Let this Graph be your Witness!

An Attestor for Verifying Java Pointer Programs

Hannah Arndt, Christina Jansen, Joost-Pieter Katoen,
Christoph Matheja, Thomas Noll

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Motivation

Writing and **analyzing** pointer programs is challenging:

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

Various verification tools focus on **memory safety** and **identifying shapes**.

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

Various verification tools focus on **memory safety** and **identifying shapes**.

Example: Reversing a list yields a list again.

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

Various verification tools focus on **memory safety** and **identifying shapes**.

Example: Reversing a list yields a list again.

But: Is it a reversal of the original list?

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

Various verification tools focus on **memory safety** and **identifying shapes**.

Example: Reversing a list yields a list again.

But: Is it a reversal of the original list?

If not, why?

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

Various verification tools focus on **memory safety** and **identifying shapes**.

Example: Reversing a list yields a list again.

But: Is it a reversal of the original list?

If not, why?

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

Various verification tools focus on **memory safety** and **identifying shapes**.

Example: Reversing a list yields a list again.

But: Is it a reversal of the original list?

If not, why?

A model checker for Java pointer programs

Motivation

Writing and **analyzing** pointer programs is challenging:

aliasing, sharing, dynamic allocation, infinite state spaces

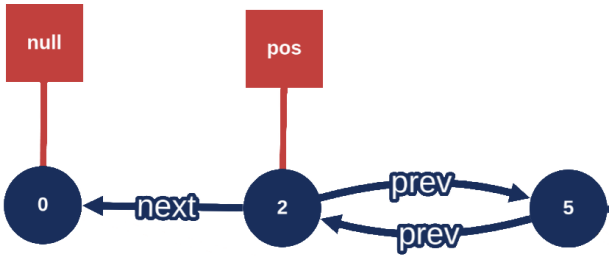
Various verification tools focus on **memory safety** and **identifying shapes**.

Example: Reversing a list yields a list again.

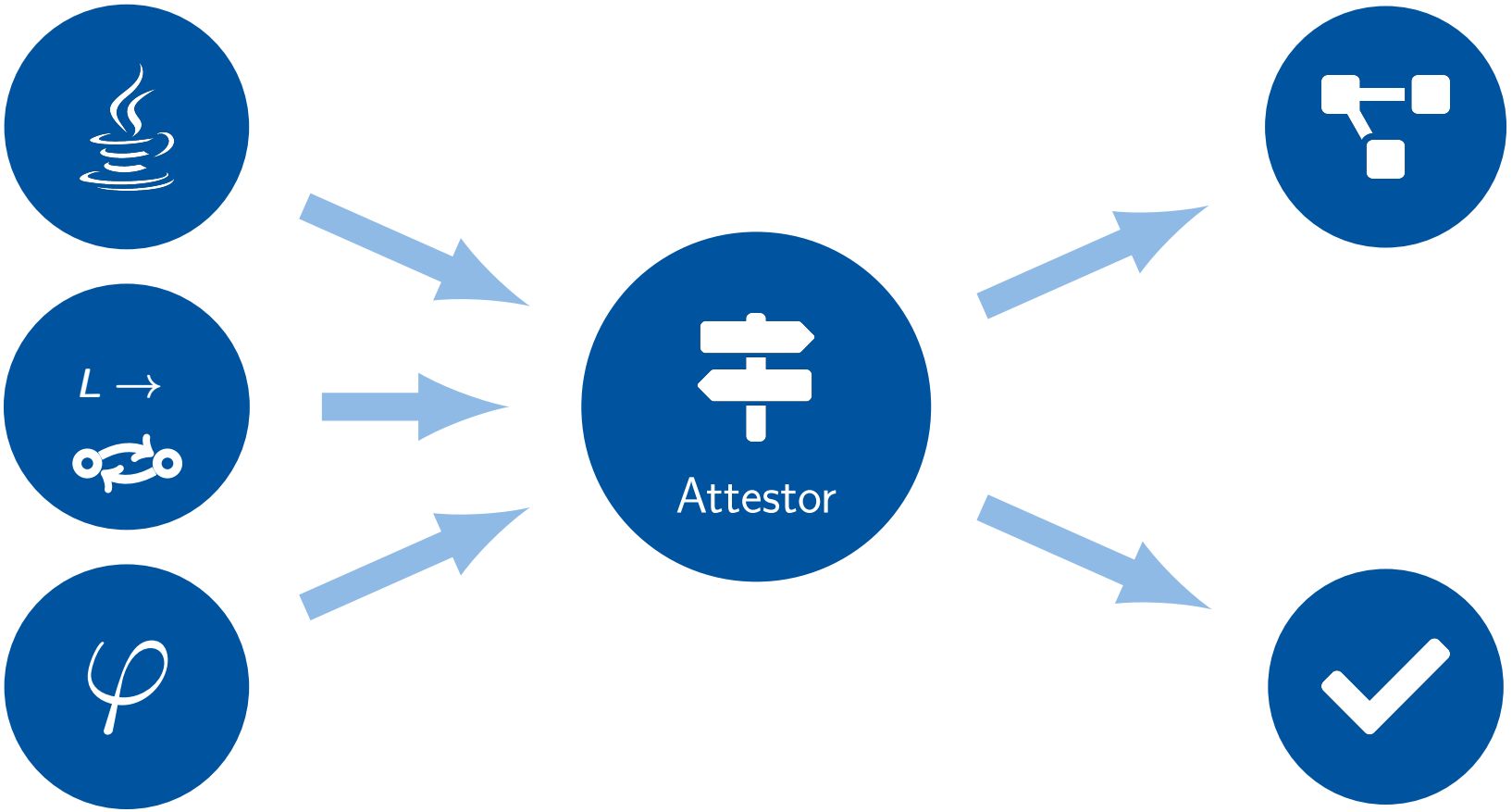
But: Is it a reversal of the original list?

