Hybrid Partial Evaluation
OOPSLA'11
(best student paper)

William R. Cook and Amin Shali
University of Texas at Austin
About me

PhD Brown University, 1989 w/Peter Wegner
  Denotational Semantics of Inheritance
  Mixins, F-Bounds, Class reorganization, ADT/OOP

AppleScript
  Lead designer and group manager

Allegis: Indirect Sales Chain Management
  150 employees, $60M in venture capital
  Customers: Microsoft, HP, Charles Schwab, etc.

University of Texas at Austin, Computer Science
  Joined 2003
Motivation

Goal
We want to write general programs/libraries
But use them in specific situations

Partial Evaluation
Can specialize a general program to some inputs

Desires
Easy to understand
Easy to implement
Works well in practice
Partial Evaluation (by hand)

Example: Power function

\[
\text{pow}(n, x) = \text{if } (n==0) \text{ then } 1 \text{ else } x \times \text{pow}(n-1, x)
\]

What if you know \(n\)?

\[
\text{pow}(3, x) = x \times \text{pow}(2, x)
\]

This depends on \(\text{pow}(2, x)\)

\[
\begin{align*}
\text{pow}(2, x) &= x \times \text{pow}(1, x) \\
\text{pow}(1, x) &= x \times \text{pow}(0, x) \\
\text{pow}(0, x) &= 1
\end{align*}
\]
Partial Evaluation (by hand)

Example: Power function

\[ \text{pow}(n, x) = \text{if } (n==0) \text{ then } 1 \text{ else } x \times \text{pow}(n-1, x) \]

What if you know \( n \)?

\[ \text{pow3}(x) = x \times \text{pow2}(x) \]

This depends on \( \text{pow}(2, x) \)

\[ \text{pow2}(x) = x \times \text{pow1}(x) \]
\[ \text{pow1}(x) = x \times \text{pow0}(x) \]
\[ \text{pow0}(x) = 1 \]
Partial Evaluation

Example: Power function

\[
pow(n, x) = \text{if } (n==0) \text{ then } 1 \text{ else } x \times pow(n-1, x)
\]

Lets call this final function “pow3”:

\[
pow3(x) = x \times x \times x
\]

Partial evaluation

Eliminates computations that depend on known inputs

Result is “residual program”

Doesn’t always work:

\[
pow(n, 19) = \text{if } (n==0) \text{ then } 1 \text{ else } 19 \times pow(n-1, 19)
\]

Useful when raising many numbers to 3rd power
Automatic Partial Evaluation

Example: Power function

\[
pow(n, x) = \text{if } (n==0) \text{ then } 1 \text{ else } x \cdot pow(n-1, x)
\]

Can we compute residual code automatically?

\[
\text{peval}(\ pow, \ 3) \rightarrow \ \text{fun}(x) \ x \cdot x \cdot x \equiv \ pow3
\]

Partial evaluation function: peval

Inputs:

- Source code of a function
- Value of the first argument

Output:

- Residual code from partially evaluating
Automatic Partial Evaluation

Example: Power function

\[ \text{pow}(n, x) = \text{if } (n==0) \text{ then } 1 \text{ else } x \times \text{pow}(n-1, x) \]

Can we compute residual code automatically?

\[ \text{peval}(\text{pow}, 3) \rightarrow \text{fun}(x) x \times x \times x \equiv \text{pow3} \]

Partial evaluation function: \text{peval}

Inputs:

- Source code of a function
- Value of the first argument

Output:

- Residual code from partially evaluating
Example

String pat = CT("(a(*|*)b)*(abb(*|*)a+b)")
Regex regex = CT(RegexParser.parse(pat));

String buffer = in.readLine(...)
regex.execute(buffer);
Pure 1\textsuperscript{st}-order Functional Language

\[ x : \text{Variable} \quad v : \text{Value} \]
\[ e = x | v | \text{if } e \text{ then } e \text{ else } e | e + e | f(e, \ldots, e) \]

\( \rho \) environment maps \textit{all} variables to values
\[ E[v] \rho = v \]
\[ E[x] \rho = \rho(x) \]
\[ E[\text{if } e_1 \text{ then } e_2 \text{ else } e_3] \rho = \text{if } E[e_1] \rho \text{ then } E[e_2] \rho \text{ else } E[e_3] \rho \]
\[ E[e_1 + e_2] \rho = E[e_1] \rho + E[e_2] \rho \]
\[ E[f(e_1, \ldots, e_n)] \rho = E[e] \rho' \]

