MEMORY MODEL SPECIFIC VERIFICATION OF SAFETY PROPERTIES OF CONCURRENT PROGRAMS

CHINMAY NARAYAN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI
May 2017

MEMORY MODEL SPECIFIC VERIFICATION OF SAFETY PROPERTIES OF CONCURRENT PROGRAMS

by

CHINMAY NARAYAN

Department of Computer Science and Engineering

Submitted

in fulfillment of the requirements of the degree of Doctor of Philosophy

to the

Indian Institute of Technology Delhi
May 2017

Certificate

This is to certify that the thesis titled **Memory Model Specific Verification of Safety Properties of Concurrent Programs** being submitted by **Chinmay Narayan** for the award of **Doctor of Philosophy** in **Computer Science and Engineering** is a record of bona fide work carried out by him under my guidance and supervision at the Department of Computer Science and Engineering, Indian Institute of Technology Delhi. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

S. Arun-Kumar
Professor
Department of Computer Science
and Engineering
Indian Institute of Technology Delhi
New Delhi, India 110 016

Acknowledgments

As a long and an arduous journey of PhD comes to an end it is time to acknowledge those who stood beside me, either in act or in prayers, during all ups and downs of this journey. Their contributions, both on professional and personal fronts, were indispensable for the completion of this journey. I was fortunate to have Prof. S.Arun-Kumar as a teacher and my thesis supervisor. As a teacher, he has an exemplary quality to explain things in a formal yet a simple to understand manner. His untiring patience to listen my questions, however simplistic they may be, and a calm attitude to answer them irrespective of the number of times they have been answered before, are some of the qualities that I will cherish and like to emulate in my life. As a researcher, he taught me the importance of formalization and the appreciation of the fact that formalism and intuition can complement each other to give very useful insights about the problem in hand. This journey was not possible without his continuous encouragement and support.

During my PhD I worked with Dr. Viktor Vafeiadis as part of my internship at MPI, Kaiserslautern. He helped me to learn a lot of technical concepts new to me, which later helped me in my research. He was always available to answer my questions even after completing my internship. I thank him for his consistent support.

I also had the privilege to know and work with Dr. Subodh Sharma. The long technical discussions with him towards the end of my PhD were very helpful in forming this thesis. He was very friendly and helpful in giving constant encouragement and technical suggestions.

I would like to thank Prof. Sanjiva Prasad for his many invaluable suggestions and feedbacks about my research work, presentations and written reports.

A significant part of my learning during PhD is attributed to Shibashis Guha, a very good friend, a keen listener and a sound advisor. We joined as PhD candidates under the same supervisor and therefore underwent the most part of our training and research together. He was always available to give valuable feedback about all ideas. I owe him a

great deal for making my PhD experience a lot more memorable.

I would also like to thank Divyanshu Bagga to make me realize the beauty and elegance that lies in proofs. He also helped me to make some of the proofs in this thesis simpler.

I was fortunate to have a group of great friends who made my stay a lot more enjoyable during PhD. Together we enjoyed many food outings, shared many emotions and discussed lot of things over uncountable cups of tea and *samosa*. They helped me immensely during different ups and downs of this journey, sometimes with just their presence. Thank you Shibashis, Shashank, Syamantak, Anamitra, Manoj, and Nisha for being such a great company.

A significant part of my PhD life was spent in the department of computer science and engineering and the verification lab in particular. I would like to thank Mr. M.M. Rathinam and Hemant for their lab related support.

My family deserves a special acknowledgement for supporting me unequivocally in taking up this long journey. I could not have completed this journey without their love, support and faith in my abilities. My parents, brother, and sister-in-law were a constant source of encouragement and motivation. I would also like to thank my in-laws for putting their faith in me by allowing me to marry their daughter during my PhD. I would like to thank Meenakshi and Homja for their unconditional love and support during all these years.

I was privileged to get many favorable circumstances which helped me to pursue my interest in research. Many of these privileges were not earned but were the result of blessings bestowed upon me by the almighty. I thank almighty for all those blessings which helped me in the pursuit of this goal.

Chinmay Narayan

Abstract

Many-core processors have already become a norm in the processor industry. Nearly every machine nowadays is equipped with these many-core processors. Most of these processors *relax* the order of execution of instructions in order to minimize the latency caused by write instructions. As a consequence, a correct program under the SC memory model might exhibit an erroneous execution under relaxed memory models. Therefore there is a need to verify the correctness of a concurrent program with respect to underlying memory model. In this work, we consider safety properties of concurrent programs as correctness specification.

