Model Checking for Symbolic-Heap Separation Logic with Inductive Predicates

James Brotherston [1] Max Kanovich [1] Nikos Gorogiannis [2]
Reuben N. S. Rowe [1]

Resource Reasoning Meeting, UCL, Wednesday 13th January 2016

[1] Programming Principles, Logic & Verification Group
Department of Computer Science, University College London

[2] Foundations of Computing Group
Department of Computer Science, Middlesex University

Motivation

Aim: *automated* verification of heap-manipulating programs

How: using *separation logic* (SL) [Reynolds, O'Hearn, many others ...]

- early successes based on (decidable) fixed abstractions
    - e.g. Smallfoot, SLAyer
- more recently, tools support user-defined predicates
    - e.g. jStar, Verifast, Cyclist (Caber), HIP/SLEEK (S2)

What: *dynamic* verification based on *model checking*

- check observed memory state matches SL assertion at given program point

Why: SL with *general* inductive predicates undecidable [Antonopoulos et al. 2014], so static analysis incomplete

Overview of our Results

For *symbolic heap* SL with arbitrary inductive predicates Φ:

- the model checking problem $(s, h) \models^?_\Phi F$ is decidable
- We identify three axes (**CV**, **DET** and **MEM**) of syntactic restriction for inductive definitions
- We prove the following complexity results:

		CV	DET	CV+DET
non-**MEM**	EXPTIME	EXPTIME	EXPTIME	$\geq$ PSPACE
MEM	NP	NP	NP	PTIME

- We provide a prototype tool implementation and experimental evaluation

Symbolic Heaps with Inductive Predicates

Terms:	$t ::= x \mid \mathbf{nil}$
Pure Formulas:	$\mathsf{Pure} \ni \pi ::= t = t \mid t \neq t \qquad \Pi \in \wp(\mathsf{Pure})$
Spatial Formulas:	$\Sigma ::= \mathbf{emp} \mid x \mapsto \boldsymbol{t} \mid P\boldsymbol{t} \mid \Sigma * \Sigma$
Symbolic Heaps:	$F ::= \exists \boldsymbol{x}. \Pi : \Sigma$

Symbolic heap $\exists \boldsymbol{x}. \Pi : \Sigma$ interpreted as a set of pairs (s, h):

- Stack s (maps variables to heap locations) must satisfy Π
- Heap h (maps locations to memory cells) described by Σ
    - **emp** is the empty heap
    - $x \mapsto \boldsymbol{t}$ denotes a singleton heap
    - $\Sigma_1 * \Sigma_2$ given by *disjoint* union
    - Predicates $P\boldsymbol{t}$ interpreted according to inductive definitions

Interpreting Inductive Definitions

We consider finite sets Φ of *inductive rules*:

$$\exists z.\Pi : \Sigma \Rightarrow Px$$

- The inductive rules allow self-reference (i.e. recursion)
- The collection of rules for a predicate P constitute the *disjunctive clauses* of its definition

An inductive rule set Φ is interpreted by a (least) fixed-point construction:

- start with the empty interpretation
- iteratively generate models using the rules and previously generated interpretation until saturation

General Model Checking Algorithm

RESULT: The model checking problem $(s, h) \models^?_\Phi F$ is decidable

General Model Checking Algorithm

There are a number of subtleties:

- A top-down rule-unfolding approach may not terminate
- We must consider infinite sets (values, heap locations, models)

General Model Checking Algorithm

There are a number of subtleties:

- A top-down rule-unfolding approach may not terminate
- We must consider infinite sets (values, heap locations, models)

We show that a bottom-up fixed-point algorithm is complete:

- It suffices to just consider instances of the form $(s, h) \models^{?}_{\Phi} P\boldsymbol{t}$
- We need only consider *sub-models* of the problem instance
- Values for existentially quantifies variables can be taken from a well-defined finite set

General Model Checking Algorithm

There are a number of subtleties:

- A top-down rule-unfolding approach may not terminate
- We must consider infinite sets (values, heap locations, models)

We show that a bottom-up fixed-point algorithm is complete:

- It suffices to just consider instances of the form $(s, h) \models^{?}_{\Phi} P\boldsymbol{t}$
- We need only consider *sub-models* of the problem instance
- Values for existentially quantifies variables can be taken from a well-defined finite set

The decision problem is **EXPTIME**-complete

- requires at most exponential number of (**PTIME**) iterations
- lower bound by a reduction from the satisfiability problem

[Brotherston et al., CSL-LICS 2014]

Syntactically Restricted Fragments

We identify three independent syntactic conditions on inductive definitions:

MEM: (Memory-consuming) rule bodies may only contain predicates if they also contain explicit memory fragments ($\mapsto$)

DET: (Deterministic) the sets of pure constraints of the rules for a given predicate P are mutually exclusive with each other

