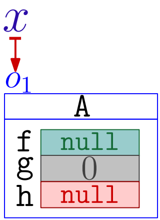
Reachability Analysis of Program Variables

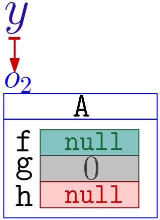
Đurica Nikolić[1,2] and Fausto Spoto[1]

1. - Dipartimento di Informatica, University of Verona (Italy)
2. - Microsoft Research - University of Trento Centre for Computational and Systems Biology

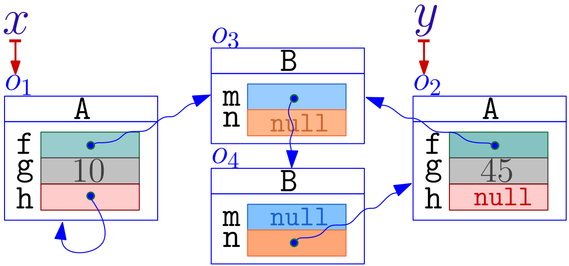
June 29th, 2012

Intuitive definition of Reachability



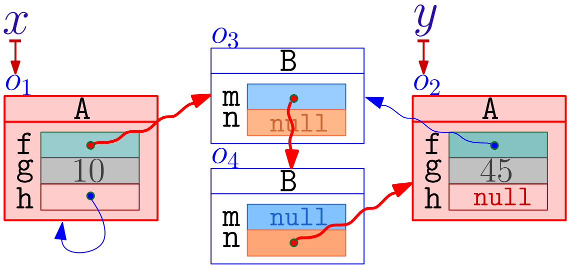


Intuitive definition of Reachability



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

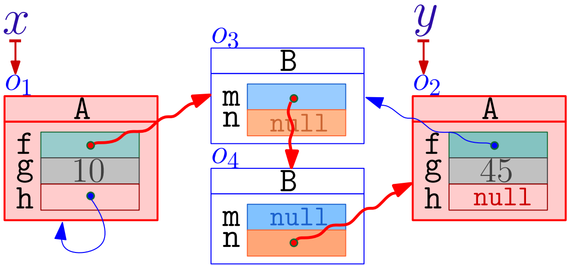
Intuitive definition of Reachability



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

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

Intuitive definition of Reachability



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

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

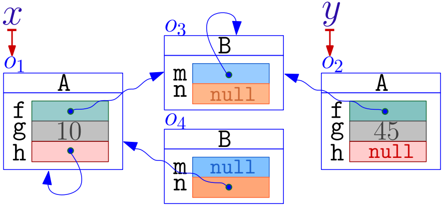
Haven't we solved this problem yet?

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

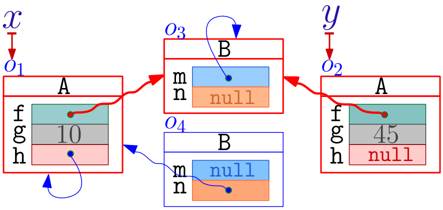
- Sharing Analysis



Haven't we solved this problem yet?

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

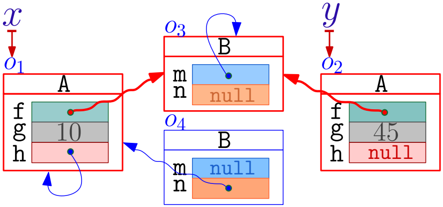
- Sharing Analysis



Haven't we solved this problem yet?

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

- Sharing Analysis

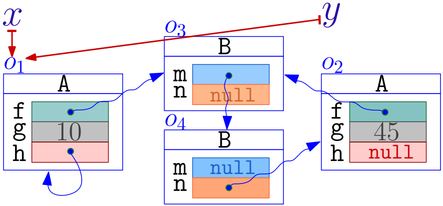


- REACHABILITY ENTAILS SHARING
- SHARING ~~ENTAILS~~ REACHABILITY

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

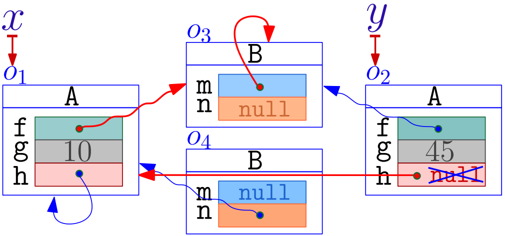
Haven't we solved this problem yet?

