Reachability Analysis of Program Variables

Đurica Nikolić

ETH - Chair of Software Engineering

November 6^{th}, 2013

Static Analysis - basic facts

- provides facts about run-time behavior of programs before their executions:
  - no division by zero
  - no null dereference
  - no infinite loops
  - ...
- numerical properties vs. memory-related properties
- over-approximations vs. under-approximations
- Abstract Interpretation [CousotCousot77] usually helps

Static Analysis - main issues

Static Analysis of real life software is extremely difficult:

- complex semantics of current programming languages
- memory-related properties required
- side-effects of method calls
- exceptional behaviors should be handled
- libraries heavily used
- annotations help, but...
- formalization vs. implementation
- proof of Soundness is difficult

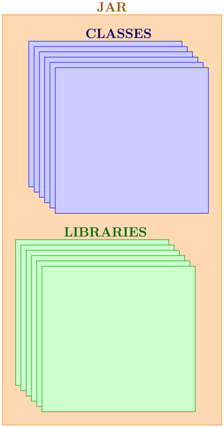
Static Analysis - main issues

Static Analysis of real life software is extremely difficult:

- complex semantics of current programming languages
- memory-related properties required
- side-effects of method calls
- exceptional behaviors should be handled
- libraries heavily used
- annotations help, but...
- formalization vs. implementation
- proof of Soundness is difficult

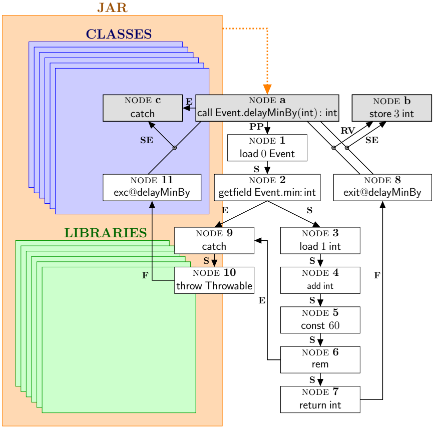
A Generic Framework for Constraint-Based Static Analyses of Java Bytecode Programs [NikolicPhD] deals with all these issues.

First steps



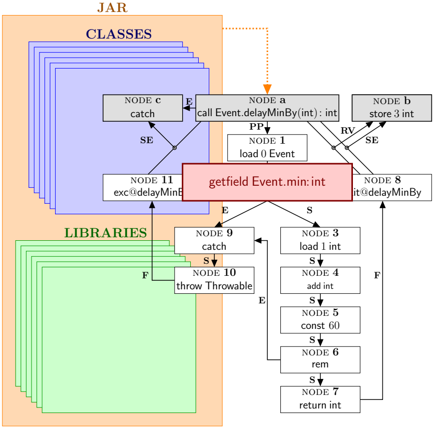
- Jar: classes and libraries

First steps



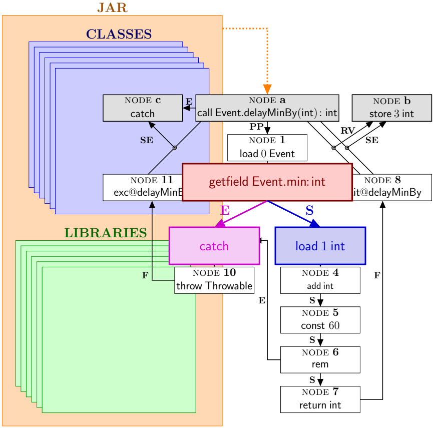
- Jar: classes and libraries
- CFG: extracted from Jar

First steps



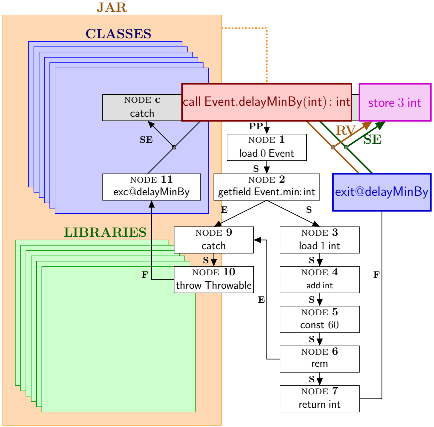
- Jar: classes and libraries
- CFG: extracted from Jar
  - nodes

First steps



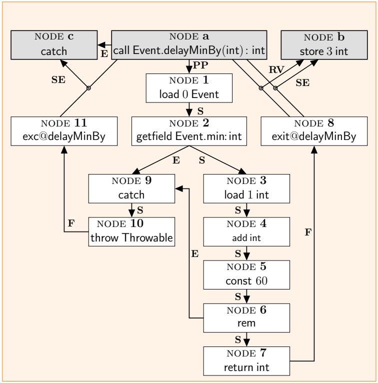
- Jar: classes and libraries
- CFG: extracted from Jar
  - nodes
  - sequential and exceptional arcs

First steps



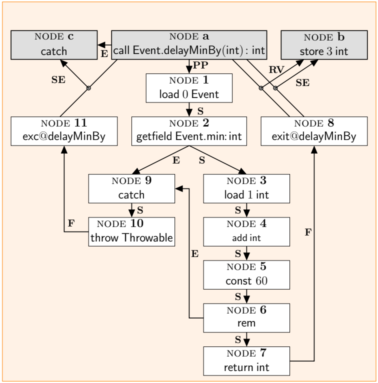
- Jar: classes and libraries
- CFG: extracted from Jar
  - nodes
  - sequential and exceptional arcs
  - return value and side-effects arcs

Abstract Constraint Graphs



Abstract Constraint Graphs

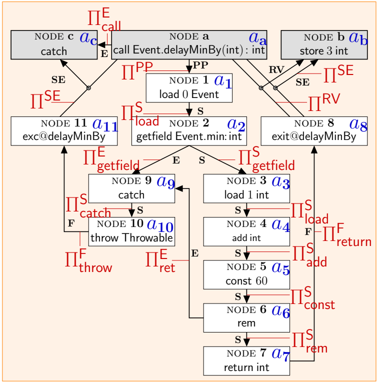
$\mathcal{A}$ - generic abstract domain



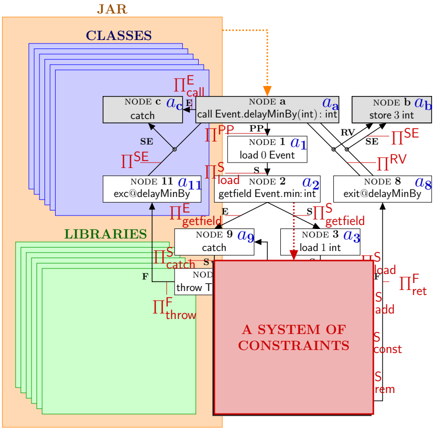
Abstract Constraint Graphs

$\mathcal{A}$ - generic abstract domain

$\Pi_{ins} : \mathcal{A} \rightarrow \mathcal{A}$ - generic propagation rule (abstract semantics of ins)

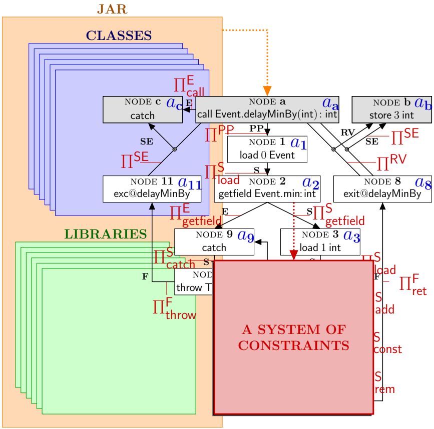


From ACG to Constraint-Based Static Analyses



- Jar: classes and libraries
- CFG: extracted from Jar
  - nodes
  - sequential and exceptional arcs
  - return value and side-effects arcs
- Abstract Constraints Graph
- a system of constraints - static analysis

From ACG to Constraint-Based Static Analyses



- Jar: classes and libraries
- CFG: extracted from Jar
  - nodes
  - sequential and exceptional arcs
  - return value and side-effects arcs
- Abstract Constraints Graph
- a system of constraints - static analysis
- Requirements!!!

