Efficient Pattern Matching in Python

Manuel Krebber, Henrik Barthels, Paolo Biantinesi
Introduction

Terms

Function Symbols: $f, g$

Constants: $a, b, c$

Variables: $x, y$

Examples: $a, f(x, b), f(g(a, b), c)$
Introduction

Pattern Matching

**Definition:** Find substitution \( \sigma \) such that \( \sigma(\text{pattern}) = \text{subject} \)

**Example:** Pattern: \( f(x,y) \)

\[
\begin{array}{c}
  x & \mapsto ? \\
  y & \mapsto ? \\
\end{array}
\]

Subject: \( f(a, g(b)) \)
**Pattern Matching**

**Definition:** Find substitution $\sigma$ such that $\sigma(\text{pattern}) = \text{subject}$

**Example:** Pattern: $f(x, y)$

\[
\begin{align*}
x & \mapsto a \\
y & \mapsto g(b)
\end{align*}
\]

Subject: $f(a, g(b))$
Introduction

Applications

• Functional programming languages
• Computer algebra systems (Mathematica, SymPy)
• Term rewriting systems
• Theorem proving
• Abstract syntax trees
Types of Matching

- Syntactic
- Associativity
- Sequence variables
- Commutativity
- Many-to-one vs. one-to-one
Types of Matching

Associativity I

\[(A \times B) \times C = A \times (B \times C) = A \times B \times C\]
Types of Matching

**Associativity II**

\[ X \times M_3 \]

\[ M_1 \times M_2 \times M_3 \]
Types of Matching

Associativity II

\[ X \times M_3 \quad (M_1 \times M_2) \times M_3 \quad M_1 \times M_2 \times M_3 \]

\[ \sigma = \{ X \mapsto (M_1 \times M_2) \} \]
Types of Matching

Sequence Variables

Can match a sequence of terms

Notation: $x^*$, $x^+$

Example: $\sigma(f(a, x^+, d)) = f(a, b, c, d)$  \hspace{1cm} $\sigma = \{x^+ \mapsto (b, c)\}$

Associativity: $\sigma(f_a(a, x, d)) = f_a(a, b, c, d)$  \hspace{1cm} $\sigma = \{x \mapsto f_a(b, c)\}$
Types of Matching

Example 1

```python
a_lt_b = CustomConstraint(lambda a, b: a < b)

pattern = Pattern([h___, b_, a_, t___], a_lt_b)

rule = ReplacementRule(pattern,
    lambda a, b, h, t: [*h, a, b, *t])

replace_all([1, 4, 3, 2], [rule])

>>> [1, 2, 3, 4]
```
Types of Matching

Example II

```python
x_sums_to_5 = CustomConstraint(lambda x: sum(x) == 5)
pattern = Pattern([___, x___, ___], x_sums_to_5)
list(match([1, 2, 3, 1, 1, 2], pattern))
>>> [{'x': (2, 3)}, {'x': (3, 1, 1)}]
```
Types of Matching

Commutativity

- \( a + b = b + a \)
- Pattern: \( x_1 + x_2 + x_3 \)
- Subject: \( a + b + c \)
- Every permutation is a match.
## Types of Matching

### Complexity

<table>
<thead>
<tr>
<th></th>
<th>Synt</th>
<th>Assoc</th>
<th>SeqVar</th>
<th>Comm</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. # matches</td>
<td>1</td>
<td>(\binom{n-1}{m-1})</td>
<td>(\binom{n+m-1}{m-1})</td>
<td>(n!)</td>
<td>(n^m)</td>
</tr>
<tr>
<td>NP complete</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

\(n = |\text{subject}|, \ m = |\text{pattern}|\)
Types of Matching

Many-to-one Matching

- Many patterns, one subject
- Speedup by simultaneous matching
- Use similarity between patterns
Discrimination Net

Patterns: \( f(1, x^*), f(1), f(y, 0) \)
Algorithms

Discrimination Net

Patterns: \( f(1, x^*), f(1), f(y, 0) \)

Subject: \( f(1, 0) \)

Match: \( f(y, 0) \) with \( \{ y \mapsto 1 \} \)
Discrimination Net

Patterns: \( f(1, x^*) \), \( f(1) \), \( f(y, 0) \)

Subject: \( f(1, 0) \)
Match: \( f(1, x^*) \) with \( \{ x^* \mapsto 0 \} \)
### Discrimination Net

Patterns: $f(1, x^*)$, $f(1)$, $f(y, 0)$

Subject: $f(1, 0)$

Match: $f(1, x^*)$ with $\{x^* \mapsto 0\}$

Tradeoff: Construction time.
Code Generation

- Inspired by parser generators.
- Performance.
- Portability.
Experiments

- MatchPy: https://github.com/hpac/matchpy [Kre17]
- Hardware: 2 Intel Xeon E5-2670 v2 2.5GHz CPUs
  - 10 cores
  - 64GB RAM
- multiprocessing

Implementations

- One-to-one.
- Many-to-one.
- Code generation.
- Parallel one-to-one.
Experiments

