Contributions to the Construction of Extensible Semantic Editors

Emma Söderberg

Doctoral Dissertation, 2012

Department of Computer Science
Lund University
Orientation

Today’s programming editors

Text editor
- gedit, Notepad, ...

Semantic editor
- Eclipse, NetBeans, ...

Background:
- Programming: From text to semantic editor
- Not all languages have semantic editors

Problems:
- Construction is time-consuming and complex
- Maintenance may be difficult (extensions)

Challenge:
How can we make it easier to construct and maintain semantic editors?
**Approach**

*Generate services from specification*

**Specification:** RAGs
- Formalism, specify semantics
- Declarative, easily modularized
- JastAddJ, JModelica, JastAdd

**RAGs – Reference Attribute Grammars**
- *Grammar* – defines program model, abstract syntax tree (AST)
- *Attributes* – computed properties of AST nodes (types, scopes, ..)
- *References*: Attribute values

**This dissertation:**
- Extend compiler with editor module
Contributions

- Paper I
- Paper II
- Paper III
- Paper IV
- Paper V
- Paper VI
- Paper VII

Specification

Robustness

Performance
Contributions

- Paper I
- Paper II
- Specification
- Robustness
- Performance
- Paper VI
- Paper VII
- Paper III
- Paper IV
- Paper V
Problem:
How to add an editor to an existing compiler?

Approach:
- Editor Framework
  - JastAdd: semantics
  - Eclipse: graphical components
- Predefined generic services
- RAG-based compiler extension

Results:
- Two demonstrators (size in LOC):
  - JastAdd: compiler 29,200, editor 4,300 (1,100)
  - PicoJava: compiler 210, editor 600 (420)

Conclusions
- Modularly defined editor services
- Compiler reuse (name analysis, type analysis, ...)

(Paper I)
Specification

Example Service: Dead assignments

Problem:
How to specify flow analysis services?

Liveness (textbook)

Let \( n \) be a node and \( \text{succ}[n] \) the set of successors for the node \( n \):

\[
\text{in}[n] = \text{use}[n] \cup (\text{out}[n] \setminus \text{def}[n])
\]

\[
\text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s]
\]

RAGs

\[
\text{syn} \quad \text{Set} \quad \text{CFGNode.in()} \quad \text{circular} \quad \text{[empty()]} = \text{use().union(out().compl(def()))};
\]

\[
\text{coll} \quad \text{Set} \quad \text{CFGNode.out()} \quad \text{circular} \quad \text{[empty()]} \quad \text{with} \quad \text{add};
\]

Stmt contributes in() to CFGNode.out() for each pred();

Expr contributes in() to CFGNode.out() for each pred();

Conclusions:
- Textbook-like definitions
- Flow analysis added modularly with few LOC.
- Precision/performance on par with Soot.

Here "a = 0" is a dead assignment because the value is not used and it could be removed.
Contributions

- **Paper I**
  Editor framework

- **Paper II**
  Flow analysis

- **Robustness**

- **Performance**

- **Paper VI**
  Paper VII

- **Specification**

- **Paper III**

- **Paper IV**

- **Paper V**
Contributions

- **Paper I**
  *Editor framework*

- **Paper II**
  *Flow analysis*

- **Paper III**

- **Paper IV**

- **Paper V**

- **Paper VI**

- **Paper VII**

**Robustness**

**Performance**

**Specification**
Performance

Faster evaluation from scratch

**Background**: At attribute evaluation
- attribute dependencies $\rightarrow$ call graph
- *no caching* - multiple evaluations $\rightarrow$ very slow
- *full caching* - at most one evaluation $\rightarrow$ faster

**Problem**: Memory/performance costs

**New idea**: Selective caching
- based on profiling
- skip caching of some attributes

**Results**: 20% speedup and 38% memory reduction
- Compared to full caching
- JastAddJ Java compiler
- Java benchmarks
Problem: How to efficiently update the program model after edits?

