Aalto University
School of Science
Master’s Programme in Computer, Communication and Information Sciences

Artem YUSHKOVSKIY

Automated Analysis of Weak Memory Models

Master’s Thesis
Espoo, ?.?.2018

Supervisor: Assoc. Prof. Keijo Heljanko
Instructor:

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Artem YUSHKOVSKIY		
Title: Automated Analysis of Weak Memory Models			
Date:	?.?.2018	**Pages:**	vi + 9
Professorship:		**Code:**	AS-116
Supervisor:	Assoc. Prof. Keijo Heljanko		
Instructor:			

In id fringilla velit. Maecenas sed ante sit amet nisi iaculis bibendum sed vel elit. Quisque eleifend lacus nec ipsum lobortis ornare. Nam lectus diam, facilisis eget porttitor ac, fringilla quis massa. Phasellus ac dolor sem, eget varius lacus. Sed sit amet ipsum eget arcu tristique aliquam. Integer aliquam velit sit amet odio tempus commodo. Quisque commodo lacus in leo sagittis vel dignissim quam vestibulum. Cras fringilla velit et diam dictum faucibus. Pellentesque at eros non mauris auctor euismod. Nullam convallis arcu vel lectus sollicitudin rutrum. Praesent consequat, nisl at pretium posuere, neque arcu dapibus lacus, ut sollicitudin elit velit ultricies libero. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae;

Nulla semper hendrerit molestie. Pellentesque blandit velit sit amet est vestibulum faucibus. Nullam massa turpis, venenatis non mollis fringilla, mattis et diam. Fusce molestie convallis elementum. Morbi nec lacus dapibus arcu mollis gravida. Aliquam erat volutpat. Nam vitae magna nunc. Nunc ut ipsum at massa porttitor vestibulum. Praesent diam lorem, ultrices nec vestibulum id, volutpat nec lacus.

Keywords:	Thesis template, master's thesis
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
??? ???

Tekijä:	Artem YUSHKOVSKIY		
Työn nimi: ?			
Päiväys:	?.?.2018	**Sivumäärä:**	vi + 9
Professuuri:	?	**Koodi:**	AS-116
Valvoja:	Assoc. Prof. Keijo Heljanko		
Ohjaaja:			

Cras tincidunt bibendum erat, vel tincidunt diam porttitor aliquam. Donec sit amet urna non felis placerat pharetra. Aenean ultrices facilisis nulla vitae semper. Nullam non libero quis dui fermentum aliquam id vel eros. Praesent elementum tortor quis sem congue iaculis sit amet eget nisl. Quisque erat tortor, condimentum eu volutpat et, blandit et augue. Phasellus erat turpis, pretium non feugiat id, posuere id velit. Vestibulum ut sapien felis, quis convallis dui.

In elementum est eu nulla hendrerit feugiat. In sodales diam vel lacus cursus tincidunt. Morbi nibh dui, imperdiet non vestibulum non, dignissim id risus. Sed sollicitudin neque lectus, porttitor sollicitudin elit. Nulla facilisi. Nullam in ante eu mi suscipit sollicitudin. Sed est velit, gravida facilisis varius eget, tempus sed urna. Aliquam erat volutpat. Nam semper condimentum nisi. Nullam scelerisque, metus nec sodales vulputate, purus augue venenatis urna, sit amet mattis turpis nisl ac metus. Mauris nec odio ut neque condimentum vulputate vel in turpis. Nulla facilisi. Nulla id tellus sapien, vitae blandit lorem.

Asiasanat:	Diplomityöpohja
Kieli:	Englanti

Acknowledgements

In id fringilla velit. Maecenas sed ante sit amet nisi iaculis bibendum sed vel elit. Quisque eleifend lacus nec ipsum lobortis ornare. Nam lectus diam, facilisis eget porttitor ac, fringilla quis massa. Phasellus ac dolor sem, eget varius lacus. Sed sit amet ipsum eget arcu tristique aliquam. Integer aliquam velit sit amet odio tempus commodo. Quisque commodo lacus in leo sagittis vel dignissim quam vestibulum. Cras fringilla velit et diam dictum faucibus. Pellentesque at eros non mauris auctor euismod. Nullam convallis arcu vel lectus sollicitudin rutrum. Praesent consequat, nisl at pretium posuere, neque arcu dapibus lacus, ut sollicitudin elit velit ultricies libero. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae;

Espoo, ?.?.2018

Artem YUSHKOVSKIY

Abbreviations

LI	Lorem Ipsum
ABC	Quisque et mi lacus, nec porta ante.
DEF	Proin pellentesque accumsan laoreet

Contents

Chapter 1

Introduction

Most modern computer systems contain large parts working concurrently. Though system parallelising can improve its performance drastically, it opens numerous of problems connected to correctness, robustness and reliability, which makes the concurrent program design one of the most difficult areas of programming [1].