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

- Sharing Analysis
- Aliasing Analysis
- Shape Analysis

Where can it be useful?

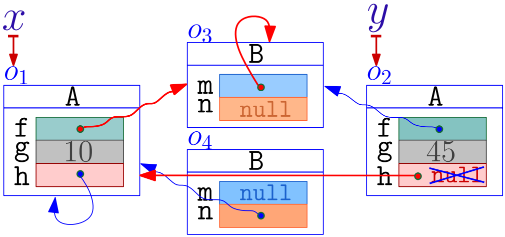
Cyclicity Analysis: An assignment $y.\mathtt{h} = x$ might make y cyclical?



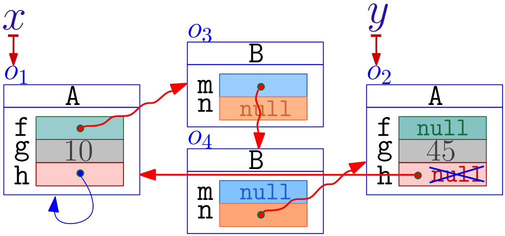
MIGHT HAPPEN
IF x AND y SHARE

Where can it be useful?

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



MIGHT HAPPEN
IF x AND y SHARE



YES
IF x REACHES y

Julia - a static analyzer for Java and Android

JULIA
software verification made easy

Reachability Analysis has been implemented inside Julia as a supporting analysis for

- ♦ Cyclicity Analysis
- ♦ Side-Effects Analysis
- ♦ Field Initialization Analysis
- ♦ Path-length Analysis

} supporting analyses of Nullness and Termination

Target language: Java bytecode

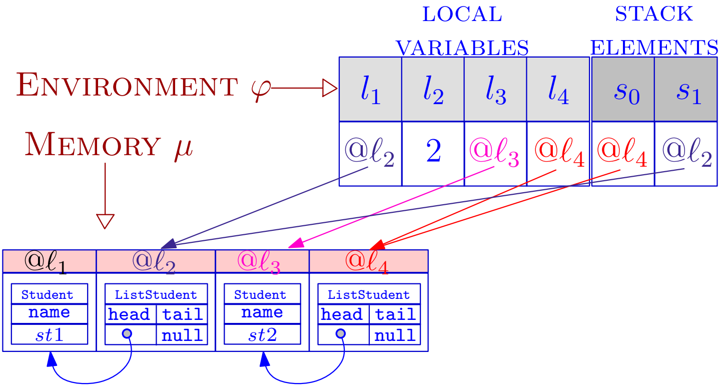
```
        ...
tmp.tail = list;
        ...
```

load 4 ListStudent
load 1 ListStudent
putfield ListStudent.tail : ListStudent

$\mathtt{tmp} \longleftrightarrow l_4$

$\mathtt{list} \longleftrightarrow l_1$

State



l_1	l_2	l_3	l_4	s_0	s_1
$@\ell_2$	2	$@\ell_3$	$@\ell_4$	$@\ell_4$	$@\ell_2$

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

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.,

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

Reachable locations and variables

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

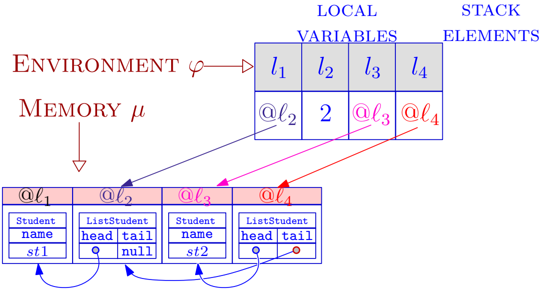
Given a state $\sigma = \langle \varphi, \mu \rangle$ and a location $@\ell$, locations reachable from $@\ell$ in σ are $\mathsf{L}_\sigma(@\ell) = \mathit{lfp}_{i \geq 0} \mathsf{L}^i_\sigma(@\ell)$, where $\mathsf{L}^i_\sigma(@\ell)$ represents the set of locations reachable from $@\ell$ in i steps,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

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

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



LOCAL VARIABLES

STACK ELEMENTS

ENVIRONMENT φ

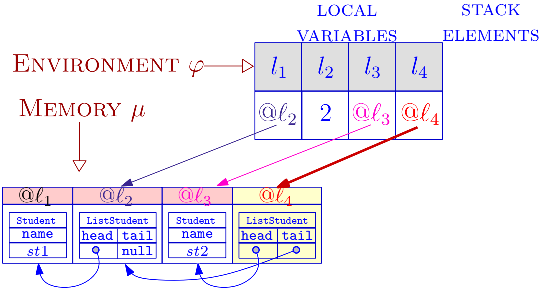
l_1	l_2	l_3	l_4
$@\ell_2$	2	$@\ell_3$	$@\ell_4$

