Field-Sensitive Unreachability and Non-Cyclicity Analysis

Enrico Scapin and Fausto Spoto

Dipartimento di Informatica - University of Verona (Italy)

BYTECODE/ETAPS 2013
Static Analysis

Definition
Static analysis consists in building compile-time techniques in order to prove properties of programs before actually running them.

Shape Analyses try to understand how the program execution manipulates the heap.

e.g.,

- sharing analysis determines if two variables might be bound to overlapping data structures.
- reachability analysis determines if exists a path in memory that links two variables.
- cyclicity analysis determines if a variable is bound to a cyclical data structure.
Reachability and Cyclicity, state of the art:

- Stefano Rossignoli and Fausto Spoto, "Detecting non-cyclicity by abstract compilation into boolean functions". In: VMCAI'06
- Samir Genaim and Damiano Zanardini, "Reachability-based Acyclicity Analysis by Abstract Interpretation". In: CoRR'12
- Đurica Nikolić and Fausto Spoto, "Reachability Analysis of Program Variables". In: IJCAR'12

\[
\text{x.next}=\text{y}; \quad \text{This assignment makes x cyclical if and only if y reaches x.}
\]
State of the Art

Reachability and Cyclicity, state of the art:

- Stefano Rossignoli and Fausto Spoto, "Detecting non-cyclicity by abstract compilation into boolean functions". In: VMCAI’06
- Samir Genaim and Damiano Zanardini, "Reachability-based Acyclicity Analysis by Abstract Interpretation". In: CoRR’12
- Đurica Nikolić and Fausto Spoto, "Reachability Analysis of Program Variables". In: IJCAR’12

\[ x.\text{next}=y; \quad \text{This assignment makes } x \text{ cyclical if and only if } y \text{ reaches } x. \]

We defined a state as \( \sigma = \langle \rho, \mu \rangle \), where:

- \( \rho \) maps variables to locations;
- \( \mu \) binds locations to objects.
Reachability and Cyclicity, state of the art:

- Stefano Rossignoli and Fausto Spoto, "Detecting non-cyclicity by abstract compilation into boolean functions". In: VMCAI’06
- Samir Genaim and Damiano Zanardini, "Reachability-based Acyclicity Analysis by Abstract Interpretation". In: CoRR’12
- Đurica Nikolić and Fausto Spoto, "Reachability Analysis of Program Variables". In: IJCAR’12

\[\text{x.next}=\text{y};\]  
This assignment makes \(x\) cyclical if and only if \(y\) reaches \(x\).

We defined a state as \(\sigma = \langle \rho, \mu \rangle\), where:

- \(\rho\) maps variables to locations;
- \(\mu\) binds locations to objects.
Scenario

Given the following Java instructions,

```java
while (x != null)
    x = x.next;
```

Does the loop halt?
Given the following Java instructions,

```java
while (x != null)
    x = x.next;
```

Does the loop halt?

- Assuming $\rho(x) = l_1$ before starting the loop.

The loop terminates in 3 iterations!
Scenario

Given the following Java instructions,

```java
while (x != null) {
    x = x.next;
}
```

Does the loop halt?

- Assuming $\rho(x) = l_1$ before starting the loop.

It depends on the cyclicity of variable $x$. The loop does not terminate!
Can we refine them?

Yes, by developing a field-sensitive analysis!

```java
while (x != null) {
    x.next = y;
    x = x.next;
}
```

Goal
For each program point, maintain a set of static fields $F$ such that a program property holds.
Can we refine them?

Yes, by developing a field-sensitive analysis!

```java
while (x != null)
    x = x.next;
    x.next = y;
```

Goal

For each program point, maintain a set of static fields $F$ such that a program property holds.

We introduce the concept of path $P$ as a tuple of fields linking two locations inside the heap $\mu$.

e.g., $l_1 \xrightarrow{P}_\mu l_4$

with $P = \langle El.next, El.next, El.next \rangle$
Field-sensitive properties

Let
- $\mathcal{F}$: set of all fields;
- $L_\sigma(x)$: set of all locations reachable from $x$.

