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 2 Goals

Libraries Development

- *Implement the core features of TSMHC in both languages*
- *Ensure type safety, modularity, and efficient memory management.*

➢ ***Expected Outcome:*** *Working libraries in Rust and Haskell*

Progress

- *Designed and Implemented a Hash-consing Library in Haskell.*
- *The library in Haskell implements a Pure version and an efficient version.*
- *Set up the Github Repo for the library which will test the code before merging any new pull request.*
- *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.*
- *Starting working on the Rust library implementation while ensuring safety and efficiency.*

Progress

Pure

```
import HashConsPure as hcp

type Expr = hcp.HC Expr'
data Expr' = Lit Int | Add Expr Expr deriving (Eq, Hashable)

-- Creating a new table
myTable = hcp.newTable

-- Hash-cons the expressions
(hcExpr1, updatedTable) = hcp.hashCons (Lit 2) myTable
(hcExpr2, updatedTable) = hcp.hashCons (Lit 5) updatedTable
(hcExpr3, updatedTable) = hcp.hashCons (Add hcExpr1 hcExpr2)
updatedTable
(hcExpr4, updatedTable) = hcp.hashCons (Add hcExpr1 hcExpr2)
updatedTable
-- Efficiently check for equality
print (hcExpr3 == hcExpr4)  -- True
```

Progress

Efficient

```
import HashCons as hc

type Expr = hc.HC Expr'
data Expr' = Lit Int | Add Expr Expr deriving (Eq, Hashable,
HashCons)
-- Constructors
lit :: Int -> Expr
lit val = hc.hashcons (Lit val)

add :: Expr -> Expr -> Expr
add exprL exprR = hc.hashcons (Add exprL exprR)

-- Hash-cons the expressions
hcExpr1 = add (lit 2) (lit 5)
hcExpr2 = add (lit 2) (lit 5)

-- Efficiently check for equality
print (hcExpr1 == hcExpr2)  -- True
```

Challenges

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

Next Step

- *Complete the Rust library implementation with accompanying tests.*
- *Consult with my advisor for review and refinement of the Rust implementation.*
- *Complete testing and benchmarking the libraries.*

Vision for final completion

- *Comprehensive documentation for both libraries.*
- *Development and execution of a thorough testing strategy.*
- *Performance benchmarking against existing implementations.*
- *Analysis of performance metrics to ensure efficiency and effectiveness.*

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