Optimality for Dynamic Patterns

Thibaut Balabonski

Laboratoire PPS
CNRS and Université Paris Diderot

PPDP 2010: July 28
Computing with data structures

```ocaml
let rec upd f t = match t with
  | Data a -> Data (f a)
  | Node l r -> Node (upd f l) (upd f r)
```

Pattern matching:

- **tests** the shape.
- **extracts** substructures.
New polymorphisms

- **Pattern** polymorphism: parameters in patterns.
- **Path** polymorphism: shape irrelevance.

```ocaml
let rec upd c f t = match t with
  | c d -> c (f d)
  | x y -> (upd c f x) (upd c f y)
  | a -> a
```

**Instantiation of the parameter** `c`

- Constructors like `Data`.
- **Functions building patterns.**
New polymorphisms

- **Pattern** polymorphism: parameters in patterns.
- **Path** polymorphism: shape irrelevance.

```ocaml
let rec upd c f t = match t with
  | Data d -> Data (f d)
  | x y -> (upd c f x) (upd c f y)
  | a     -> a
```

**Instantiation of the parameter** \( c \)

- Constructors like `Data`.
- **Functions building patterns.**
New polymorphisms

- **Pattern** polymorphism: parameters in patterns.
- **Path** polymorphism: shape irrelevance.

```ocaml
let rec upd c f t = match t with
  | (fun x -> Node (Data x) (Data _)) d ->
  | x y -> (upd c f x) (upd c f y)
  | a      -> a
```

**Instantiation of the parameter** $c$

- Constructors like `Data`.
- **Functions building patterns.**
New polymorphisms

- **Pattern** polymorphism: parameters in patterns.
- **Path** polymorphism: shape irrelevance.

let rec upd c f t = match t with
  | Node (Data d) (Data _) -> Node ...
  | x y -> (upd c f x) (upd c f y)
  | a     -> a