If not, why?

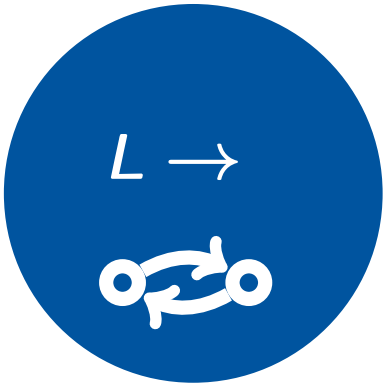
A model checker for Java pointer programs



Input / Output



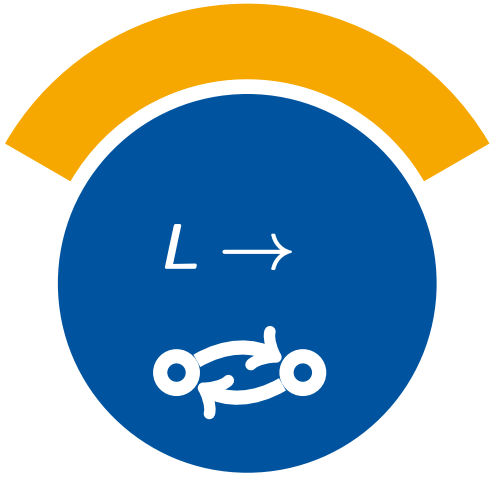
Input: Graph Grammar



Input: Graph Grammar

Program states

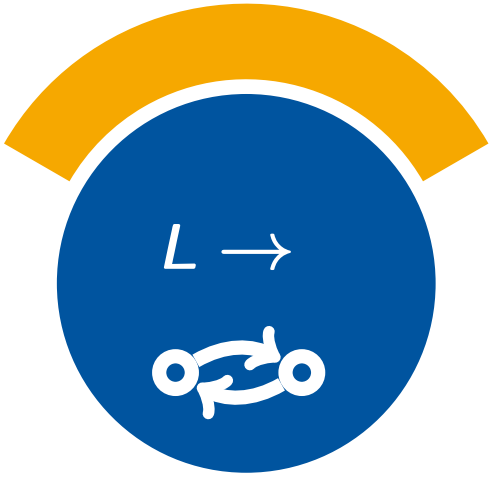
program states are graphs



Input: Graph Grammar

Program states

abstract program states are languages of graphs



Input: Graph Grammar

Program states

abstract program states are languages of graphs

Graph Grammars

(indexed) HRG

Input: Graph Grammar

Program states

abstract program states are languages of graphs

Graph Grammars

(indexed) HRG

built-in or user-supplied

Input: Graph Grammar

Program states

abstract program states are languages of graphs

Graph Grammars

(indexed) HRG

built-in or user-supplied

data structure spec.