MEMORY μ

$@\ell_1$	$@\ell_2$	$@\ell_3$	$@\ell_4$
Student name $st1$	ListStudent head tail null	Student name $st2$	ListStudent head tail

WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

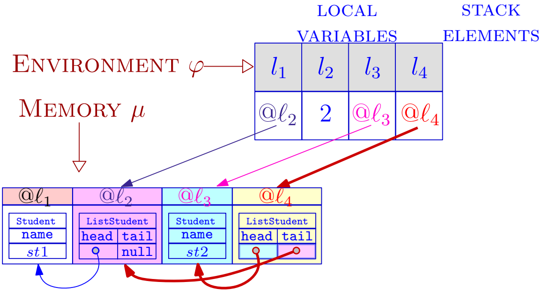
Reachable locations and variables



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

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

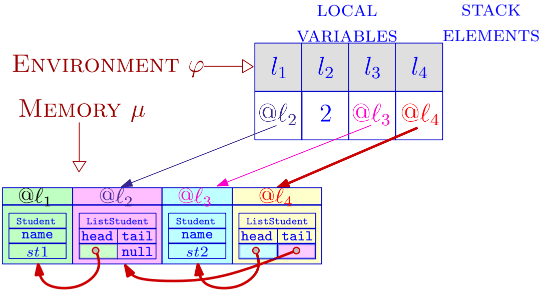
Reachable locations and variables



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

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

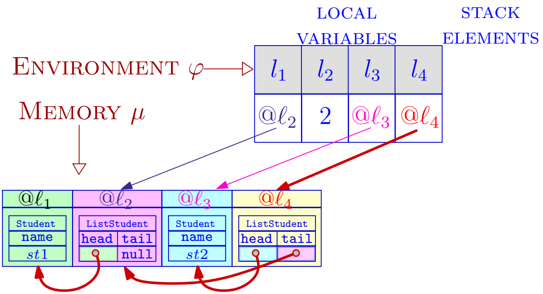
Reachable locations and variables



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

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

Reachable locations and variables



WHICH LOCATIONS ARE REACHABLE FROM $@\ell_4$?

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

$$\begin{aligned}
\varphi(l_4) = @\ell_4 &\quad\Rightarrow\quad l_4 \rightsquigarrow^\sigma l_4\\
\varphi(l_1) = @\ell_2 &\quad\Rightarrow\quad l_4 \rightsquigarrow^\sigma l_1\\
\varphi(l_3) = @\ell_3 &\quad\Rightarrow\quad l_4 \rightsquigarrow^\sigma l_3
\end{aligned}$$

Abstract Interpretation Framework [CousotCousot77]

CONCRETE DOMAIN

ABSTRACTION MAP

CONCRETIZATION MAP

ABSTRACT DOMAIN

$$\begin{tikzcd}
C \arrow[r, squiggly, "f"] \arrow[d, shift right, "\alpha"'] & C \arrow[d, shift right, "\alpha"'] \\
A \arrow[u, shift right, "\gamma"'] \arrow[r, squiggly, "f^{\sharp}"'] & A \arrow[u, shift right, "\gamma"']
\end{tikzcd}$$

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

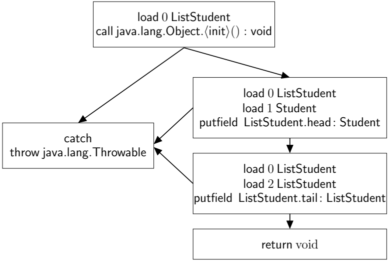
- Σ - set of all states
- V - set of all variables
- Concrete Domain: $C = \langle \wp(\Sigma), \subseteq \rangle$
- Abstract Domain: $A = \langle \wp(V \times V), \subseteq \rangle$
  - an abstract element $R \in A$ represents those concrete states whose reachability information is over-approximated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

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

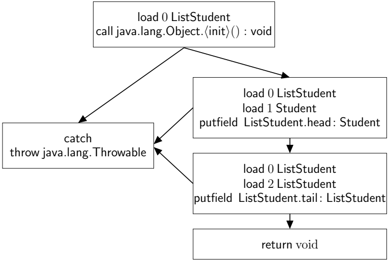
CONSTRAINT-BASED STATIC ANALYSIS - EXAMPLE

