Matej Tomášek
m.tomasek@student.tudelft.nl

Supervisor: Jaro Reinders
Responsible Professor: Casper Bach Poulsen

Algebraic Effects and Handlers for Software Transactional Memory

1. >Introduction

Algebraic effects and handlers [1] is a technique for modular effectful programming where **side effects** are modelled by separating their **set of algebraic operations - syntax**, and their **implementation - handlers**.

However, many mainstream functional programming languages, like Haskell, lack built-in frameworks to specify and implement effects, such as Software Transactional Memory (STM) [2], in this manner.

2. >Research Questions

How can we implement and reason about an algebraic effect model of STM in Haskell?

- How can an implementation of algebraic effects that implements the intended behavior of STM look like in correspondence with the literature?
- What are the mathematical laws describing the intended behavior of STM and prove the proposed implementation is correct with respect to them?
- How do the operations of our STM interface interact with the operations of other effects, and what the "extension" of transactional memory to other effects looks like?

3. >Methodology

Algebraic Effects and Handlers

- Modelled as **free monads [3]**
- Effect syntax described by **signature functors**
- Composition of multiple effects using **co-products** (+) and **subtyping** (<)
- Handlers are implemented by recursively folding over the free monad structure

Software Transactional Memory

- STM is a concurrency abstraction for lock-free communication between threads using database-like atomic transactions.
- The transactions modify transactional variables shared between threads and commit to the global memory - optimistic execution
- Allows retrying, choice with orElse and concurrent execution with atomically

```haskell
limitedWithdraw :: TVar Int -> Int -> STM ()
limitedWithdraw account amount = do
    balance <- readTVar account
    if amount > 0 && amount > balance then
        retry
    else
        writeTVar acc (balance - amount)
```

Figure 1: Limited withdrawal example from [2]

4. >Implementation

Sequential model (Free monad)

- The handled result is either accumulated heap modifications or failed computation on retry
- Offers transactional semantics from effect interaction

```haskell
newtype TVar a = ...
type Heap = ...

alloc :: (Show a, Eq a) => a -> Heap -> (TVar a, Heap)
update :: (Show a, Eq a) => TVar a -> a -> Heap -> Heap
lookup :: (Show a, Eq a) => TVar a -> Heap -> a

data STM k where
    New     :: (Show a, Eq a) => a -> (TVar a -> k) -> STM k
    Read    :: (Show a, Eq a) => TVar a -> (a -> k) -> STM k
    Write   :: (Show a, Eq a) => TVar a -> a -> k -> STM k
    Retry   :: STM k
    OrElse  :: k -> k -> STM k

hSTM :: Functor f => StatefulHandler STM a Heap f (Maybe (a, Heap))
```

Figure 2: STM interface

Concurrent model (Hefty trees [5])

- Using modified implementation of higher-order (scoped) multi-threading effect from [6], providing operations atomic, fork and wait
- Devised two evaluation options: round-robin (real) scheduling with execute, and modelling all possible interleavings with non-determinism in executeAll
- atomically embeds the handler hSTM to evaluate the transaction and its changes are either committed to the shared memory in State Heap, or it recursively call itself with the same transaction for re-execution at a later time

```haskell
atomic :: Thread <: h => Hefty h a -> Hefty h a
fork :: Thread <: h => Hefty h a -> Hefty h ()
wait :: (Alg Out <: h, Thread <: h) => Int -> Hefty h ()

execute :: (Alg Err <: h, Alg Out <: h)
    => Hefty (Thread :+: h) a -> Hefty h a
executeAll :: (Alg Err <: h, Alg NonDet <: h, Alg Out <: h)
    => Hefty (Thread :+: h) a -> Hefty h a

atomically :: (Thread <: h, Alg (State Heap) <: h)
    => Free (STM + End) a -> Hefty h a
```

