Type-Safe Modular Hash consing Library

A Comparative study in Haskell and Rust

Karan Ahluwalia — Advisor: Dr. Matthew Fluet

Haskell

- *Pure Functional Language*
- *Emphasizes type safety*
- *Ideal for constructing pure versions*

Rust

- *Systems programming language*
- *Concurrency & safety focused*
- *Suitable for performance-critical applications*

Implementations

- ***Pure Version:*** *Emphasizing functional purity and correctness.*
- ***Efficient Single-threaded Version:*** *Utilizing weak references for optimized garbage collection.*
- ***Efficient Thread-safe Version:*** *Ensures safe handling in multi-threading.*
- ***Single-threaded:*** *Optimized for single-core performance.*
- ***Thread-safe:*** *Ensuring safety and performance in concurrent settings.*

Haskell Hash consing

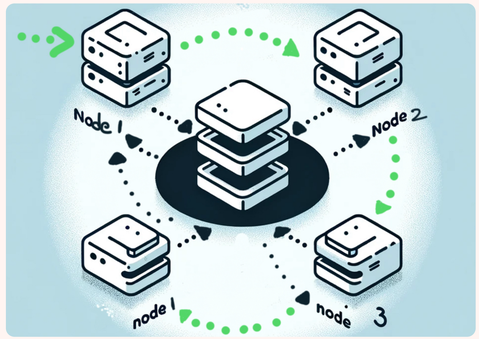
1. ***Pure Version:***
   User-managed table.
   Maintains functional purity, avoiding backend side-effects.
2. ***Single-threaded with Weak Pointers:***
   Optimized memory management using weak pointers.
   Designed for performance in non-concurrent scenarios.
3. ***Thread-safe with Weak Pointers:***
   Ensures safe and efficient handling in multi-threading
   Uses weak pointers for optimized garbage collection.

Rust Hash consing

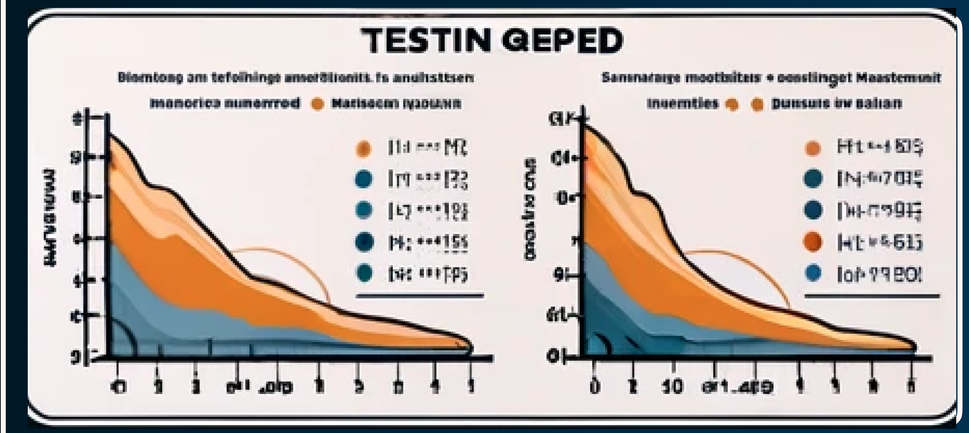
1. ***Single-threaded Version:***
   Optimized for fast performance in non-concurrent applications.
   Avoids overhead of thread safety mechanisms
2. ***Thread-safe Version:***
   Designed to ensure safety in multi-threaded contexts.
   Balances between performance and concurrency.

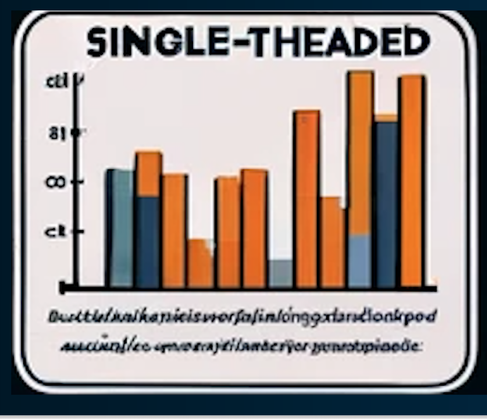
Hash consing

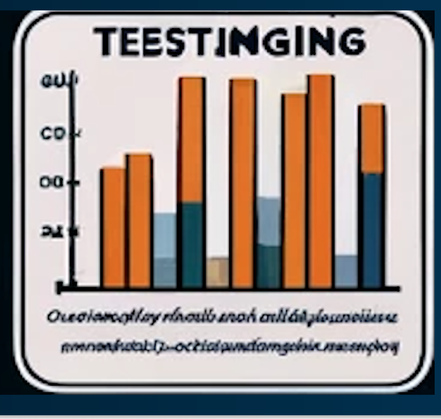
A memory optimization technique that eliminates duplicate sub-expressions by sharing common structures. It stores each unique sub-expression in a hash table, ensuring that identical structures point to the same memory location.



More explanation and a better diagram.







- *Understand the interplay of type safety and modularity in hash consing.*
- *Explore performance variations across languages and implementations.*
- *Highlight the strengths of Haskell and Rust in this domain.*

Why This Study?

Testing Strategies

1. ***Boolean Formulae:***
   Assess the library's effectiveness in handling logical expressions.
   Measure performance when working with complex boolean constructs.
2. ***Abstract Syntax Trees (ASTs):***
   Test the ability to process and optimize structured code representation
   Benchmark efficiency in tree traversal and manipulation.

Conclusions

Memory Saving and other performance metrics.

References

All the references.