A review of Peyton Jones, S., Eber, J.M. and Seward, J., 2000, September. Composing contracts: an adventure in financial engineering (functional pearl). In ACM SIGPLAN Notices (Vol. 35, No. 9, pp. 280-292). ACM.

There are different types of financial contracts exist in financial markets. In this paper, they have proposed a combinatory library to represent and valuate those financial contracts.

Their combinatory library is implemented in Haskell. They used Haskell because of many reasons. One of the main reason is that Haskell and This contract language are booth functional languages and they follow the same syntax patterns. And because Haskell is a lazy language, it was become easier to build recursive functions and lattice data structures.

In the proposed library, they have introduced set of combinators so that they can be used to combine contracts together and create much larger contracts. It consists with 10 primitive combinators and 5 primitive observables. An observable is a time varying quantity. And in any particular time the observable need to have a value that both parties to the contract will agree. Those 10 primitive combinators are zero, one, give, and, or, truncate, then, scale, get, anytime. 5 observables that they were introduced are konst, lift, lift2, instance and time. To prove that this language can be applied to real-world contracts, they used 3 types of contracts. Namely, they are Zero coupon bonds, European options, and American options. But they have generalized the language for other contracts as well.

When building those combinators, they have introduced two technical words called acquisition date and horizon. The date at which contract is acquired is called acquisition date and horizon is the last date at which it can be acquired. They have used them to precisely describe combinators. Truncate combinator was used to change the acquisition date and the horizon.

They have created a model to valuate contracts written in this language. This model used lattice valuation method. For each combinator, evaluation semantics were introduced. Those evaluation semantics convert a contract into a value process. A value process is a partial function from time to a random variable. To valuate a contract, this model needs the contract definition, interest rate model and the set of evaluation semantics. In the paper, implementations of this language and valuation engine have done using language Haskell. But they have suggested that it can be implemented using other languages as well. So the proposed model is independent of the language Haskell. And one drawback of most of the domain specific imbedded languages is the memorization problem. Proposed language also suffers from this problem. The have given this as a feature work that needs to be solved.

This paper is well written and structured well enough so that anyone can understand the content. In the first part of the paper they have introduced the language and it's syntaxes, semantics. Then a financial valuation model was applied to valuate contracts. In the introduction, they introduced few basic combinators and when try to represent different types of contracts they have introduced other combinators as well.

They have mapped that evaluation semantics to the actual financial world and its behaviors. But one of the drawbacks in this paper is that it lacks the formal proof for the combinators they have introduced. And in this paper, they claimed that any contract can be represented using this language. But only proven for three types of contracts only. And another drawback is that proposed language cant represent contracts which have time gaps after the acquisition date. Because of this, some American options can't be represented using this language. Snell function is not clearly explained in this paper.