Efficient Pattern Matching in Python

Manuel Krebber, **Henrik Barthels**, Paolo Bientinesi

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Introduction

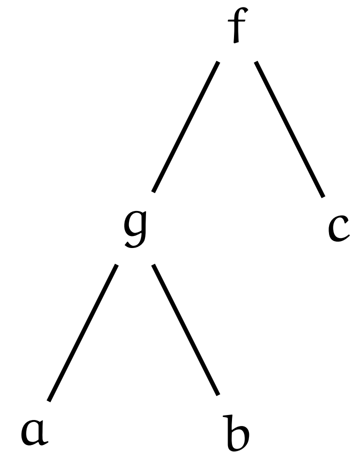
Terms

Function Symbols: f, g

Constants: a, b, c

Variables: $\mathbf{x}, \mathbf{y}$

Examples: a, $f(\mathbf{x}, b)$,
$f(g(a, b), c)$



Introduction

Pattern Matching

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

Example: Pattern: $f(\mathbf{x}, \mathbf{y})$

$\mathbf{x} \mapsto ?$

$\mathbf{y} \mapsto ?$

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

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Pattern Matching

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

Example: Pattern: $f(\mathbf{x}, \mathbf{y})$

$$\Bigg\downarrow \begin{array}{l} \mathbf{x} \mapsto a \\ \mathbf{y} \mapsto g(b) \end{array}$$

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

Introduction

Applications

- Functional programming languages
- Computer algebra systems (Mathematica, SymPy)
- Term rewriting systems
- Theorem proving
- Abstract syntax trees

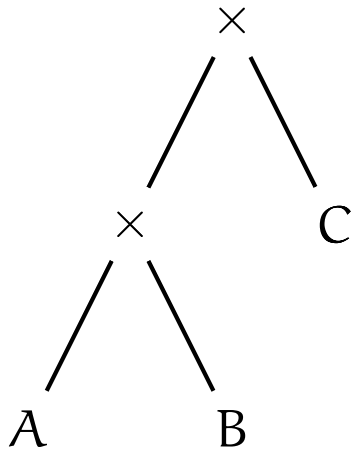
Types of Matching

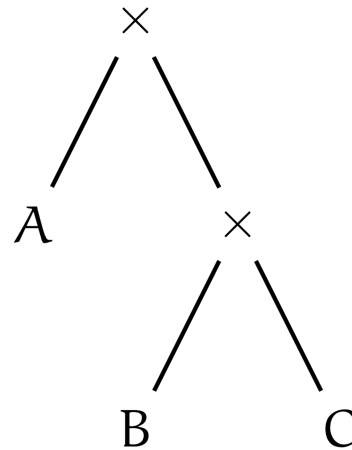
- Syntactic
- Associativity
- Sequence variables
- Commutativity
- Many-to-one vs. one-to-one