$\mathcal{A}$ satisfies ACC, each Π_{ins} monotonic, each Π_{ins} soundly approximates ins
$\Rightarrow$ Soundness!!!

User vs. Framework

User	Framework
	• Extract CFG from a Jar
• Instantiate $\mathcal{A}$ (property) • Instantiate Π_{ins} for each ins (abstract semantics of ins)	
	• Construct ACG using Π_{ins}s • Extract constraints from ACG
• Show that $\mathcal{A}$ and each Π_{ins} meet framework's Requirements	
	• Existence of the least solution • Soundness of the solution

JULIA - A STATIC ANALYZER FOR JAVA AND ANDROID

SEVERAL CONSTRAINT-BASED STATIC ANALYSES HAVE BEEN IMPLEMENTED INSIDE JULIA. THEY ARE USED LIKE SUPPORTING ANALYSES FOR JULIA'S NULLNESS AND TERMINATION TOOLS AND IMPROVE THEIR PRECISION. WWW.JULIASOFT.COM

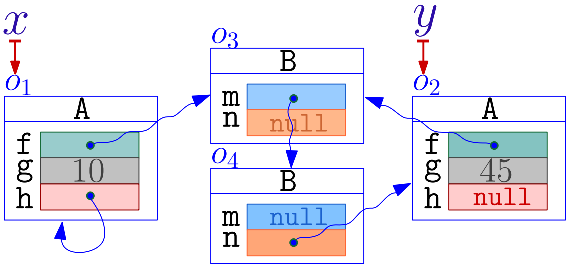
- DEFINITE ALIASING ANALYSIS
- POSSIBLE SHARING ANALYSIS [SAS 2008]
- POSSIBLE SIDE EFFECTS ANALYSIS
- POSSIBLE CREATION POINT ANALYSIS
- POSSIBLE REACHABILITY ANALYSIS [IJCAR 2012, TOPLAS 2013]
- DEFINITE EXPRESSION ALIASING ANALYSIS [ICTAC 2012]

Reachability Analysis of Variables: An Example of Constraint-Based Static Analysis

[IJCAR 2012, TOPLAS 2013]

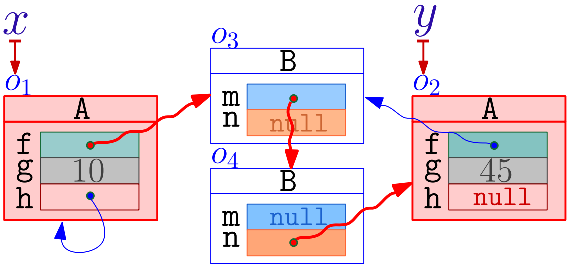
Intuitive definition of Reachability

Intuitive definition of Reachability



Is there a sequence of fields $\mathtt{f}_1, \ldots, \mathtt{f}_k$ such that $x.\mathtt{f}_1.\ldots.\mathtt{f}_k = y$?

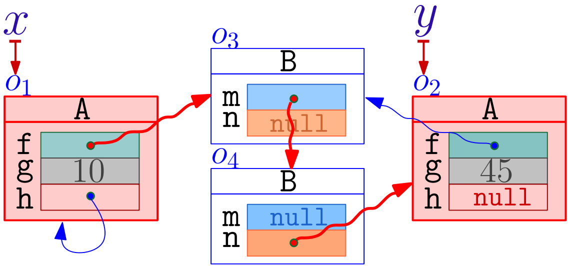
Intuitive definition of Reachability



Is there a sequence of fields $\mathtt{f}_1, \ldots, \mathtt{f}_k$ such that $x.\mathtt{f}_1.\ldots.\mathtt{f}_k = y$?

$x.\mathtt{f}.\mathtt{m}.\mathtt{n} = y$

Intuitive definition of Reachability



Is there a sequence of fields $\mathtt{f}_1, \ldots, \mathtt{f}_k$ such that $x.\mathtt{f}_1.\ldots.\mathtt{f}_k = y$?

$x.\mathtt{f}.\mathtt{m}.\mathtt{n} = y \quad \Rightarrow \quad x$ reaches y

A Simple Example - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public static void main(String[] args) {
    List<Student> list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>();
      tmp.head = student;
      tmp.tail = list;
      list = tmp;
    }
  }
}
```

A Simple Example - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public static void main(String[] args) {
    List<Student> list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>();
      tmp.head = student;
      tmp.tail = list;
      list = tmp;
    }
  }
}
```

Reachability

a reaches b, i.e., $a \rightsquigarrow b$ iff

a reaches a location bound to b

A Simple Example - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public static void main(String[] args) {
    List<Student> list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>();
      tmp.head = student;
      tmp.tail = list;
      list = tmp;
    }
  }
}
```

Reachability

a reaches *b*, i.e., $a \rightsquigarrow b$ iff

a reaches a location bound to *b*

tmp $\rightsquigarrow$ student

A Simple Example - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public static void main(String[] args) {
    List<Student> list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>();
      tmp.head = student;
      tmp.tail = list;
      list = tmp;
    }
  }
}
```

Reachability

a reaches *b*, i.e., $a \rightsquigarrow b$ iff

a reaches a location bound to *b*

tmp	$\rightsquigarrow$	student
list	$\rightsquigarrow$	tmp

A Simple Example - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public static void main(String[] args) {
    List<Student> list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>();
      tmp.head = student;
      tmp.tail = list;
      list = tmp;
    }
  }
}
```

Reachability

a reaches b, i.e., $a \rightsquigarrow b$ iff

a reaches a location bound to b

tmp	$\rightsquigarrow$	student
list	$\rightsquigarrow$	tmp
list	$\rightsquigarrow$	student

A Simple Example 2 - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public List(Student head, List<Student> tail) {
    this.head = head;
    this.tail = tail;
  }

  public static void main(String[] args) {
    ListStudent list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>(student, list);
      list = tmp;
    }
  }
}
```

A Simple Example 2 - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public List(Student head, List<Student> tail) {
    this.head = head;
    this.tail = tail;
  }

  public static void main(String[] args) {
    ListStudent list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>(student, list);
      list = tmp;
    }
  }
}
```

Reachability

a reaches *b*, i.e., $a \rightsquigarrow b$ iff

a reaches a location bound to *b*

A Simple Example 2 - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public List(Student head, List<Student> tail) {
    this.head = head;
    this.tail = tail;
  }

  public static void main(String[] args) {
    ListStudent list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>(student, list);
      list = tmp;
    }
  }
}
```

