

CS455 Compiler Project

Phillip Michalak

December 11, 2003

Overview

- Goals
- Features
- Implementation
- Lessons

My Goals

- Rapid development (minimum effort)
- Extensibility (minimum effort)
- Implementation challenges (learn)
- Redundancy redundancy elimination
- Yes, Chen, correctness and termination

Features

- Modularity/Extensibility
- Configurable optimization at runtime
 - i.e. gcc `-Ox` functionality
- Variable detail debug output

Modularity

- Compiler = Sequencer + Optimizations
- Sequencer orders optimizations
- Each optimization:
 - Is a transformation of AST and communal information (CFG, etc.)
 - Has no knowledge of other transformations
 - Tells the sequencer whether it made changes
- This paradigm gives us...

Runtime Configurability

- Select optimizations to run at run-time
- Simple sequencer runs optimizations in user specified order

[types]

ast.dataflow.ControlFlowGraph

ast.dataflow.Dominators

ast.dataflow.LiveSets

ast.dataflow.ReachingDefinitions

ast.dataflow.AvailSets

[transformations]

ast.LinearizeCode

ast.optimize.FoldConstants

ast.optimize.Simplify

ast.optimize.ConstantPropagation

ast.optimize.RedundancyElimination

ast.optimize.DeadCode

ast.optimize.UselessControlFlow

ast.optimize.RemoveTrivialJumps

ast.instrument.CountStatements

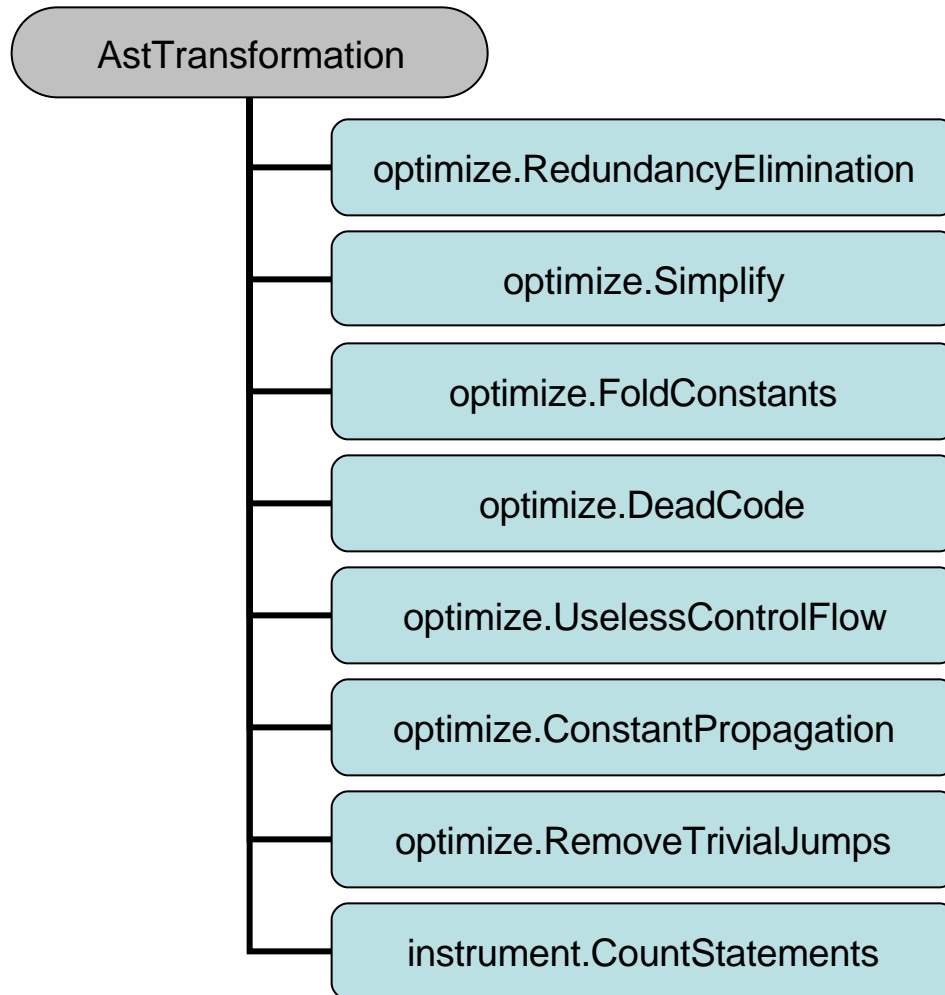
Debug Output

- What the @\$*! is going on?
- Variable level trace output provides more or less detailed information as requested
- Useful for debugging
 - When things are working, less is more
 - When it's broken, need detailed accounting