**Unreachability** for each path from $x$ to $y$ in state $\sigma$, the fields in $F$ are not part of that path.

$$\forall \mathcal{P} \subseteq \mathcal{F} \ (x \xrightarrow{P}_{\sigma} y \implies \mathcal{P} \cap F = \emptyset) \equiv x \not\xrightarrow{F}_{\sigma} y$$

**Non-cyclicity** for each cycle reachable from $x$ in state $\sigma$, the fields in $F$ are not part of the cycle.

$$\forall \ell \in L_\sigma(x), \forall \mathcal{P} \subseteq \mathcal{F} \ (\ell \xrightarrow{P}_{\mu} \ell \implies \mathcal{P} \cap F = \emptyset) \equiv x \xrightarrow{\emptyset F}_{\sigma}$$
Abstract Interpretation

In order to make our analysis computable, we use the general framework of Abstract Interpretation.
Concrete and Abstract Domains

- $\Sigma$ - set of all states
- $\mathcal{V}$ - set of all variables
- $\mathcal{F}$ - set of all program fields

- Concrete domain: $\mathcal{C} = \wp(\Sigma)$
- Abstract domain: $\mathcal{A} = \wp(\mathcal{V} \times \mathcal{V} \times \wp(\mathcal{F})) \cup \wp(\mathcal{V} \times \wp(\mathcal{F}))$
- Concretization map $\gamma: \mathcal{A} \rightarrow \mathcal{C}$

\[
\gamma(I \in \mathcal{A}) = \left\{ \sigma \in \Sigma \mid \left( \forall a \ni^F b \in I, \exists F' \subseteq \mathcal{F}. a \ni^F_{\sigma} b \land F \subseteq F' \right) \land \left( \forall c \ni^F \not\ni^F \in I, \exists F' \subseteq \mathcal{F}. c \ni^F_{\sigma} F' \land F \subseteq F' \right) \right\}
\]

Our properties are under-approximated by the information in $I$.  

Unreachability & Non-Cyclicity Analysis
Methodology

Program Under Analysis

class Element{
    private Object value;
    private Element prec, next;

    public Element(Object value){
        this.value=value;
    }
    public Element(Object value, Element prec){
        this.value=value;
        this.prec=prec;
        prec.next=this;
    }
}

public class MWexample{
    public static void main(String[] args){
        Element top = new Element(new Integer(0));
        for(int i=1;i<=3;i++)
            top = new Element(new Integer(i),top);
    }
}
### Program Under Analysis

```java
class Element {
    private Object value;
    private Element prec, next;

    public Element(Object value) {
        this.value = value;
    }

    public Element(Object value, Element prec) {
        this.value = value;
        this.prec = prec;
        prec.next = this;
    }

    public class MWexample {
        public static void main(String[] args) {
            Element top = new Element(new Integer(0));
            for (int i = 1; i <= 3; i++)
                top = new Element(new Integer(i), top);
        }
    }
}
```

### Java Bytecode

```java
invokespecial #1 <Object/<init>()V>
aload_0
aload_1
putfield #2 Element.value : Object
aload_0
aload_2
putfield #3 Element.prec : Element
aload_2
aload_0
putfield #4 Element.next : Element
return
```
Methodology

1. Program Under Analysis

class Element{
    private Object value;
    private Element prec, next;

    public Element(Object value) {
        this.value = value;
    }

    public Element(Object value, Element prec) {
        this.value = value;
        this.prec = prec;
        prec.next = this;
    }

    public class MWexample {
        public static void main(String[] args) {
            Element top = new Element(new Integer(0));
            for (int i = 1; i <= 3; i++)
                top = new Element(new Integer(i), top);
        }
    }
}

2. Java Bytecode

    invokespecial #1 <Object/<init>()V>
    aload_0
    aload_1
    putfield #2 Element.value: Object
    aload_0
    aload_2
    putfield #3 Element.prec: Element
    aload_2
    aload_0
    putfield #4 Element.next: Element
    return

3. Control Flow Graph
Constraint Based Static Analysis

From the Control Flow Graph we build the Abstract Constraint Graph

- **Nodes** represent bytecode instructions.
- **Arcs** represent the abstract semantics.
- Each node is decorated with an abstract set $I$.
- Each arc is decorated with a propagation rule.
- **Propagation Rules $\#i$**
  - defined for each type of arc, depending on its sources;
  - state how the information in each node is propagated.
Propagation Rules

- Their definitions can become complex whenever they exploit other static analyses.
- The unreachability and non-cyclicity information is propagated along the arcs of the ACG until reaching a fix-point.
- It exists since they are all monotonic functions.
- The fix-point is the maximal solution of the ACG with respect to the partial order $\supseteq$. 