Reachability

a reaches *b*, i.e., $a \rightsquigarrow b$ iff

a reaches a location bound to *b*

tmp	$\rightsquigarrow$	student
tmp	$\rightsquigarrow$	list

A Simple Example 2 - List of Students

```
public class Student {
  String name;
  ...
}
 public class List<Student> {
  public Student head;
  public List<Student> tail;

  public List(Student head, List<Student> tail) {
    this.head = head;
    this.tail = tail;
  }

  public static void main(String[] args) {
    ListStudent list = null;
    for (int i = 1; i <= n; i++) {
      Student student = new Student(i);
      List<Student> tmp = new List<Student>(student, list);
      list = tmp;
    }
  }
```

Reachability

a reaches *b*, i.e., $a \rightsquigarrow b$ iff

a reaches a location bound to *b*

tmp	$\rightsquigarrow$	student
list	$\rightsquigarrow$	student
list	$\rightsquigarrow$	tmp

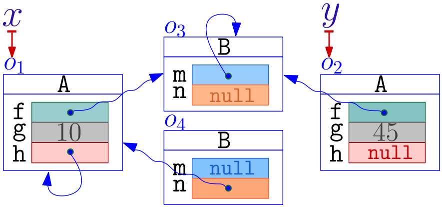
Haven't we solved this problem yet?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

Haven't we solved this problem yet?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

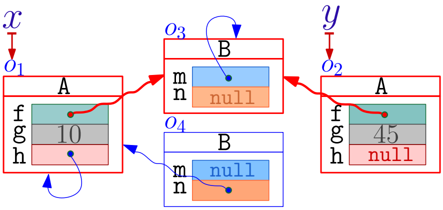
- Sharing Analysis



Haven't we solved this problem yet?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

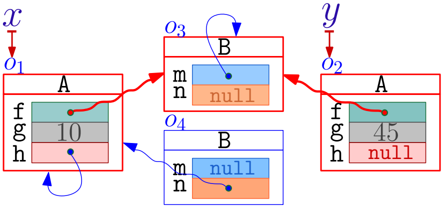
- Sharing Analysis



Haven't we solved this problem yet?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

- Sharing Analysis

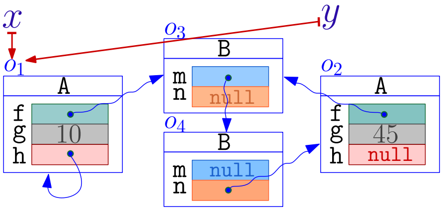


- REACHABILITY ENTAILS SHARING
- SHARING ~~ENTAILS~~ REACHABILITY

Haven't we solved this problem yet?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

- Sharing Analysis
- Aliasing Analysis



- ALIASING ENTAILS REACHABILITY
- REACHABILITY ~~ENTAILS~~ ALIASING

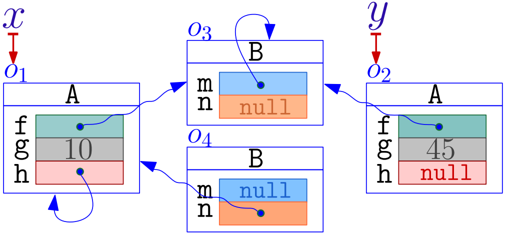
Where can it be useful?

Cyclicity Analysis: An assignment $y.\mathrm{h} = x$ might make y cyclical?

Where can it be useful?

Cyclicity Analysis: An assignment $y.\mathtt{h} = x$ might make y cyclical?

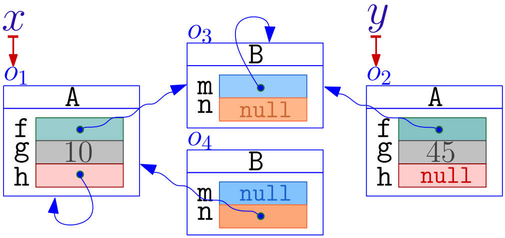
"Sharing" approach



Where can it be useful?

Cyclicity Analysis: An assignment y.h $= x$ might make y cyclical?

"Sharing" approach

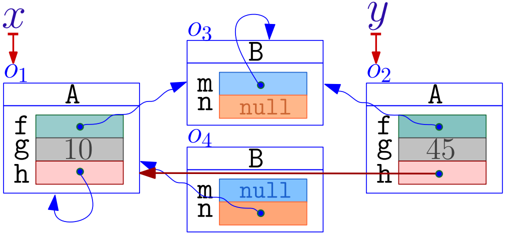


y.h$=x$ makes y cyclical?

Where can it be useful?

Cyclicity Analysis: An assignment y.h $= x$ might make y cyclical?

"Sharing" approach

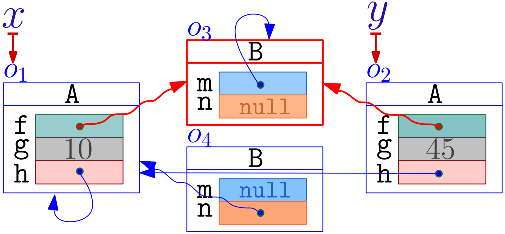


y.h$=x$ makes y cyclical?

Where can it be useful?

Cyclicity Analysis: An assignment y.h $= x$ might make y cyclical?

"Sharing" approach

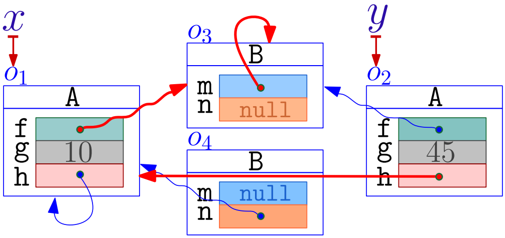


y.h$=x$ makes y cyclical?
if x shares with y?

Where can it be useful?

Cyclicity Analysis: An assignment y.h $= x$ might make y cyclical?

"Sharing" approach

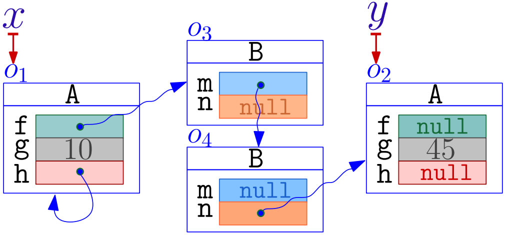


y.h$=x$ makes y cyclical? No!

Where can it be useful?

Cyclicity Analysis: An assignment $y.\mathtt{h} = x$ might make y cyclical?

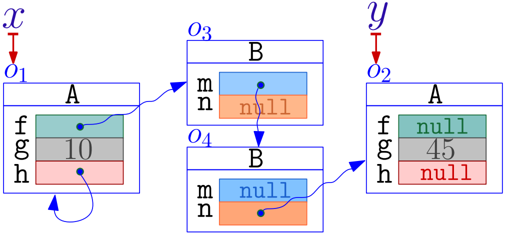
"Reachability" approach



Where can it be useful?

Cyclicity Analysis: An assignment y.h $= x$ might make y cyclical?