\textbf{lookup function definition:} \( f(x_1, \ldots, x_n) = e \)
\[ \rho' = \{ x_1 = E[e_1] \rho, \ldots, x_n = E[e_n] \rho \} \]
Evaluation to Partial Evaluation

The type of eval

\[ E : \text{Expression} \rightarrow \text{Environment} \rightarrow \text{Value} \]

\[ \text{Environment} = \text{Variable} \rightarrow \text{Value} \]

\[ \text{FreeVars}(e) \subseteq \text{Domain}(v) \]

All variables are bound

What about a partial evaluator?

\[ P : \text{Expression} \rightarrow \text{Environment} \rightarrow \text{Expression} \]

Result might not be a complete value

\[ P[x+y] \ {x=3, \ y=2} \rightarrow 5 \]

\[ P[x+y] \ {x=3} \rightarrow [3+y] \]
Online Partial Evaluator $\mathcal{P}$

$x : \text{Variable} \quad v : \text{Value}$

$e = x \mid v \mid \textit{if } e \text{ then } e \text{ else } e \mid e + e \mid f(e, \ldots, e)$

environment maps $\textit{some}$ variables to values

$\mathcal{P}[v]\rho = v$

$\mathcal{P}[x]\rho = \textit{if } x \in \text{dom}(\rho) \text{ then } \rho(x) \text{ else } [x]$

returns code $[x]$ if the variable is not defined
Online Partial Evaluator $P$

$x : \text{Variable} \quad v : \text{Value}$

$e = x \mid v \mid \text{if } e \text{ then } e \text{ else } e \mid e+e \mid \mathcal{f}(e, \ldots, e)$

environment maps some variables to values

$P[v] \rho = v$

$P[x] \rho = \text{if } x \in \text{dom}(\rho) \text{ then } \rho(x) \text{ else } \ll x \rr$

$P[\text{if } e_1 \text{ then } e_2 \text{ else } e_3] \rho =$

\[
\text{case } P[e_1] \rho \text{ of } \\
\quad v \rightarrow \text{if } v \text{ then } P[e_2] \rho \text{ else } P[e_3] \rho \\
\quad e \rightarrow \ll \text{if } e \text{ then } P[e_2] \rho \text{ else } P[e_3] \rho \rr
\]

if its a boolean $v$, then pick branch.
else create a new if statement
Online Partial Evaluator $P$

$P[v] \rho = v$

$P[x] \rho = \text{if } x \in \text{dom}(\rho) \text{ then } \rho(x) \text{ else } \ll x \gg$

$P[\text{if } e_1 \text{ then } e_2 \text{ else } e_3] \rho =$

\text{case } P[e_1] \rho \text{ of}

\begin{align*}
v & \rightarrow \text{if } v \text{ then } P[e_2] \rho \text{ else } P[e_3] \rho \\
e & \rightarrow \ll \text{if } e \text{ then } P[e_2] \rho \text{ else } P[e_3] \rho \gg
\end{align*}

$P[e_1 + e_2] \rho =$

\begin{align*}
v_1 + v_2 & \text{ if } v_i = P[e_i] \rho \\
\ll e'_1 + e'_2 \gg & \text{ if } e'_i = P[e_i] \rho
\end{align*}

apply operator if arguments are both are values
otherwise generate new expression
Function Calls

\[ P[\vec{f}(e_1, \ldots, e_n)]\rho = \langle \vec{f}_\rho (e'_{d_1}, \ldots, e'_{d_k}) \rangle \]

1. lookup function definition: \( \vec{f}(x_1, \ldots, x_n) = e \)
2. Partially evaluate the arguments
   \( e'_i = P[e_i] \)
3. partition arguments into “compile time (CT)” and “runtime”
   \( \{ s_1, \ldots, s_j \} \cup \{ d_1, \ldots, d_k \} = \{ 1, \ldots, n \} \)
   \( \forall \{ e'_{s_1}, \ldots, e'_{s_j} \} = \{ v_1, \ldots, v_j \} \)
4. create environment with CT variables
   \( \rho_s = \{ x_{s_1} = v_{s_1}, \ldots, x_{s_j} = v_{s_j} \} \)
5. create new function specialized by CT values
   \( \vec{f}_\rho (x_{d_1}, \ldots, x_{d_k}) = E[e] \rho_s \)
6. Residual code is call with runtime arguments
   \( \langle \vec{f}_\rho (e'_{d_1}, \ldots, e'_{d_k}) \rangle \)
Hybrid Partial Evaluation

