Improving Type Inference in the C# Language

Tomáš Husák
Author | husaktomas98@gmail.com

Mgr. Pavel Ježek Ph.D.
Supervisor

Charles University
Faculty of Mathematics and Physics

Motivation

C# is a strongly typed language utilizing type inference to save type annotations. However, the current type inference has several weaknesses.

The strength of the method type inference is lower than we can see in other programming languages. Concretely, it uses only arguments' types to deduce the method's type arguments. We can see the weakness in cases where type arguments only depend on target type or type parameter restrictions. The mentioned weakness is more noticeable because of the "*all or nothing*" principle, where the method type inference infers either all of the type arguments or nothing.

Another place where type arguments are often used is a generic object creation where there is no type inference at all.

The mentioned issues give us a reason to think about improving type inference. The following improvement is described as a proposal consisting of the language feature design and implemented in a fork of the official C# compiler, **Roslyn**.

Language Feature Design

The design has to follow the following goals to make it more likely to be accepted by the C# language committee.

- Not introducing a breaking change
- Not to slow down the compilation process significantly
- Intuitive and coherent syntax and semantics

These goals are achieved by **partial type inference**, introducing an *underscore* character, skipping inferrable type arguments in the argument list of an *invocation expression* and an *object creation expression*, and allowing the specification of just ambiguous ones.

```
M<_, object>(42, null);  // _ = int
void M<T1, T2>(T1 t1, T2 t2) { ... }
```

It also improves the type inference in the case of an object creation expression by leveraging type bounds obtained from the target and type parameter constraint clauses.

```
using System.Collections.Generic;
IList<int> temp = new List<_>(); // _ = int
```

Implementation

Implementation is done in the Roslyn fork accompanied by the set of tests verifying the functionality.

Related Work

An important part of this design is big exploration of type inference in existing similar languages and applicability on the C# language which has its own limitations. We used the **Hindley-Milner** type inference theory to observe these limitations. Together with already existing ideas on GitHub discussions, we merged them into one language feature proposal presented in this poster.

Results

The implementation passed basic Roslyn tests, which signalizes robustness and no breaking change.
The proposal was presented to the C# language committee, which gave positive feedback and wanted to continue with further discussions.
In the state of writing, there were already two language design meetings discussing the design in detail and possible alternatives.
There are also a few reactions to this proposal on GitHub from the C# community.

GitHub fork https://github.com/TomatorCZ/roslyn
Language change pull request https://github.com/dotnet/csharplang/pull/7582#pullrequestreview-2004342942