"Reachability" approach

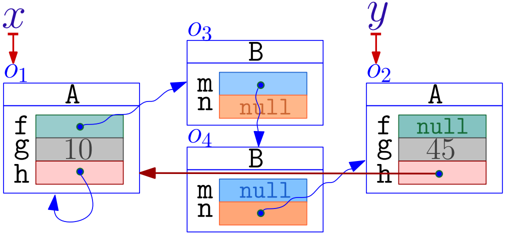


y.h$=x$ makes y cyclical?

Where can it be useful?

Cyclicity Analysis: An assignment $y.\mathtt{h} = x$ might make y cyclical?

"Reachability" approach

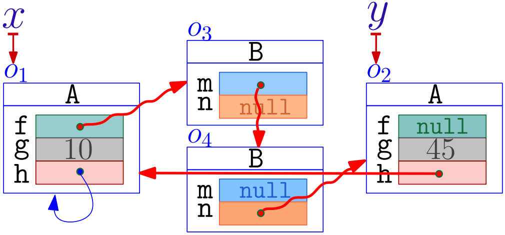


$y.\mathtt{h}{=}x$ makes y cyclical?
if x reaches y

Where can it be useful?

Cyclicity Analysis: An assignment y.h $= x$ might make y cyclical?

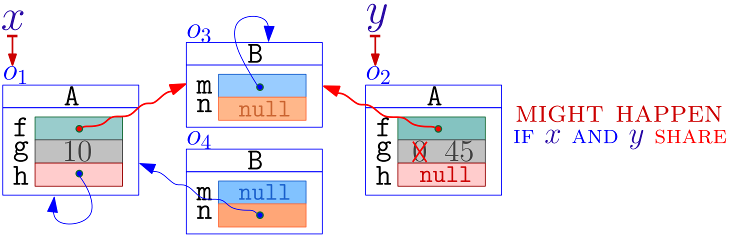
"Reachability" approach



y.h$=x$ makes y cyclical? Yes!

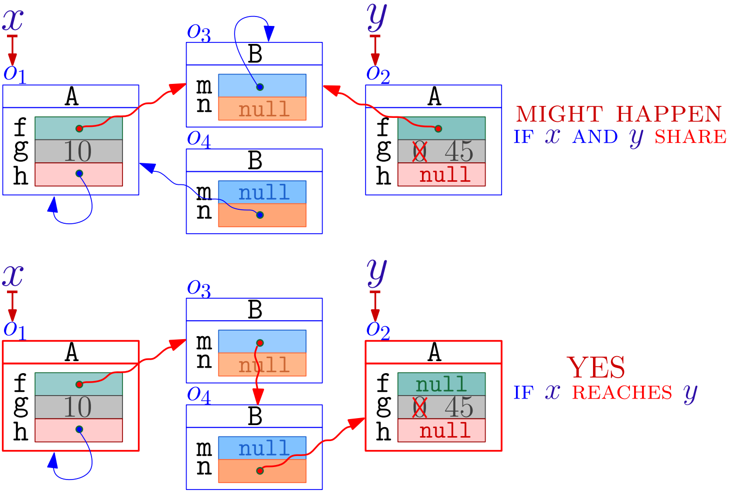
Where can it be useful?

Side-effects Analysis: An assignment $y.g = 45$ might affect a parameter x of a method m?



Where can it be useful?

Side-effects Analysis: An assignment y.g $= 45$ might affect a parameter x of a method m?



Target language: a fragment of java bytecode

`const` v
`dup` t
`load` k t
`store` k t
`ifne` t
`new` κ
`getfield` $\kappa.f{:}\mathsf{t}$
`putfield` $\kappa.f{:}\mathsf{t}$
`throw` κ
`catch`
`exception_is` K

Target language: a fragment of Java bytecode

const v
dup t
load k t   Basic Instructions
store k t
ifne t
new κ
getfield $\kappa.f$:t
putfield $\kappa.f$:t
throw κ
catch
exception_is K

Target language: a fragment of Java bytecode

`const` v
`dup` t
`load` k t
`store` k t
`ifne` t
`new` κ
`getfield` $\kappa.f$:t   Object-manipulating
`putfield` $\kappa.f$:t
`throw` κ
`catch`
`exception_is` K

Target language: a fragment of Java bytecode

`const` v
`dup` t
`load` k t
`store` k t
`ifne` t
`new` κ
`getfield` $\kappa.f$: t
`putfield` $\kappa.f$: t
`throw` κ
`catch` Exception-handling
`exception_is` K

Target language: a fragment of Java bytecode

`const` v
`dup` t
`load` k t
`store` k t
`ifne` t
`new` κ
`getfield` $\kappa.f$:t
`putfield` $\kappa.f$:t
`throw` κ
`catch`
`exception_is` K

Our implementation handles all Java types and bytecodes.

Target language: A Fragment of Java Bytecode

```
...
tmp.tail = list;
...
```

tmp ⟷ l_4
list ⟷ l_1

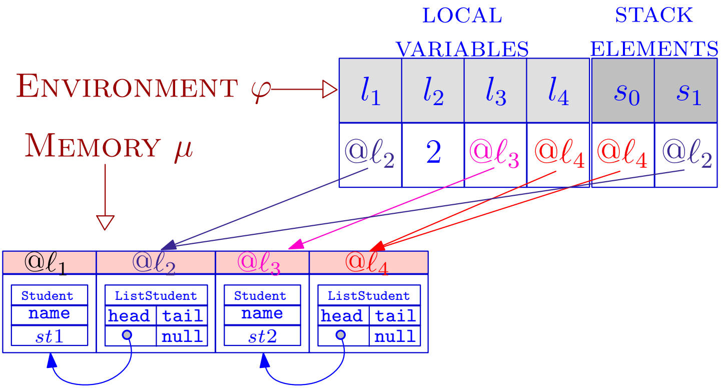
```
load 4 List
load 1 List
putfield List.tail : List
```

State

Some definitions:

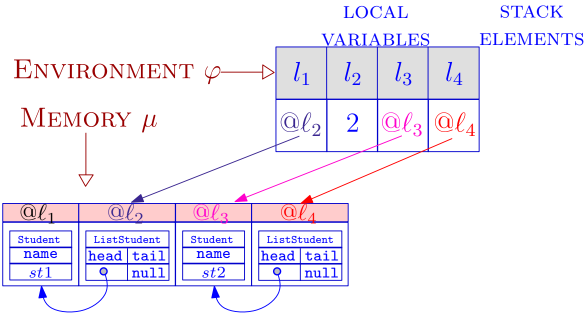
- we distinguish local ($\mathsf{L} = \{l_0, l_1, \ldots\}$) and stack ($\mathsf{S} = \{s_0, s_1, \ldots\}$) variables;
- values can be integers ($\mathbb{Z}$), locations ($\mathbb{L} = \{@\ell_1, \ldots\}$) and $\mathtt{null}$;
- objects contain fields and have methods;
- environments map variables into values $\varphi : \mathsf{L} \cup \mathsf{S} \to \mathbb{Z} \cup \mathbb{L} \cup \{\mathtt{null}\}$;
- memories μ map locations to objects;
- states are tuples $\langle \varphi, \mu \rangle$;
- Σ denotes the set of all possible states.