- 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


load 0 ListStudent
call java.lang.Object.⟨init⟩() : void
load 0 ListStudent
load 1 Student
putfield ListStudent.head : Student
catch
throw java.lang.Throwable
load 0 ListStudent
load 2 ListStudent
putfield ListStudent.tail : ListStudent
return void

Constraint-based static analysis - example



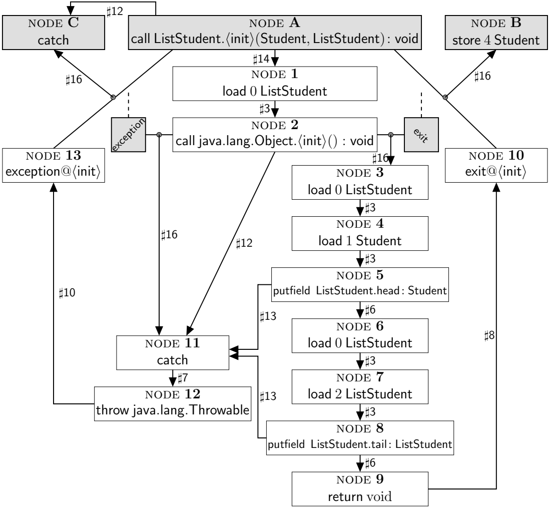
load 0 ListStudent
call java.lang.Object.⟨init⟩() : void

load 0 ListStudent
load 1 Student
putfield ListStudent.head : Student

catch
throw java.lang.Throwable

load 0 ListStudent
load 2 ListStudent
putfield ListStudent.tail : ListStudent

return void



NODE C
catch

NODE A
call ListStudent.⟨init⟩(Student, ListStudent) : void

NODE B
store 4 Student

NODE 1
load 0 ListStudent

NODE 2
call java.lang.Object.⟨init⟩() : void

NODE 13
exception@⟨init⟩

NODE 10
exit@⟨init⟩

NODE 3
load 0 ListStudent

NODE 4
load 1 Student

NODE 5
putfield ListStudent.head : Student

NODE 6
load 0 ListStudent

NODE 7
load 2 ListStudent

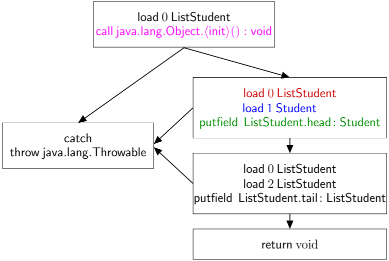
NODE 8
putfield ListStudent.tail : ListStudent

NODE 9
return void

NODE 11
catch

NODE 12
throw java.lang.Throwable

Constraint-based static analysis - example



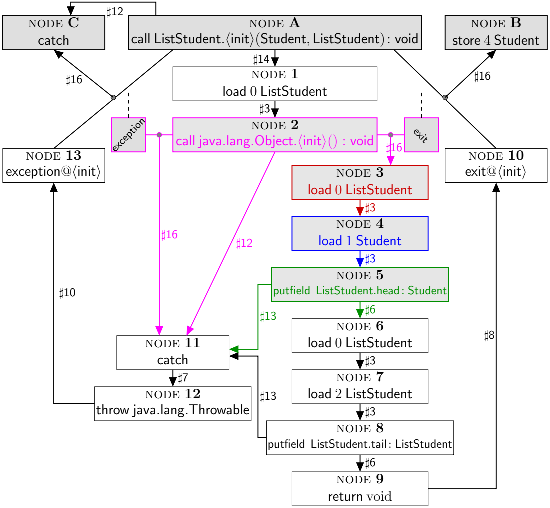
load 0 ListStudent
call java.lang.Object.⟨init⟩() : void

catch
throw java.lang.Throwable

load 0 ListStudent
load 1 Student
putfield ListStudent.head : Student

load 0 ListStudent
load 2 ListStudent
putfield ListStudent.tail : ListStudent

return void



NODE C
catch

NODE A
call ListStudent.⟨init⟩(Student, ListStudent) : void

NODE B
store 4 Student

NODE 1
load 0 ListStudent

NODE 2
call java.lang.Object.⟨init⟩() : void

exception

exit

NODE 13
exception@⟨init⟩

NODE 10
exit@⟨init⟩

NODE 3
load 0 ListStudent

NODE 4
load 1 Student

NODE 5
putfield ListStudent.head : Student

NODE 6
load 0 ListStudent

NODE 7
load 2 ListStudent

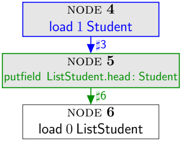
NODE 8
putfield ListStudent.tail : ListStudent

