Type-Based Heap Cloning through Flow-Sensitivity

Candidature Asessment 1

Mohamad Barbar

date?

Outline

1. Review of first year
2. Background
3. Problem
4. Approach
5. Evaluation
6. Future

Review of first year

General

- Completed TRP (90%) and TRM (72%)
- Two accepted conference submissions on Control Flow Integrity
  - Short paper @ **ACISP 2018**
  - Post @ **ICSE 2018**
- Participated in the FEIT Research Showcase 2018
  - Third in group
- Helping supervise an undergraduate capstone student
- Fortnightly reading seminars at Data61

General

- Completed TRP (90%) and TRM (72%)
- Two accepted conference submissions on Control Flow Integrity
  - Short paper @ **ACISP 2018**
  - Post @ **ICSE 2018**
- Participated in the FEIT Research Showcase 2018
  - Third in group
- Helping supervise an undergraduate capstone student
- Fortnightly reading seminars at Data61

General

- Completed TRP (90%) and TRM (72%)
- Two accepted conference submissions on Control Flow Integrity
  - Short paper @ **ACISP 2018**
  - Post @ **ICSE 2018**
- Participated in the FEIT Research Showcase 2018
  - Third in group
- Helping supervise an undergraduate capstone student
- Fortnightly reading seminars at Data61

General

- Completed TRP (90%) and TRM (72%)
- Two accepted conference submissions on Control Flow Integrity
  - Short paper @ **ACISP 2018**
  - Post @ **ICSE 2018**
- Participated in the FEIT Research Showcase 2018
  - Third in group
- Helping supervise an undergraduate capstone student
- Fortnightly reading seminars at Data61

General

- Completed TRP (90%) and TRM (72%)
- Two accepted conference submissions on Control Flow Integrity
  - Short paper @ **ACISP 2018**
  - Post @ **ICSE 2018**
- Participated in the FEIT Research Showcase 2018
  - Third in group
- Helping supervise an undergraduate capstone student
- Fortnightly reading seminars at Data61

General

- Completed TRP (90%) and TRM (72%)
- Two accepted conference submissions on Control Flow Integrity
  - Short paper @ **ACISP 2018**
  - Post @ **ICSE 2018**
- Participated in the FEIT Research Showcase 2018
  - Third in group
- Helping supervise an undergraduate capstone student
- Fortnightly reading seminars at Data61

Technical

- Using NLP to analyse incomplete programs
- Cheap and precise call graph construction
  - Between pointer analysis and CHA/RTA/etc.
- Using types for faster analysis
- Exploring flow-sensitivity for type-based alias analysis
- **Type-based heap cloning through flow-sensitivity**

Technical

- Using NLP to analyse incomplete programs
- Cheap and precise call graph construction
  - Between pointer analysis and CHA/RTA/etc.
- Using types for faster analysis
- Exploring flow-sensitivity for type-based alias analysis
- **Type-based heap cloning through flow-sensitivity**

Technical

- Using NLP to analyse incomplete programs
- Cheap and precise call graph construction
  - Between pointer analysis and CHA/RTA/etc.
- Using types for faster analysis
- Exploring flow-sensitivity for type-based alias analysis
- **Type-based heap cloning through flow-sensitivity**

Technical

- Using NLP to analyse incomplete programs
- Cheap and precise call graph construction
  - Between pointer analysis and CHA/RTA/etc.
- Using types for faster analysis
- Exploring flow-sensitivity for type-based alias analysis
- **Type-based heap cloning through flow-sensitivity**

Technical

- Using NLP to analyse incomplete programs
- Cheap and precise call graph construction
  - Between pointer analysis and CHA/RTA/etc.
- Using types for faster analysis
- Exploring flow-sensitivity for type-based alias analysis
- **Type-based heap cloning through flow-sensitivity**

Background

Pointer analysis

Points-to analysis

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Alias analysis

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Alias analysis

- Answers: do `p` and `q` point to the same object?

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Alias analysis

- Answers: do `p` and `q` point to the same object?
  - Must they?

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Alias analysis

- Answers: do `p` and `q` point to the same object?
  - Must they?
  - May they?

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Alias analysis

- Answers: do `p` and `q` point to the same object?
  - Must they?
  - May they?
  - Must they not?

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Alias analysis

- Answers: do `p` and `q` point to the same object?
  - Must they?
  - May they?
  - Must they not?
- Can use points-to sets to compute, but more flexible

Pointer analysis

Points-to analysis

- Answers: what objects does pointer `p` point to?
- Computes *points-to sets* for every pointer

Alias analysis

- Answers: do `p` and `q` point to the same object?
  - Must they?
  - May they?
  - Must they not?
- Can use points-to sets to compute, but more flexible
  - We're only after a yes/no answer!