Scapin and Spoto (univr.it) Unreachability & Non-Cyclicity Analysis BYTECODE’13 11 / 15
Example: putfield $\kappa.f:t$

\[ \lambda \langle \langle l, s_{j-1} :: s_{j-2} :: s \rangle, \mu \rangle . \langle \langle l, s \rangle, \mu[(\mu(s_{j-2}) \phi)(f) \rightarrow s_{j-1}] \rangle \]

local vars \hspace{1cm} stack vars

\[ \rho \]

$S_{j-2} \xrightarrow{f} S_{j-1}$

It changes the paths between locations!

How to correctly propagate the information w.r.t this instruction?

**KEY IDEA:** exploit the result of the possible reachability analysis.

\[ \langle x, y \rangle \notin M_R \implies x \not\sim y \]
e.g., field-sensitive unreachability

- for each $d \not{\rightarrow}^F w$ such that $d \not{\rightarrow} s_{j-2} \lor s_{j-1} \not{\rightarrow} w$, $F$ does not change after the putfield node.
- for each $a \not{\rightarrow}^F x$ such that $\langle a, s_{j-2} \rangle, \langle s_{j-1}, x \rangle \in \mathcal{MR}_\tau$, $F$ probably changes:
  - for sure, after the putfield, $F$ does not contain the field $f$!
Conclusions

1. Build an under-approximated analysis to state two field-sensitive properties.
Conclusions

1. Build an under-approximated analysis to state two field-sensitive properties.
2. Exploit the abstract interpretation framework to prove its correctness.

Future works: implementing this analysis in Julia Tool to improve the precision of its termination checker.
Conclusions

1. Build an under-approximated analysis to state two field-sensitive properties.

2. Exploit the abstract interpretation framework to prove its correctness.
   - each propagation rule $\Pi^{\#i}$ correctly approximates the set of states obtained by the correspondent instruction $\text{ins}^{\#i}$ execution:
     for each $I \in A$, \( \text{ins}(\gamma(I)) \subseteq \gamma(\Pi(I)) \)

Future works:
Implementing this analysis in Julia Tool to improve the precision of its termination checker.

Scapin and Spoto (univr.it)
Conclusions

1. Build an under-approximated analysis to state two field-sensitive properties.

2. Exploit the abstract interpretation framework to prove its correctness.
   - each propagation rule $\Pi^i$ correctly approximates the set of states obtained by the correspondent instruction $\text{ins}^i$ execution:
     
     $$
     \text{for each } I \in A, \quad \text{ins} \left( \gamma (I) \right) \subseteq \gamma (\Pi (I))
     $$
   
   - the analysis correctly approximates the semantics of the program with respect to the two properties defined:
     
     let $\Rightarrow^* \left< \text{ins} \parallel \sigma \right>$ be an execution and $l_{\text{ins}}$ the approx information,
     
     $$
     \sigma \in \gamma (l_{\text{ins}})
     $$

Future works:

- implementing this analysis in Julia Tool to improve the precision of its termination checker.
Conclusions

1. Build an under-approximated analysis to state two field-sensitive properties.

2. Exploit the abstract interpretation framework to prove its correctness.
   - each propagation rule \( \Pi^i \) correctly approximates the set of states obtained by the correspondent instruction \( \text{ins}^i \) execution:
     
     for each \( l \in A \), \( \text{ins} (\gamma(l)) \subseteq \gamma(\Pi(l)) \)
   - the analysis correctly approximates the semantics of the program with respect to the two properties defined:
     
     let \( \Rightarrow^* \langle \text{ins} \parallel \sigma \rangle \) be an execution and \( l_{\text{ins}} \) the approx information, \( \sigma \in \gamma(l_{\text{ins}}) \)

Future works: implementing this analysis in Julia Tool to improve the precision of its termination checker.
Thank You!