Omar Thabet (o.thabet@student.tudelft.nl)
Supervisors: Casper Bach Poulsen, Aron Zwaan
CSE 3000 2023/05

Phased Type Checker For Java

A Type Checker For a Subset of Java Built On Scope Graph Semantics

TU Delft

Introduction:

- Type checking plays a crucial role in programming, ensuring the correctness of programs.
- Scope graphs have emerged as a novel approach for representing program scoping and name bindings.
- Ministatix, is an implementation of Statix which allows for constructing type checkers based on the programming language specification.
- Ministatix was use to implement a type checker for a subset of the Java Programming language where the steps of type checking different components are interleaved and their order is abstracted.
- Another approach to type checking using scope graphs is to manually schedule the order of the type checking in phases.

Contribution:

This study makes the following contributions:

- Demonstration of the phased approach through the implementation of a type checker for a subset of Java.
- Discussion of the differences between the phased type checker and Ministatix.
- Evaluation of the testing methodology [3] and comparison of the supported Java features in the aforementioned type checkers.
- Comparison between the declarativity and extendability of the code in the two type checkers.

Research Question:

- How does the phased approach compare to the Ministatix implementation?
- How many phases are required for effective implementation?
- How easily can the program be extended to include additional Java features, and will this necessitate additional phases?

Scope Graphs:

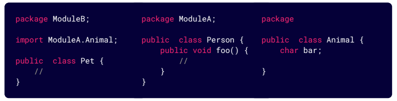
- The phased type checker makes use of the Phased Haskell library [2] to construct and query the scope graph.
- There are two types of Nodes as shown in figure 1:
  - Scope nodes are oval shaped and represent scopes.
  - Sink nodes are rectangular and represent declarations and contain data such as a scope reference in case of class declaration sinks or type in case of variable declaration sinks.
- The scope graph uses directed and labeled edges to represent the relationships between two nodes.

Type Checking In Three Phases:

1. Convert the Java Syntax an Abstract Syntax Tree represented by datatypes.
2. Type check the program in three phases:
   a. Explore all packages and classes in the program.
   b. For all classes, resolve Imports then type check class member declarations (Fields, Methods, Constructors) while ignoring initial field values and method bodies.
   c. For all class members, type check method bodies and field initial values

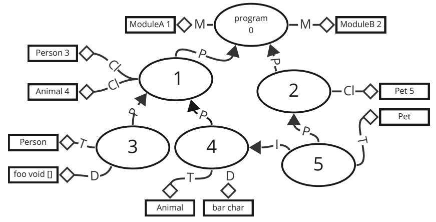
An Example Program and it's corresponding Scope Graph:

Fig 1



```
package ModuleB;            package ModuleA;            package

import ModuleA.Animal;      public  class Person {      public  class Animal {
                                public void foo() {         char bar;
public  class Pet {                 //
    //                          }                       }
}                           }
```



Monotonic queries:

Every type checker have to implicitly or explicitly ensure that variable name resolution results remain consistent as the program environment is extended:

- In scope graphs, a query is not a part of the graph but it's an algorithm that traverses the graph following a path with a given regular expression.
- In Ministatix, the ordering of queries is abstracted using language-independent critical edges.
- The phased type checker ensures stable query results by ordering the phases such that no further additions could effect the outcome of the query.

References:
[1] Rouvoet, A., Van Antwerpen, H., Poulsen, C. B., Krebbers, R., & Visser, E. (2020). Knowing when to ask: sound scheduling of name resolution in type checkers derived from declarative specifications. *Proceedings of the ACM on Programming Languages, 4*(OOPSLA), 1–28. https://doi.org/10.1145/3428248
[2] https://github.com/heft-lang/hmg
[3] https://github.com/OmarTheMostWanted/scope-graph-scheduling-bsc/blob/master/lang-java/tests/Main.hs

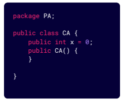
Discussion:

Java Features Support:

Both type checkers support a different subset of the Java features, the main differences are:

Phased TC supports:	Ministatix supports:
• Loops • If statements • Method invocation • Method overloading • Constructor overloading • Arrays	• Hierarchical package and class structures • Interfaces • Sub typing • Nested classes • Named and on demand imports

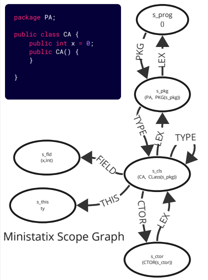
Scope Graph Structure and labels:



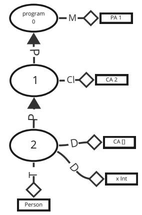
```
package PA;

public class CA {
    public int x = 0;
    public CA() {
    }

}
```



Ministatix Scope Graph



Phased Type Checker Scope Graph

Ministatix requires the inclusion of edges in two directions as the scope graph needs to be traversable in both directions.

The phased type checker utilizes a simpler scope graph structure, where traversal occurs in a single direction from lower scopes to higher scopes.

Label Purpose	Ministatix	Phased TC
Lexical Parent	Lex	P
Import	IMP-ST,IMP-OD	I
Package Declaration	PKG	M
Class Declaration	Type	Cl
Scope Type	THIS	T
Constructor Declaration	CTOR	D
Method Declaration	METHOD	D
Field Declaration	FIELD	D
Variable Declaration	VAR	D

Conclusion: Declarativity, easy of use and limitations:

Phased TC:

- Less Declarative due to the predefined order of execution.
- Easier to follow the flow of the program due the clearly defined phases.
- The addition of more Java features such as sub-typing requires additional phases and further modifications to the current phases.
- Has a single bug that is difficult to fix: The wrong error message is given when a class with the same name is imported.

Ministatix:

- More declarative as the order of execution is abstracted with the help of Critical edges [1].
- More difficult to debug due to interleaved process.

Future Work:

- Fix bugged error:
- Extend the supported feature set to match Ministatix.
- Implement parallel type checking.
- Early termination of queries for a more efficient implementation.

<- Code on Github