Strategic Programming
by Model Interpretation
and Partial Evaluation

William R. Cook
UT Austin

Presented at ACL2 Meeting
Oct 2, 2008
Web Applications

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

Addresses key problem: integrating multiple models
Web Page Wireframe

Drug Free Communities
Program Monitoring and Management System

Home | Assessment | Capacity | Planning | Implementation | Evaluation | Administration

Home > Assessment > Community needs assessment > Edit community needs assessment

Edit community needs assessment

Target community name

Target geographic areas:

(Hold down Ctrl to select multiple)

Specific targeted geographic areas

(Hold down Ctrl to select multiple)

Further describe the geographic areas selected above (e.g., names of cities, counties, etc.).

Assessment summary

Targeted gender

Targeted grade

(Hold down Ctrl to select multiple)
Partial Evaluation of Web Interpreter

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

- static
- dynamic

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

- Dynamic web site (wiki-style) or compiled
- Possible to generate both web and GUI
Partial Evaluation

Given a function of two arguments

(define (pow n m) ;; computes m^n
    (cond ((= n 0) 1)
          (t (* m (pow (- n 1) m))))))

If the first argument is known statically, then

(pow 3 x) == (* x x x)
(pow 6 x) == (* x x x x x x)

Eliminates the recursive and conditional overhead
Partial Evaluation

A better strategy

\[
\text{(define (pow2 n m) ;; computes } m^n
\text{(cond ((= n 0) 1)
\text{   ((= (mod n 2) 0) (square (pow2 (/ n 2) m)))
\text{   (t (* m (pow2 (- n 1) m)))})
}\]

Gives a faster residual program

\[
\text{(pow2 8 n) == (square (square (square n)))
\text{(pow2 6 n) == (square (* n (square n)))}
\]
Partial Evaluation Function

Traditionally called **mix**

\[(\text{mix } \text{pow } 3) = (\lambda (n) (* n n n))\]
\[(\text{mix } \text{pow2 } 3) = (\lambda (n) (* n (\text{square } n)))\]

**mix** : \((A \times B \rightarrow C) \times A \rightarrow (B \rightarrow C)\)

Looks like uncurried version of the curry function

**curry** : \((A \times B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C))\)

**Specification**

\[(\text{defthm mix-behavior} \]
\[\text{(equal } ((\text{mix } f \ a) b) (f a b)))\]

* oops, its higher-order*
List Comprehensions in Scheme (first-order map)

Borrowed from Haskell

\( (\text{for } \text{var } \text{items } \text{op } \text{result}) \)
\[ \text{op } [ \text{result } | \text{var } \in \text{items} ] \]

\( (\text{for } \text{var } \text{items } \text{op } (\text{if } \text{cond } \text{result})) \)
\[ \text{op } [ \text{result } | \text{var } \in \text{items}, \text{cond } ] \]

Examples

\( (\text{for } x \ (1 \ 2 \ 3) + x) \) \( \Rightarrow 6 \)
\( (\text{for } x \ (1 \ 2 \ 3) \text{ list } (* x x)) \) \( \Rightarrow (1 \ 4 \ 9) \)
\( (\text{for } x \ (1 \ 2 \ 3) \text{ and } (\text{even? } x)) \) \( \Rightarrow \#f \)
\( (\text{for } x \ (1 \ 2 \ 3) \text{ list } (\text{if } (\text{odd? } x))) \) \( \Rightarrow (1 \ 3) \)
\( (\text{for } x \ (1 \ 2 \ 3) \text{ list } (\text{if } (\text{odd? } x) \ x (- x))) \) \( \Rightarrow (1 \ -2 \ 3) \)
A Normal Scheme Evaluator (first-order)

x : Variable      v : Value
e : Expression = x | v | if e e e | f(e, ..., e) | op(e, ..., e)
ρ : Environment = Variable → Value
Ε : Expression → Environment → Value

Ε[ v ] ρ = v
Ε[ x ] ρ = ρ(x)
Ε[ if e_1 e_2 e_3 ] = if Ε[e_1] then Ε[e_2] else Ε[e_3]
Ε[ f(e_1, ..., e_n) ] ρ = Ε[e] ρ'

lookup definition of f: f(x_1, ..., x_n)=e
ρ' = [x_1=Ε[e_1]ρ, ..., x_n=Ε[e_n]ρ]
Ε[ op(e_1, ..., e_n) ] ρ = apply(op, Ε[e_1]ρ, ..., Ε[e_n]ρ)
From Full Evaluation to Partial Evaluation

The type of eval

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

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

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

All variables are bound

What about a partial evaluator?