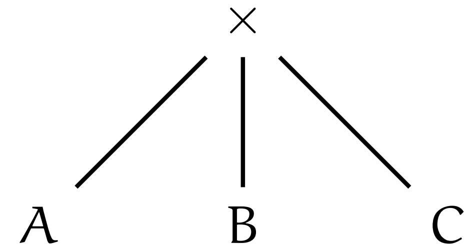
Types of Matching

Associativity I

$$(A \times B) \times C = A \times (B \times C) = A \times B \times C$$

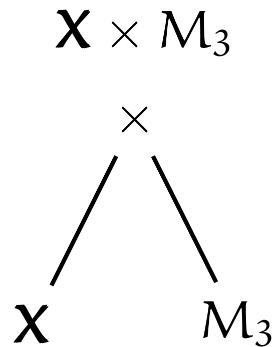


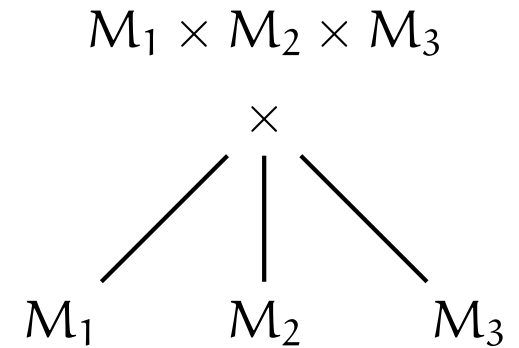




Types of Matching

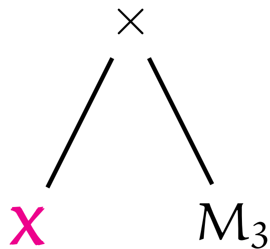
Associativity II



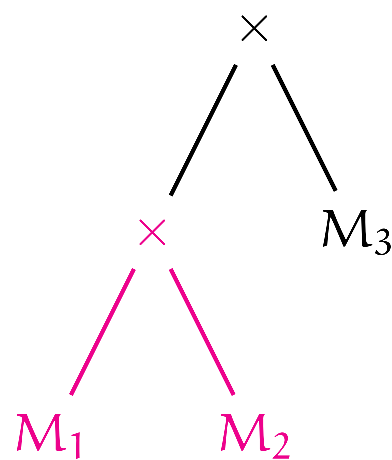


Associativity II

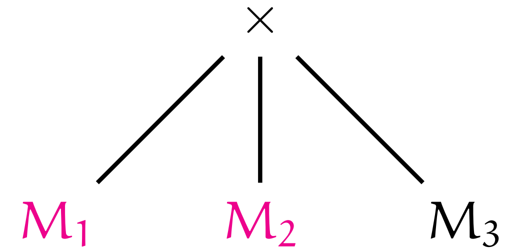
$\mathbf{X} \times M_3$



$(M_1 \times M_2) \times M_3$



$M_1 \times M_2 \times M_3$



$$\sigma = \{\mathbf{X} \mapsto (M_1 \times M_2)\}$$

HPAC High Performance and Automatic Computing

RWTH AACHEN UNIVERSITY

Types of Matching

Sequence Variables

Can match a sequence of terms

Notation: $\mathbf{x}^*, \mathbf{x}^+$

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

Associativity: $\sigma(f_a(a, \mathbf{x}, d)) = f_a(a, b, c, d)$ $\quad \sigma = \{\mathbf{x} \mapsto f_a(b, c)\}$