NODE 9
return void

NODE 11
catch

NODE 12
throw java.lang.Throwable

Constraint-based static analysis - example



NODE 4
load 1 Student

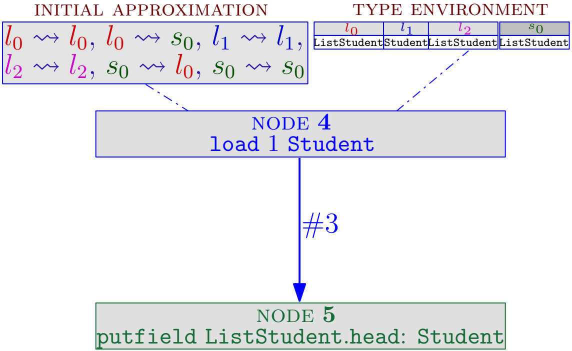
$\sharp 3$

NODE 5
putfield ListStudent.head : Student

$\sharp 6$

NODE 6
load 0 ListStudent

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,\ l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,\ s_0 \rightsquigarrow s_0$

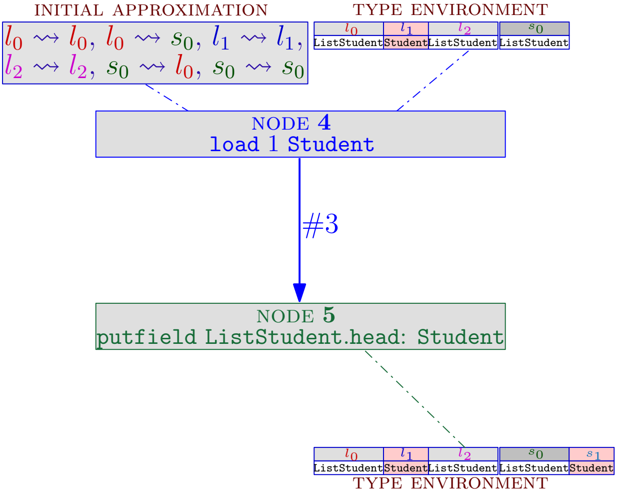
TYPE ENVIRONMENT

l_0	l_1	l_2	s_0
ListStudent	Student	ListStudent	ListStudent

NODE **4**
load 1 Student

NODE **5**
putfield ListStudent.head: Student

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,\ l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,\ s_0 \rightsquigarrow s_0$

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0
ListStudent	Student	ListStudent	ListStudent

NODE **4**

load 1 Student

$\#3$

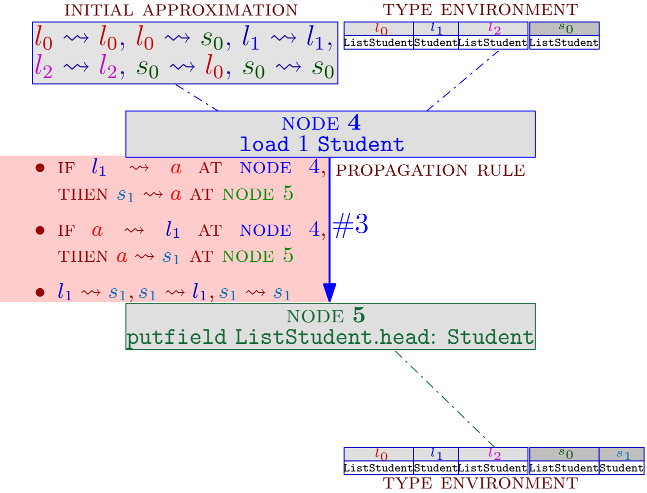
NODE **5**

putfield ListStudent.head: Student

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

TYPE ENVIRONMENT

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,$
$l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,\ s_0 \rightsquigarrow s_0$

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0
ListStudent	Student	ListStudent	ListStudent