CV: (Constructively Valued) the values of the existentially quantified variables in rule bodies are (uniquely) determined by the parameters

Syntactic Restrictions: Examples

- Acyclic linked list (MEM+CV+DET):

$$x = \mathsf{nil} : \mathsf{emp} \Rightarrow \mathsf{List}(x) \qquad \exists y.\, x \neq \mathsf{nil} : x \mapsto y * \mathsf{List}(y) \Rightarrow \mathsf{List}(x)$$

Syntactic Restrictions: Examples

- Acyclic linked list (MEM+CV+DET):

$$x = \mathsf{nil} : \mathsf{emp} \Rightarrow \mathsf{List}(x) \qquad \exists y.\, x \neq \mathsf{nil} : x \mapsto y * \mathsf{List}(y) \Rightarrow \mathsf{List}(x)$$

- Possibly cyclic linked list segment (MEM+DET):

$$x = y : \mathsf{emp} \Rightarrow \mathsf{rls}(x, y) \qquad \exists z.\, x \neq y : \mathsf{rls}(x, z) * z \mapsto y \Rightarrow \mathsf{rls}(x, y)$$

Syntactic Restrictions: Examples

- Acyclic linked list (MEM+CV+DET):

$$x = \mathbf{nil} : \mathbf{emp} \Rightarrow \mathbf{List}(x) \qquad \exists y.\, x \neq \mathbf{nil} : x \mapsto y * \mathbf{List}(y) \Rightarrow \mathbf{List}(x)$$

- Possibly cyclic linked list segment (MEM+DET):

$$x = y : \mathbf{emp} \Rightarrow \mathbf{rls}(x, y) \qquad \exists z.\, x \neq y : \mathbf{rls}(x, z) * z \mapsto y \Rightarrow \mathbf{rls}(x, y)$$

- Binary tree and tree context (MEM+CV):

$$x = \mathbf{nil} : \mathbf{emp} \Rightarrow \mathbf{tree}(x) \quad \exists y, z.\, x \neq \mathbf{nil} : x \mapsto (y, z) * \mathbf{tree}(y) * \mathbf{tree}(z) \Rightarrow \mathbf{tree}(x)$$

$$x = y : \mathbf{emp} \Rightarrow \mathbf{tree\_ctxt}(x, y)$$

$$\exists v, w.\, x \neq y : x \mapsto (v, w) * \mathbf{tree}(v) * \mathbf{tree\_ctxt}(w, y) \Rightarrow \mathbf{tree\_ctxt}(x, y)$$

$$\exists v, w.\, x \neq y : x \mapsto (v, w) * \mathbf{tree}(w) * \mathbf{tree\_ctxt}(v, y) \Rightarrow \mathbf{tree\_ctxt}(x, y)$$

Complexity of Model Checking Restricted Fragments

		CV	DET	CV+DET
non-MEM	EXPTIME	EXPTIME	EXPTIME	$\geq$ PSPACE
MEM	NP	NP	NP	PTIME

- **NP** upper bound for **MEM** rules:
  - top-down procedure with sub-model restriction
- lower bounds given by:

  (MEM+CV) reduction from the 3-partition problem

  (MEM+DET) reduction from 3-SAT
- For **MEM+CV+DET**, top-down procedure is *deterministic* and polynomially bounded in the size of the heap

Implementation

- Implemented both algorithms in OCaml
- Formulated 'typical performance' benchmark suite:
  - 6 annotated programs from the Verifast test suite
    - Covers almost all fragments (CV+DET missing)
    - Assertions taken from 15 different program points
  - Harvested over 2150 concrete models at runtime
    - ranging in size from 0 – 100 memory cells
- Also tested worst-case performance
  - using the harvested models and predicates requiring the generation of all possible submodels
- Tested PTIME algorithm on relevant benchmark instances

Experimental Results

- Tests carried out on 2.93GHz Intel i7 (8GB memory)
- Running times for the general algorithm demonstrate exponential behaviour:
    - for 10 heap cells – between 5 and 60ms
    - for 30 heap cells – between 10ms and 10s
    - some instances with 100 heap cells still checking in ~100ms
- Worst-case benchmark instances: ~40s for 20 heap cells
- All runs of the PTIME algorithm took <10ms

Conclusions & Future Work

- We show decidability of the model checking problem for symbolic heap SL with general inductive predicates
  - EXPTIME complexity reduced to NP or PTIME with natural restrictions on the inductive predicates
- Prototype OCaml implementation and experimental evaluation
  - PTIME algorithm shows promise for run-time verification
  - General algorithm may still be useful for off-line (unit) testing
- Future work:
  - How does adding *classical* conjunction affect the results?
  - Model checking may facilitate *disproving* of entailments

Thank you for listening!

Implementation available at:
`github.com/ngorogiannis/cyclist`