. Bestowing or withholding some of these properties gives rise to different categories of intersection types. In the original systems [8, 13–15], $\cap$ is treated as an ACI constructor. Non-idempotent intersection types [10–12, 20], also called multi types or quantitative types, with applications to cost models for λ-calculus [1, 3, 9, 24, 25], consider $\cap$ as an AC constructor. Finally, considering $\cap$ solely as an A constructor gives rise to ordered systems [2], which ensure an ordering of the occurrences of variables in the body.

Each category of intersection types behaves as a specific data structure: ACI intersections behave as sets, AC as multisets and A as lists, or strings. In fact, these algebraic properties, stated as axioms, give rise to equational theories. Unification of terms, taking into account these equational theories, is an established research field under the name of E-unification. ACI-unification, also called set unification, has been studied in [18, 19, 26], and shown sound and complete [19]. AC-unification, also called multiset unification, is usually reduced to solving diophantine equations and has been studied in [26], also shown sound and complete. A-unification, also called string (or word) unification, has been studied in [22, 23, 27, 30, 32] and shown decidable and infinitary [22, 23], i.e. it is possible to obtain the minimal and complete set of unifiers, even though these may be infinite. Reviews of these topics can be found in [5–7, 33].

The previous statements suggest that E-unification algorithms can be incorporated into type inference algorithms for intersection types, specifically to solve constraints between intersections. The correspondence between algebraic properties and equational theories guides this incorporation: set unification can be incorporated in type inference algorithms for ACI intersections, multiset unification for AC intersections, and string unification for A intersections. The results stated previously strengthen these claims, by suggesting type inference will naturally inherit soundness and completeness properties of E-unification algorithms.

In fact, previous work of ours [4] has confirmed this for ACI intersections. For example, for the term $(\lambda x.xx)\ (\lambda y.y)$, the following constraints are generated: $\{\alpha_1 \doteq \alpha_2 \rightarrow \alpha_3, \alpha_1 \wedge \alpha_2 \doteq \alpha_5 \rightarrow \alpha_5 \wedge \alpha_6 \rightarrow \alpha_6\}$. Note the second constraint, constraining intersection types, can be reduced to solving a set unification problem. Applying the set unification algorithm from [19], as suggested in [4], we obtain the following solution: $\{\alpha_5 \doteq \alpha_6 \rightarrow \alpha_6, \alpha_2 \doteq \alpha_5, \alpha_3 \mapsto \alpha_5, \alpha_1 \doteq \alpha_2 \rightarrow \alpha_3\}$. Hence, we uncover the principal type: $\emptyset \vdash_\wedge (\lambda x\ .\ x\ x)\ (\lambda y\ .\ y) : \alpha_6 \rightarrow \alpha_6$.

Our goal is then to incorporate E-unification algorithms into type inference algorithms for intersection types, according to the different algebraic properties of these. As far as we know, this sort of approach has not been done for both AC and A intersections. Afterwards, our goal is to unite these approaches in a single (rank-2 intersection) type inference algorithm, parameterized by the algebraic properties of intersection types. According to the three different categories, ACI, AC or A, the type inference algorithm calls the corresponding E-unification algorithm in the type constraint solving phase.

This work was partially financially supported by the portuguese Fundação para a Ciência e a Tecnologia, under the PhD grant number SFRH/BD/145183/2019 and by Base Funding - UIDB/00027/2020 of the Artificial Intelligence and Computer Science Laboratory – LIACC - funded by national funds through the FCT/MCTES (PIDDAC).

WITS '23, August 28, 2023, Braga, Portugal

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

REFERENCES

[1] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2018. Tight Typings and Split Bounds. *Proc. ACM Program. Lang.* 2, ICFP, Article 94 (jul 2018), 30 pages. https://doi.org/10.1145/3236789

[2] Sandra Alves and Mário Florido. 2022. Structural Rules and Algebraic Properties of Intersection Types. In *Theoretical Aspects of Computing – ICTAC 2022*, Helmut Seidl, Zhiming Liu, and Corina S. Pasareanu (Eds.). Springer International Publishing, Cham, 60–77. https://doi.org/10.1007/978-3-031-17715-6_6

[3] Sandra Alves, Delia Kesner, and Daniel Ventura. 2020. A Quantitative Understanding of Pattern Matching. In *25th International Conference on Types for Proofs and Programs (TYPES 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 175)*, Marc Bezem and Assia Mahboubi (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 3:1–3:36. https://doi.org/10.4230/LIPIcs.TYPES.2019.3

[4] Pedro Ângelo and Mário Florido. 2022. Type Inference for Rank-2 Intersection Types Using Set Unification. In *Theoretical Aspects of Computing – ICTAC 2022*, Helmut Seidl, Zhiming Liu, and Corina S. Pasareanu (Eds.). Springer International Publishing, Cham, 462–480. https://doi.org/10.1007/978-3-031-17715-6_29

[5] Franz Baader and Tobias Nipkow. 1998. *Term Rewriting and All That.* Cambridge University Press. https://doi.org/10.1017/CBO9781139172752

[6] Franz Baader and Jörg H. Siekmann. 1994. *Unification Theory.* Oxford University Press, Inc., USA, 41–125.

