Motivation

Consolidation
- Analyses in different tools
- Analyses in different formalisms

Extensibility
- Basis for compiler optimizations and further static analyses
  - Constant Propagation
  - Null analysis / exhaustive matching
- Unification of pointer analyses, location types etc.
SACO

Some analyses (pointer) already implemented in SACO

Integrating SACO is difficult

- Code not modular — huge dependency to all of SACO
- Prolog-based — difficult to maintain at university
- ABS frontend unreliable (and unmaintained?)
SACO
Some analyses (pointer) already implemented in SACO

Integrating SACO is difficult
- Code not modular — huge dependency to all of SACO
- Prolog-based — difficult to maintain at university
- ABS frontend unreliable (and unmaintained?)

- We do not aim to reimplement SACO
- Only auxiliary analyses, nothing with resources
## Approach

<table>
<thead>
<tr>
<th>Pure Java</th>
<th>Access ABS compiler as library</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Can be easily integrated into abstools</td>
<td>▶ No new aspects defined (yet)</td>
</tr>
<tr>
<td></td>
<td>▶ Analyses don’t need to be defined in aspects</td>
</tr>
</tbody>
</table>
Implementation

Different levels of control flow graphs

Intraprocedural CFG

- Nodes represent statements
- Top-down order modified by control flow statements
Implementation

Different levels of control flow graphs

Intraprocedural CFG
- Nodes represent statements
- Top-down order modified by control flow statements

Interprocedural CFG
- Nodes represent blocks
- Order defined by calls, returns, throws
Implementation

Different levels of control flow graphs

Intraprocedural CFG
- Nodes represent statements
- Top-down order modified by control flow statements

Interprocedural CFG
- Nodes represent blocks
- Order defined by calls, returns, throws

Analyses may use either
May also use custom graph implementation
Intraprocedural CFG

```c
{  
    Int x = 4;
    Int y = 2;

    if (b) {
        y = x;
        x = 1;
    } else {
        x = 2;
        y = 4;
    }
}
```
Intraprocedural CFG

```java
{ 
    Int x = 4;
    Int y = 2;

    if (b) {
        y = x;
        x = 1;
    } else {
        x = 2;
        y = 4;
    }
}
```
Interprocedural CFG

Contains only reachable nodes, starting from main block

```java
interface I { Int ident(Int i); }

class C implements I {
    Int ident(Int i) {
        return i;
    }

    Unit run() {
        Int one = this.ident(1);
        Int alsoOne = one;
    }
}

I i = new C();
Fut<Int> identFut = i.ident(1);
await identFut?;
Int alsoOne = identFut.get;
println(toString(alsoOne));
```
**Interprocedural CFG**

```java
interface I { Int ident(Int i); }

class C implements I {
    Int ident(Int i) {
        return i;
    }
    Unit run() {
        Int one = this.ident(1);
        Int alsoOne = one;
    }
}

I i = new C();
Fut<Int> identFut = i!ident(1);
await identFut?;
Int alsoOne = identFut.get;
println(toString(alsoOne));
```
Defining an analysis
Three implementations required

Required implementations

- Knowledge
- FlowState
- Flow

- Abstractions to model different parts of analysis
- Open and generic, yet insightful for framework
- Implementation not necessarily an analysis
  - e.g. constant propagation applies result of reaching definitions analysis
Defining an analysis

Knowledge holds the analysis information

<table>
<thead>
<tr>
<th>Required implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knowledge</strong></td>
</tr>
<tr>
<td><strong>FlowState</strong></td>
</tr>
<tr>
<td><strong>Flow</strong></td>
</tr>
</tbody>
</table>

- Represents the information an analysis aggregates
- Example: `VarDecl` => `Set<Exp>`
- Immutable data object
- Mathematically implementation defines a semilattice
- `combine(Knowledge)` method defines merging two instances
  - Usually intersection or union
Defining an analysis
FlowState manages transitions

Required implementations

- Knowledge
- FlowState
- Flow

- Represents the state of an analysis at a certain CFG node
- Keeps track of outgoing Knowledge
- `withIn(Knowledge)` computes new outgoing Knowledge
  - Handles all transitional logic during analysis
Defining an analysis

Flow is the main entry point for an analysis

Required implementations

- Knowledge
- FlowState
- Flow

- Defines the execution logic for analysis
- Base class handles generic data flow execution:
  - ForwardFlow or BackwardFlow
- Creates initial states for all nodes
- Then hands off control to base class
ReachingFlow
Intraprocedural reaching definitions analysis

```java
{ 
    Int x = 4;
    Int y = 2;

    if (b) {
        y = x;
        x = 1;
    } else {
        x = 2;
        y = 4;
    }
}
```
ReachingFlow
Intraprocedural reaching definitions analysis

```java
{  
    Int x = 4;
    Int y = 2;

    if (b) {
        y = x;
        x = 1;
    } else {
        x = 2;
        y = 4;
    }
}
```
ConstantPropagationFlow
Interprocedural data flow

Partially evaluates expressions and method calls

```
interface I { Int ident(Int i); }

class C implements I {
    Int ident(Int i) {
        return i;
    }

    Unit run() {
        Int one = this.ident(1);
        Int alsoOne = one;
    }
}
```

```
{  
    Int i = new C();
    Fut<Int> identFut = i.ident(1);
    await identFut?;
    Int alsoOne = identFut.get;
    println(toString(alsoOne));
}
```
interface I { Int ident( Int i ); }

class C implements I {
    Int ident( Int i ) {
        return i;
    }
    Unit run() {
        Int one = this.ident(1);
        Int alsoOne = one;
    }
}

I i = new C();
Fut<Int> identFut = i.ident(1);
await identFut?;
Int alsoOne = identFut.get;
println (toString(alsoOne));
Build structure
Enforcing good practices

- Checkstyle
  - Indentation width, trailing spaces, ...
  - Everything public has to be documented
- Good test coverage
- Built with Maven
  - Makes import in any common IDE easy
  - Failure on style violations or SpotBugs findings
Outlook

- Data-flow in general
  - Complete pointer analysis
  - More fine-grained context-sensitivity
- MHP/MHF
  - Very useful, but under active development in SACO
  - Probably implementation without advanced features
- Causality
- Framework Behavioral Types