Input: Graph Grammar

Program states

abstract program states are languages of graphs

Graph Grammars

(indexed) HRG

built-in or user-supplied

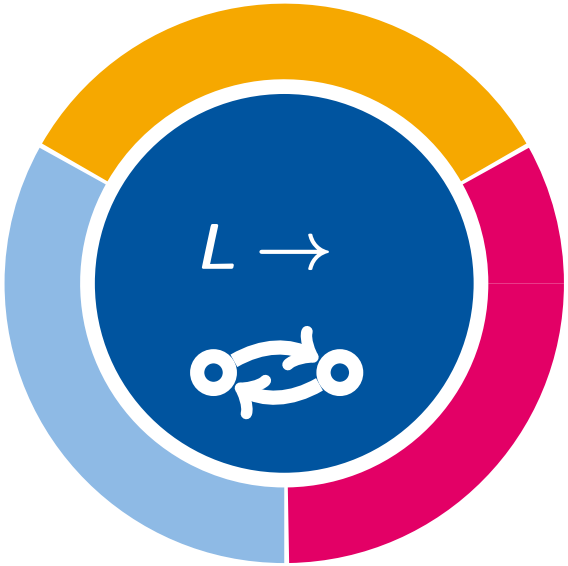
data structure spec.

guide abstraction

Input: Graph Grammar

Program states

abstract program states are languages of graphs



Graph Grammars

(indexed) HRG

built-in or user-supplied

data structure spec.

guide abstraction

Examples

(doubly-)linked lists

lists of lists

skip lists

(balanced) trees

Input: LTL Specification

Input: LTL Specification

Specifications

LTL

Examples

$\mathcal{G}\,\mathcal{F}\,\texttt{empty}$

Input: LTL Specification

Specifications

LTL

$+$ points-to relations

Examples

$\mathcal{G}\,\mathcal{F}\,$`empty`

`x.next != null`

Input: LTL Specification

Specifications

LTL

+ points-to relations

+ reachability

Examples

$\mathcal{G}\,\mathcal{F}\,\texttt{empty}$

`x.next != null`

$x \xrightarrow{\texttt{next}} y$

Input: LTL Specification

Specifications

LTL

+ points-to relations

+ reachability

+ grammar languages

Examples

$\mathcal{G}\,\mathcal{F}\,\texttt{empty}$

`x.next != null`

$x \xrightarrow{\texttt{next}} y$

heap tree shaped

Input: LTL Specification

Specifications

LTL

+ points-to relations

+ reachability

+ grammar languages

+ object-tracking

Examples

$\mathcal{G}\,\mathcal{F}\,\texttt{empty}$

`x.next != null`

$x \xrightarrow{\texttt{next}} y$

heap tree shaped

$\forall e : \mathcal{F}\,(x == e)$

Attestor: State Space Generation

Attestor: State Space Generation

Materialization

grammar derivations

Attestor: State Space Generation

Materialization

grammar derivations

Execution

graph transformations

procedure modular

Attestor: State Space Generation

Materialization

grammar derivations

Execution

graph transformations

procedure modular

Abstraction

inverse derivations

Attestor: State Space Generation

Materialization

grammar derivations

Execution

graph transformations

procedure modular

Labeling

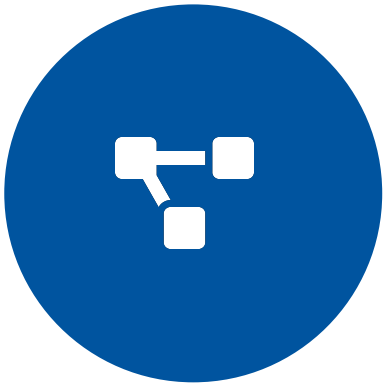
model checking problem

uses heap automata [ESOP'17]