NODE 4
`load 1 Student`

- IF $l_1 \rightsquigarrow a$ AT NODE 4, THEN $s_1 \rightsquigarrow a$ AT NODE 5
- IF $a \rightsquigarrow l_1$ AT NODE 4, THEN $a \rightsquigarrow s_1$ AT NODE 5
- $l_1 \rightsquigarrow s_1, s_1 \rightsquigarrow l_1, s_1 \rightsquigarrow s_1$

PROPAGATION RULE

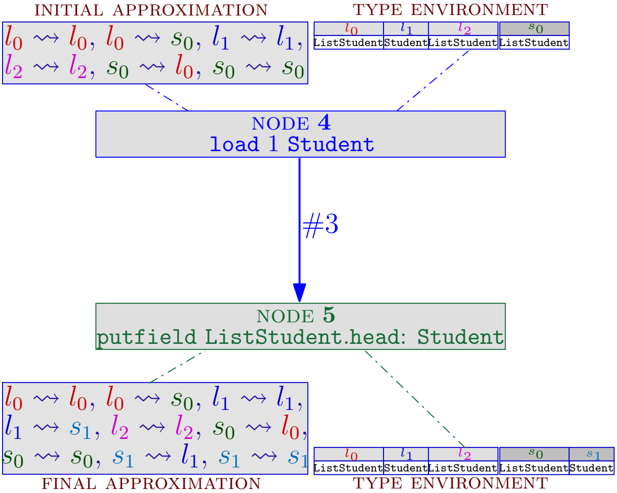
#3

NODE 5
`putfield ListStudent.head: Student`

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

TYPE ENVIRONMENT

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,$
$l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,\ s_0 \rightsquigarrow s_0$

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0
ListStudent	Student	ListStudent	ListStudent

NODE 4
load 1 Student

#3

NODE 5
putfield ListStudent.head: Student

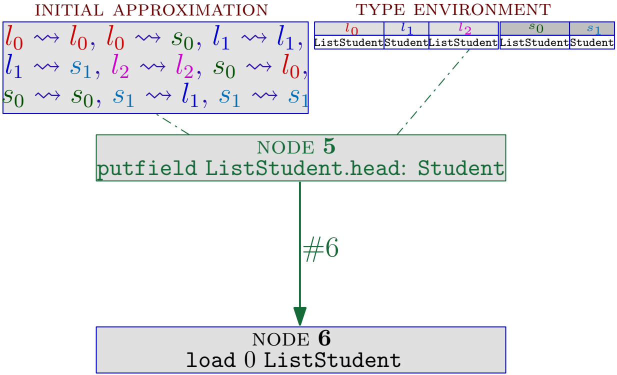
$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,$
$l_1 \rightsquigarrow s_1,\ l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,$
$s_0 \rightsquigarrow s_0,\ s_1 \rightsquigarrow l_1,\ s_1 \rightsquigarrow s_1$

FINAL APPROXIMATION

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

TYPE ENVIRONMENT

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,$
$l_1 \rightsquigarrow s_1,\ l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,$
$s_0 \rightsquigarrow s_0,\ s_1 \rightsquigarrow l_1,\ s_1 \rightsquigarrow s_1$

TYPE ENVIRONMENT

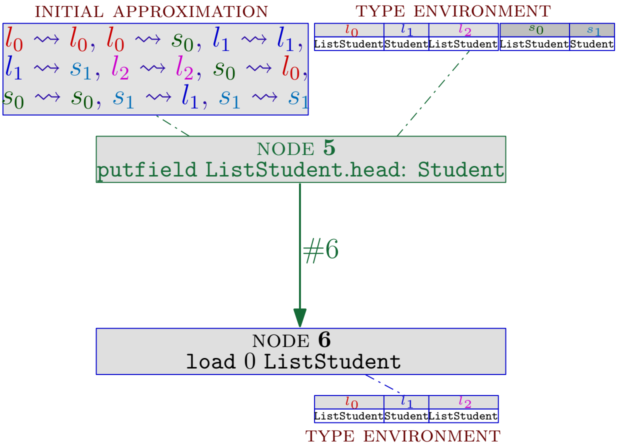
l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

NODE **5**
`putfield ListStudent.head: Student`

#6

NODE **6**
`load 0 ListStudent`

PROPAGATION RULES - EXAMPLE



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,\ l_1 \rightsquigarrow s_1,\ l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,\ s_0 \rightsquigarrow s_0,\ s_1 \rightsquigarrow l_1,\ s_1 \rightsquigarrow s_1$

