A Decision Tree-based Approach to Dynamic Pointcut Evaluation

Robert Dyer and Hridesh Rajan

Department of Computer Science
Iowa State University
{rdyer,hridesh}@cs.iastate.edu

October 19, 2008
Motivation: Dynamic PCD Evaluation

Approach: Decision-tree based Matching

Technical Contributions:
- Formalization of the PCD Evaluation problem
- Algorithms using Decision-tree structures for faster matching
- Use of implication relationships for partial evaluation of type predicates
Terminology

PCD Evaluation

\[ \begin{align*}
  a & \in \mathcal{A}, \text{the set of attributes} \\
  o & \in \mathcal{O}, \text{the set of operators} \\
  v & \in \mathcal{V}, \text{the set of values}
\end{align*} \]
\( a \in \mathcal{A} \), the set of attributes
\( o \in \mathcal{O} \), the set of operators
\( v \in \mathcal{V} \), the set of values

\[
\text{pred} ::= (a, o, v) \\
\text{fact} ::= (a, v)
\]
\[ a \in \mathcal{A}, \text{the set of attributes} \]
\[ o \in \mathcal{O}, \text{the set of operators} \]
\[ v \in \mathcal{V}, \text{the set of values} \]

\[
\begin{align*}
pred & ::= (a, o, v) \\
fact & ::= (a, v) \\
PCD & ::= pred \\
& | (PCD) \\
& | pred \&\& PCD \\
& | pred \|\| PCD \\
join point & ::= fact \\
& | fact \&\& join point
\end{align*}
\]
\[ A ::= \{ \text{modifier, type, name} \} \]
\[ V ::= \{ v : v \text{ is a modifier, type or name in the program} \} \]
\[ O ::= \{ ==, != \} \]
\[ A ::= \{\text{modifier}, \text{type}, \text{name}\} \]
\[ V ::= \{v : v \text{ is a modifier, type or name in the program}\} \]
\[ O ::= \{==, !=\} \]

**Example PCD**

\[(\text{modifier}, ==, \text{public}) \land (\text{type}, !=, \text{void}) \land (\text{name}, ==, "\text{Set}"")\]
\[ \mathcal{A} ::= \{ \text{modifier, type, name} \} \]
\[ \mathcal{V} ::= \{ v : v \text{ is a modifier, type or name in the program} \} \]
\[ \mathcal{O} ::= \{ ==, != \} \]

**Example PCD**

(mostly, ==, public) && (type, !=, void) && (name, ==, "Set")

**Example join point**

(modifier, public) && (type, FElement) && (name, "Set")
Decision Tree-based Dynamic PCD Evaluation
2 ways of viewing the problem

- PCDEval'

```
PCD1
```

```
JP1  JP2  JP3  JP4  ...  JPn
```
2 ways of viewing the problem

- **PCDEval**

```
JP1
```

```
PCD1  PCD2  PCD3  PCD4  ...  PCDn
```
Evaluation Algorithm overview
- Order predicates for efficiency
- Create PCD evaluation tree(s)
- Add predicates to decision trees
- Create links to parents
Consider the following PCD: $\text{Pred1} \parallel (\text{Pred2} \&\& \text{Pred3})$
Order predicates for efficiency
  - Modifiers are simple to match
  - Makes other decision-trees disjoint (smaller)
Overview
Partial Evaluation of Types

Robert Dyer and Hridesh Rajan

Decision Tree-based Dynamic PCD Evaluation
Goal: Reduce size of decision-trees
Idea: Partially evaluate predicates
Known: $B \prec C$

Evaluate: $A \prec B, A \prec C$
Known: $B \ll C$

Evaluate: $A \ll B$, $A \ll C$

$A \ll B \land B \ll C$
Known: $B \ll C$
Evaluate: $A \ll B, A \ll C$

$A \ll B \land B \ll C \rightarrow A \ll C$
Known: $B \preceq C$

Evaluate: $A \preceq B$, $A \preceq C$

$A \preceq B \land B \preceq C \rightarrow A \preceq C$

Partially Evaluate: $A \preceq B$
Created implementation in Nu virtual machine
- Bind and Remove primitives for deploying/un-deploying advice
- Synthetic micro-benchmark
  - Measures time to Bind (add to trees) and match
  - Varies type hierarchy depth
Decision Tree-based Dynamic PCD Evaluation
Old matching code - (∼40 µs constant)
Old matching code - average case 3-50x slower
worst case 3-88x slower
Related Work

- Efficient Matching Techniques
- Dynamic Residue Evaluation
- Partial Evaluation Techniques
Future Work

- Example Implementation(s)
- Real-world Evaluations
Motivation: Dynamic PCD Evaluation

- PCDs arrive dynamically
- PCDs might be removed later
- Matching the whole (loaded) system against a PCD is too slow

Approach: Decision-tree based Matching

- Order evaluations based on cost
- Partially evaluate wherever possible

Technical Contributions:

- Formalization of the PCD Evaluation problem
- Algorithms using Decision-tree structures for faster matching
- Use of implication relationships for partial evaluation of type predicates
Questions?

http://www.cs.iastate.edu/~nu/
class C {
    public static void run() {
        measure { Bind.. // to methods returning C1 }
        measure { Bind.. // to methods returning C2 }
        measure { Bind.. // to methods returning C3 }

        measure { C1.testMethod }
        measure { C2.testMethod }
        measure { C3.testMethod }
    }

    public C testMethod() { return NULL }
}

C.run
C1.run
C2.run
C3.run
...