Type-Safe Modular Hash-Consing Library in Rust and Haskell

BY KARAN AHLUWALIA
ADVISED BY PROF. MATTHEW FLUET

Type-Safe Modular Hash-Consing

Technique to save memory and speed up certain operations by sharing instances of immutable values.

Goal of hash-consing is to optimize memory usage.

How Hash-consing works:

- *Hashing*
- *Equality Checking*
- *Sharing*

Proposed Work

- *Develop a robust and efficient Type-Safe Modular Hash-Consing library in Haskell and Rust using unique features of the languages.*
- *Demonstrate how Rust and Haskell handle hash-consing in different ways.*
- *Collect extensive performance and memory usage data for benchmarking and comparison.*

Milestone 3 Goals

Libraries Development, Testing and Documentation

- *Implement the core features of TSMHC in both languages*
- *Ensure type safety, modularity, and efficient memory management.*
- *Thoroughly document the libraries*
- *Implement comprehensive testing strategies.*

➢ ***Expected Outcome:*** *Well-documented completed libraries*

Progress

- *Designed and Implemented a Hash-consing Library in Rust.*
- *The library in Rust implements a Single-Threaded version and a Thread-safe version.*
- *Designed the testing strategies for the libraries which will showcase the usage of the library in real life.*
- *Developed a design plan for the library in Rust.*

Rust

- Rust emphasizes memory safety without using a garbage collector, aiming to prevent common programming bugs like null pointer dereferencing and buffer overflows.
- It offers powerful features to safely handle concurrent programming, making it easier to write programs that run efficiently on modern multicore processors.
- As a system programming language, Rust offers performance comparable to C and C++, making it suitable for performance-critical applications.
- Rust provides high-level abstractions without sacrificing efficiency, allowing for expressive code that doesn't compromise on performance.
- Includes a built-in package manager (Cargo), a robust compiler with helpful error messages, and a growing ecosystem of libraries and tools.

Progress

How it works

```
enum BoolExpr {
           Const(bool),
           And(Hc<BoolExpr>, Hc<BoolExpr>),
           Or(Hc<BoolExpr>, Hc<BoolExpr>),
           Not(Hc<BoolExpr>),
           }

let table: HCTable<BoolExpr> = HCTable::new(); --initializing table
let expr1: BoolExpr = BoolExpr::Const(true);
let expr2: BoolExpr = BoolExpr::Const(true);
let hc1: Hc<BoolExpr> = table.hashcons(expr1);
let hc2: Hc<BoolExpr> = table.hashcons(expr2);

println!("{}", hc1 == hc2); // true
```

Challenges

- *Determining the appropriate data structures for the library.*
- *Designing the library architecture to align with Rust's strengths.*

Expected outcome

- *Programs using Hash consing will have a smaller memory print than the program with same without Hash consing.*
- *Single-threaded programs works faster with single-threaded version of the library.*

Future Work

- *Automate the cleanup without dropping the value every time from the table.*

- *Create custom Hash maps to increase performance and efficiency.*

Background

- *Jean-Christophe Filliâtre and Sylvain Conchon. 2006. Type-safe modular hash-consing. In Proceedings of the 2006 workshop on ML (ML '06). Association for Computing Machinery, New York, NY, USA, 12–19. https://doi.org/10.1145/1159876.1159880*
- *ZHOU, N., & HAVE, C. (2012). Efficient tabling of structured data with enhanced hash-consing. Theory and Practice of Logic Programming, 12(4-5), 547-563. doi:10.1017/S1471068412000178*
- *Braibant, T., Jourdan, JH., Monniaux, D. (2013). Implementing Hash-Consed Structures in Coq. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds) Interactive Theorem Proving. ITP 2013. Lecture Notes in Computer Science, vol 7998. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39634-2_36*

Thank you