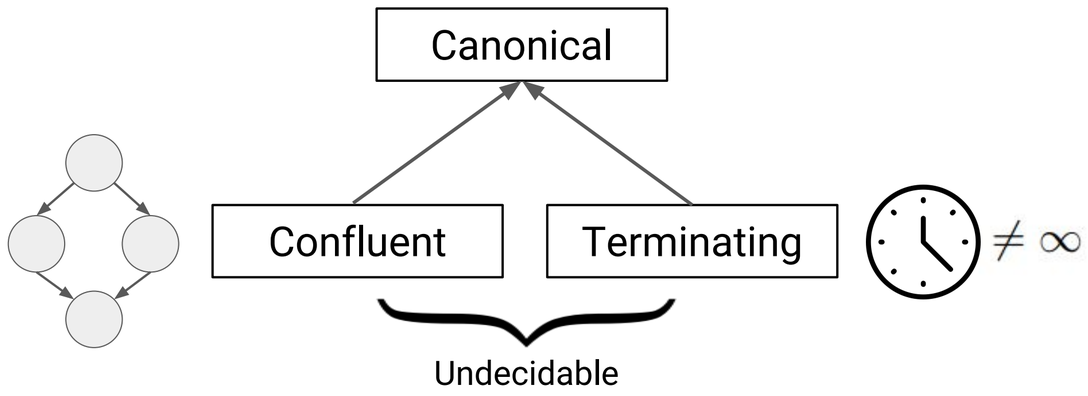
Term Rewriting Using Rewrite.jl

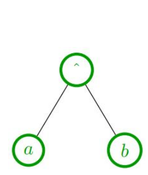
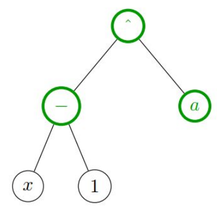
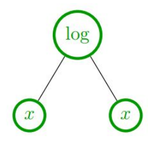
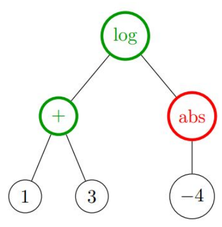
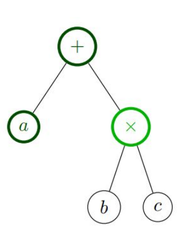
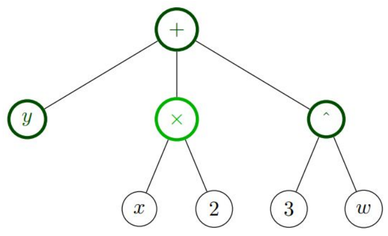
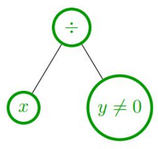
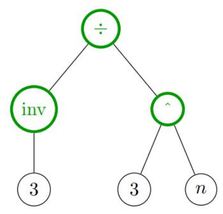
Harrison Grodin and Yijia Chen

Goal

Create a **modular**, **configurable** term rewriting library capable of generating **canonical** rewrite systems for simplifying expressions using **custom identities**.



Matching

Description	Pattern	Subject
Semantic Match: The function names and arities are equivalent, and each subject argument matches its corresponding pattern argument.		
Semantic Match Failure: The function names and arities are equivalent, but the two arguments of the subject are not structurally equivalent and fail to match.		
Property-Based Match: The function names are equivalent. The variable pattern argument matches a sequence of subject arguments, and the remaining argument is matched semantically.		
Constraint-Based Match: The first argument matches as usual, and the second argument is determined to be always nonzero and matches.		

