Free Trade: Composable Smart Contracts

Student: Ross Gardiner, Supervisor: Dr. Nicolas Wu, Project Type: Research

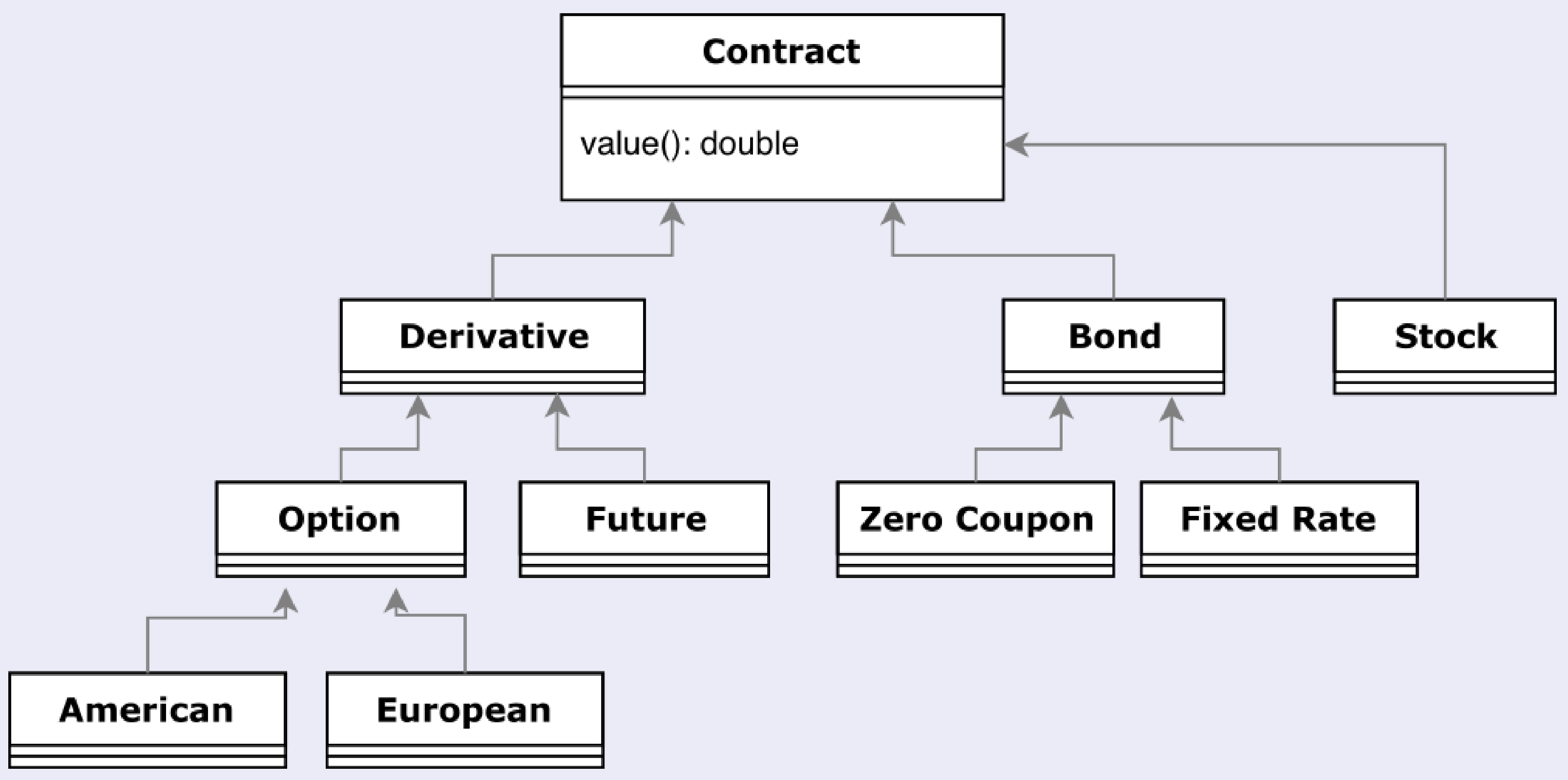
University of Bristol, Department of Computer Science

Composable Contracts

Programmatic representation of financial contracts is a critical tool for banks and other financial institutions. For example, it enables:

- realtime portfolio valuation
- realtime portfolio risk analysis
- trade execution without human intervention

However, traditional systems are inflexible. You can imagine this inheritance diagram:



What if you want something with properties of both an option and a bond? Either you duplicate code or allow multiple inheritance. Worse yet—you have to wait for the IT department to implement it!
'Composable contracts' were first proposed by Peyton Jones, Eber and Seward in 2000. They suggested a library of simple primitives that can be combined together to express many different types of financial contract. For example:

```
-- immediately pay the holder one unit of currency
one :: Currency -> Contract
-- invert rights and obligations of a contract
give :: Contract -> Contract
-- scale rights and obligations by an observable
scale :: Obs -> Contract -> Contract
```

These allow declarative construction and valuation of very complex contracts.

Smart Contracts

The concept of 'smart contracts' appeared in the 1990s. The crucial difference between 'smart' and traditional contracts is that a smart contract can track and/or enforce its parties' performance of their obligations.
Current implementations of smart contracts employ blockchain technology to achieve global, irrevocable consensus about the state of a contract. Ethereum is the leading implementation. It offers a Turing-complete VM to which users can deploy their contracts.
Here's a simple example, written in the Solidity language:

```
contract Greeter {
  address creator;
  string greeting;

  function Greeter(string _greeting) public {
    creator = msg.sender;
    greeting = _greeting;
  }

  function greet() constant returns (string) public
   {
    return greeting;
  }

  function setGreeting(string _newgreeting) public {
    require(msg.sender == creator);
    greeting = _newgreeting;
  }
}
```

When deployed, this contract allows anyone to call its `greet()` function, which returns the stored data. Anyone can also call `setGreeting`, but the greeting will not be updated unless the caller is the person who deployed the contract.
More complex contracts can represent voting systems, transparent auctions, tradeable tokens and a wide variety of other useful constructs.

1. Project Aims

The aim of this project is to **reimplement** the original composable contracts idea in a **modern Haskell style**, including:

- implementing the contract combinators in a Free monadic style,
- demonstrating the flexibility this adds by implementing several different interpreters,
- and making the combinator language modular by adopting a 'Data Types à la Carte' approach.

The other primary aim of this project is to **write a compiler** that allows these contracts to be **deployed on the Ethereum blockchain**.

2. Research Proposal

Smart contracts are currently of great interest to both researchers and large businesses. R3, which develops the Corda platform, has corporate partners including Barclays, JPMorgan, Goldman Sachs, Intel and Microsoft.
Future research is likely to focus on some of the following areas:

- Enforcement of debt obligations
- Multi-party contracts
- Additional language primitives
- Transaction cost-efficiency of blockchain implementations

3. Progress and Status

Complete

- 'Free' implementation of combinator language
- Implementation of basic interpreters for combinator language

In progress

- Reimplementation of interpreter for contract valuation
- Modularisation of combinator language

Remaining

- Reimplementation of interpreter for contract valuation
- Modularisation of combinator language
- Compiler to Ethereum (Solidity)