In this thesis we propose a *trace partitioning based approach* as a method of proving correctness of concurrent programs under those memory models which have a well-defined operational semantics. We take up the SC, TSO, and PSO memory models in this work. We give an alternating finite automata (AFA) based construction to represent a set of *similar* traces. Such a similarity measure allows us to partition the set of all traces of a concurrent finite data domain program in a finite number of equivalence classes. To extend this approach for the TSO and PSO memory models we propose an equivalent alternate semantics of these models. Such a semantics allow us to construct a set of symbolic traces of a program under these memory models. These traces are then partitioned using AFA in a number of equivalence classes. To handle the unbounded buffer size of TSO and PSO memory models we prove the existence of a fixed point k_0 for every program P such that it is sufficient to prove the correctness of P with buffer size k_0. This theorem makes our approach complete for the TSO and the PSO memory models. We implemented this approach in a tool `ProofTraPar` which works very competitively against state-of-the-art tools.

For language memory models which are defined axiomatically, like C11, we explore the possibility of using concurrent separation logic (CSL) as a feasible proof method. We consider the C11 memory model and extend CSL to propose *Relaxed Separation*

Logic (RSL). We apply the idea of the *transfer of ownership* from CSL to model different atomics proposed in the C11 model.

सारांश

कई–कोर प्रोसेसर पहले से ही प्रोसेसर उद्योग में बहुत प्रचिलित हो गए हैं। आजकल लगभग हर मशीन इन कई–कोर प्रोसेसर से बनी है। इनमें से अधिकतर प्रोसेसर काम की गति को बढ़ाने के लिए निर्देशों के क्रम को बदल देते हैं परिणामस्वरूप एक ऐसा प्रोग्राम जो बिना क्रम बदले सही साबित किया गया था, क्रम बदलने के बाद एक गलत परिणाम दे सकता है । इसलिए इन प्रोग्राम्स को सही साबित करने के लिए बदले हुए क्रम को ध्यान में रखना बहुत जरूरी हो जाता है । इस काम में हम कुछ मेमोरी मॉडल्स, जो कि निर्देशों के क्रम को अलग–अलग तरीके से बदलते हैं, को समझते हैं और यह देखते हैं कि वह किसी प्रोग्राम के सही काम करने के गुण को कैसे प्रभावित करते हैं।

इस थीसिस में हम ट्रेस विभाजन आधारित एक विधि को प्रस्तावित करते हैं जो कि यह मालूम करती है कि कोई प्रोग्राम किसी दिए हुए मेमोरी मॉडल में सही तरीके से काम करेगा या नहीं। इस ट्रेस विभाजन की विधि की प्रभावशीलता दिखने के लिए हम तीन मेमोरी मॉडल्स के साथ काम करते हैं जिनका नाम एससी, टीएसओ, और पीएसओ है। किसी प्रोग्राम के हर क्रियान्वन का प्रतिनिधित्व करने के लिए एक अल्टेरनेटिंग फाइनाइट आटोमैटा आधारित निर्माण करते हैं। इस अल्टेरनेटिंग फाइनाइट ऑटोमेटा के द्वारा एक तरह के सभी क्रियान्वन एक साथ बनाये जा सकते हैं। सीमित डाटा डोमेन होने की वजह से इस तरह के ऑटोमेटा भी सीमित ही बनते हैं। इस विधि को एससी मेमोरी मॉडल में प्रयोग करने के बाद हम इस विधि को टीएसओ और पीएसओ मेमोरी मॉडल के लिए भी आगे संशोधित करते हैं। इस क्रम में हम यह साबित करते हैं कि असीमित मेमोरी बफर में प्रोग्राम को सही दिखने के लिए उसे केवल एक निश्चित बफर आकार तक ही सही दिखाना पर्याप्त होता है । इस विधि की दक्षता को वास्तविक प्रोग्राम्स पर दिखने के लिए हमने इसका एक प्रोग्राम बनाया जिसका नाम **प्रूफट्रेपर** रखा। यह प्रोग्राम दूसरे समकालीन प्रोग्राम्स के समक्ष काफी जल्दी और अच्छा काम करता है। इसके अलावा इस थीसिस में हमने C++ भाषा के मेमोरी मॉडल पर भी काम किया और दिखाया कि स्वामित्व हस्तांतरण का प्रयोग करके इस भाषा में लिखे प्रोग्राम्स की सत्यता के बारे में कुछ महत्वपूर्ण तथ्य साबित किये जा सकते हैं ।