Types of Matching

Example I

```
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]
```


HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Types of Matching

Example II

```
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: $\mathbf{x_1} + \mathbf{x_2} + \mathbf{x_3}$
- Subject: $a + b + c$
- Every permutation is a match.

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Types of Matching

Complexity

	Synt	Assoc	SeqVar	Comm	All
Max. # matches	1	$\binom{n-1}{m-1}$	$\binom{n+m-1}{m-1}$	$n!$	n^m
NP complete	no	yes	yes	yes	yes

$$n = |\text{subject}|, \; m = |\text{pattern}|$$

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

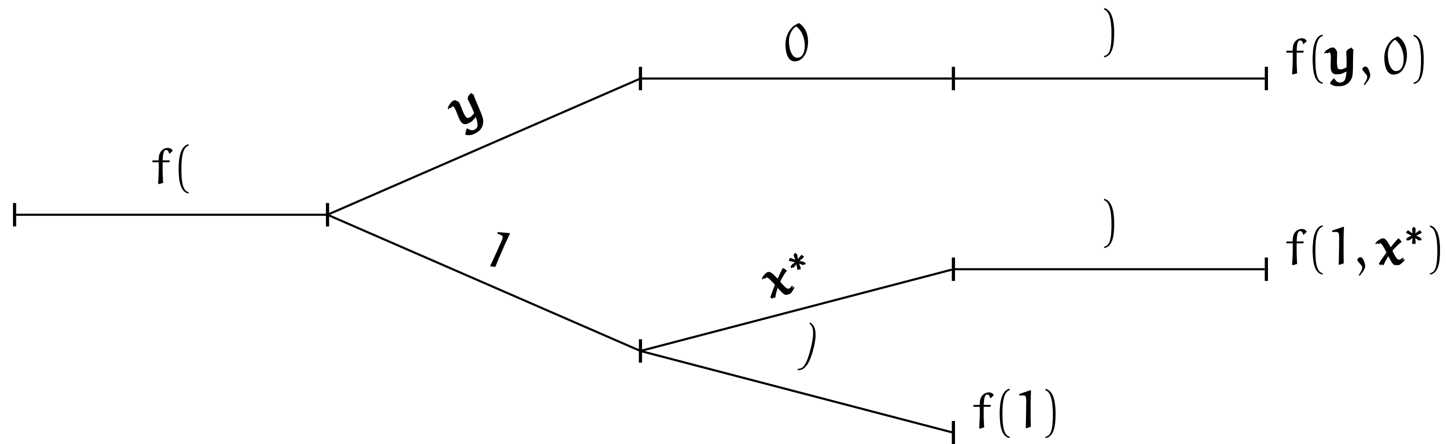
Types of Matching

Many-to-one Matching

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

Discrimination Net

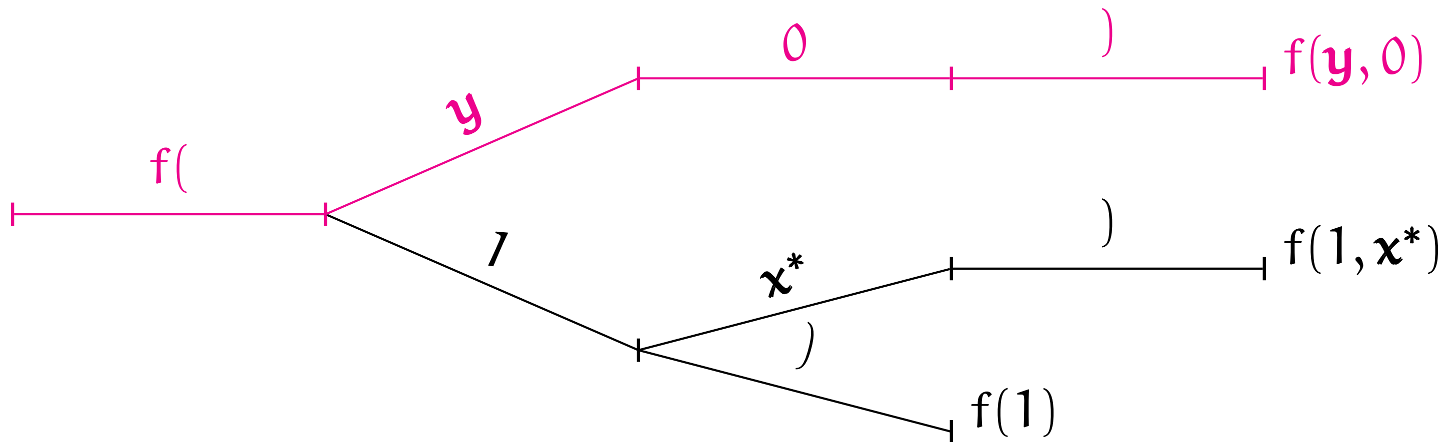
Patterns: $f(1, x^*)$, $f(1)$, $f(y, 0)$



HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Discrimination Net

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

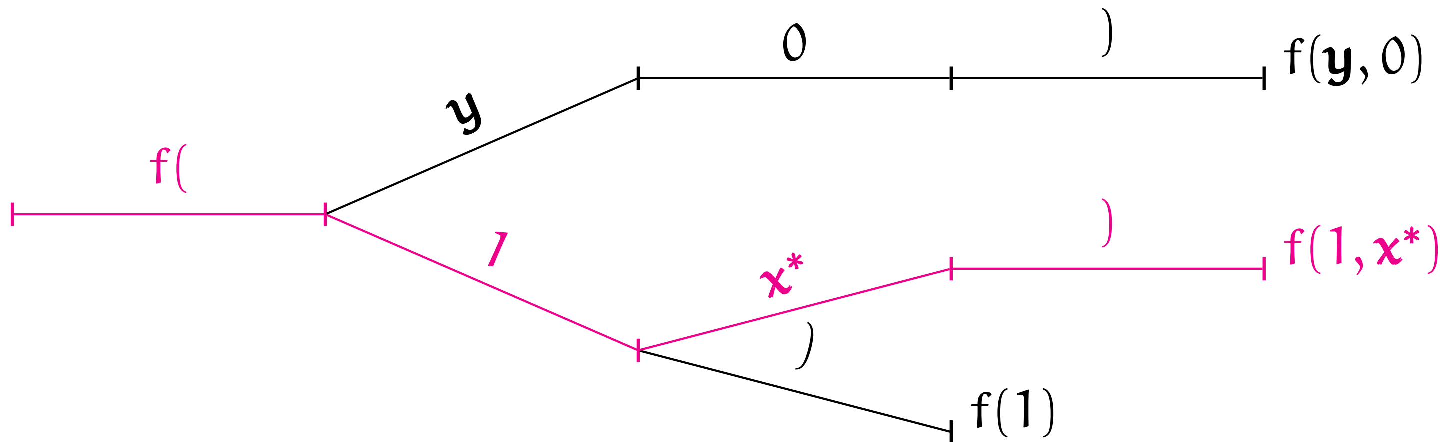


Subject: $f(1, 0)$
Match: $f(y, 0)$ with $\{y \mapsto 1\}$

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Discrimination Net