State



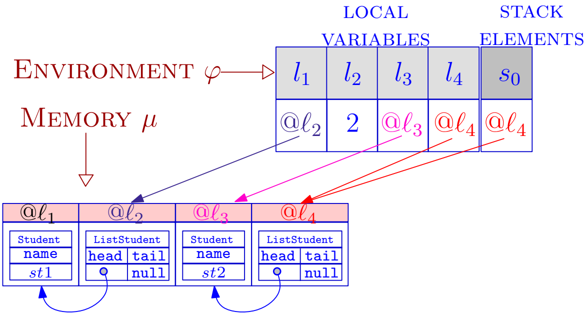
$$\sigma = \langle\langle \underbrace{[@\ell_2, 2, @\ell_3, @\ell_4]}_{L} \,||\, \underbrace{@\ell_2 :: @\ell_4}_{S} \rangle, \mu\rangle$$

Semantics of tmp.tail = list at bytecode level



load 4 List
load 1 List
putfield List.tail: List

Semantics of $\texttt{tmp.tail} = \texttt{list}$ at bytecode level

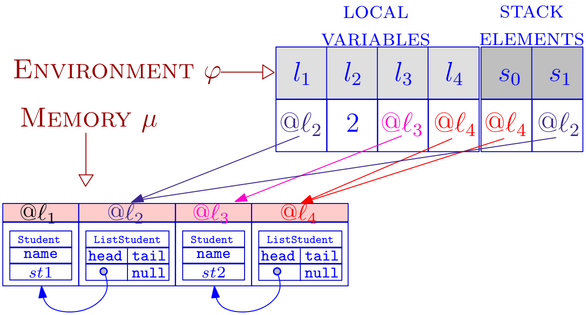


load 4 List

load 1 List

putfield List.tail: List

Semantics of tmp.tail = list at bytecode level

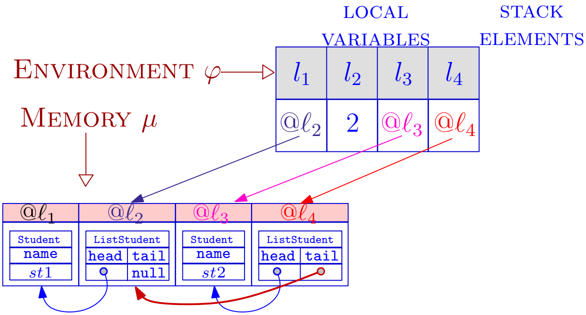


load 4 List

load 1 List

putfield List.tail: List

Semantics of tmp.tail = list at bytecode level



load 4 List
load 1 List
putfield List.tail: List

Reachable locations and variables

Reachable locations $L_\sigma(a)$

Given a state $\sigma = \langle \varphi, \mu \rangle$ and a location $@\ell$, locations reachable from $@\ell$ in σ are $L_\sigma(@\ell) = lfp_{i \geq 0} L_\sigma^i(@\ell)$, where $L_\sigma^i(@\ell)$ represents the set of locations reachable from $@\ell$ in i steps, i.e.,

$$L_\sigma^i(@\ell) = \begin{cases} \{@\ell\} & \text{if } i = 0 \\ \bigcup_{@\ell_1 \in L_\sigma^{i-1}(@\ell)} (\mathrm{rng}(\mu(@\ell_1).\phi) \cap \mathbb{L}) \cup L_\sigma^{i-1}(@\ell) & \text{otherwise.} \end{cases}$$

Reachable locations and variables

Reachable locations $\mathsf{L}_\sigma(a)$

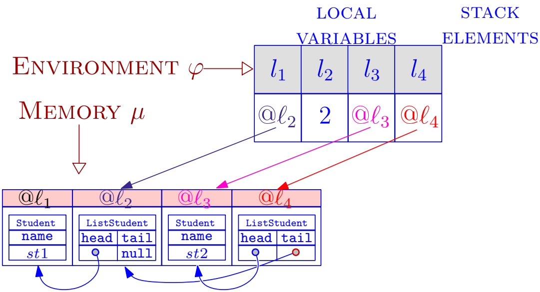
Given a state $\sigma = \langle \varphi, \mu \rangle$ and a location $@\ell$, locations reachable from $@\ell$ in σ are $\mathsf{L}_\sigma(@\ell) = \mathit{lfp}_{i \geq 0} \mathsf{L}^i_\sigma(@\ell)$, where $\mathsf{L}^i_\sigma(@\ell)$ represents the set of locations reachable from $@\ell$ in i steps, i.e.,

$$\mathsf{L}^i_\sigma(@\ell) = \begin{cases} \{@\ell\} & \text{if } i = 0 \\ \bigcup\limits_{@\ell_1 \in \mathsf{L}^{i-1}_\sigma(@\ell)} (\mathrm{rng}(\mu(@\ell_1).\phi) \cap \mathbb{L}) \cup \mathsf{L}^{i-1}_\sigma(@\ell) & \text{otherwise.} \end{cases}$$

Reachability of variables $a \rightsquigarrow^\sigma b$

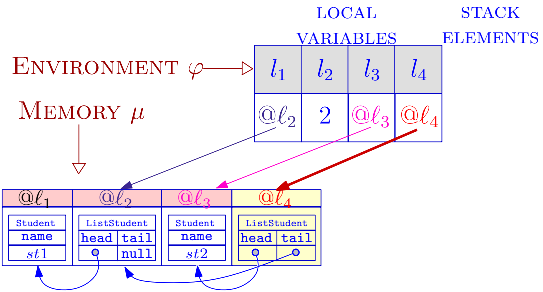
We say that a variable b is reachable from a variable a in σ, and we denote it $a \rightsquigarrow^\sigma b$ iff $\varphi(a), \varphi(b) \in \mathbb{L}$ and $\varphi(b) \in \mathsf{L}_\sigma(a)$.

Reachable locations and variables



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

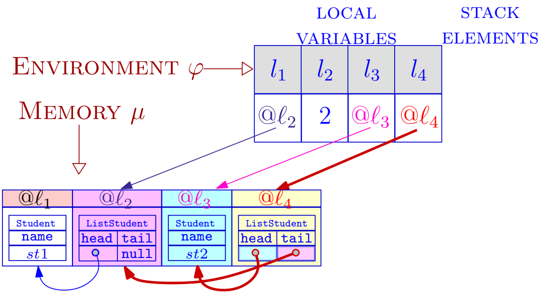
REACHABLE LOCATIONS AND VARIABLES



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

$$\mathsf{L}^0_\sigma(@\ell_4) \quad = \quad \{@\ell_4\}$$

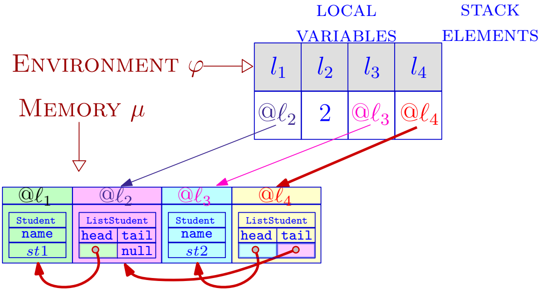
Reachable locations and variables



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

$$\mathsf{L}^0_\sigma(@\ell_4) = \{@\ell_4\}$$
$$\mathsf{L}^1_\sigma(@\ell_4) = \{@\ell_2, @\ell_3, @\ell_4\}$$

