Experiences working with Verifast, Predator and Forester

dsOli/DSI Day, 15th January 2016

Jan Boockmann

table of contents

shape analysis

- subtopic of program analysis
- static code analysis technique (used at compile time)
- verify properties of linked, dynamically allocated data structures (reason about the heap)
- applicable in the field of compile-time optimization and program verification

shape analysis (cont'd)

- prove termination
- (partial/total) correctness (i.e. does this function really sort a list)
- check concurrent programs (deadlocks or mutex relations)
- verification of safety properties (memory leaks, null dereference, multiple frees, array out of bound errors)
- may-alias, must-alias, sharing, reachability, disjointness, cyclicity, etc.

shape analysis (cont'd)

- most of the approaches are graph based
- to achieve scalable tools heap abstraction is required

Predator

- tool for automated formal verification of sequential C programs operating with pointers and linked lists
- was successful in the last four SVCOMPs (1G/2S/1B)
- inspired by works on separation logic, but now purely graph-based
- supported operations: pointer arithmetic, reinterpretation of memory contents, address alignment, …
- “hunts” for memory safety errors
- similar tools: SpaceInvader[Calcagno2009]

Forester

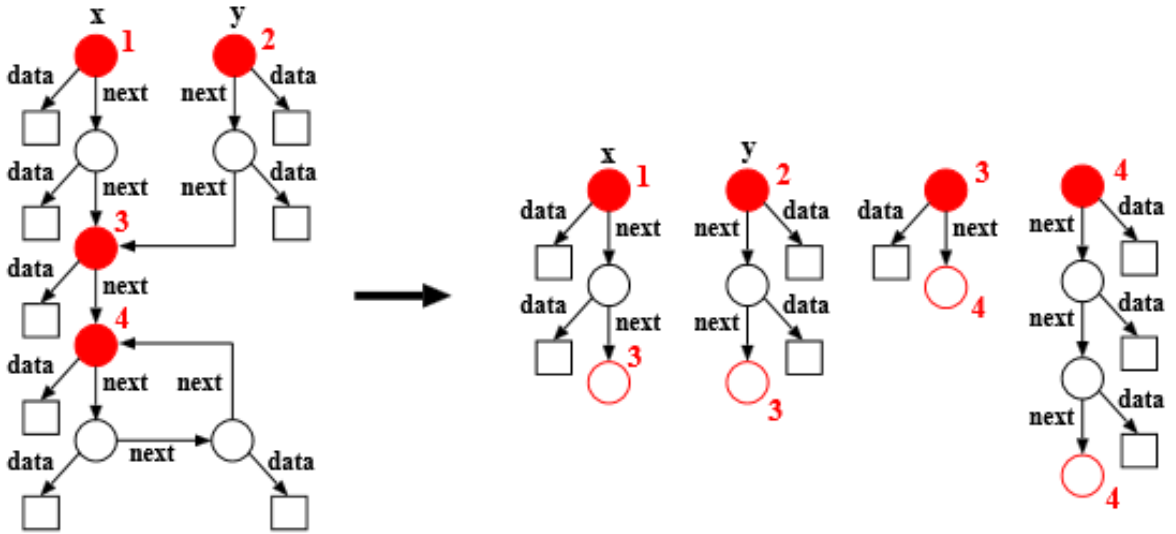
- tool for checking manipulation of dynamic data structures in sequential C programs
- using forest automata
- searches for the same kind of errors as Predator
- really difficult to understand

Forester (cont'd)

- heap is split at cut-points intro tree components

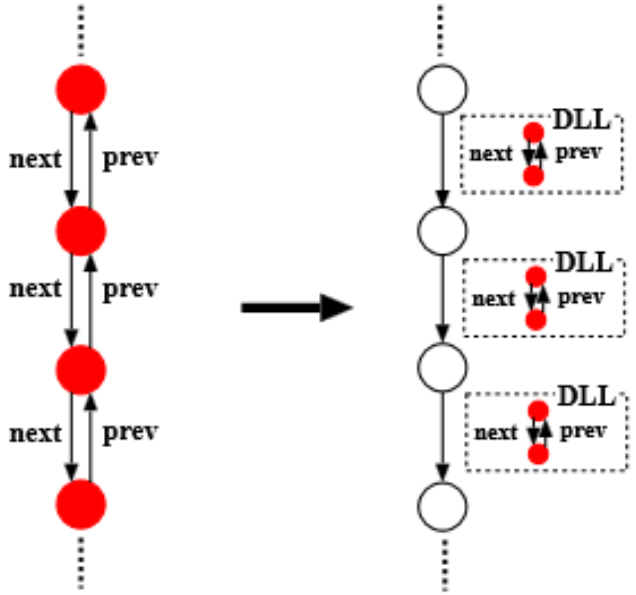
  →inspired by separation logic

- tree automata represent set of tree components
- forest automata represent tuples of tree automata



Forester (cont'd)

- structures with unboundedly many cut-points are represented hierarchically → boxes



example

Verifast

*"**VeriFast** is a **verifier** for **single-threaded and multithreaded C and Java programs annotated with preconditions and postconditions** written in **separation logic**. […] The programmer may define **inductive datatypes**, primitive recursive pure **functions over these datatypes**, and abstract separation logic **predicates**. To enable verification of these rich specifications, the programmer may write **lemma functions.** […] Since neither VeriFast itself **nor** the underlying SMT solver need to do any* **significant search**, ***verification time is predictable and low**."*

http://people.cs.kuleuven.be/~bart.jacobs/verifast/

separation logic[Reynolds2002]

- extension of Hoare logic
- introduces the ‘separating conjunction’ and the ‘frame rules’
- allows modular reasoning

$$\{P\}S\{Q\} \qquad P * Q \qquad \frac{\{P\}S\{Q\}}{\{P*R\}S\{Q*R\}}$$

example

experiences & comparison

- soundness
- definition of memory safety

- automation
- “debuggability”

outlook

- writing annotations is complex and time consuming
- average of 2.17 lines of C code verified per hour [Philippaerts2014]
- automating parts of this process can greatly enhance the use of VeriFast
- dsOli could only generate annotations for non-nested data structures [Mühlberg2015]
- DSI is more powerful and can cope nested data structures

literature

- Jan Tobias Mühlberg, David H. White, Mike Dodds, Gerald Lüttge and Frank Piessens. 2015. Learning Assertions to Verify Linked-List Programs. 13th Intl. Conf. on Software Engineering and Formal Methods. Springer-Verlag
- Philippaerts, P., Mühlberg, J. T., Penninckx, W., Smans, J., Jacobs, B., and Piessens, F. Software verifcation with VeriFast: Industrial case studies. Science of Computer Programming, 82:77 97, 2014.
- Cristiano Calcagno, Dino Distefano, Peter O'Hearn, and Hongseok Yang. 2009. Space Invading Systems Code. In *Logic-Based Program Synthesis and Transformation*, Michael Hanus (Ed.). Lecture Notes In Computer Science, Vol. 5438. Springer-Verlag, Berlin, Heidelberg 1-3
- John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science* (LICS '02). IEEE Computer Society, Washington, DC, USA, 55-74.
- Predator/Forester GIT Repository: https://github.com/kdudka/predator