Traditionally, studies related to concurrent programming concern mostly classical cases of implementing race-free and lock-free parallel programs, asynchronous data structures and synchronisation primitives of a programming language. Unfortunately, when it comes to analysis of the real-world concurrent programs, the algorithmic level of abstraction is not enough for guaranteeing their properties of correctness and reliability. The reasons of this fact lie in the code optimisations that both compiler and hardware perform in order to increase performance as much as possible. For instance, some processor architecture performs reordering of memory access instructions in a thread, while it should not be considered as a valid code transformation with respect to the logic of program (//TODO: example). It is said that this kind of processor exploits *relaxed*, or *weak memory model (WMM)*, that allow certain amount of memory operations reordering while preserving consistency. Weak memory models serve as set of guarantees made by designers of hardware (compilers, operation environment, etc.) for programmers on which behaviours of their concurrent code they should expect.

necessity of Weak Memory Models

- hardware wmms
- wmms of programming languages

- specific wmms like for kernel

wmm as a formal way to define guaranties that a hw, programming language, execution environment provide for programmers.

considering wmm as a set of allowed behaviours, the latter wmms are the supersets

wmm allows and disallows optimisations: partial sync of memory buffers, out-of-order execution (reordering), <more> => more behaviours that are unallowed in SC.

question possible to answer with wmm: which behaviours (in addition to SC) are allowed? which new states are allowed? Consequently, correctness, absence of data races, deadlocks or portability issues, etc.

existing tools: herd, diy7 – exhaustive search approach for exploring the state space.

another approach: using smt solver, e.g. for answering questions like

base paper: aims to investigate portability of small programs written in a C-like pseudocode and provides the proof-of-concept tool PORTHOS [link]. As input, it takes a program and two memory models in CAT language. Then it encodes programs and memory models into an smt formula and tries to solve it via z3. Current thesis aims to extend this tool functionality to process the real C code, therefore it proposes different modula program architecture and multiple optimisations.

1.1 Thesis structure

...

Chapter 2

Weak Memory Models

briefly what it is, as in Intro

Some examples of what wmms allow to do

Lamport's sequentially consistent memory model – global order "a global order, even if they are actually executed in different threads running in parallel in different processors. In this model, each read from some memory location is guaranteed to see a value written by the last write to this location." ([c11 by natasha]). Produces same result as sequential program.

2.1 The CAT language

the CAT language [ref to Jade's paper] (hard part: to decipher the Alglave's paper).

the event representation

2.2 Examples of WMM

briefly known hw memory models: X86-TSO, Alpha, POWER, – ref to Jade; language memory models: Java, C++; library-level kernel memory model, ref to github with tests

Relationship between different models `http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_506_Spring_2013/10c_ks`

Chapter 3

Implementation

This chapter describes the architecture of the tool `mousquitaires` ...
language: java

3.1 Program Requirements

- stability (tests)
- scalability (new features of language, new models, new tasks for a program)
- transparency
- efficiency

3.2 Program Components

Big view

3.3 C11 to YTree parser

below: mostly mock text.

- The language-dependent syntax tree: - for now it's the C subset language which I called 'Cmin'; as a base, I used the C11 grammar from ANTLR github repository, then I simplified it a lot, cutting off many unnecessary C syntax features and making it more convenient for parsing. When developing the Cmin language, I kept in mind C elements that are necessary for processing the linux kernel code, though for now not the whole grammar element described in file 'Cmin.g4' are being implemented; - later I am going to add the litmus grammar as well; - in future, it will be not a problem to add any new C-like language;

- The language-independent abstract syntax tree (aliased 'Ytree', where 'Y' resembles branching of the tree): - all tree nodes in my code are prefixed with 'Y', see tentative (yet almost complete) class hierarchy in picture 'YEntity.png'; - this AST contains very basic language elements according to the C execution model (statements and expressions); - converting the language-dependent syntax tree to the language-independent syntax tree is performed by Visitor pattern (e.g., for Cmin->Ytree conversion is made by 'CminToYtreeConverterVisitor') - minor changes are performed by converting to ytree representation: desugaring the target code, etc.

