An Interface for exploring the application of rewrite rules in Lift

Euan McGrevey

Context

Hardware has become increasingly parallel in recent years

Resulting in -> Performance Portability and Programmability.

Lift - Research Project (University of Glasgow, Edinburgh, and more)

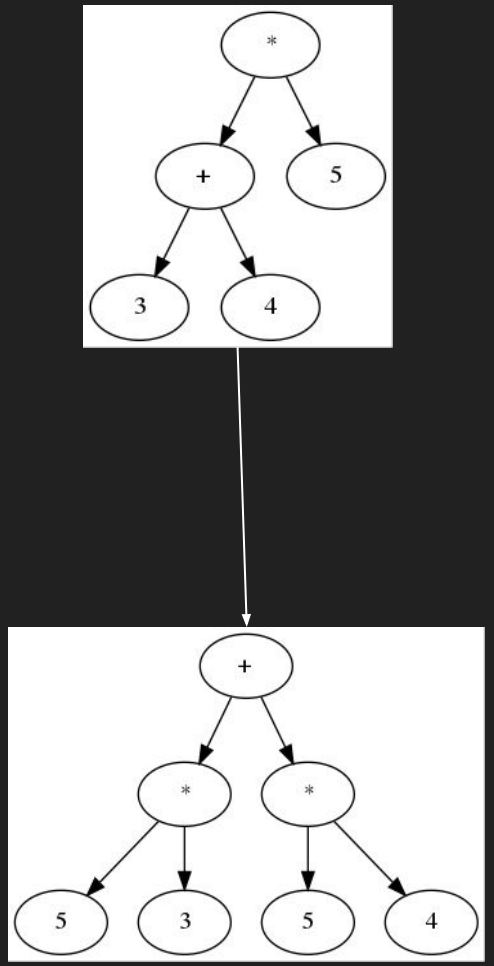
High level functional language + Rewrite rules to tackle these problems

Motivation

Transforming expressions - what does that mean?

Rewrite rules as a way to encode optimisation choices

Why do we care about turning some expression into another?



Requirements

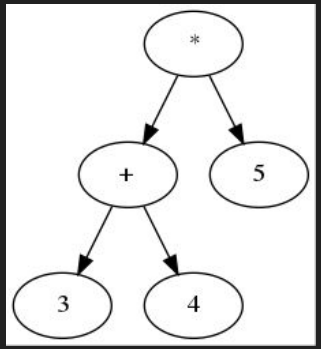
Given some starting expression and some goal expression, and a set of rewrite rules to use, can we get from the start to the goal?

- Determine whether it is possible to turn a given expression into some other expression.

- Bonus: Provide a means to replicate the transformation.

Design

- Expressions as Binary Trees - Extension to full program ASTs.



- Scala? - Pattern Matching is key.

- Rewrite rules and Rewrite results

Implementation

- Series of modular functions

- Disregard efficiency for sake of ease of programmability

- Worst case time complexity is exponential, so limiting factor.

Case Study: Arithmetic Expressions

- If the transformation is possible/impossible, the function will output so.
- However, nonoptimal solutions in most cases

Conclusion

- Improvements

- Future Work