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

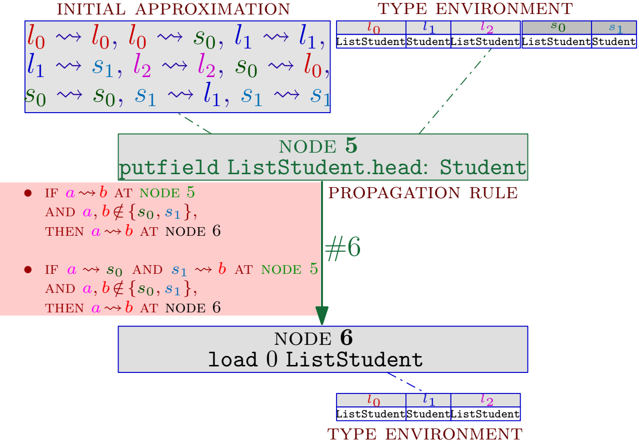
NODE **5**
putfield ListStudent.head: Student

NODE **6**
load 0 ListStudent

l_0	l_1	l_2
ListStudent	Student	ListStudent

TYPE ENVIRONMENT

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,\ l_1 \rightsquigarrow s_1,\ l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,\ s_0 \rightsquigarrow s_0,\ s_1 \rightsquigarrow l_1,\ s_1 \rightsquigarrow s_1$

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

NODE **5**
`putfield ListStudent.head: Student`

PROPAGATION RULE

#6

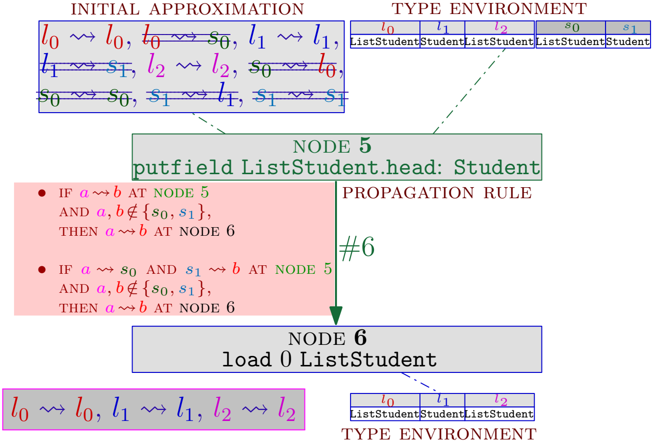
- IF $a \rightsquigarrow b$ AT NODE 5 AND $a, b \notin \{s_0, s_1\}$, THEN $a \rightsquigarrow b$ AT NODE 6
- IF $a \rightsquigarrow s_0$ AND $s_1 \rightsquigarrow b$ AT NODE 5 AND $a, b \notin \{s_0, s_1\}$, THEN $a \rightsquigarrow b$ AT NODE 6

NODE **6**
`load 0 ListStudent`

l_0	l_1	l_2
ListStudent	Student	ListStudent

TYPE ENVIRONMENT

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0$, ~~$l_0 \rightsquigarrow s_0$~~, $l_1 \rightsquigarrow l_1$, ~~$l_1 \rightsquigarrow s_1$~~, $l_2 \rightsquigarrow l_2$, ~~$s_0 \rightsquigarrow l_0$~~, ~~$s_0 \rightsquigarrow s_0$~~, ~~$s_1 \rightsquigarrow l_1$~~, ~~$s_1 \rightsquigarrow s_1$~~

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

NODE **5**
putfield ListStudent.head: Student

PROPAGATION RULE

- IF $a \rightsquigarrow b$ AT NODE 5 AND $a, b \notin \{s_0, s_1\}$, THEN $a \rightsquigarrow b$ AT NODE 6
- IF $a \rightsquigarrow s_0$ AND $s_1 \rightsquigarrow b$ AT NODE 5 AND $a, b \notin \{s_0, s_1\}$, THEN $a \rightsquigarrow b$ AT NODE 6

