Karlsruhe Institute of Technology

Institute for Program Structures
and Data Organization (IPD)

Gradual Verification and maybe something about implicit dynamic frames

Master's Thesis of

Johannes Bader

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer: Prof. Dr.-Ing. Gregor Snelting, Karlsruhe Institute of Technology - Karlsruhe, Germany

Advisors: Assoc. Prof. Jonathan Aldrich, Carnegie Mellon University - Pittsburgh, USA
Assoc. Prof. Éric Tanter, University of Chile - Santiago, Chile

Duration: 2016-05-10 – 2016-09-28

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text, and have followed the rules of the KIT for upholding good scientific practice.

Karlsruhe, 2016-09-??

.....

(**Johannes Bader**)

Abstract

Formal verification using Hoare logic is a powerful tool to prove properties of imperative computer programs.
However, in practice programmers often face situations ... rigid... not flexible... - incomplete information about parts of the program - laziness, forced to annotate everything - unable to express due to limited syntax - unable to prove something facing undecidability
To counteract these limitations we introduce the notion of gradual formulas with an unknown part "?".
The main contribution of this work is presenting a gradual verification logic that covers the full range between completely unannotated programs and fully annotated programs. We prove the soundness of this logic and ... Siek et al. (2015).

Acknowledgments

I wish to thank my advisors Jonathan Aldrich and Éric Tanter for offering me this topic and for their patient assistance throughout the past few months. In any situation and every way, their remarks and thoughts guided me in the right direction.

Also I am very grateful to all my family and friends who encouraged and supported me throughout my life.

Contents

1 Introduction

1.1 Motivation

1.1.1 As extension to unverified setting

Motivating example:

```
public static boolean hasLegalDriver(Car c)
{
    return c.driver.age >= 18;
}
```

Motivating example with potential leak:

```
public static boolean hasLegalDriver(Car c)
{
    allocateSomething();
    boolean result = c.driver.age >= 18;
    releaseSomething();
    return result;
}
```

Motivating example with argument validation:

```
public static boolean hasLegalDriver(Car c)
{
    if (c == null || c.driver == null)
        throw new IllegalArgumentException("something␣smart");

    // business logic (requires 'c.driver.age' to evaluate)
}
```

Motivating example with declarative approach (JML syntax):

```
//@ requires c != null && c.driver != null;
public static boolean hasLegalDriver(Car c)
{
    // business logic (requires 'c.driver.age' to evaluate)
}
```

There are two basic ways to turn this annotation into a guarantee:

Static Verification (run ESC/Java [1]) In the unlikely event that the verifier can prove the precondition at all call sites, our problem is solved. Otherwise, we have to enhance the call sites in order to convince the verifier. Choices:

- Add parameter validation, effectively duplicating the original runtime check across the program.

- Add further annotations, guiding the verifier towards a proof. This might not always work due to limitations of the verifier or decidability in general.

There are obvious limitations to this approach, static verification tends to be invasive. At least there is a performance benefit: The runtime check that was originally part of every call is now only necessary in places where verification would not succeed otherwise.

Runtime Assertion Checking (RAC, run JML4c [TODO: http://www.cs.utep.edu/cheon/download/jm
)] This approach basically converts the annotation back into a runtime check equivalent to our manual argument validation. It is therefore less invasive, not requiring further changes to the code, but also lacks the advantages of static verification.

2 Background and Related Work

2.1 Abstract Gradual Typing

2.2 Implicit Dynamic Frames

2.2.1 Self-Framing

2.3 Hoare Logic

...as formal setting

```
class Point
{
    int manhattenDistance(Point p)
        requires \phi_{pre};
        ensures  \phi_{post};
    {
        s1;
        s2;
        .
        .
        .
    }
}
```

$$\texttt{this} : \texttt{Point}, \texttt{p} : \texttt{Point}, \texttt{result} : \texttt{int} \vdash \{\phi_{pre}\} s1; s2; ... \{\phi_{post}\}$$

3 A Static Verification System

3.1 Syntax

3.2 Static Semantics

3.3 Dynamics Semantics

4 A Gradual Verification System

5 Evaluation

TODO

6 Conclusion

6.1 Evaluation

6.1.1 Limitations

6.2 Future Work

7 Appendix

Bibliography

[1] K Rustan M Leino, Greg Nelson, and James B Saxe. Esc/java user's manual. *ESC*, 2000:002, 2000.