### Instantiation of the parameter c

- Constructors like Data.
- **Functions building patterns.**
New polymorphisms

- **Pattern** polymorphism: parameters in patterns.
- **Path** polymorphism: shape irrelevance.

```ml
let rec upd c f t = match t with
  | c d -> c (f d)
  | x y -> (upd c f x) (upd c f y)
  | a -> a

Instantiation of the parameter `c`
- Constructors like `Data`.
- Functions building patterns.
Usual compile-time analyzes

- Node 1 (Data d) \rightarrow
- Node (Data d) (Node _ _) \rightarrow
- Node (Node l r) (Node _ _) \rightarrow

This requires static patterns!

Goal: Run-time sharing mechanisms for dynamic patterns.
Usual compile-time analyzes

<table>
<thead>
<tr>
<th>Node l (Data d)</th>
<th>(Data d) -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node (Data d)</td>
<td>(Node _ _)</td>
</tr>
<tr>
<td>Node (Node l r)</td>
<td>(Node _ _)</td>
</tr>
</tbody>
</table>
Usual compile-time analyzes

| Node l (Data d) | (Node _ _) | ->
| Node (Data d)   | (Node _ _) | ->
| Node (Node l r) | (Node _ _) | ->

This requires static patterns!

Goal

Run-time sharing mechanisms for dynamic patterns.
Roadmap

1. Is optimal sharing a reasonable solution?

2. Families of terms: learning from history

3. (Some syntax)

4. Recording the history

5. From history to memory
Is optimal sharing a reasonable solution?

There is some *mismatch* between theoretical optimality and practical efficiency.

Families of terms: learning from history

(Some syntax)

Recording the history

From history to memory
The trap of optimality

Usual implementations

Extra cost

Sharing

β-Optimality

Usual implementations
The trap of optimality

Extra cost

β-Optimality

Usual implementations

Sharing
The trap of optimality

Extra cost

Usual implementations

Sharing

$\beta$-Optimality
Weak reduction

Usual implementations

- Passing arguments to functions.
- No partial evaluation.

⇒ No evaluation under a binder ($\lambda$).

Confluent weak reduction

No reduction between a bounded variable and its binder.
Is optimal sharing a reasonable solution?

Families of terms: learning from history

Terms with same history should not be considered different.

(Some syntax)

Recording the history

From history to memory
Optimality and history

Optimal optimality?

Families and history

Optimality commandment

At each step you shall reduce a whole family.

- **Family**: all redexes sharing a common history.
- **History**: all events that contributed to the creation of a redex.
- **Redex**: REDucible EXpression.
Which reduction steps are relevant?

Back to basics
- analysis of direct contribution.
- how can redexes be created?

And later
- record contributions in labels.
Is optimal sharing a reasonable solution?
Families of terms: learning from history

(Some syntax)

Digression: the Weak Pure Pattern Calculus.
Simple syntax, complex reduction.

Recording the history
From history to memory
Syntactic trees

- Optimal optimality?
- Families and history

(Some syntax)

Recording the history

History and memory

T. Balabonski — Optimality for Dynamic Patterns

PPDP 2010: July 28 — 10/20
Pattern matching: a complex operation

Failure
If $A$ and $P$ incompatible, reduction to $\bot$.

Partial definition
Matching undefined for improper terms.
Pattern matching: a meta-operation

\[
\begin{align*}
\{a/\theta \hat{x}\} &= \{x \leftarrow a\} & x & \in \theta \\
\{\hat{x}/\theta \hat{x}\} &= \{\} & x & \notin \theta \\
\{a_1a_2/\theta p_1p_2\} &= \{a_1/\theta p_1\} \cup \{a_2/\theta p_2\} & a_1a_2, p_1p_2 & \text{matchable} \\
\{a/\theta p\} &= \bot & a, p & \text{matchable}
\end{align*}
\]

Undefined otherwise!

Matchable forms

- \(\hat{x} \ t_1 \ldots \ t_n\)
- \([\theta]p \rightarrow b\)
Recording the history

*Labels to record contributions and define redex families.*

Is optimal sharing a reasonable solution?

Families of terms: learning from history

(Some syntax)
Contributions can be recorded by labels.

- **Family**: all redexes sharing a common label.
Forward: The consequences of my deeds

\[ \lambda \theta \]

\[ x_i \quad x_j \]

\[ \Omega \]

initial labels, £, …

atomic labels α ::= ¡ | ⌜ Ω ⌝ | | Ω

labels Γ, ∆, Ω ::= α₁ α₂ … αₙ
Forward: The consequences of my deeds
Forward: The consequences of my deeds
Forward: The consequences of my deeds

\[ \lambda \theta \]

\[ x_i, x_j \]

\[ \Omega \]

\[ W \]

\[ \lambda \]

\[ \alpha \]

initial labels, \¡, ¢, £...

atomic labels

\[ \alpha_1 \alpha_2 \ldots \alpha_n \]

labels \( \Gamma, \Delta, \Omega \)
Forward: The consequences of my deeds

Initial labels: $\alpha, \beta, \gamma, \ldots$

Atomic labels: $\alpha$ ::= $\alpha$ | $\neg \Omega \neg$ | $|\Omega, \alpha|$}

Labels: $\Gamma, \Delta, \Omega$ ::= $\alpha_1 \alpha_2 \ldots \alpha_n$
Backward: Where do I come from?

Name of the redex

\[ \Omega = \alpha \Gamma \Delta_{pat} \Delta_{arg} \]
Pattern matching and history extraction

\[
\begin{align*}
\{Y/\theta \alpha : Z\}_m &= \alpha : \{Y/\theta Z\}_m & m \in \{l, w\} \\
\{Y/\theta \hat{x}\}_l &= (\epsilon, \{x \leftarrow Y\}) & x \in \theta \\
\{Y/\theta \hat{x}\}_w &= (|Y|, \{x \leftarrow Y\}) & x \in \theta \\
\{Y/\theta ([\tau]p \rightarrow b)\}_l &= (\epsilon, \bot) \\
\{Y/\theta ([\tau]p \rightarrow b)\}_w &= (|Y|, \bot) \\
\{\alpha : Z/\theta X\}_m &= \alpha : \{Z/\theta X\}_m & \text{matchability} \\
\{\hat{x}/\theta \hat{x}\}_m &= (\epsilon, \{\}) & x \not\in \theta \\
\{A_1A_2/\theta P_1P_2\}_m &= \{A_1/\theta P_1\}_w \uplus \{A_2/\theta P_2\}_l & \text{matchability} \\
\{Y/\theta X\}_m &= (|Y|\mid X|, \bot) & \text{matchability} \\
\{Y/\theta X\}_m &= \text{undefined} & \text{otherwise}
\end{align*}
\]

\[
\begin{array}{l|llll}
|\theta| P \rightarrow B & = \varepsilon & \oplus & (\Delta_2, \sigma_2) & (\Delta_2, \bot) & \text{undef.} \\
|\hat{x}| & = \varepsilon & (\Delta_1, \sigma_1) & (\Delta_1 \Delta_2, \sigma_1 \uplus \sigma_2) & (\Delta_1 \Delta_2, \bot) & \text{undef.} \\
|T_1 T_2| & = |T_1| & (\Delta_1, \bot) & (\Delta_1, \bot) & (\Delta_1, \bot) & (\Delta_1, \bot) & \text{undef.} \\
|\alpha : Z| & = \alpha\mid Z| & \text{undef.} & \text{undef.} & \text{undef.} & \text{undef.}
\end{array}
\]
Is optimal sharing a reasonable solution?

Families of terms: learning from history

(Some syntax)

Recording the history

From history to memory

*From labelled terms to directed acyclic graphs.*
Labels as memory locations

Sharing property: equal labels $\Rightarrow$ equal terms.
Labels as memory locations

Graph reduction

Sharing property: equal labels $\Rightarrow$ equal terms.
Labels as memory locations

Labelled development?

Graph reduction

Sharing property: equal labels $\Rightarrow$ equal terms.
From labels to graph reduction

Complete labelled reduction:
development of all redexes with a given label.

Sharing theorem

Complete labelled reduction preserves the sharing property.

Hypothesis: start with initial labels.
Key: contribution is characterized by labels.
Conclusion

Present results

A causality analysis can yield

- a concrete DAG implementation.
- a proof of optimality of call-by-need.

Through PPC, this work reaches as well λ-calculus and pattern matching à la ML.

Future work

- Sharing in functions with multiple cases.
- Effective reduction strategy.
A broader picture

Optimal optimality?
Families and history
(Some syntax)
Recording the history
History and memory

Pure Pattern Calculus

Sharing in Weak PPC PPDP’10

Explicit Matching HOR’10

Alternatives

Call-by-need

Efficient Evaluation

...and questions...
A broader picture

Optimal optimality?
Families and history
(Some syntax)
Recording the history
History and memory

Pure Pattern Calculus

Sharing in Weak PPC PPDP’10

Explicit Matching HOR’10

Alternatives

Efficient Evaluation

Call-by-need

...and questions...