#6

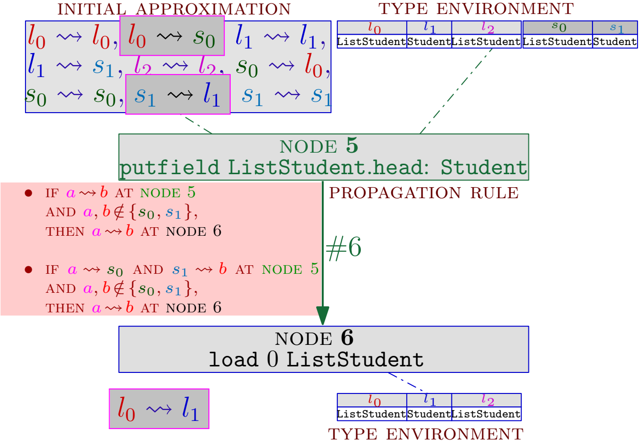
NODE **6**
load 0 ListStudent

$l_0 \rightsquigarrow l_0$, $l_1 \rightsquigarrow l_1$, $l_2 \rightsquigarrow l_2$

l_0	l_1	l_2
ListStudent	Student	ListStudent

TYPE ENVIRONMENT

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,$ $\boxed{l_0 \rightsquigarrow s_0}$ $l_1 \rightsquigarrow l_1,$
$l_1 \rightsquigarrow s_1,$ $l_2 \rightsquigarrow l_2,$ $s_0 \rightsquigarrow l_0,$
$s_0 \rightsquigarrow s_0,$ $\boxed{s_1 \rightsquigarrow l_1}$ $s_1 \rightsquigarrow s_1$

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

NODE **5**
putfield ListStudent.head: Student

PROPAGATION RULE

- IF $a \rightsquigarrow b$ AT NODE 5
AND $a, b \notin \{s_0, s_1\}$,
THEN $a \rightsquigarrow b$ AT NODE 6
- IF $a \rightsquigarrow s_0$ AND $s_1 \rightsquigarrow b$ AT NODE 5
AND $a, b \notin \{s_0, s_1\}$,
THEN $a \rightsquigarrow b$ AT NODE 6

#6

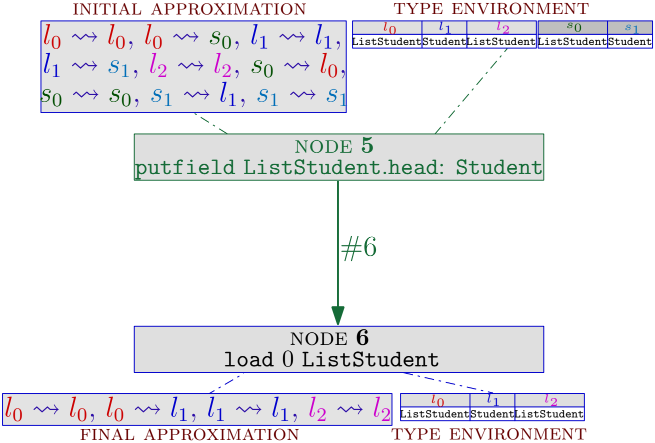
NODE **6**
load 0 ListStudent

$\boxed{l_0 \rightsquigarrow l_1}$

l_0	l_1	l_2
ListStudent	Student	ListStudent

TYPE ENVIRONMENT

Propagation rules - example



INITIAL APPROXIMATION

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow s_0,\ l_1 \rightsquigarrow l_1,\ l_1 \rightsquigarrow s_1,\ l_2 \rightsquigarrow l_2,\ s_0 \rightsquigarrow l_0,\ s_0 \rightsquigarrow s_0,\ s_1 \rightsquigarrow l_1,\ s_1 \rightsquigarrow s_1$

TYPE ENVIRONMENT

l_0	l_1	l_2	s_0	s_1
ListStudent	Student	ListStudent	ListStudent	Student

NODE **5**
putfield ListStudent.head: Student

#6

NODE **6**
load 0 ListStudent

$l_0 \rightsquigarrow l_0,\ l_0 \rightsquigarrow l_1,\ l_1 \rightsquigarrow l_1,\ l_2 \rightsquigarrow l_2$

FINAL APPROXIMATION

l_0	l_1	l_2
ListStudent	Student	ListStudent

TYPE ENVIRONMENT

REACHABILITY ANALYSIS	SIDE-EFFECTS ANALYSIS	FIELD INITIALIZAT. ANALYSIS

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

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

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

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%

Goal: define, formally prove correct and implement a Reachability Analysis of Program Variables for Java bytecode

1. definition a concrete operational semantics of Java bytecode;
2. formal definition a notion of reachability;
3. a constraint-based inter-procedural static analysis based on abstract interpretation;
4. formal proof of correctness of the analysis;
5. implementation of our inter-procedural analysis for full Java bytecode;
6. experimental evaluation of our approach.

THANK YOU!!!