State of the art: - Hand-coded solutions, complex, error-prone

Challenge: - Automatically update model, RAGs

Earlier work: - Optimal automatic updates for AGs - No handling of references

New results: - Dynamic algorithm for RAGs. - Build dynamic dependency graph during evaluation - Use graph to uncache affected attributes after edits
Performance

Incremental evaluation comparison

Example

```plaintext
int a, b;
int c;
a = 42;
```

Legend

- **Name**: AST node
- **name**: Attribute instance
- "..": Token with value ".."
- **edited**: affected
- **examined**: skipped
- -- AST edge
- ← Attribute dep.
- → Attribute dep.
  (examined)

[Paper V]
Contributions

- **Paper I**
  Editor framework

- **Paper II**
  Flow analysis

- **Paper III**
  Efficient caching

- **Paper IV**
  Incremental evaluation

- **Paper V, Paper VI, Paper VII**
Contributions

- **Paper I**
  Editor framework

- **Paper II**
  Flow analysis

- **Paper III**
  Efficient caching

- **Paper IV**
  Incremental evaluation

- **Paper VI**

- **Paper VII**

**Robustness**

**Performance**

**Specification**
Robustness

How to handle erroneous input?

Problem:
Scope errors cause recovery to fail

Idea:
- Use layout for recovery
- Aid existing recovery with preprocessor

New Algorithm:
Bridge parsing

Editor
```
class A {
    void m() {
        int a
        int b;
    }
}
```

Program Model

Error Recovery:
Error Productions
Automatic Recovery

Parser

Outline
Folding
Completion

Island grammars:
- Islands: Interesting
- Water: Uninteresting

Bridge Parsing
- Reefs: Patterns for recovery
- Bridges – Scopes

New Algorithm:
Bridge parsing

```
class C {
    void m() {
        //...
    }
    void n() {
        //...
    }
}
```

Problem:
Scope errors cause recovery to fail

Idea:
- Use layout for recovery
- Aid existing recovery with preprocessor

New Algorithm:
Bridge parsing

```
class.. {1 1 void.. {2 #.. 1 } 1 void.. {2 #.. 1 } 0 }
```

[Paper VI]
Robustness

Bridge Parsing Algorithm

class C {
    void m() {
        // ...
    }
    void n() {
        // ...
    }
}

Algorithm:
1. Tokenizer
2. Bridge Builder
3. Bridge Repairer
Robustness

Results from adding bridge parsing

**Antlr** – a well-known LL-based parser generator

![Graph showing distance to ideal AST for plain ANTLR and with Bridge Parsing](image)

- **Distance to ideal AST**
- **Test cases**
- **Legend**:
  - Red diamond: plain ANTLR
  - Black square: with Bridge Parsing

[Paper VI]
Robustness

*Bridge Parsing ideas in JSGLR*

**Collaboration:** TU Delft

**Problem:**
Provide error recovery for Scannerless GLR (SGLR)

**SGLR:**
- *Generalized* LR: Arbitrary CFG
- *Scannerless* – include tokens in the grammar
- Language composition, e.g., Java-SQL and enum
- JSGLR – implementation of SGLR in Java

**Method:**
Recovery using island grammars, layout and bridge parsing

**Results:**
Recovery quality on par with the Eclipse Java parser.

![Context Free Grammars](image)

- **LL**
- **LR**
- **GLR**

**Recovery Quality**

- **Eclipse/Java**
- **JSGLR**

[Paper VII] Bridge Parser Parts
Contributions

- **Paper I**
  Editor framework

- **Paper II**
  Flow analysis

- **Paper III**
  Efficient caching

- **Paper IV**
  Incremental evaluation

- **Paper V**

- **Paper VI**
  Bridge parsing  
  Layout-sensitive recovery

- **Paper VII**
Contributions

- Paper I
  *Editor framework*

- Paper II
  *Flow analysis*

- Paper III
  *Efficient caching*

- Paper IV
  *Incremental evaluation*

- Paper V

- Paper VI

- Paper VII
  *Bridge parsing*
  *Layout-sensitive recovery*