Environment gives values to \textit{some} variables

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

Result might not be a complete value

\[
(\text{mix } '(+ x y) '(x=3 \ y=2)) \Rightarrow 5^S
\]

\[
(\text{mix } '(+ x y) '(x=3)) \Rightarrow (+ 3 y)^D
\]
Online Partial Evaluator (first-order)

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

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

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

\[ \mathcal{P}[\text{if } e_1 \ e_2 \ e_3 ] \rho = \text{case } \mathcal{P}[e_1] \rho \text{ of} \]
\[ \quad v \rightarrow \text{if } v \text{ then } \mathcal{P}[e_2] \rho \text{ else } \mathcal{P}[e_3] \rho \]

\[ e \rightarrow [[\text{if } e \ \mathcal{P}[e_2] \rho \ \mathcal{P}[e_3] \rho ]] \]

\[ \mathcal{P}[f(e_1, ..., e_n)] \rho = [[\langle f, \rho_S.\text{vals} \rangle(\rho_D.\text{vals})]] \]

lookup definition of f: \[ f(x_1, ..., x_n)=e \]
\[ \rho_S = [ (x_i, v_i) \mid i \in 1..n, v_i = \mathcal{P}[e_i] \rho ] \]
\[ \rho_D = [ (x_i, e'_i) \mid i \in 1..n, e'_i = \mathcal{P}[e_i] \rho, e \notin \text{Value} ] \]

create function: \( \langle f, \rho_S.\text{vals} \rangle(\rho_D.\text{vars}) = \mathcal{P}[e] \rho_S \)

\[ \mathcal{P}[\text{op}(e_1, ..., e_n)] \rho = \]
\[ \{ \text{apply}(\text{op}, v_1, ..., v_n) \quad \text{if } v_i = \mathcal{P}[e_i] \rho \}
\[ [[\text{op}(\mathcal{P}[e_1] \rho, ..., \mathcal{P}[e_n] \rho)]] \quad \text{otherwise} \]
Specialization Example

Specialization

(mix '(pow 2 *))

Residual code

(define (pow-0-* m) 1)
(define (pow-1-* m) (* m (pow-0-* m)))
(define (pow-2-* m) (* m (pow-1-* m)))

Residual code (with actual post-processing)

(define (pow-2-* m) (* m m))

Issues

In general does not terminate
Might fail to specialize (more in a minute)
Correctness

(defun ev-pe
  (implies (and (= (free-vars e) (bound-vars v))
               (= v (append v1 v2)))
  (= (S (eval e v))
    (mix (lift (mix e v1)) v2))

Evaluation in a complete environment gives the same result as partial evaluation with part of the environment, followed by partial evaluation with the result of the environment.

Not proven!
Consider an interpreter
\[ \text{interpret}(\text{program}, \text{data}) \]
Use the same program for many data inputs: program is static

**First Futamura Projection**
\[ \text{mix}(\text{interpret}, \text{program}) = \text{compiled} \]
Such that \( \text{compiled}(\text{data}) = \text{interpret}(\text{program}, \text{data}) \)

**Second Futamura Projection**
\[ \text{mix}(\text{mix}, \text{interpret}) = \text{compiler} \]
Such that \( \text{compiler}(\text{program}) = \text{compiled} \)

**Third Futamura Projection**
\[ \text{mix}(\text{mix}, \text{mix}) = \text{compilerGenerator} \]
Such that \( \text{compilerGenerator}(\text{interpret}) = \text{compiler} \)
Example: Web Applications

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

Addresses key problem: **integrating multiple models**
Partial Evaluation of Web Interpreter

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

\[ \text{static} \quad \text{dynamic} \]

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

Supports both dynamic interpretation and compiled execution in same framework
Strategic Programming

A different form of modularity

Factor programs into:
- General strategies
- Specifics of problem at hand ➔ model

Examples (no particular order)
- Parsers (Yacc)
- Semantics Data Models
- Dependency models (Make)
- State machines (statecharts)
- Security models
- User interface models (web)
- Workflow models
Traditional Modularity: Procedural Decomposition

Why not just use a library?
- Grammars
  - First/follow set computation
- Regular expressions in Java
  - variable binding problems
- User interfaces
  - instantiate components, then run

Use general-purpose language?
- More difficult to analyze, optimize
  - Static becomes dynamic

Fairly clean in Haskell combinator libraries
Models

“A schematic *description* of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics.”  

