Information Hiding and Package Abstraction for Go

Bachelor Thesis Project Description

Quoc Truong Tommy Ho

Supervisors: Felix Wolf, Prof. Dr. Peter Müller
Department of Computer Science, ETH Zurich

Start: 1st November 2022
End: 1st May 2023

1 Introduction

Go programs are categorized into packages, and each of these packages consists of declarations of functions, types, variables, etc. Members of packages are made available to other packages by importing the package using the `import` keyword. The visibility of members is defined by writing the first letter in uppercase or lowercase, which indicates that the declaration is exported or not-exported respectively. An exported member is visible for other packages if the package that the member is declared in is imported, whereas non-exported members stay not visible and cause undefined errors if they are referenced.

Gobra is a verifier for Go, based on the Viper verification infrastructure. However, Gobra currently treats all members as always visible. For instance, Gobra verifies the code in Figure 1. The variable `answer` is written in lowercase, and is therefore not visible outside of package `bar`. But package `foo`, which imported package `bar`, uses the variable `answer` in the assertion. This example illustrates that Gobra does not check visibility.

The aim of this thesis is to extend Gobra to support Go's visibility feature and to improve Gobra's support for package abstractions.

2 Motivation

Go programs can consist of many different packages. Verifying and managing these large structures can cause many difficulties, and visibility can reduce this overhead:

- When we change the specifications inside some package `A` then all packages that import package `A` need to be re-verified. This can cause a chain reaction where all packages that depend on re-verified packages have to be re-verified themselves.
- The effect of changes inside packages also induces a maintenance problem. If packages are changed, then we do not want to change all the specifications that use private variables. For example in Figure 1, if we change the constant in package `bar` (e.g.

```
1       package bar
2       const answer int = 42
```

```
1       package foo
2       import bar
3       func main(){
4               assert bar.answer == 42
5       }
```

Figure 1: There are two packages: `foo` and `bar`. The program constant variable `answer` in `bar` is written in lowercase letters and is not exported by Go, but verifies without error in Gobra.

`answer` to `question`) then we need to change the specification in package `foo` too. However, `answer` is not visible in package `foo`, so there should not be a need to change it across different packages. By adding visibility to Gobra, we can reduce the need to maintain across different packages, when we modify private items in one package.

- Without visibility, a package `B` that imports package `A` imports all the definitions in package `A` to package `B`. As a result, the complexity of the verified program grows larger with the larger definitions in package `A`. This can have a substantial impact on the performance of the verification process. For example, we can avoid importing private fields of structs, thus reducing the complexity of the encoding of the struct and thus improving the verification performance.

However by adding visibility various complications arise and need to be handled, like the visibility of specifications, permissions, interfaces, etc. Some problems also need additional functionalities to improve upon the verification process.

3 Existing Works

To solve the various problems caused by the introduction of visibility, we take inspiration from the wory by Leavens and Müller [1]. This work describes a variety of rules and lemmas for visibility applied to the Java Modeling Language (JML). Since Java is a more standard object-oriented language with more complex visibility constructs (public, default, protected, private) than Go, not all the rules introduced by the work are applicable to Gobra.

4 Core Goals

- **Designing a language extension to support visibility in Gobra.** For this goal, we will develop an extension of Gobra's annotation language that enables Gobra to verify code with private members. To develop the language extension, we will investigate how existing work for the Java Modeling Language [1] can be applied to Gobra. Furthermore, as part of this goal, we will adapt Gobra's encoding of private state. The aim is that in the encoding, private state is abstracted away as much as possible to reduce the complexity of the generated Viper program.

- **Developing a language extension to specify package interfaces.** For this goal, we will design an annotation language that let's users of Gobra define how members of a package are exported. We will investigate relevant Go usage patterns and existing Gobra case studies to determine how package members should be exported by default and which other usage patterns are important on top of that. The aim of the annotation language is to set the right default, but to also enable the other identified important usage patterns. In case no other important usage patterns are identified, setting the right default without an additional annotation language is sufficient.

- **Implementation.** We will implement the language extensions chosen for the first two core goals. The objective is to have a well-documented and maintainable implementation that is merged into Gobra's main repository.

- **Evaluation.** For this goal, we will evaluate both expressiveness and performance of our implemented extensions.

  - For expressiveness, we will collect several relevant examples that illustrate the expressiveness of our introduced extensions. Furthermore, we will measure the annotation overhead introduced by our extensions.
  - For performance, we will evaluate how Gobra's verification runtime improves on examples using private state due to our improved encoding.

5 Extended Goals

- **Interfaces with private methods.** For this extension goal, we will extend the language designs of the first two core goals to also support public interfaces with private methods. In particular, the extension will support that in Gobra, public interfaces with private fields can be implemented outside the package defining the interface.
- **Supporting multiple package interfaces.** For this extension goal, we will extend the language design of the second core goal to support defining multiple package abstractions. The idea is that for an imported package, the client can specify which package abstraction of the ones defined in the imported package they want to use.
- **Improving the support for global variables.** For this extension goal, we will extend the language designs of the first two core goals to improve the support for global variables. In Gobra, global variables produce additional proof obligations. We will investigate how these obligations can be simplified for private global variables and adapt Gobra's encoding accordingly.

References

[1] G. T. Leavens and P. Müller, "Information hiding and visibility in interface specifications," in *International Conference on Software Engineering (ICSE)*, 2007.