Derivable Partial Locking for Algebraic Data Types

Boldizsár Németh and Zoltán Kelemen

Concurrency is one of the most actively researched fields of Computer Science. Writing concurrent programs is challenging. The causes are the need for synchronization and solving possible race conditions and deadlocks while avoiding to unnecessary waiting and overhead. This can be very difficult when a transaction needs to lock multiple data elements, even with using previously defined concurrent data structures [1].

Algebraic Data Types are composite data structures that naturally support pattern matching. When the type is parametric then the ADTs support writing generic algorithms without additional complexity.

The integrity of the program data can be archived by providing locks for a data structure or using concurrent data structures. Central locking is a safe strategy to avoid race conditions, but it has a high cost because globally used objects are frequently accessed by different threads. Hierarchic locking allows transactions to lock exactly the data elements that they need [2]. This makes small transactions take small locks and work in parallel without waiting for each other.

This article focuses on a method that helps the implementation of thread-safe programs with ADTs. By transforming the data model of the application to thread-safe data structures with a built-in locking mechanism, a programmer can focus on the business logic of his application when writing the program [3].

First, we need to transform the original ADT to the thread-safe version. For this, the programmer can configure what parts of the data structure should be locked. Too many locks cause performance loss. Second, we define a frontend to access the locked parts of the data structure. This must be done with minimal syntactical noise, to ease the development of the application.

We implement our solution to this problem in Haskell. We use the Haskell concurrency primitive mvar [4] to create a concurrent version of the data model. Two tools are inspected that are capable of transforming the data structure to a thread-safe version. The generics of GHC provide a way to derive type class instances for ADTs, like how it can be done with built-in type classes. This can be done by decomposing the structure of the ADT and defining the meaning of the functions on these primitive blocks. The second tool is the Template Haskell compiler extension. With TH the internal representation of the program can be inspected and modified. This grants more freedom to transparently change the program, but there is the danger of confusing users and committing errors while transforming the program.

Acknowledgements

Supported by EITKIC 12-1-2012-0001.

References

[1] Ohad Shacham, Nathan Bronson, Alex Aiken, Mooly Sagiv, Martin Vechev, and Eran Yahav. 2011. Testing Atomicity of Composed Concurrent Operations. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications (OOPSLA '11). (51-64).

[2] Goetz Graefe. Hierarchical locking in B-tree indexes.

[3] Peter Hawkins, Alex Aiken, Kathleen Fisher, Martin Rinard, and Mooly Sagiv. 2012. Concurrent Data Representation Synthesis. Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation. Pages 417-428

[4] Simon Peyton Jones, Andrew Gordon, Sigbjorn Finne. 1996. Concurrent Haskell. 23rd ACM Symposium on Principles of Programming Languages.