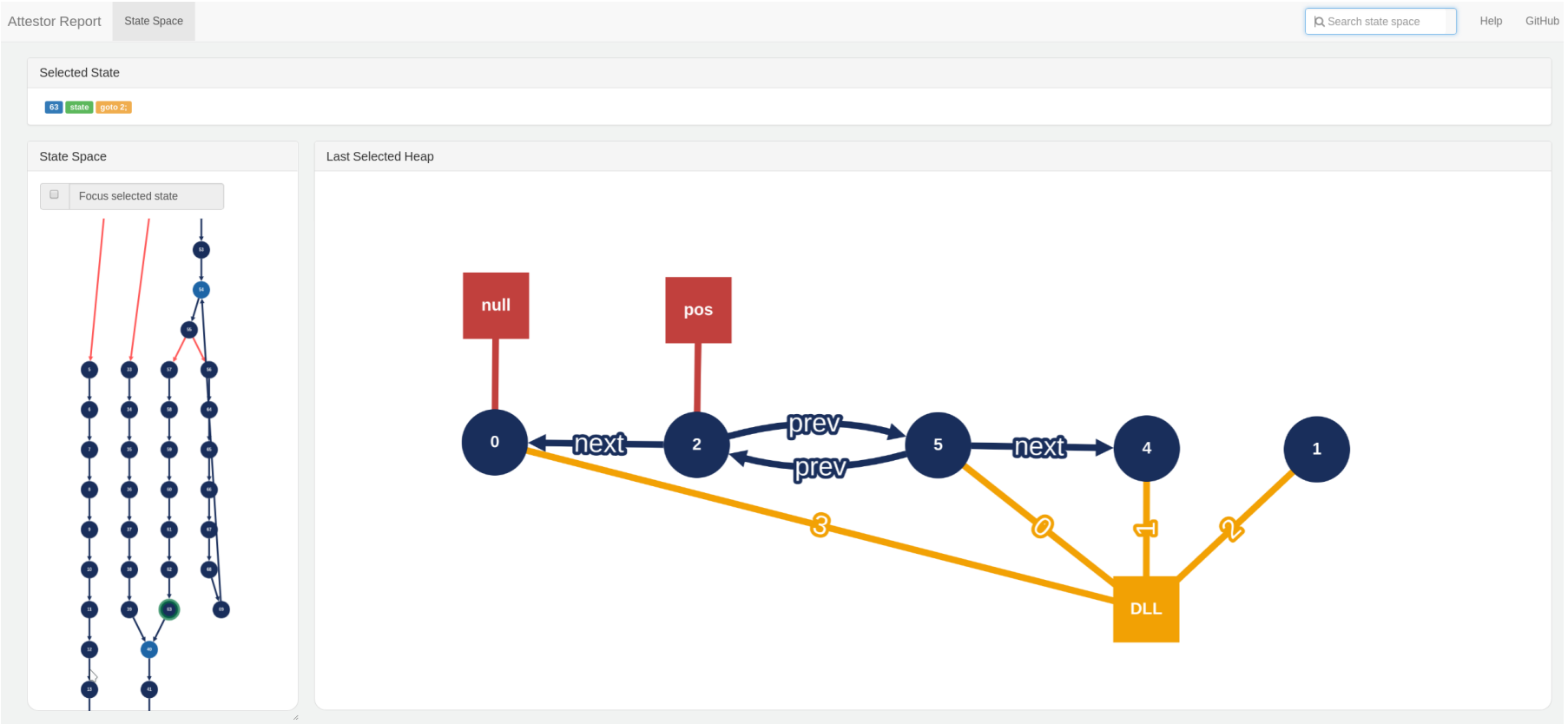
Abstraction

inverse derivations

Output: State Space



Output: State Space



Output: Specification Satisfied?

Output: Specification Satisfied?

Built-in LTL Model Checker

yes + proof

Output: Specification Satisfied?

Built-in LTL Model Checker

yes + proof

don't know

Output: Specification Satisfied?

Built-in LTL Model Checker

yes + proof

don't know

no + counterexample

Output: Specification Satisfied?

Built-in LTL Model Checker

yes + proof

don't know

no + counterexample

Counterexamples

possibly spurious

Output: Specification Satisfied?

Built-in LTL Model Checker

yes + proof

don't know

no + counterexample

Counterexamples

possibly spurious

verified by Attestor

Output: Specification Satisfied?

Built-in LTL Model Checker

yes + proof

don't know

no + counterexample

Counterexamples

possibly spurious

verified by Attestor

generate non-spurious initial states

Experiments

75 benchmarks in total, 7 different data structures

Experiments

75 benchmarks in total, 7 different data structures

Properties: memory safety, shape, reachability, full traversal, output = input

Experiments

75 benchmarks in total, 7 different data structures

Properties: memory safety, shape, reachability, full traversal, output = input

Lindstrom tree traversal (no recursion, no stack)

memory safety	229 states	0.119 seconds
output = input	67941 states	8.901 seconds

Recursive tree traversal

memory safety	91 states	0.075 seconds
output = input	21738 states	1.714 seconds

SLL to AVL tree

balanced tree	7166 states	2.728 seconds

Conclusion

Conclusion

~20k LOC (Java)

Conclusion

~20k LOC (Java)

build with
Apache Maven

Conclusion

~20k LOC (Java)

build with
Apache Maven

https://moves-rwth.github.io/attestor