Online style (no static analysis)
Annotations to begin partial evaluation
No termination guarantee
Object-oriented language
  Imperative (mutable state)
  Objects “live” at compile-time or runtime (not both)
  Specialize methods and classes
Formalized in Haskell (It's fun, check it out!)
Real compiler for Java (Part of Batches project)
Partial Evaluation Annotations

Two stages:

Compile-time: pre execution during partial evaluation/compilation
Runtime: normal execution

Annotation

CT(e)
Marks expression e for execution at compile time

Example

Regex regex = CT(Regex.parse("(a|b)*"));
regex.execute(buffer);
Objects

Any object can be instantiated at compile time

if (config.loggingEnabled) ...
  dead code elimination
if (config.enabled("logging")) ...
  dead code elimination
if (config.enabled(userInput)) ...
  inlines “enabled” method using config state
x = new System(config1);
y = new System(config2);
  specializes System class, once for each config!
dynamicHashMap.put("c1", config)
  compile-time error: Config cannot jump to runtime
Objects exists in exactly one stage

Cannot move
(from compile time to runtime)

Primitive values can move
Mutable State

Mutable state
Supported!
A mutable object is either compile-time or runtime
Within stage, all mutations happen in correct order

```java
Regex regex = CT(Regex.parse("(a|b)*"));
regex.execute(buffer);
```

Behavior maybe different with/without PE
Annotated program may have different behavior
  e.g. Print statements may happen at compile time
Program may be rejected (if statements)
Annotations create a new language
Reflection

Reflection becomes static

String name = CT(“getSize”);
Method m = o.getClass().getMethod(name);
m.invoke(o);

converts to:

o.getSize();
Termination

No Termination Guarantee

The compiler may diverge

This is true of C++ compiler

Hit Ctrl-C and modify the program....
Example Model Interpreter

Interpreter

```java
int run(State current) {
    print(current.label);
    String input = in.readLine();
    nxt = current.trans.lookup(input);
    run(nxt);
}
```
Example Model Interpreter

Interpreter

```java
int run(State current) {
    print(current.label);
    String input = in.readLine();
    nxt = current.trans.lookup(input);
    run(nxt);
}
```

 compile time

Open

Close d

Runtime
"The Trick"
(Binding-time improvement)

Interpreter

```java
int run(State current) {
    print(current.label);
    String input = in.readLine();
    for (Trans t : current.trans)
        if (t.event == input)
            return run(t.to);
    return run(current);
}
```
Partial Evaluation

```java
int runOpen() {
    print("Open");
    String input = in.readLine();
    if ("close" == input)
        return runClosed();
    return runOpen();
}

int runClosed() {
    print("Closed");
    String input = in.readLine();
    if ("open" == input)
        return runOpen();
    return runClosed();
}

int run(State current) {
    print(current.label);
    String input = in.readLine();
    for (Trans t : current.trans)
        if (t.event == input)
            return run(t.to);
    return run(current);
}
```
Goal: Partial Evaluation of WebDSL

web(UI, Schema, db, request) : HTML

UI : description of user interface (pages, sections)
schema: description of data (constraints, etc)
db : data store (described by schema)
request : an HTTP request
web : interpreter, with design knowledge
Partial Evaluation

\[ \text{web}_{\text{UI, Schema}}(db, \text{request}) : \text{HTML} \]
 compiletime runtime

\[ \text{web}_{\text{UI, Schema}} \] is partial evaluation of web with respect to UI model and data schema

Supports both dynamic interpretation and compiled execution in same framework
Civet: A Partial Evaluator for Java
Civet: A Partial Evaluator for Java

Usable but not complete
  Implemented as extension to javac

Testing on Real Applications
  ModelTalk -- dynamic pricing application
  more?