Linear Algebra

- Patterns: \( \sim 200 \)
- BLAS/LAPACK kernels, e.g. \( \alpha A^T B + \beta C, \lambda^{-1} B \)
- Subjects: \( \sim 140 \) test problems
- Taken from the linear algebra compiler Linnea.
Experiments

Side Note: Linnea

- How to compute the following expressions?
  
  \[ b := (X^T X)^{-1} X^T y \]
  
  \[ x := (A^{-T} B^T B A^{-1} + R^T [\Lambda(R_z)] R)^{-1} A^{-T} B^T B A^{-1} y \]
  
  \[ x_{ij} := A_i B c_j \]

- Tradeoff: Ease of use vs. performance.

\[ w = A*(B\backslash c) \]

```python
ml0 = A; ml1 = B; ml2 = c;
potrf!('L', ml1)
trsv!('L', 'N', 'N', ml1, ml2)
trsv!('L', 'T', 'N', ml1, ml2)
ml3 = Array{Float64}(n)
gemv!('N', 1.0, ml0, ml2, 0.0, ml3)
w = ml3
```
### Experiments

#### Speedup over One-to-one Matching

<table>
<thead>
<tr>
<th>Matching</th>
<th>Average time [ms]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-to-one</td>
<td>15.75</td>
<td>1</td>
</tr>
<tr>
<td>Many-to-one</td>
<td>0.46</td>
<td>47</td>
</tr>
<tr>
<td>Code gen</td>
<td>0.28</td>
<td>119</td>
</tr>
<tr>
<td>Parallel</td>
<td>16.13</td>
<td>0.98</td>
</tr>
</tbody>
</table>
Experiments

Abstract Syntax Trees

- `lib2to3`
- Patterns: 36 `lib2to3` → 1200 MatchPy
- Subjects: 900 lines of code from unittests
## Experiments

### Speedup over One-to-one Matching

<table>
<thead>
<tr>
<th>Matching</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many-to-one</td>
<td>165</td>
</tr>
<tr>
<td>Code gen</td>
<td>215</td>
</tr>
<tr>
<td>Parallel</td>
<td>0.28</td>
</tr>
</tbody>
</table>

### Speedup over `lib2to3` Matching

<table>
<thead>
<tr>
<th>Matching</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many-to-one</td>
<td>3.8</td>
</tr>
<tr>
<td>Code gen</td>
<td>4.9</td>
</tr>
</tbody>
</table>
Experiments

Logic

• Convert boolean formulas to algebraic normal form (ANF) [KM01].
• Patterns: 10 rules
• Example: $\neg x \to x \oplus \top$
• Subjects: Randomly generated.
## Experiments

### Speedup over One-to-one Matching

<table>
<thead>
<tr>
<th>Matching</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many-to-one</td>
<td>1.08</td>
</tr>
<tr>
<td>Code gen</td>
<td>1.45</td>
</tr>
</tbody>
</table>
Experiments

Replacement Rules

```python
# and(x, \top) \rightarrow x
...
# and(x, \bot) \rightarrow \bot
...
# and(x, x) \rightarrow x
...
# and(x, \text{xor}(y, z)) \rightarrow \text{xor}(\text{and}(x, y), \text{and}(x, z))
ReplacementRule(
    Pattern(LAnd(x_, LXor(y_, z_))),
    lambda x, y, z: LXor(LAnd(x, y), LAnd(x, z))
),
```
Experiments

Handwritten Code

```python
if isinstance(expr, LAnd):
    if LBot in args:
        return LBot
    args = set(args)
    args.discard(LTop)
    args = list(args)
    for i, o in enumerate(args):
        if isinstance(o, LXor):
            and_args = args[:i] + args[i+1:]
            xor_args = [LAnd(*and_args, arg) for arg in o]
            return hand_coded_simplify(LXor(*xor_args))
    if not args:
        return LTop
    return LXor(*args)
```
Experiments

Speedup over Generated Code

<table>
<thead>
<tr>
<th>Matching</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand-coded</td>
<td>1510</td>
</tr>
</tbody>
</table>
Symbolic Integration

- Rule-based symbolic integrator [RJ09].
- 6000 replacement rules.
- Example: $x^m \rightarrow \frac{1}{m+1}x^{m+1}$ with $x$ not in $m$, and $m + 1 \neq 0$.
- We use 151 patterns.
- 100 subjects from test problems.
Experiments

Speedup over One-to-one Matching
Conclusions

Contributions

• Many-to-one matching, i.e. generalized discrimination nets for
  – sequence variables
  – associativity/commutativity
• Open source implementation
Thank you for your attention

Questions?
Hélène Kirchner and Pierre-Etienne Moreau. 
Promoting Rewriting to a Programming Language: A Compiler for Non-Deterministic Rewrite Programs in Associative-Commutative Theories. 

Manuel Krebber. 
Non-linear Associative-Commutative Many-to-One Pattern Matching with Sequence Variables.  

Albert D Rich and David J Jeffrey. 
A Knowledge Repository for Indefinite Integration Based on Transformation Rules. 