*American Heritage Dictionary*

A map of Texas  
Molecular model of DNA  
Economic model  
Regular expression  
SAP Database = model of a business  
Excel model of finances (project, home, etc)
Meta-Models

Description of the structure of a model

A map of Texas
  Planar boundaries, points, labels, elevation, vegetation, etc
Molecular model of DNA
  Sticks and balls, connectivity
Economic model
  Math (equations)
Regular expression
  Grammar: $e ::= c \mid ee \mid e\cdot e \mid e^*$
Database = model of a business
  Named tables with columns, constraints, etc
Excel model of finances (project, home, etc)
  Grid of values and equations, formatting
Model

Person: name=“Bob” manager=Sue
Person: name=“Sue” employees={Bob, ...}

Meta-Model

Type: name=“Person” fields={
  Field: name=“name” type=String
  Field: name=“manager” type=Person optional
    inverse=employees
  Field: name=“employees” type=Person many
    inverse=manager
}

Meta-Meta Model

Type: name=“Type” fields={
  Field: name=“name” type=String key
  Field: name=“fields” type=Field many inverse=owner
}
Type: name=“Field” fields={
  Field: name=“name” Field: name=“owner” type=Type
  Field: name=“type” type=Type
  Field: name=“inverse” optional type=Field
  Field: name=“cardinality” type=one,optional,many
}
What Kind of Models?

User Interface
  Mapping to/from data for web & GUI

Security/Authorization
  Enforced as checks, provide metadata to UI

Triggers and Workflow
  React to conditions, concurrent workflow

Semantic Data Models (graphs)
  Graphs: constraints, computations, relationships

Generic Operations
  Read, write, parse, compare, diff, merge, ...
Create & Modify

View

Security/Constraints

Model

Create & Modify

Render & Notify

Reaction Workflow

24
One Example of Potential Benefit

Acrobat API

6000 pages of documentation (+ 1310 PDF)
395 structures, 1293 functions, 89 enumerations

Microsoft Word Automation API

638 interfaces, 6621 functions, 325/3127 enums

SAP

20,000 to 50,000 tables

Each with UI, security, workflow, consistency...
BoundedValue
  x : Integer
  enforce \( \text{min} \leq x \land x \leq \text{max} \)
  min: read-only Integer
  max: Integer

Semantic Data Model (1978)
(model = description of something of interest)
class BoundedValue {
    private int x;
    private int min;
    private int max;

    // read-only values
    public BoundedValue(int min) {
        this.min = min;
    }

    // x : Integer
    int getX() { return x; }
    void setX(int val) {
        if (!(min <= val && val <= max))
            error("Constraint Violation");
        x = val;
    }

    // min: read-only Integer
    int getMin() { return min; }

    // max: Integer
    int getMax() { return max; }
    void setMax(int val) {
        if (val > max)
            error("Constraint Violation");
        max = val;
    }
}
TransJava(Type T) {
    write(" class " + T.name + " {");
    // omit generation of member variables
    for ( Field F in T.fields ) {
        write(" public " + F.type + " get" + capitalize(F.name) + "() {");
        write(" return " + t.name + ";");
        write(" }");
        if (! F.read-only ) {
            write(" public " + F.type + " set" + capitalize(F.name));
            write("(" + F.type + " val) {");
            for ( Field S in T.Fields )
                if ( affects(F.name, S.constraint ))
                    ... // generate constraint code with x replaced by val
            write(" " + t.name + " = val;");
            write(" }");
        }
    }
    write("}");
}