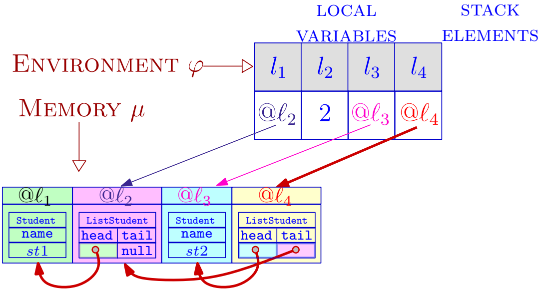
REACHABLE LOCATIONS AND VARIABLES



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

$$\begin{aligned}
\mathsf{L}^0_\sigma(@\ell_4) &= \{@\ell_4\}\\
\mathsf{L}^1_\sigma(@\ell_4) &= \{@\ell_2, @\ell_3, @\ell_4\}\\
\mathsf{L}^2_\sigma(@\ell_4) &= \{@\ell_1, @\ell_2, @\ell_3, @\ell_4\} \quad \Rightarrow \boxed{\mathsf{L}_\sigma(@\ell_4) = \{@\ell_1, @\ell_2, @\ell_3, @\ell_4\}}
\end{aligned}$$

Reachable locations and variables



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

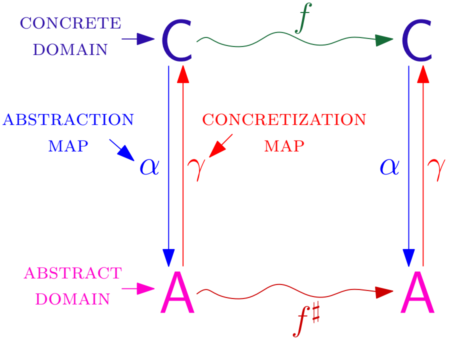
$$\mathsf{L}^0_\sigma(@\ell_4) = \{@\ell_4\}$$
$$\mathsf{L}^1_\sigma(@\ell_4) = \{@\ell_2, @\ell_3, @\ell_4\}$$
$$\mathsf{L}^2_\sigma(@\ell_4) = \{@\ell_1, @\ell_2, @\ell_3, @\ell_4\} \quad \Rightarrow \quad \mathsf{L}_\sigma(@\ell_4) = \{@\ell_1, @\ell_2, @\ell_3, @\ell_4\}$$

$$\varphi(l_4) = @\ell_4 \quad \Rightarrow \quad l_4 \rightsquigarrow^\sigma l_4$$
$$\varphi(l_1) = @\ell_2 \quad \Rightarrow \quad l_4 \rightsquigarrow^\sigma l_1$$
$$\varphi(l_3) = @\ell_3 \quad \Rightarrow \quad l_4 \rightsquigarrow^\sigma l_3$$

Formal definition **depends on the current program state**, i.e., on **ONE PARTICULAR EXECUTION**.

We want to determine an **approximation** of the reachability holding for **ANY POSSIBLE EXECUTION**.

Abstract Interpretation Framework [CousotCousot77]



BEST CORRECT APPROXIMATION: $f^{bca} = \alpha \circ f \circ \gamma$

IN PRACTICE: $f^\sharp$ IS LESS PRECISE THAN f^{bca} AND INTRODUCES OVER-APPROXIMATION

Concrete and Abstract Domains

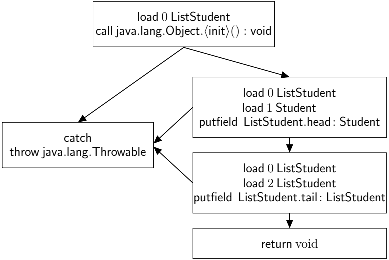
- Σ - set of all states
- V - set of all variables
- Concrete Domain: $C = \langle \wp(\Sigma), \subseteq \rangle$
- Abstract Domain: $A = \langle \wp(V \times V), \subseteq \rangle$
  - an abstract element $R \in A$ represents those concrete states whose reachability information is over-approximated by the pairs of variables in R
  - we write $a \rightsquigarrow b$ to denote $\langle a, b \rangle$
- Concretization Map:

$$\gamma(R) = \{\sigma \in \Sigma \mid \forall a, b \in V. a \rightsquigarrow^{\sigma} b \Rightarrow a \rightsquigarrow b \in R\}$$

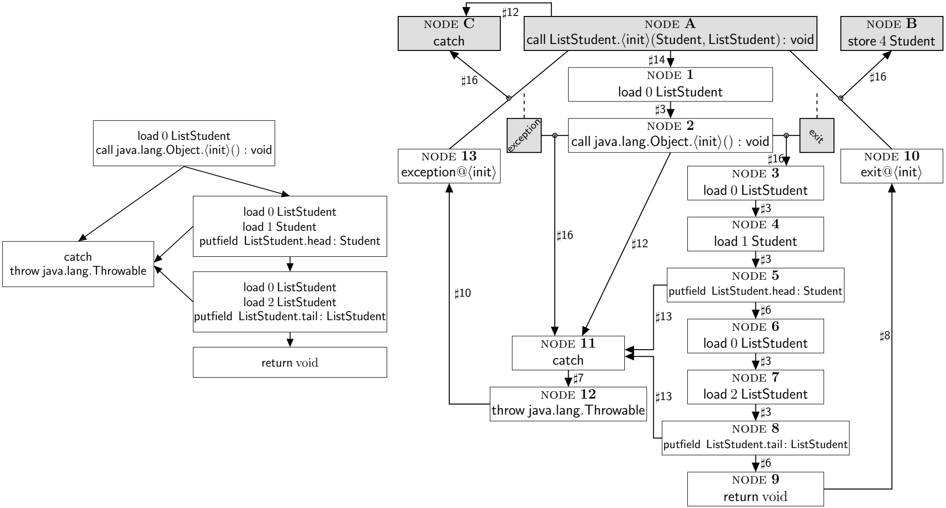
Constraint-based static analysis - example

- Abstract Constraint Graph (ACG= $\langle V, E \rangle$) gives rise to an over-approximation of the reachability information at each point of a program P.
- The CFG of P gives rise to the nodes and arcs of the ACG, i.e., there is a node for every bytecode and there is an arc between 2 nodes if their corresponding bytecodes are adjacent in the CFG.
- Each node is decorated by an abstract element, i.e., by a set of ordered pairs of variables representing an over-approximation of the reachability information at that point.
- Arcs propagate approximations of the reachability of their sources, i.e., they represent abstract semantics of bytecodes.
- The reachability information of the initial node, corresponding to the beginning of the `main` method is $\emptyset$, and it is propagated through the ACG.

