UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Bachelor thesis

Jacob A. Siegumfeldt, Laust K. Dengsøe

Optimizing Futhark's Type Checker

Advisor: Troels Henriksen

Handed in: June 10, 2025

Contents

1 Background

1.1 Constraint-Based Typing

There are different ways of solving the type inference problem in programming languages with a Hindley-Milner type system. One approach is to use *Algorithm W* [1] but this algorithm intertwines the process of generating constraints and solving them, leading to substitutions being applied (too) often and making it "hardly efficient". For this project, a better approach is to generate every constraint that can be inferred from a given expression and then solve these constraints afterwards.

Formally, we write a *type constraint* as

$$\langle \tau_1, \tau_2 \rangle$$

where τ_1 and τ_2 are both monomorphic types, while a *constraint set* C is a set of such equalities. We say that a *type substitution* σ *unifies* the constraint $\langle \tau_1, \tau_2 \rangle$ if $\sigma(\tau_1)$ is *syntactically* equal to $\sigma(\tau_2)$. Extending the notion of solvability to constraint sets, a type substitution σ solves a constraint set C if σ solves every type constraint in C. A substitution σ that solves a constraint set C is called a *solution* or *unifier* for C. As there might be multiple unifiers for a constraint set [2], we define $\mathcal{U}(C)$ to be the set of all unifiers for C. Furthermore, a type substitution ρ is called a *most general unifier* (MGU) of a set of type constraints C if $\rho \in \mathcal{U}(C)$ and for every $\sigma \in \mathcal{U}(C)$ there exists a type substitution σ' such that $\rho = \sigma' \circ \sigma$. In other words, ρ is an MGU if

1.2 Union-Find

Union-find is an imperative data structure that relies on disjoint sets to represent sets that are disjoint, meaning that two distinct sets cannot contain the same elements. It is useful for applications that require the grouping of multiple distinct elements into disjoint sets. The data structure supports three basic operations: creating a set, union of two sets, and finding the set that a given element is part of. Each set has a representative or root, which can sometimes just be any arbitrary element in the given set, however in some cases (as in ours) a particular element is picked based on some property or heuristic.

The operation of making a set simply creates a set given some element. This element is made the root of the newly created set. Given some element, the find operation returns a pointer to the root of the set that the element is part of. The union operation is the most "complex" operation, as it first finds the roots of the sets, that the two elements to be joined, say x and y, are part of. Then a new set, S, is created containing all the elements in both S_x and S_y yielding $S = S_x \bigcup S_y$. As mentioned previously, the root of S can be arbitrarily chosen among the elements or chosen by some property or heuristic.

Known implementations of the union-find data structure uses linked-lists or disjoint-set forests. The latter allows for a faster runtime by using the proper heuristics [3, p. 527], therefore we will only describe this way of implementing the data structure.

1.2.1 Union-Find for Unification

1.2.2 Union-Find in Haskell

1.3 An Algorithm for Unification using Union-Find

1.4 Futhark Quirks

References

[1] *Hindley–Milner type system*, en, Page Version ID: 1279817262, Mar. 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hindley%E2%80%93Milner_type_system&oldid=1279817262 (visited on 05/14/2025).

[2] J. Hoffmann, "Lectures 5–7: Type Inference," en, [Online]. Available: https://www.cs.cmu.edu/~janh/courses/ra19/assets/pdf/lect03.pdf.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, eng, Fourth edition. Cambridge, Massachusetts London: The MIT Press, 2022, isbn: 978-0-262-04630-5 978-0-262-36750-9.