Can also be done with code quoting
Strategic Programming with Interpreters

Translate : Model → Code
   Run the code to create objects
Define data abstraction using an interpreter
Factory : Model → (Message → Value)
Factory (Type T) {
    D = new Hashtable();
    return lambda (Message m) {
        // iterate over fields in type description
        for (Field F in T.fields)
            if (m.match("get" + capitalize(F.name))
                return D.get(F.name)
        if (!F.read-only) {
            if (m.match("set" + capitalize(F.name), val)) {
                for (Field S in T.Fields)
                    if (affects(F.name, S.constraint))
                        if (!eval(S.constraint, {F.name = val}))
                            error("Constraint violation")
                D.set(F.name, val);
            return;
        }
    }
}
BoundedValue_Factory() {
  D = new Hashtable();
  return lambda (Message m) {
    if (m.match("getX"))
      return D.get("x")
    if (m.match("setX", val))
      if (!(D.get("min") ≤ val ∧ val ≤ D.get("max")))
        error ("Constraint violation")
      D.set("x", val)
    if (m.match("getMin"))
      return D.get("min")
    if (m.match("getMax"))
      return D.get("max")
    if (m.match("setMax", val))
      if (!((D.get("x") ≤ val))
        error ("Constraint violation")
      D.set("max", val)
  }
}
Generic Operations

Tree
  value : Integer
  left : Tree
  right : Tree

Equality

Equal_{Tree}(a, b)
  return Equal_{Integer}(a.value, b.value)
  && Equal_{Tree}(a.left, b.left)
  && Equal_{Tree}(a.right, b.right)

06/27/08
Generic Equality

Equal(M, a, b)
  if (M.primitive)
    cast(M.type, a) == cast(M.type, b)
  else
    for all F in M.fields
      Equal(F.type, a.(F.name), b.(F.name))
Equal(M, a, b)
  if (M.primitive)
    cast(M.type, a) == cast(M.type, b)
  else
    for all F in M.fields
      Equal(F.type, a.(F.name), b.(F.name))

Partially Evaluate Equal(Tree, a, b)...

Equal_{Tree}(a, b)
  return Equal_{Integer}(a.value, b.value)
  && Equal_{Tree}(a.left, b.left)
  && Equal_{Tree}(a.right, b.right)
Issues with Partial Evaluation

Mixing dynamic and static values
evaluate all calls where all arguments are static

Example: Finite State Machine interpreter

```
(define EvenOdd
  '( ((even 1) odd)
      ((even 0) even)
      ((odd 1) even)
      ((odd 0) odd)))
```
Issues with Partial Evaluation

Mixing dynamic and static values

\[
\text{(define (run FSM state input)}
\\text{(if (null? input) state)}
\\text{(let (((new (assoc (list state (car input)) FSM)))})}
\\text{(if new ;; new is dynamic, not static)}
\\text{(run FSM (cadr new) (cdr input))}
\\text{`(error ,state ,(car input))))})
\]

Assoc is called with a dynamic and a static value

Static table: \( A \rightarrow B \) and a dynamic key \( a \)

\[ b = \text{lookup(table, a)} \]

Result is *dynamic*, so little specialization takes place
The “Trick”

Convert lookup to *iteration*

```
(define (run FSM state input)
  (if (null? input) state
   (for trans FSM first
    (if (and (equal? (caar trans) state)
         (equal? (cadar trans) (car input)))
      (run FSM (cadr trans) (cdr input)))
    (run FSM (cadr trans) (cdr input)))
  `(error ,state ,(car input))))
```

Unrolls the static table into a series of conditions
in effect, a case statement

Allows specialization of the table results
(define (run-even input)
  (if (null? input)
      'even
      (if (equal? 1 (car input))
          (run-odd (cdr input))
          (if (equal? 0 (car input))
              (run-even (cdr input))
              (list 'error 'even (car input))))))

(define (run-odd input)
  (if (null? input)
      'odd
      (if (equal? 1 (car input))
          (run-even (cdr input))
          (if (equal? 0 (car input))
              (run-odd (cdr input))
              (list 'error 'odd (car input))))))
Deforestation

Separate logical structure from rendering
  web – creates S-Expression representation of web page
  html – renders S-Expression as text

Web server composes these functions
  server(request) = html(web(UI, DM, db, request))

Need deforestation to remove intermediate structures
Partially Static Values

Consider

\[
\text{web: (.... (list 'TABLE '((BORDER 1)) (web body...))) )}
\]
\[
\text{html: (... (print "<") (print (car x))}
\]
\[
\text{ (for attr (cadr x) ...) (print ")")}
\]
\[
\text{(html (caddr x))}
\]
\[
\text{(print "/") (print (car x)) (print ">")})
\]

