Survey of Type Inference Algorithms for Statically Typed Languages

TU Delft

Saulius Jakovonis - s.jakovonis@student.tudelf.nl

Supervisors: Jesper Cockx, Bohdan Liesnikov

What?

Type inference in programming languages refers to automatic type detection based on surrounding context.

- Applies when there is a **known expression** with an **unknown type** which needs to be determined.
- Usually occurs at **compile time**

```
for (auto light : scene.lights)
  draw(light);
```

Example 1: using “auto” keyword in C++

```
ghci>  add x y z = (x + y) : z
ghci>  :type add
add :: Num a => a -> a -> [a] -> [a]
```

Example 2: inferring function types in Haskell

Why?

Type inference maintains **type-checking** even without requiring any explicit type annotations. As a result it

- Reduces the **verbosity** of a programming language - making the code faster to write.
- Reduces **redundant information**, making the code more concise and easier to read.
- Reduces the **cognitive effort** required to write programs, since the programmer has to worry less about what types to use.

.. while maintaining **type-safety** of statically typed languages

Type Inference Algorithms

A. Hindley-Milner Based (1978-…)

Algorithm W (Original):

- One of the first type inference algorithms, yet **popular** and **influential** to this day
- Originally for **ML**, but appears in many other languages (e.g. **OCaml**, **Haskell**, **F#**)
- Uses **Hindley-Milner** type system and Robinson’s **unification** algorithm
- Designed around **parametric polymorphism**

Pros:

1. **Simple** and **efficient**
2. Type annotations are **never needed**
3. Always produces **most general type** for **any** well-typed expression

Cons:

1. Does **not** allow for more complex features (i.e. Subtyping, Type Classes/Ad-Hoc polymorphism, First-Class Polymorphism…)
2. Known for poor **error message** locality [1]

Extensions of Algorithm W:

Subtyping:

MLsub (2017)

Simple-sub (2020)

Type Classes & GADTs:

OutsideIn(X) (2011)

First-Class Polymorphism:

ML_F (2003)

HM_F (2008)

FreezeML (2020)

Error Localization:

Algorithm M (1998)

SOLVE (2002)

“Practical Error Localization” (2015)

B. Bidirectional Type Checking (1999-…)

- More recent approach to type inference that has become **very popular for new languages** [1]
- Combines **type-checking** and **type inference** into one process [2]
- Allows for more complex language features to be supported by **requiring type annotations where needed** [3]
- In practice used by languages such as **Scala** to enable subtyping and **Haskell** to enable first-class polymorphism

Pros:

1. More **robust** & **flexible** for complex features
2. Better **error locality** [1] [2]

Cons:

1. Some **annotations** are usually needed [2]
2. Scope is restricted to **local expressions** [2]

(Some) Bidirectional Algorithms:

Subtyping:

Local Type Inference (1999)

Colored Local Type Inference (2001)

First-Class Polymorphism:

QuickLook (2020)

Goals

Main goal is to produce a **survey** of the existing algorithms for **type inference** for statically typed languages proposed in **literature**. Broken down into the following **subquestions**:

1. What are the common **issues** to implementing type inference?
2. What are the proposed **solutions** to these issues?
3. How do these solutions **compare**? Identify **advantages** and **limitations**.
4. How were these methods adopted in **practice**?

Method

- The information is sourced from existing literature with emphasis on peer reviewed research papers, but **official documentiation** and **reputable blog posts** also considered.
- Algorithms are compared based on the **evaluations** present in the **original research**, as well as issues identified by their **successors**.
- The identified algorithms are **categorized** and **compared** based on their techniques, limitations and advantages.

References

[1] J. Dunfield and N. Krishnaswami, “Bidirectional Typing,” *ACM Computing Surveys*, vol. 54, pp. 1–38, May 2021.
[2] B. C. Pierce and D. N. Turner, “Local type inference,” ACM Trans. Program. Lang. Syst., vol. 22, no. 1, pp. 1–44, Jan. 2000.
[3] S. Peyton Jones, D. Vytiniotis, S. Weirich, and M. Shields, “Practical Type Inference for Arbitrary-Rank Types,” J. Funct. Program., vol. 17, pp. 1–82, Jan. 2007