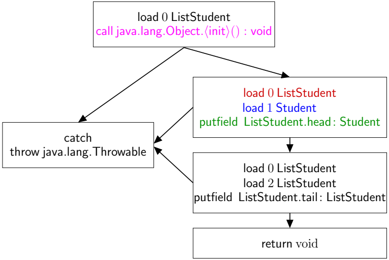
Constraint-based static analysis - example

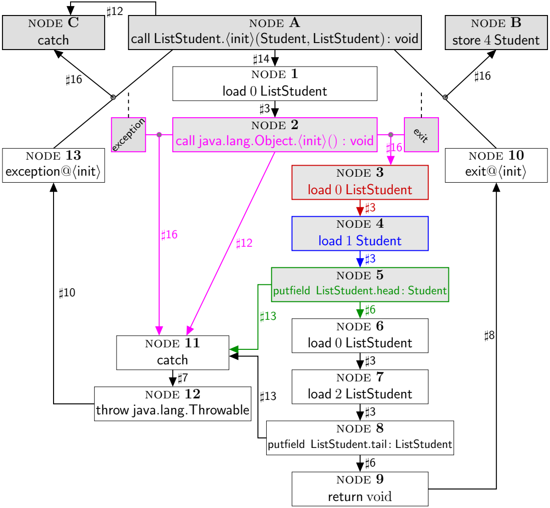


Constraint-based static analysis - example

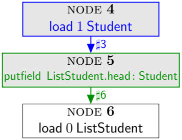


Constraint-based static analysis - example

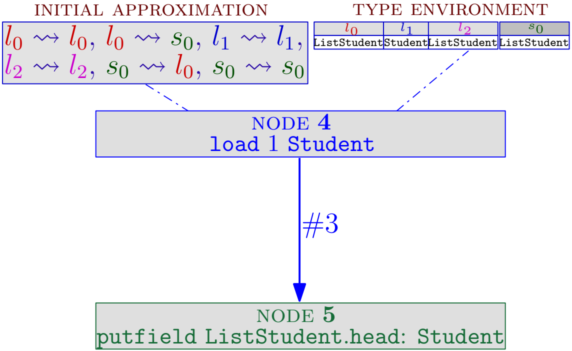




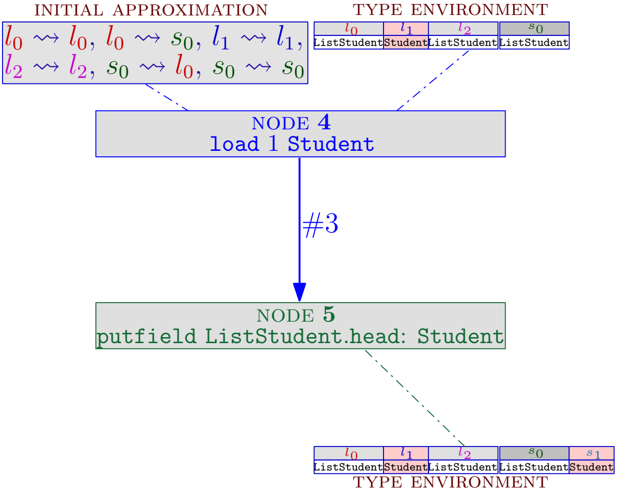
Constraint-based static analysis - example



