Global Type Inference for Featherweight Generic Java – Prototype Implementation (Artifact)

Andreas Stadelmeier
Duale Hochschule Baden-Württemberg Stuttgart, Campus Horb, Germany

Martin Plümicke
Duale Hochschule Baden-Württemberg Stuttgart, Campus Horb, Germany

Peter Thiemann
Institut für Informatik, Universität Freiburg, Germany

Abstract

We implemented a prototype of the type inference algorithm described in the paper "Global Type Inference for Featherweight Generic Java". Our type inference algorithm for Featherweight Generic Java (GFJ) is able to calculate the missing types in a Typeless Featherweight Generic Java (FGJ-GT) program. Inserting those types generates a valid GFJ program. We demonstrate this with a prototype implementation. The prototype is a web application which accepts GFJ-GT programs as input and shows the respective GFJ program after the type inference.

2012 ACM Subject Classification Software and its engineering → Language features

Keywords and phrases type inference, Java, subtyping, generics

Digital Object Identifier 10.4230/DARTS.8.2.18

Related Article Andreas Stadelmeier, Martin Plümicke, and Peter Thiemann, "Global Type Inference for Featherweight Generic Java", in 36th European Conference on Object-Oriented Programming (ECOOP 2022), LIPIcs, Vol. 222, pp. 28:1–28:27, 2022.
`https://doi.org/10.4230/LIPIcs.ECOOP.2022.28`

Related Conference 36th European Conference on Object-Oriented Programming (ECOOP 2022), June 6–10, 2022, Berlin, Germany

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2022 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

The artifact is a prototype implementation of the type inference algorithm presented in the corresponding paper. The idea of this prototype is to showcase our type inference algorithm for featherweight generic java. It can be used to get a feeling of what the type inference algorithm is capable of. All of the examples presented in the paper can also be verified with this implementation. Also see the inputs in appendix C.

Our prototype also demonstrates a way to implement the nondeterministic parts of the type inference algorithm.The part in question is the selection of the correct typing described in chapter 4. We did not implement this by adding backtracking, as supposed by the paper. Our prototype implementation resolves this by internally keeping all possible types for the same method when processing multiple classes (see appendix B).

2 Content

The artifact package includes:

- Scala source code as a sbt-project

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://github.com/JanUlrich/FeatherweightTypeInference`.

4 Tested platforms

The web application should run in any up-to-date web browser with javascript support. We succesfully tested it with Firefox and Google Chrome.

5 License

The artifact is available under MIT license.

6 MD5 sum of the artifact

321b5b2c4ba5318c25dc9dd1fbcab6f7

7 Size of the artifact

60.0 KiB

A Documentation

A.1 Running the web application

The web application can be started by opening `index.html` in a web browser. It is also possible to access the web application under the following link:

`https://janulrich.github.io/FeatherweightTypeInference/index.html`

A.2 Building

The project is a standard scala sbt project. If you are familiar with scala you can compile the project to javascript by running `fullLinkJS` in the sbt console. See the documentation on scala.js: `https://www.scala-js.org/doc/tutorial/basic/index.html`

Compile the project with `sbt fullLinkJS`. This will generate a javascript file in `./target/scala-2.13/fj-typeinference-opt/`, which is referenced by `index.html`.

A.3 Howto Use

Just enter a proper FGJ-TI program on the left side and see the type infered result on the right side. The implementation is currently in a prototype state. It only works if you supply a valid input. The input has to be according to the syntax given in figure 6 of our paper and satisfy the type rules given in figure 8-10. If a incorrect input is given the right side will either show no result at all or a short error message. It also is possible that the algorithm fails with an uncaught exception. This exception is only displayed in the error console of your browser and will not be shown in the web application.

Hints:

- Every class has to have an `extends` clause
  - `class Example extends Object ...` for example
- Every generic variable has to have an `extends` clause as well
  - `class Example<A extends Object> extends Object ...`
- The classes are inferred in the order given in the input
  - the first class can only access its own methods
  - the subsequent classes can also access the methods of their predecessors
- see Example inputs below

B Differences to the algorithm described in our paper

The algorithm described in our paper is nondeterministic. The function described in chapter 4 selects only one of the solutions from the Unify function. Our prototype implementation internally computes every possible solution. It refines the solutions by filtering out unnecessary and duplicate solutions and then will show every solution by overloading the input methods.

For example the input:

```
class Example<A extends Object> extends Object{
    A f;
    m(){ return new Example(this);}
}
```

will generate two solutions for the method `m`:

```
class Example<A extends Object> extends Object{
    A f;
    Example<Object> m() {
        return new Example(this);
    }
    Example<Example<A>> m() {
        return new Example(this);
    }
}
```

C Example inputs

```
class Function<B extends Object, A extends Object> extends Object{
A a;

A apply(B p){
    return this.a;
}
}

class Box <S extends Object> extends Object {
S val;
map(f) {
    return new Box( f.apply( this.val ) ) ;
}
}
```

```
class True extends Object{
}
class False extends Object{
}

class Nand1 extends Object{
False nand(True a, True b){ return new False(); }
}
class Nand2 extends Object{
True nand(False a, True b){ return new True(); }
}
class Nand3 extends Object{
True nand(True a, False b){ return new True(); }
}
class Nand4 extends Object{
True nand(False a, False b){ return new True(); }
}

class SATExample extends Object{
True f;

sat(v1, v2, v3, o1, o2){
    return o1.nand(v1, o2.nand(v2, v3));
}

forceSATtoTrue(v1, v2, v3, o1, o2){
    return new SATExample(this.sat(v1, v2, v3, o1, o2));
}
}
```

```
class Identity extends Object{
  id(a){
    return a;
  }
}
```

```
class List<A extends Object> extends Object{
  A head;
  List<A> tail;
  add( a){
    return new List(a, this);
  }
  get(){
    return this.head;
  }
}

class PrincipleType extends Object {
  function(a){
    return a.add(this).get();
  }
}
```