Contents

List of Figures xiii

1 Introduction 1

1.1 Related Work . . . 3

1.2 Overview of the thesis . . . 8

2 Trace Partitioning for the SC Memory Model 11

2.1 Preliminaries . . . 15

2.1.1 Program Model . . . 15

2.1.2 Weakest Precondition . . . 17

2.1.3 Alternating Finite Automata (AFA) . . . 21

2.2 Overview of our approach . . . 22

2.2.1 Constructing the AFA from a Trace and a Formula . . . 23

2.2.2 Constructing the weakest precondition from $\hat{\mathcal{A}}_{\sigma,\phi}$ . . . 28

2.2.3 Enlarging the set of words accepted by $\hat{\mathcal{A}}_{\sigma,\phi}$ . . . 32

2.2.4 Putting All Things Together For Safety Verification . . . 36

2.3 Experimental Evaluation . . . 38

2.3.1 Discussion . . . 40

3 Trace Partitioning for the TSO Memory Model 43

3.1 Preliminaries . . . 46

3.1.1 Program Model . . . 46

3.1.2 The TSO Semantics . . . 47

3.2 Overview of our approach . . . 49

3.2.1 Unbounded Buffer Analysis . . . 49

3.2.2 Symbolic trace construction . . . 56

3.2.3 Fence Insertion . . . 68

3.2.4 Extension to the PSO memory model 70
3.3 Experimental Evaluation 70
3.3.1 Discussion 72

4 RSL: A Proof System for the C11 Memory Model 75
4.1 Preliminaries 76
4.1.1 Program Model 76
4.1.2 The C11 Memory Model 78
4.2 Relaxed Separation Logic 83
4.2.1 Overview of RSL 83
4.2.2 Examples 91
4.3 Dealing with Relaxed Memory Accesses 94
4.4 Semantics and Soundness 97
4.4.1 Semantics of Assertions 97
4.4.2 Semantics of Hoare triples 104
4.4.3 Memory Safety and Race Freedom 109
4.4.4 Soundness of the Proof Rules 116
4.5 Remarks 124

5 Conclusion and Future Directions 125
5.1 Conclusion 125
5.2 Future Directions 126

Bibliography 129

List of Figures

1.1 Peterson's algorithm for two processes . . . 3

1.2 Related Work: An overview . . . 5

2.1 Property specific trace partitioning . . . 12

2.2 Peterson's algorithm for two processes . . . 13

2.3 Comparison with inductive Data Flow Graphs [FKP13] . . . 14

2.4 Specification of Peterson's algorithm . . . 16

2.5 Weakest precondition axioms . . . 17

2.6 Trace partitioning approach for the SC memory model in a nutshell . . . 23

2.7 Transition function used in Definition 2.2.1 . . . 24

2.8 A trace from Peterson's algorithm . . . 25

2.9 AFA of trace given in Figure 2.8 and $\phi = \neg(\ell_2 = 2)$ . . . 27

2.10 Rules for `HMap` construction . . . 29

2.11 Examples illustrating the rational behind defining stability in terms of logical equivalence . . . 32

2.12 `HMap` construction for the running example . . . 33

2.13 Example showing the conversion of universal states to existential states . . . 34

2.14 Rules for adding more edges . . . 35

2.15 AFA of Figure 2.12 after Modification . . . 36

2.16 Comparison with THREADER[GPR11b], and Lazy-CSeq [ITF+14] . . . 40

2.17 An example program where bug is exposed in three context switches . . . 40

3.1 Challenges in extending the trace partitioning approach for TSO and PSO 43

3.2 Extending the Trace partitioning approach for the TSO and PSO memory models in a nutshell . . . 46

3.3 TSO semantics for *k-bounded* buffers . . . 47

3.4 Transition rules for symbolic trace construction 58
3.5 Reasoning in presence of the read operation from the buffer 62
3.8 Comparison of our tool with Memorax[AAC+12]. 71

4.1 Simple spinlock implementation. 77
4.2 Semantics of closed program expressions. 80
4.3 Axioms satisfied by consistent C11 executions 81
4.4 Sample executions violating coherency conditions [BOS+11]. 81
4.5 Message passing example showing transfer of ownership 83
4.6 Standard proof rules supported by RSL. 85
4.7 Example illustrating splittable acquire permissions 88
4.8 Verification of the lock module. 91
4.9 Verification of the message passing example. 92
4.10 Example illustrating fractional ownership and transfer thereof. 94
4.11 Program with a possible dependency cycle. 94
4.12 Execution exhibiting the dependency cycle. 94
4.13 Program without a dependency cycle. 96
4.14 Definition of heap composition, $h_1 \oplus h_2$. 98
4.15 Definition of the semantics of assertions. 102
4.16 Cases for the race-freedom proof. 113