Patterns: $f(1, \mathbf{x}^*)$, $f(1)$, $f(\mathbf{y}, 0)$



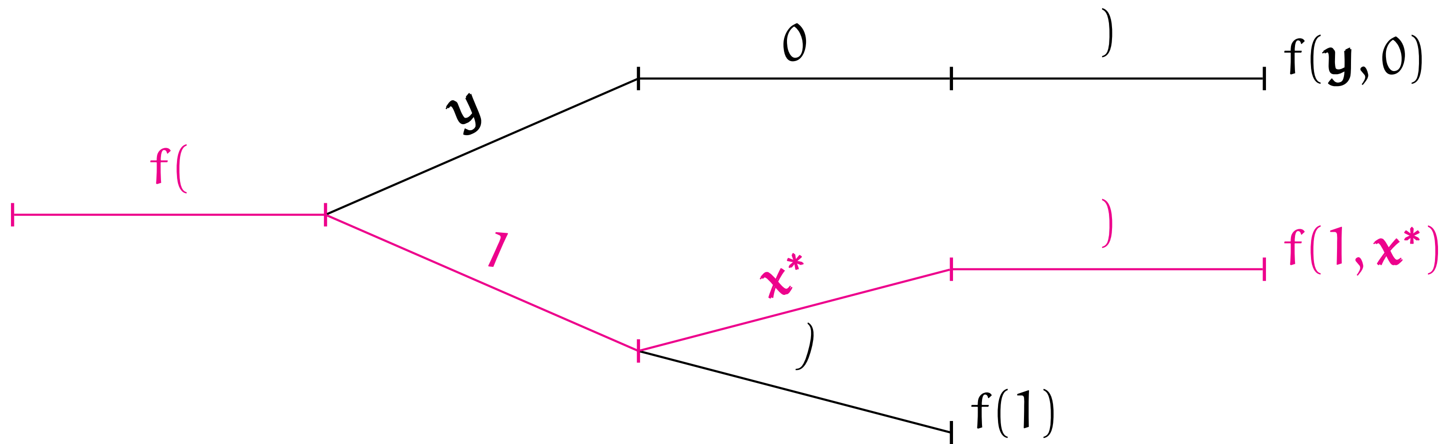
Subject: $f(1, 0)$

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

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Discrimination Net

Patterns: $f(1, \mathbf{x}^*)$, $f(1)$, $f(\mathbf{y}, 0)$



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

Tradeoff: Construction time.

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Algorithms

Code Generation

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

HPAC High Performance and Automatic Computing | RWTH AACHEN UNIVERSITY

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: ~ 200
- BLAS/LAPACK kernels, e.g. $\alpha A^T B + \beta C$, $A^{-1}B$
- Subjects: ~ 140 test problems
- Taken from the linear algebra compiler *Linnea*.

HPAC High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

Side Note: Linnea

- How to compute the following expressions?

$$b := (X^T X)^{-1} X^T y$$

$$x := \left(A^{-T} B^T B A^{-1} + R^T [\Lambda(Rz)] R\right)^{-1} A^{-T} B^T B A^{-1} y$$

$$x_{ij} := A_i B c_j$$

- Tradeoff: Ease of use vs. performance.