Figure 3: Operations of the concurrency model

References

[1] Gordon Plotkin and John Power. Algebraic Operations and Generic Effects. Applied Categorical Structures, 11(1):69–94, 2003
[2] Simon Peyton Jones. Beautiful Concurrency. O'Reilly, Beautiful Code Edition, 01 200
[3] Casper Bach Poulsen. Algebras of Higher-Order Effects in Haskell, 08 2023. URL: http://casperbp.net/posts/2023-08-algebras-of-higher-order-effects/.
[4] Johannes Borgström, Karthikeyan Bhargavan, and Andrew D. Gordon. A compositional theory for STM Haskell. In Proceedings of the 2nd ACM SIGPLAN Symposium on Haskell, Haskell '09, pages 69–80, New York, NY, USA, 2009. Association for Computing Machinery
[5] Casper Bach Poulsen and Cas van der Rest. Hefty algebras: Modular elaboration of higher-order algebraic effects. Proc. ACM Program. Lang., 7(POPL), jan 2023
[6] Nicolas Wu, Tom Schrijvers, and Ralf Hinze. Effect Handlers in Scope. In Proceedings of the 2014 Haskell Symposium, Haskell '14, New York, NY, USA, 2014. ACM

5. >Proof of Correctness

- To ensure correctness, the proposed implementation was verified on a set of equivalences for STM abstractions [4]

$$(\texttt{readTVar}\ a >\!\!>\!\!=_{\mathrm{STM}} \lambda x.\ \texttt{writeTVar}\ a\ x) \leftrightarrow \texttt{return}_{\mathrm{STM}}() \quad (1)$$

$$(\texttt{writeTVar}\ a\ M >\!\!>_{\mathrm{STM}} \texttt{writeTVar}\ b\ N) \leftrightarrow (\texttt{writeTVar}\ b\ N >\!\!>_{\mathrm{STM}} \texttt{writeTVar}\ a\ M)\ \text{if}\ a \neq b \quad (2)$$

$$(\texttt{readTVar}\ a >\!\!>\!\!=_{\mathrm{STM}} \lambda x.\ \texttt{writeTVar}\ b\ M >\!\!>\!\!=_{\mathrm{STM}} \texttt{return}_{\mathrm{STM}}\ x) \leftrightarrow (\texttt{writeTVar}\ b\ N >\!\!>_{\mathrm{STM}} \texttt{readTVar}\ a)\ \text{if}\ a \neq b \quad (3)$$

$$\texttt{orElse}\ \texttt{retry}\ M \leftrightarrow M \quad (4)$$

$$\texttt{orElse}\ M\ \texttt{retry} \leftrightarrow M \quad (5)$$

$$\texttt{orElse}\ M_1\ (\texttt{orElse}\ M_2\ M_3) \leftrightarrow \texttt{orElse}\ (\texttt{orElse}\ M_1\ M_2)\ M_3 \quad (6)$$

Figure 4: The mathematical laws used to verify our implementation

6. >Applications

- The implementation was used to recreate the Dining Philosophers problem solution in Haskell, which uses native STM transactions - with the round-robin scheduler evaluation, we were able to achieve the same functionality.

```
> Running the philosophers. Press Ctrl-C to quit.
> Aristotle is hungry.
> Kant is hungry.
> Spinoza is hungry.
> Aristotle got forks 1 and 2 and is now eating.
> Marx is hungry.
> Russel is hungry.
> Spinoza got forks 3 and 4 and is now eating.
...
```

Figure 5: Output of the recreated solution in our framework

- The extension of atomically was provided for other effects, under the assumption that they interact with their version of transactional memory.
- Delegation of retry and orElse operations to the Transactional effect

```haskell
atomically' :: (Eq w, Functor f, Thread <: h, Alg (State w) <: h)
    => (forall f'. Functor f' => StatefulHandler f a w f' (a, w))
    -> Free (f + Transactional + End) a -> Hefty h a
```

Figure 6: atomically' signature

7. >Conclusions and Limitations

- Despite implementing correct semantics, coupling of syntax and handlers reduces modularity [5][6]
- Our concurrency model heavily relies on global state semantics, as local scopes in forked threads were inaccessible from the main thread
- For future work, improving the concurrency model's robustness to handle non-terminating transactions, e.g. consisting of only a single retry operation, and extending the formal reasoning to atomically

TU Delft
Delft University of Technology