Implementation

- 43 java files (12 dataflow, 10 opt, 8 graph)
- Package structure
 - util
 - util.dataflow
 - util.graph
 - ast
 - ast.dataflow
 - ast.instrument
 - ast.optimize

Implementation: Optimizations



Implementation: Optimizations

FILE: AstTransformation.java

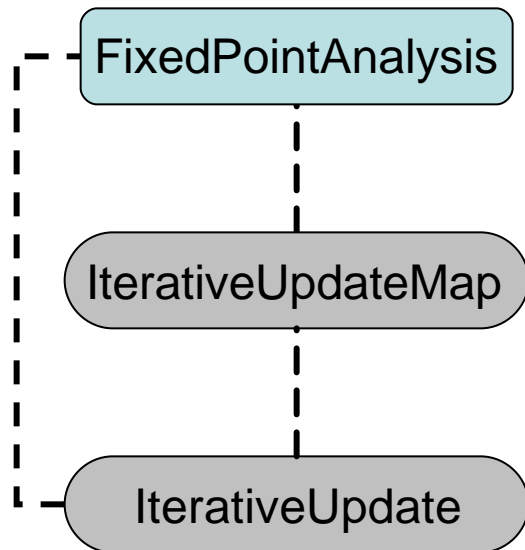
```
package usr.michalak.ast;
```

```
import java.lang.Exception;
```

```
import ast.parser.AstNode;
```

```
public interface AstTransformation {  
    public boolean transform(AstNode ast) throws Exception;  
}
```

Implementation: Dataflow



```
package usr.michalak.util.dataflow;
```

```
import java.lang.*;
```

```
import java.io.*;
```

```
import java.util.*;
```

```
import usr.michalak.util.graph.*;
```

```
import usr.michalak.util.Log;
```

```
public class FixedPointAnalysis {
```

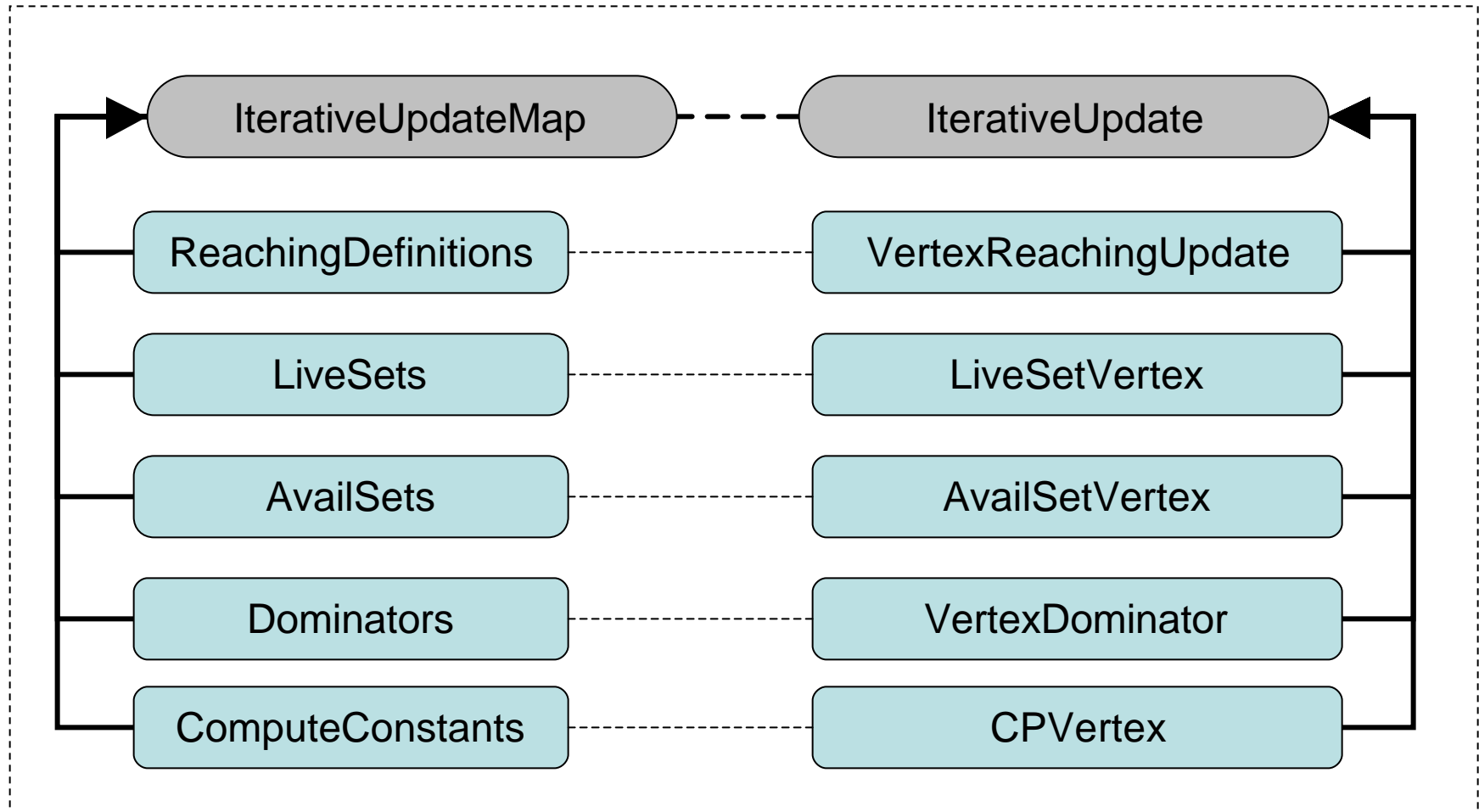
```
    public static void analyzePostOrder(Graph g,  
        IterativeUpdateMap m) throws Exception;
```

```
    public static void analyzeReversePostOrder(  
        Graph g, IterativeUpdateMap m) throws Exception;
```

```
    public static void analyze(Graph g, Collection els,  
        IterativeUpdateMap m) throws Exception;
```

```
}
```