3.4 YTree to XGraph event converter

- Then, the AST is being interpreted and converted to event-based representation (aliased 'Xrepr' for eXecution representation): - more low-level code representation (or high-level assembly); - I try to keep this representation close to the one you described in your papers: basic load & store events, branching events, fence events; - this representation is being implementing these days, I've just started doing it (see current class hierarchy in the picture 'XEntity.png');

- After we acquired the event-based representation, we can perform some modifications/simplifications/optimisations on it (separately, allowing user to manage them): - converting to SSA form as one of necessary steps before encoding; - (more? – I'm not thinking about it yet);

3.4.1 Loop unrolling

The original program encoded into the `XGraph` represents a *flow graph*, a connected cyclic directed graph with single source node `[ENTRY]` (usually for convenience all leaves are connected to the sink node `[EXIT]`). The cycles are caused by low-level jump instructions, obtained from non-linear high-level control-flow statements (such as `while`, `do-while`, `for`, etc.). However, the cyclic flow graph cannot be encoded into SMT formula since ... //TODO:REFERENCE.

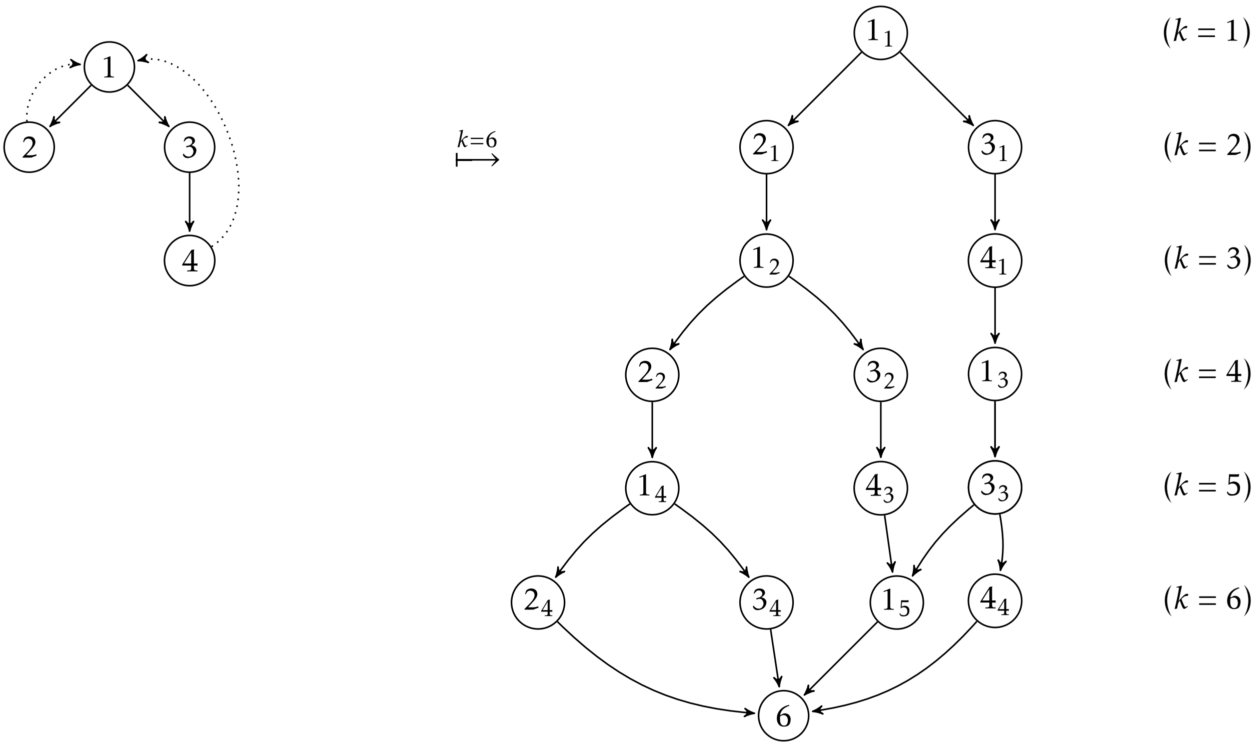

Figure 3.1: *Example of the flow graph from the Figure* **??**, *unwinded up to the bound* $k = 6$

3.5 XGraph to ZFormula (SMT) encoder

- Then, this modified event-representation is being encoded to SMT formula and sent to the solver.

3.6 Optimisations

... performed on each stage

Chapter 4

Evaluation

4.1 Comparison with PORTHOS

4.1.1 Unique Features

4.1.2 Performance

4.2 Comparison with HERD

4.2.1 Unique Features

4.2.2 Performance

Chapter 5

Summary

Bibliography

[1] Paul E McKenney. *Is parallel programming hard, and, if so, what can you do about it?(v2017. 01.02 a)*. 2017.