Initial results for ModelTalk (3rd party test)
  Original: 3,153 ms
  Optimized: 293 ms
Performance of RegEx example

Uses Derivative-based interpreter for regular expressions

<table>
<thead>
<tr>
<th>Program</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original regex state machine</td>
<td>1189</td>
</tr>
<tr>
<td>Specialized regex state machine</td>
<td>573</td>
</tr>
<tr>
<td>dk.brics.automaton regex library</td>
<td>816</td>
</tr>
</tbody>
</table>
Paradox of Performance with Partial Evaluation

- Slow Version
- Hand Optimized (general)
- Partial Eval (Specialized)

Faster
Future

Currently AST based transformation

Should move to byte-code partial evaluation
Conclusion

Hybrid Partial Evaluation
- Online strategy
- Offline power

Practicalities
- Annotations to begin partial evaluation
- No termination guarantee
- Compile-time structures are never residualized
- Imperative effects within each stage
- Programmers must be aware of “the trick”
- No self-application (First Futamura projection only)
Futamura (1971)

Partial evaluation of an interpreter with respect to a program is a compiled version of the program.
Partial Evaluation of Interpreters

Interpreter

python(“notify.pl”, “in.txt”) → o
Futamura Projections I

**Interpreter**

\[
\text{python(“notify.pl”, “in.txt”)} \rightarrow o
\]

**First Futamura projection**

\[
\text{peval(python, “notify.pl”)} \rightarrow g \quad \text{where } g(“in.txt”) = o
\]

\[g \quad \text{is compiled version of notify.pl}\]
Futamura Projections (pattern)

Interpreter

\[
\text{python}("\text{notify.pl}", "\text{in.txt}") \rightarrow o
\]

First Futamura projection

\[
\text{peval}(\text{python}, "\text{notify.pl}") \rightarrow g \quad \text{where } g("\text{in.txt}") = o
\]

g is compiled version of notify.pl
Futamura Projections II

Interpreter

\[
\text{python(“notify.pl”, “in.txt”) } \rightarrow \ o
\]

First Futamura projection

\[
\text{peval(python, “notify.pl”) } \rightarrow \ g \quad \text{where } g(“in.txt”) = o
\]

\[
g \text{ is compiled version of notify.pl}
\]

Second Futamura projection

\[
\text{peval(peval, python) } \rightarrow \ c \quad \text{where } c(“notify.pl”) = g
\]

\[
c \text{ is a python compiler}
\]
Futamura Projections III

Interpreter

\[
\text{python(“notify.pl”, “in.txt”) } \rightarrow o
\]

First Futamura projection

\[
\text{peval(python, “notify.pl”)} \rightarrow g \quad \text{where } g(“in.txt”) = o
\]

\[g\] is compiled version of notify.pl

Second Futamura projection

\[
\text{peval(peval, python)} \rightarrow c \quad \text{where } c(“notify.pl”) = g
\]

\[c\] is a python compiler

Third Futamura projection

\[
\text{peval(peval, peval)} \rightarrow z \quad \text{where } z(\text{python}) = c
\]

\[z\] is a compiler compiler!
We only need First Projection

Interpreter
\[
\text{python(“notify.pl”, “in.txt”) } \rightarrow \text{ o}
\]

First Futamura projection
\[
\text{peval(python, “notify.pl”) } \rightarrow \text{ g } \quad \text{where g(“in.txt”) = o}
\]
\[
g \text{ is compiled version of notify.pl}
\]

Second Futamura projection
\[
\text{peval(peval, python) } \rightarrow \text{ c } \quad \text{where c(“notify.pl”) = g}
\]
\[
c \text{ is a python compiler}
\]

Third Futamura projection
\[
\text{peval(peval, peval) } \rightarrow \text{ z } \quad \text{where z(python) = c}
\]
\[
z \text{ is a compiler compiler!}
\]
Avoid Need for Self-Applicable peval

Interpreter
\[
\text{python(“notify.pl”, “in.txt”) } \rightarrow \text{ o}
\]

First Futamura projection
\[
\text{peval(python, “notify.pl”) } \rightarrow \text{ g where g(“in.txt”) = o}
\]
g is compiled version of notify.pl

Second Futamura projection
\[
\text{peval(peval, python)} \rightarrow \text{ c where c(“notify.pl”) = g}
\]
c is a python compiler

Third Futamura projection
\[
\text{peval(peval, peval) } \rightarrow \text{ z where z(python) = c}
\]
z is a compiler compiler!
Futamura in Practice

Interpreters have “good” behavior
  Control flow depends on program first, then input
    just like pow(n, x): control flow depends on n

Can’t make good compilers via 2nd/3rd Futamura
  Trying to make a C compiler via Futamura will fail
  Was that the right goal?
    Be careful what you pick as challenge problem

Hypothesis:
  First Futamura projection will work well enough for
    model interpreters
    solves real problem, simple partial evaluator