If we can bring web and html together:
\[
\text{html-web: (... (print "<TABLE BORDER=1>"})}
\]
\[
\text{(html-web-body ...)
\]
\[
\text{(print "/TABLE>"})}
\]

Note that \[(\text{list ... })\] is partially static

The top level of the list is static
Deforestation

;; rendering to HTML text
(define (html x)
  (if (not (pair? x))
    (print x)
    (begin
      (print "<")
      (print (car x))
      (for attr (cadr x) begin
        (print "=")
        (print (cadr attr))
      )
      (print ">")
      (for c (cddr x) begin (html c))
      (print ">
    "))
)

;; logical structure of table
`((TABLE ((padding 0))
  (TR () ,@(for sub (@ layout 'contents) cons
    `(TD () ,(render sub 'label))))
  ,(for item part cons
    `(TR () ,@(for sub (@ layout 'contents) cons
      `(TD () ,(render sub type item request))))))

;; supercompiled version
(supercompiled version
  (print "<TABLE padding=0><TR>")
  (for sub (@ layout 'contents) begin
    (print "<TD>")
    (print (@ sub 'label))
    (print "</TD>\n")
  )
  (print "</TR>\n")
  (for item (@ part 'items) begin
    (print "<TR>")
    (for sub (@ layout 'contents) begin
      (print "<TD>")
      (html-render sub type item request)
      (print "</TD>\n")
    )
    (print "</TR>\n")
  )
  (print "</TABLE>\n")
)
Deforestation with Partial Evaluation

Integrate with partial evaluation

Merge: \((f\ a..\ (g\ b..)\ c..)\) \implies (f-g\ a..\ b..\ c..)

Driving: \((f\ (if\ a\ b\ c))\) \implies (if\ a\ (f\ b)\ (f\ c))

Specialize on partially static values

(cons x y ) is partially-static; use axioms

\[(\text{null? (cons x y))} \implies \#f\)

\[(\text{car (cons x y))} \implies x\]

\[(\text{cdr (cons x y))} \implies y\]

Environment = Variable \(\rightarrow\) (Value\(^S\) + Expr\(^D\))
Generic Reader

Read(Model, Data, Factory) : Object

Specialized reader

Read_{Model}(Data, Factory)

Read then patch up circularity

Object graph instantiator

Read_{Model, Data}(Factory)

Fixup table is static when data is static

But dynamic objects must be inserted into it
Operators to control binding time

(dynamic e [test])
  Force e to be dynamic
    If test is not given, or is dynamic
  (dynamic (make-table) data)
    Make a static table if data is static

(indirect (var e) body)
  Bind dynamic e to a static identifier within body

(plugin e)
  Evaluate a piece of code from the model
Aspects and Features

Aspect = change to an interpreter
   AspectJ = Modifying residual code!

Change on multiple levels
   Models, Interpreters, Metamodels

Evolution of models
   Tracking differences, Refactoring, Upgrades

Feature Modularity
   Schema is partitioned by “features”
   General composition operators (Batory)
Both features and strategies are cross-cutting
Pummel Language and System

Interpreters
First-order imperative fragment of Scheme with objects

Partial Evaluation “sweet spot?”
The models are static, fully eliminated Non-Turing complete language

Bootstrapping and self-hosting
Change fundamental model/interpretation
Rebuild new version of system
Conclusion

Strategic Programming

- Interpretation of descriptions
- Aspects as interpreter extensions
- Compilation by partial evaluation + deforestation
- Self-implemented system (Scheme)

Not for *all* kinds of programming
- Not for unique programs

Good when problem is *scale*, not intricacy
Related Work/Concepts

Aspects: modify the interpreter
Reflection inversion: description → code
Features: compose models, modify interpreters
Polytypic programming: models generalize types
Template metaprogramming: use partial eval instead
Design patterns: elicit in meta-level interpreter
Debugging: debug the interpreters
Adaptive Programming: meta-strategies?
Types: not sure... untyped meta, typed object?
Applications: eliminate boilerplate
Evolution: compare and transform, rebuild
Description

\[ \text{Object} \]

*and partial evaluation*
Do not Design Your Programs
Program Your Designs!