Implementation: Dataflow



Implementation

FILE: IterativeUpdateMap.java

```
package usr.michalak.util.dataflow;
```

```
import java.lang.*;
```

```
import java.io.*;
```

```
import java.util.*;
```

```
import usr.michalak.util.graph.*;
```

```
public interface IterativeUpdateMap {  
    public void initialize(Graph g) throws Exception;  
    public IterativeUpdate getUpdate(Graph g, Object graphElement);  
}
```

Implementation: Dataflow

```
package usr.michalak.util.dataflow;
```

```
import java.lang.*;
```

```
import java.io.*;
```

```
import java.util.*;
```

```
import usr.michalak.util.graph.*;
```

```
public interface IterativeUpdate {
```

```
    public void initialize(Graph g, Object graphElement) throws Exception;
```

```
    public boolean update(Graph g, Object graphElement) throws  
        Exception;
```

```
    public void cleanup(Graph g, Object graphElement) throws Exception;
```

```
}
```

Runtime Output

```
michalak@lisbon as4> make test/fibonacci.out.c
/usr/staff/lib/java/jdk1.3/bin/java -classpath "src/output:/u/xshen/255/public/src/"
  usr.michalak.ast/TransformAst test/fibonacci.adap -f config -d 1
cmdline config option
cmdline debug option
LOGLEVEL=1
CONFIGURATION=config
Registering types for usr.michalak.ast.dataflow.ControlFlowGraph
Registering type CFG
Registering types for usr.michalak.ast.dataflow.Dominators
Registering type DOMINATORS
Registering types for usr.michalak.ast.dataflow.LiveSets
Registering...
Transforming program with usr.michalak.ast.LinearizeCode
Transforming program with usr.michalak.ast.optimize.FoldConstants
Transforming program with usr.michalak.ast.LinearizeCode
Transforming program with usr.michalak.ast.optimize.FoldConstants
Transforming program with usr.michalak.ast.optimize.Simplify
Transforming program with usr.michalak.ast.optimize.ConstantPropagation
Transforming...
```

Lessons

- Extensible/Modular in direct conflict with minimum effort!
- CFG is a very useful abstraction
- Difficult synchronizing CFG and AST during transformations
- Most difficulty with useless control flow (CFG transformations), but it was a worthwhile optimization