Objects (pointees)

Objects which can be pointed to are allocated as:

Objects (pointees)

Objects which can be pointed to are allocated as:

Stack objects:

```
    int i;
```

Objects (pointees)

Objects which can be pointed to are allocated as:

Stack objects:

```
int i;
```

Global objects:

```
int i;
```

Objects (pointees)

Objects which can be pointed to are allocated as:

Stack objects:

```
int i;
```

Global objects:

```
int i;
```

Heap objects:

```
int *x;
x = malloc(sizeof(int));
```

Problem

Problem

The heap is modelled **imprecisely**.

Imprecision #1. Single abstract object may represent *too* many concrete objects.

Problem

The heap is modelled **imprecisely**.

Imprecision #1. Single abstract object may represent *too* many concrete objects.

```
void *xmalloc(size_t s) {
    void *x = malloc(s); // returns abstract object 'o'
    assert(x && "allocation failed!");
    return x;
}
```

Problem

The heap is modelled **imprecisely**.

Imprecision #1. Single abstract object may represent *too* many concrete objects.

```
void *xmalloc(size_t s) {
    void *x = malloc(s); // returns abstract object 'o'
    assert(x && "allocation failed!");
    return x;
}
```

```
int           *a = xmalloc(sizeof(int          ));
std::string   *b = xmalloc(sizeof(std::string  ));
struct time_t *c = xmalloc(sizeof(struct time_t));
```

Problem

The heap is modelled **imprecisely**.

Imprecision #2. How can virtual calls from a heap object be resolved if heap objects are untyped?

Problem

The heap is modelled **imprecisely**.

Imprecision #2. How can virtual calls from a heap object be resolved if heap objects are untyped?

```
class A {
    void foo() { std::cout << "A::foo\n"; }
}

class A : public B {
    void foo() { std::cout << "B::foo\n"; }
}
```

Problem

The heap is modelled **imprecisely**.

Imprecision #2. How can virtual calls from a heap object be resolved if heap objects are untyped?

```
class A {
    void foo() { std::cout << "A::foo\n"; }
}

class A : public B {
    void foo() { std::cout << "B::foo\n"; }
}
```

```
A *x = malloc(sizeof(B));  // returns abstract object 'o'
new(x) B();
x->foo();  // What will be printed?
```

- **How can types improve static analysis precision?**

- **How can types improve static analysis precision?**

- **How can type-based analysis replace other precision improvement mechanisms?**

- **How can types improve static analysis precision?**
- **How can type-based analysis replace other precision improvement mechanisms?**
- **How can types be used to perform fast analyses?**

Approach

Evaluation

Looking forward

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

July 2019 - January 2020

- Flow-sensitivity for type-based alias analysis
  - Partly worked on
- Aim for CC 2020
- Maybe extend to March (aim for OOPSLA or SAS)

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

July 2019 - January 2020

- Flow-sensitivity for type-based alias analysis
  - Partly worked on
- Aim for CC 2020
- Maybe extend to March (aim for OOPSLA or SAS)

Onward

- Investigations

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

July 2019 - January 2020

- Flow-sensitivity for type-based alias analysis
  - Partly worked on
- Aim for CC 2020
- Maybe extend to March (aim for OOPSLA or SAS)

Onward

- Investigations
  - Optimisations based on type-based analysis

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

July 2019 - January 2020

- Flow-sensitivity for type-based alias analysis
  - Partly worked on
- Aim for CC 2020
- Maybe extend to March (aim for OOPSLA or SAS)

Onward

- Investigations
  - Optimisations based on type-based analysis
  - Applying subgraph mining to flow-sensitive pointer analysis

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

July 2019 - January 2020

- Flow-sensitivity for type-based alias analysis
  - Partly worked on
- Aim for CC 2020
- Maybe extend to March (aim for OOPSLA or SAS)

Onward

- Investigations
  - Optimisations based on type-based analysis
  - Applying subgraph mining to flow-sensitive pointer analysis
  - Fast rebuilding of the SVFG for subsequent analyses

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

July 2019 - January 2020

- Flow-sensitivity for type-based alias analysis
  - Partly worked on
- Aim for CC 2020
- Maybe extend to March (aim for OOPSLA or SAS)

Onward

- Investigations
  - Optimisations based on type-based analysis
  - Applying subgraph mining to flow-sensitive pointer analysis
  - Fast rebuilding of the SVFG for subsequent analyses
  - Parallelisation of flow-sensitive pointer analysis

Plan

Now - July 2019

- Complete type-based heap cloning project
  - Complete implementation
  - Complete evaluation
  - Finish writing
  - Submit (to POPL)

July 2019 - January 2020

- Flow-sensitivity for type-based alias analysis
  - Partly worked on
- Aim for CC 2020
- Maybe extend to March (aim for OOPSLA or SAS)

Onward

- Investigations
  - Optimisations based on type-based analysis
  - Applying subgraph mining to flow-sensitive pointer analysis
  - Fast rebuilding of the SVFG for subsequent analyses
  - Parallelisation of flow-sensitive pointer analysis
- Dissertation!