Data-Flow Analysis for ABS

Björn Petersen, Eduard Kamburjan

May 28, 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Integrating SACO is difficult

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

SACO

Some analyses (pointer) already implemented in SACO

TECHNISCHE UNIVERSITÄT DARMSTADT

Integrating SACO is difficult

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

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

Content

Approach

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Pure Java

- Can be easily integrated into abstools

Access ABS compiler as library

- No new aspects defined (yet)
- Analyses don't need to be defined in aspects

Implementation

Different levels of control flow graphs

TECHNISCHE UNIVERSITÄT DARMSTADT

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

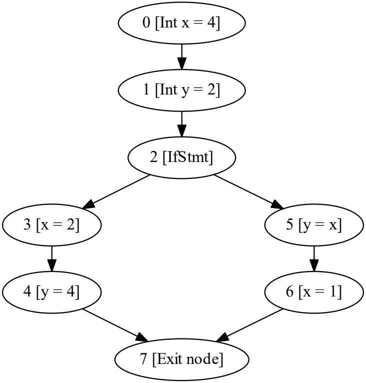
Intraprocedural CFG

TECHNISCHE UNIVERSITÄT DARMSTADT

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

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

Intraprocedural CFG

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

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



Interprocedural CFG

TECHNISCHE UNIVERSITÄT DARMSTADT

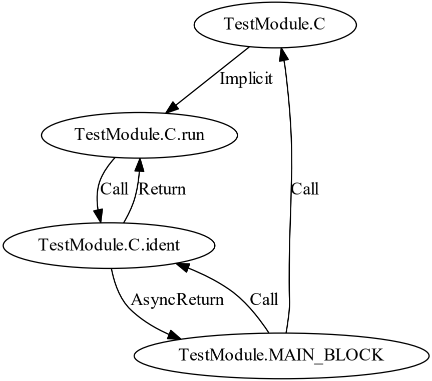
- Contains only reachable nodes, starting from main block

```
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


```
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

TECHNISCHE UNIVERSITÄT DARMSTADT

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Required implementations

- **Knowledge**
- FlowState
- Flow

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

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

TECHNISCHE UNIVERSITÄT DARMSTADT

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

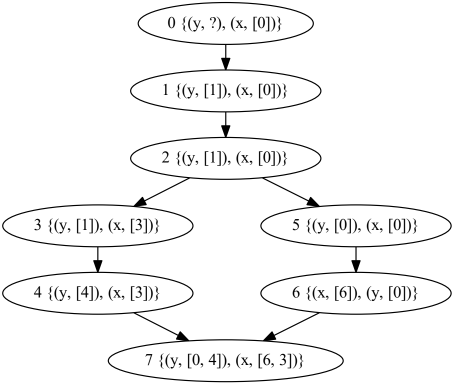
TECHNISCHE UNIVERSITÄT DARMSTADT

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

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

ReachingFlow

Intraprocedural reaching definitions analysis

```
{
    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;
    }
}
```

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

ConstantPropagationFlow

Interprocedural data flow

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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));
}
```

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

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

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

Build structure

Enforcing good practices

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 Behavorial Types