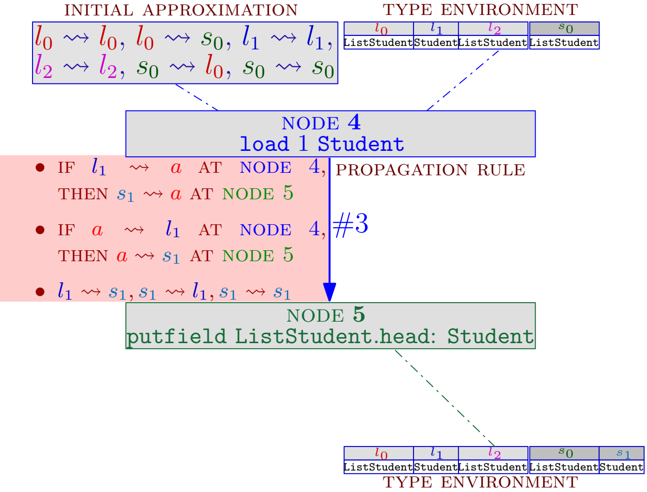
Propagation rules - example



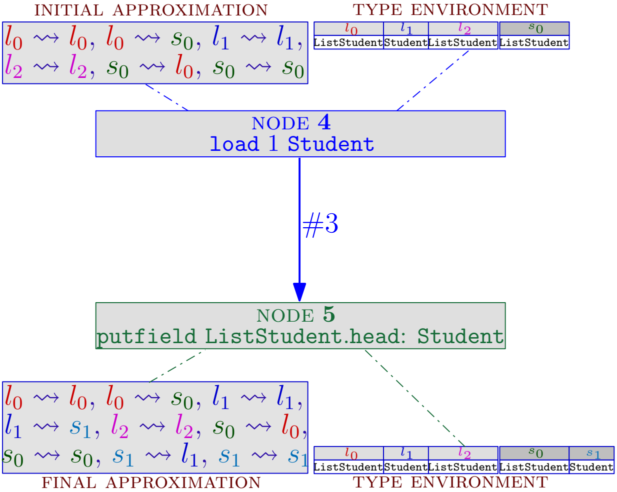
Propagation rules - example



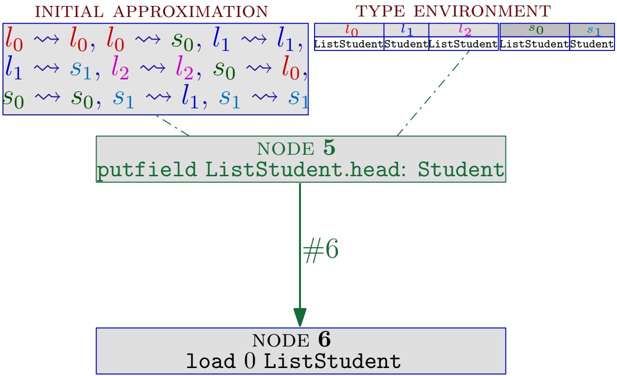
Propagation rules - example



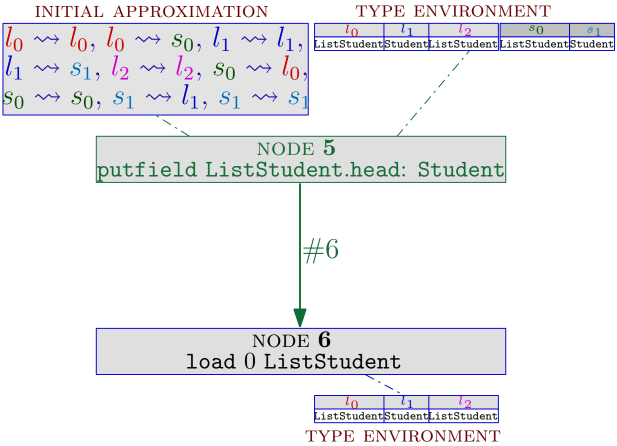
Propagation rules - example



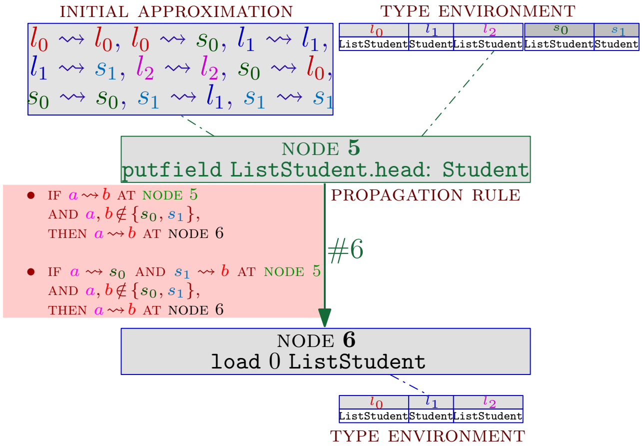
Propagation rules - example



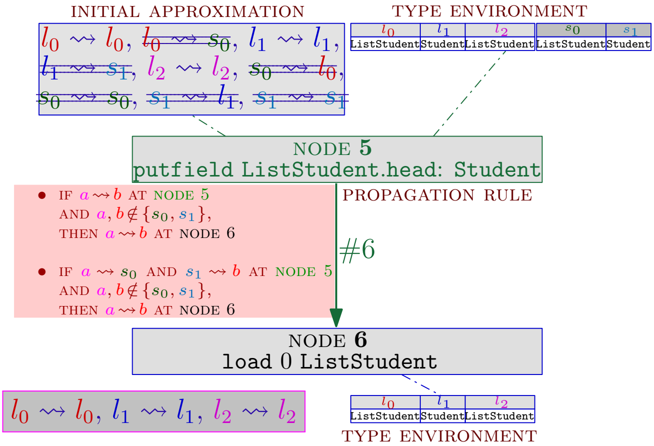
Propagation rules - example



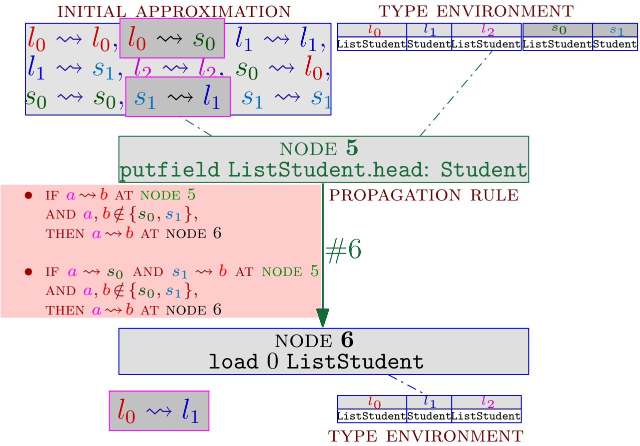
Propagation rules - example



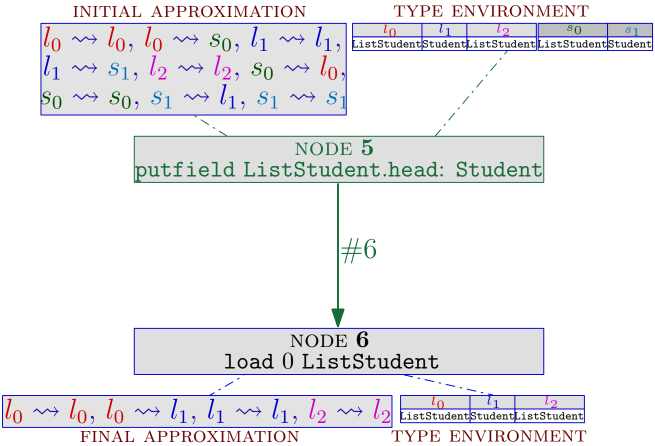
Propagation rules - example



PROPAGATION RULES - EXAMPLE



Propagation rules - example

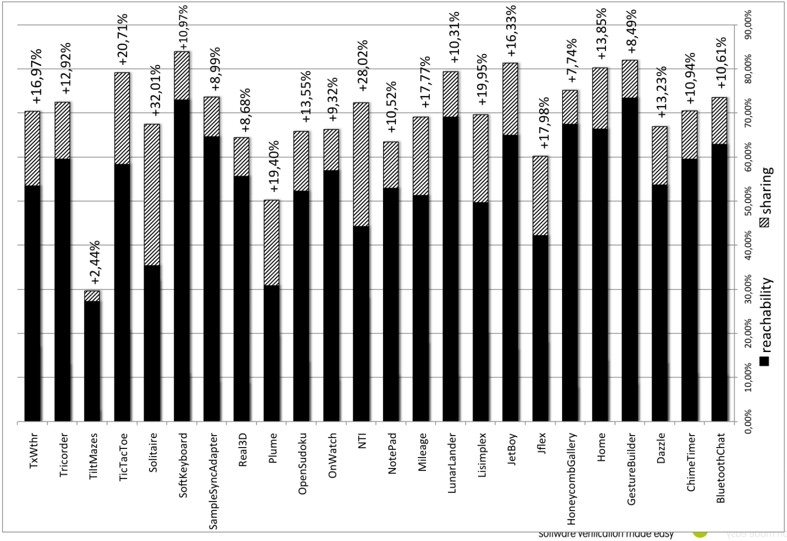


Soundness of our approach

Let ins and $\sigma \in \Sigma$ be a bytecode instruction and a state reached by an execution of the main method of a program, and let $R_{\mathsf{ins}} \in \mathsf{A}$ be the reachability approximation computed by our analysis at ins. Then,

$$\sigma \in \gamma(R_{\mathsf{ins}}).$$

Experimental evaluation with Julia - Sharing vs. Reachability



Experimental evaluation with Julia - Impact on other analyses

Reachability analysis	Side-effects analysis	Field initializat. analysis

Experimental evaluation with Julia - Impact on other analyses

REACHABILITY ANALYSIS	SIDE-EFFECTS ANALYSIS	FIELD INITIALIZAT. ANALYSIS
45.07%		

the ratio of pairs of variables $\langle v, w \rangle$ such that the analysis concludes that v might reach w, over the total number of pairs of variables of reference type: the lower the ratio, the higher the precision

Experimental evaluation with Julia - Impact on other analyses

REACHABILITY ANALYSIS	SIDE-EFFECTS ANALYSIS	FIELD INITIALIZAT. ANALYSIS
45.07%	−23.47%	

which parameters p of a method might be affected by its execution: the method might update a field of an object reachable from p:

the lower the numbers, the better the precision

Experimental evaluation with Julia - Impact on other analyses

REACHABILITY ANALYSIS	SIDE-EFFECTS ANALYSIS	FIELD INITIALIZAT. ANALYSIS
45.07%	−23.47%	+3.46%

the number of fields of reference type proven to be always initialized before being read, in all constructors of their defining class:
the higher the numbers, the better the precision

software verification made easy

Experimental evaluation with Julia - Impact on other analyses

Reachability analysis	Side-effects analysis	Field initializat. analysis
45.07%	−23.47%	+3.46%

	Nullness analysis	Termination analysis
runtime	−7.77%	−1.62%
warnings	−3.38%	0%

Static Analysis - main issues

Static Analysis of real life software is extremely difficult:

- complex semantics of current programming languages
- memory-related properties required
- side-effects of method calls
- instructions' exceptional behaviors
- libraries heavily used
- annotations help, but...
- formalization vs. implementation
- proof of Soundness is difficult

Static Analysis - main issues

Static Analysis of real life software is extremely difficult:

- complex semantics of current programming languages | Java bytecode
- memory-related properties required | reachability,sharing,aliasing,side-effects
- side-effects of method calls | ACG's SE arcs deal with them
- instructions' exceptional behaviors | ACG's exceptional arcs deal with them
- libraries heavily used | our CFG includes them
- annotations help, but... | we do not use annotations
- formalization vs. implementation | done
- proof of Soundness is difficult | our framework simplifies these proofs

Questions?