```
w = A*(B\c)
```

```
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

Matching	Average time [ms]	Speedup
One-to-one	15.75	1
Many-to-one	0.46	47
Code gen	0.28	119
Parallel	16.13	0.98

Abstract Syntax Trees

- `lib2to3`
- Patterns: 36 `lib2to3` $\rightarrow$ 1200 MatchPy
- Subjects: 900 lines of code from unittests

power
trailer
arglist
atom
testlist_gexp
isinstance (any , (any . . . any))

Experiments

Speedup over One-to-one Matching

Matching	Speedup
Many-to-one	165
Code gen	215
Parallel	0.28

Speedup over `lib2to3` Matching

Matching	Speedup
Many-to-one	3.8
Code gen	4.9

Experiments

Logic

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

Speedup over One-to-one Matching

Matching	Speedup
Many-to-one	1.08
Code gen	1.45

Experiments

Replacement Rules

```
# and(x, ⊤) → x
...
# and(x, ⊥) → ⊥
...
# and(x, x) → x
...
# and(x, xor(y, z)) → xor(and(x, y), and(x, z))
ReplacementRule(
    Pattern(LAnd(x_, LXor(y_, z_))),
    lambda x, y, z: LXor(LAnd(x, y), LAnd(x, z))
),
```

Experiments

Handwritten Code

```
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)
```

Speedup over Generated Code

Matching	Speedup
Hand-coded	1510

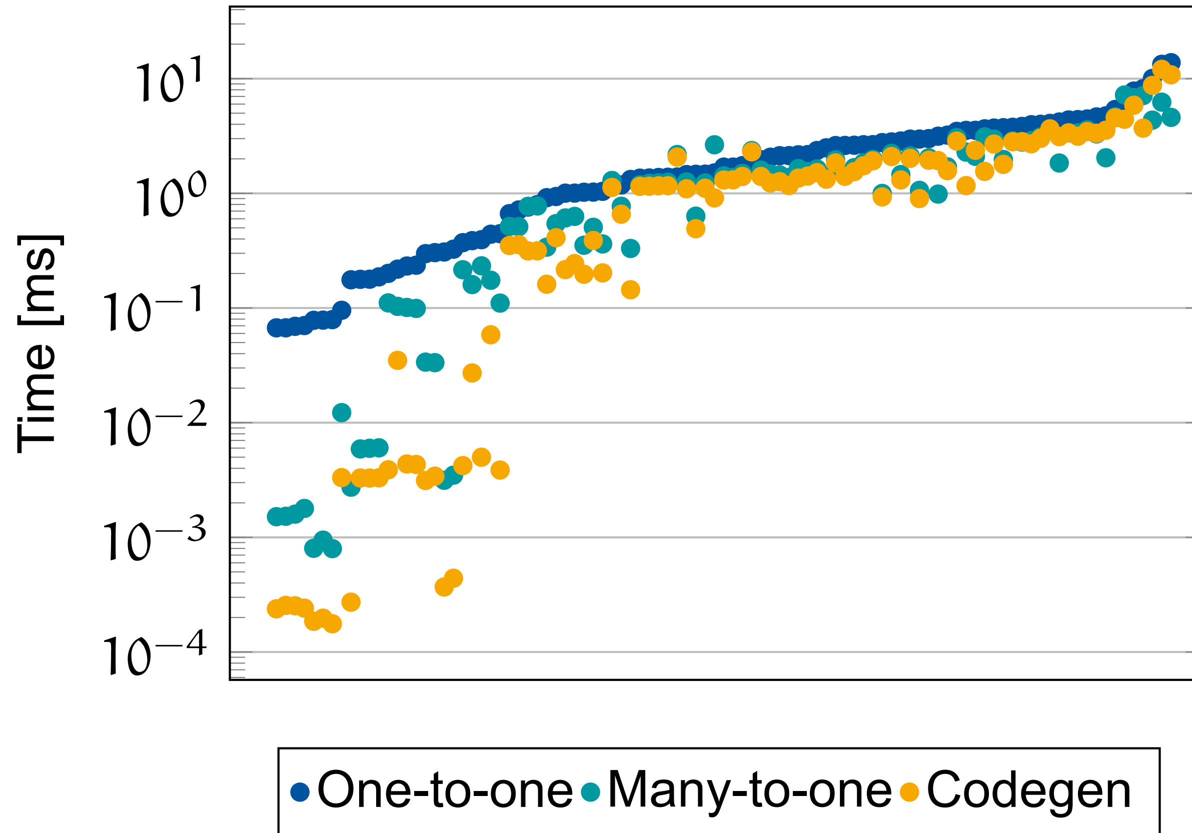
Experiments

Symbolic Integration

- Rule-based symbolic integrator [RJ09].
- 6000 replacement rules.
- Example: $x^m \to \frac{1}{m+1}x^{m+1}$ 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?

References

- Hélène Kirchner and Pierre-Etienne Moreau.
  Promoting Rewriting to a Programming Language: A Compiler for Non-Deterministic Rewrite Programs in Associative-Commutative Theories.
  Journal of Functional Programming, 11(2):207–251, 2001.
- Manuel Krebber.
  Non-linear Associative-Commutative Many-to-One Pattern Matching with Sequence Variables.
  CoRR, abs/1705.00907, 2017.
- Albert D Rich and David J Jeffrey.
  A Knowledge Repository for Indefinite Integration Based on Transformation Rules.
  Calculemus/MKM, 5625:480–485, 2009.