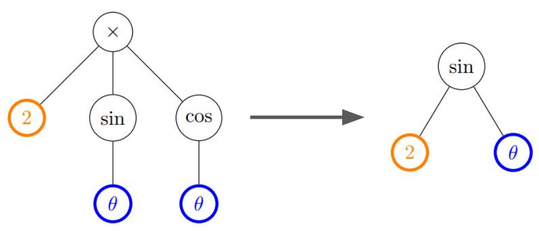
Properties of Common Functions:

Function Name	+	×	&	|
Properties	flat, orderless	flat	flat, orderless	flat, orderless

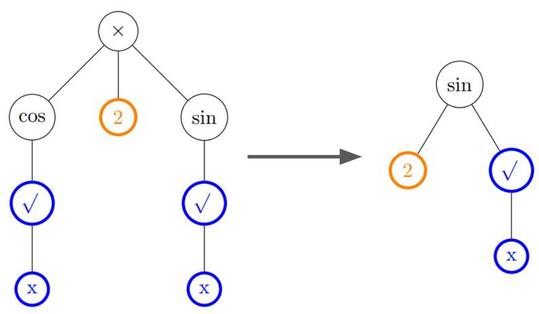
Normalization

Rule:
A normalization rule is composed of an input pattern and an output pattern.

- Often included in rule sets by default, evaluation rules compute specific functions with all constant arguments, such as 1+2.



Term Normalization:
If a term matches the input pattern of a rule, the resulting substitution is used to replace variables in the output pattern of the rule. This process is repeated until the term does not match the input patterns of any rule.

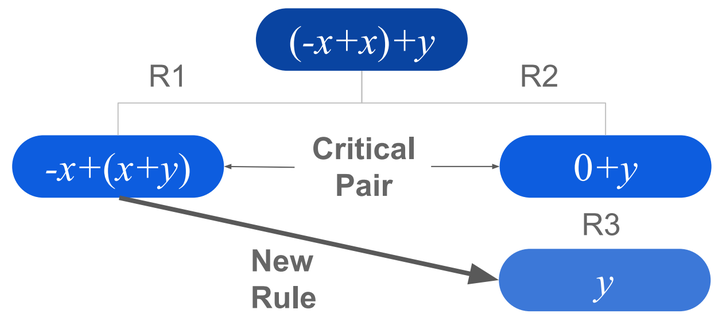


Completion

Rewrite System Generation:

1. An axiom is oriented into a rule based on a provided reduction order.
2. Term intersections of the input patterns of the current rules, called **critical terms**, are found and rewritten by their source rules. Results that are not equivalent form a **critical pair**.
3. A new rule is generated between each critical pair based on the reduction order.

The rewrite system is completed when no axioms or critical terms remain.



Results

```
In [1]: using Rewrite

In [2]: normalize(@term(cos(π/2 - x) / sin(π/2 - x)))
Out[2]: @term(tan(x))

In [3]: normalize(@term(log(b, 1 / b^37)))
Out[3]: @term(-37)

In [4]: using SpecialSets

        x = Variable(:x, TypeSet(Int))
        normalize(@term(diff(sqrt(3*$x + 1), $x)))
Out[4]: @term(3 * inv(2) * inv(sqrt(1 + 3x)))

In [5]: normalize(@term(abs(2^x)))
Out[5]: @term(2 ^ x)

In [6]: normalize(@term(x^2*1 + 0*y))
Out[6]: @term(x ^ 2)

In [7]: ex = :((sin(x)^2 + cos(x)^2) ^ 5)
        ex_norm = parse(normalize(convert(Term, ex)))

        x = rand(Float64, (2, 2))

        @show eval(ex);
        @show eval(ex_norm);

        eval(ex) = [1.0 -2.77556e-16; 0.0 1.0]
        eval(ex_norm) = [1.0 0.0; 0.0 1.0]
```

Future Work

Features:

- Derive term image sets based on wrapper functions
- Convert Julia function calls on term objects to *Fn* objects
- Named rules and automated theorem proving
- Custom REPL mode
- Color and LaTeX display modes
- Support for infinite terms

Improvement:

- Compatibility with existing libraries
- Context construction and storage
- Unions, set differences, and new sets in SpecialSets.jl
- Efficiency of flat and orderless matching

Track progress or contribute at: github.com/HarrisonGrodin/Rewrite.jl

References

- "Term Rewriting and All That" (Baader and Nipkow)
- "MatchPy" and related research (Krebber et al.)
- "An Introduction to Knuth–Bendix Completion" (A. J. J. Dick)
- "A complete proof of correctness of the Knuth-Bendix completion algorithm" (Gérard Huet)
- Combinatorics.jl (JuliaMath organization)
- DiffRules.jl (JuliaDiff organization)