[7] Franz Baader, Wayne Snyder, Paliath Narendran, Manfred Schmidt-Schauss, and Klaus Schulz. 2001. Chapter 8 - Unification Theory. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). North-Holland, Amsterdam, 445–533. https://doi.org/10.1016/B978-044450813-3/50010-2

[8] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. 1983. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic* 48, 4 (1983), 931–940. https://doi.org/10.2307/2273659

[9] Alexis Bernadet and Stéphane Lengrand. 2011. Complexity of Strongly Normalising λ-Terms via Non-idempotent Intersection Types. In *Foundations of Software Science and Computational Structures*, Martin Hofmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 88–107. https://doi.org/10.1007/978-3-642-19805-2_7

[10] Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. 2018. Inhabitation for Non-idempotent Intersection Types. *Logical Methods in Computer Science* Volume 14, Issue 3 (Aug. 2018). https://doi.org/10.23638/LMCS-14(3:7)2018

[11] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. 2017. Non-idempotent intersection types for the Lambda-Calculus. *Logic Journal of the IGPL* 25, 4 (07 2017), 431–464. https://doi.org/10.1093/jigpal/jzx018 arXiv:https://academic.oup.com/jigpal/article-pdf/25/4/431/19368065/jzx018.pdf

[12] Daniel de Carvalho. 2007. *Sémantiques de la logique linéaire et temps de calcul.* Ph. D. Dissertation. http://www.theses.fr/2007AIX22066 Thèse de doctorat dirigée par Ehrhard, Thomas Mathématiques discrètes et fondements de l'informatique Aix-Marseille 2 2007.

[13] Mario Coppo and Mariangiola Dezani-Ciancaglini. 1980. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic* 21, 4 (10 1980), 685–693. https://doi.org/10.1305/ndjfl/1093883253

[14] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. 1980. Principal Type Schemes and Lambda-calculus Semantics. In *To H.B.Curry: Essays on Combinatory Logic, Lambda-calculus and Formalism.* Academic Press, 535–560. http://www.di.unito.it/~dezani/papers/CDV80.pdf

[15] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. 1981. Functional Characters of Solvable Terms. *Mathematical Logic Quarterly* 27, 2-6 (1981), 45–58. https://doi.org/10.1002/malq.19810270205 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/malq.19810270205

[16] Haskell B. Curry. 1969. Modified basic functionality in combinatory logic. *Dialectica* 23, 2 (1969), 83–92.

[17] Luis Damas and Robin Milner. 1982. Principal Type-schemes for Functional Programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Albuquerque, New Mexico) *(POPL '82)*. ACM, New York, NY, USA, 207–212. https://doi.org/10.1145/582153.582176

[18] Agostino Dovier, Eugenio Omodeo, Enrico Pontelli, and Gianfranco Rossi. 1996. A Language for Programming in Logic with Finite Sets. *J. Log. Program.* 28 (01 1996), 1–44.

[19] Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. 2006. Set unification. *Theory and Practice of Logic Programming* 6, 6 (2006), 645–701. https://doi.org/10.1017/S1471068406002730

[20] Philippa Gardner. 1994. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software*, Masami Hagiya and John C. Mitchell (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 555–574. https://doi.org/10.1007/3-540-57887-0_115

[21] R. Hindley. 1969. The Principal Type-Scheme of an Object in Combinatory Logic. *Trans. Amer. Math. Soc.* 146 (1969), 29–60.

[22] Michael Hoche, Jörg Siekmann, and Peter Szabo. 2008. String unification is essentially infinitary. https://doi.org/10.22028/D291-25201

[23] Joxan Jaffar. 1990. Minimal and Complete Word Unification. *J. ACM* 37, 1 (jan 1990), 47–85. https://doi.org/10.1145/78935.78938

[24] Delia Kesner and Daniel Ventura. 2014. Quantitative Types for the Linear Substitution Calculus. In *Theoretical Computer Science*, Josep Diaz, Ivan Lanese, and Davide Sangiorgi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 296–310. https://doi.org/10.1007/978-3-662-44602-7_23

[25] Delia Kesner and Pierre Vial. 2020. Non-idempotent types for classical calculi in natural deduction style. *Logical Methods in Computer Science* Volume 16, Issue 1 (Jan. 2020). https://doi.org/10.23638/LMCS-16(1:3)2020

[26] M. Livesey and Jörg Siekmann. 1976. Unification of sets and multisets. https://doi.org/10.22028/D291-36868

[27] G S Makanin. 1977. THE PROBLEM OF SOLVABILITY OF EQUATIONS IN A FREE SEMIGROUP. *Mathematics of the USSR-Sbornik* 32, 2 (feb 1977), 129. https://doi.org/10.1070/SM1977v032n02ABEH002376

[28] Robin Milner. 1978. A theory of type polymorphism in programming. *J. Comput. System Sci.* 17, 3 (1978), 348 – 375. https://doi.org/10.1016/0022-0000(78)90014-4

[29] Martin Odersky, Martin Sulzmann, and Martin Wehr. 1999. Type Inference with Constrained Types. *Theor. Pract. Object Syst.* 5, 1 (Jan. 1999), 35–55. https://doi.org/10.1002/(SICI)1096-9942(199901/03)5:1%3C35::AID-TAPO4%3E3.0.CO;2-4

[30] Wojciech Plandowski. 2004. Satisfiability of Word Equations with Constants is in PSPACE. *J. ACM* 51, 3 (may 2004), 483–496. https://doi.org/10.1145/990308.990312

[31] J. A. Robinson. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM* 12, 1 (Jan. 1965), 23–41. https://doi.org/10.1145/321250.321253

[32] Klaus U. Schulz. 1992. Makanin's algorithm for word equations-two improvements and a generalization. In *Word Equations and Related Topics*, K. U. Schulz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 85–150. https://doi.org/10.1007/3-540-55124-7_4

[33] Jörg H. Siekmann. 1989. Unification theory. *Journal of Symbolic Computation* 7, 3 (1989), 207–274. https://doi.org/10.1016/S0747-7171(89)80012-4 Unification: Part 1.