Towards Certified Management of Financial Contracts*

Patrick Bahr, Jost Berthold and Martin Elsman

University of Copenhagen
Dept. of Computer Science (DIKU)
{paba,berthold,mael}@di.ku.dk

1 Introduction

Banks and financial institutions nowadays often use domain-specific languages (DSLs) for describing complex financial contracts, in particular, for specifying how asset transfers for a specific contract depend on underlying observables, such as interest rates, currency rates, and stock prices.

The seminal work by Peyton-Jones and Eber on financial contracts [4] shows how an algebraic approach to contract specification can be used for valuation of contracts (when combined with a model of the underlying observables) and for managing how contracts evolve under so-called fixings[1] and decision-taking, with the contracts eventually evaporating into the empty contract, for which no party have further obligations. The ideas have emerged into Eber's company *LexiFi*, which has become a leading software provider for a range of financial institutions, with all contract management operations centralised around a domain-specific contract language hosted in MLFi [3], a derivative of the functional programming language OCaml.

In this paper, we present a small simple contract language, which rigorously relegates any artefacts of modelling and computation from its core. The language shares the same vision as the previously mentioned work with the addition that it (a) allows for specifying multi-party contracts (such as entire portfolios), (b) has good algebraic properties, well suited for formal reasoning, and yet (c) allows for expressing many interesting contracts appearing in real-world portfolios, such as various variations of so-called barrier options.

We show that plenty of information can be derived from, and useful manipulations defined on, just the symbolic contract specification, independent of any stochastic aspects of the modelled contracts. Contracts modelled in our language are analysed and transformed for management according to a precise cash-flow semantics, modelled and checked using the Coq proof assistant.

Implementations of the contract language in Haskell and Coq are available online[2] together with machine-checkable proofs (in Coq) of the key properties of the contract language.

2 Contract Language

2.1 Language Constructs

Financial contracts essentially define future transactions (*cash-flows*) between different parties who agree on a contract. Amounts in contracts may be *scaled* using real-valued expression ($\mathsf{Expr}_{\mathbb{R}}$), which may refer to *observable* underlying values, such as foreign exchange rates, stock prices, or market indexes. Likewise, contracts can contain *alternatives* depending on Boolean predicates ($\mathsf{Expr}_{\mathbb{B}}$), which may refer to these observables, as well as external *decisions* taken by the parties involved.

Observables and choices in our contract language are "observed" with a given offset from the current day. In general, all definitions use

*This work has been partially supported by the Danish Council for Strategic Research under contract number 10-092299 (Hiperfit [2]), and the Danish Council for Independent Research under Project 12-132365.

[1] Underlying observables gradually become fixed when time passes by.

[2] See https://github.com/HIPERFIT/contracts.

relative time, aiding the compositionality of contracts.

A common contract structure is to *repeat* a choice between alternatives until a given end date. As a simple example, consider an *FX option* on US dollars: Party X may, within 90 days, decide whether to buy 100 US dollars for a fixed rate r of Danish Kroner from Party Y.

$$\begin{aligned} \mathit{option} &= \mathit{checkWithin}(\mathit{chosenBy}(X, 0), 90, \mathit{trade}, \mathit{zero}) \\ \mathit{trade} &= \mathit{scale}(100, \mathit{both}(\mathit{transfer}(Y, X, \mathsf{USD}), \mathit{pay})) \\ \mathit{pay} &= \mathit{scale}(r, \mathit{transfer}(X, Y, \mathsf{DKK})) \end{aligned}$$

The *checkWithin* construct which generalises an alternative by iterating the decision (*chosenBy*) of party X. If X chooses at one day before the end (90 days), the *trade* comes into effect, consisting of two transfers (*both*) between the parties. Otherwise, the contract becomes empty (*zero*) after 90 days.

The contracts atoms and combinators of the language are:

$$\begin{aligned} \mathit{zero} &: \mathsf{Contr} \\ \mathit{transfer} &: \mathsf{Party} \times \mathsf{Party} \times \mathsf{Currency} \to \mathsf{Contr} \\ \mathit{scale} &: \mathsf{Expr}_{\mathbb{R}} \times \mathsf{Contr} \to \mathsf{Contr} \\ \mathit{translate} &: \mathbb{N} \times \mathsf{Contr} \to \mathsf{Contr} \\ \mathit{checkWithin} &: \mathsf{Expr}_{\mathbb{B}} \times \mathbb{N} \times \mathsf{Contr} \times \mathsf{Contr} \to \mathsf{Contr} \\ \mathit{both} &: \mathsf{Contr} \times \mathsf{Contr} \to \mathsf{Contr} \end{aligned}$$

translate(n) simply translates a contract n days into the future.

In the expression language, we also define a special expression *acc* which *accumulates* a value over a given number of days from today.

$$\mathit{acc} : (\mathsf{Expr}_\alpha \to \mathsf{Expr}_\alpha) \to \mathbb{N} \to \mathsf{Expr}_\alpha \to \mathsf{Expr}_\alpha$$

The accumulator can be used to compute averages (for so-called Asian options), or more generally to carry forward a state while computing values.

2.2 Denotational Semantics

The semantics of a contract is given by its cash-flow, which is a partial mapping from time to transfers between two parties:

$$\begin{aligned} \mathsf{Trans} &= \mathsf{Party} \times \mathsf{Party} \times \mathsf{Currency} \to \mathbb{R} \\ \mathsf{Flow} &= \mathbb{N} \rightharpoonup \mathsf{Trans} \end{aligned}$$

The cash-flow is a partial mapping since it may not be determined due to insufficient knowledge about observables and external decisions, provided by an environment $\rho \in \mathsf{Env}$:

$$\begin{aligned} \mathsf{Env} &= \mathbb{Z} \to X \rightharpoonup \mathbb{R} \cup \mathbb{B} \\ \mathcal{C}\,[\![\cdot]\!]. &: \mathsf{Contr} \times \mathsf{Env} \to \mathsf{Flow} \end{aligned}$$

Note that the environment is a partial mapping from $\mathbb{Z}$, i.e. it may provide information about the past.

This denotational semantics is the foundation for the formalisation of symbolic contract analyses, contract management and transformations.

An important property of the semantics of contracts is monotonicity, i.e.

$$\mathcal{C}\,[\![c]\!]_{\rho_1} \subseteq \mathcal{C}\,[\![c]\!]_{\rho_2} \text{ if } \rho_1 \subseteq \rho_2$$

where $\subseteq$ denotes the subset inclusion of the graph of two partial functions.

2.3 Contract Analysis

When dealing with contracts we are interested in a number of semantic properties of contracts, e.g. causality (Does the cash-flow at each time t depend only on observables at time $\leq t$?), horizon (From which time onwards is the cash-flow always zero?) and dependencies (Which observables does the cash-flow depend on?). Such properties can be characterised precisely using the denotational semantics. For example a contract c is causal iff for all $t \in \mathbb{N}$ and $\rho_1, \rho_2 \in \mathsf{Env}$ such that $s \leq t$ implies $\rho_1(s) = \rho_2(s)$ for all $s \in \mathbb{Z}$, we have that $\mathcal{C}\,[\![c]\!]_{\rho_1}(t) = \mathcal{C}\,[\![c]\!]_{\rho_2}(t)$. That is, the cash-flows at any time t do not depend on observables and decisions after t.

It is in general undecidable whether a contract is causal, but we can provide conservative approximations. For instance we have an inductively defined predicate Causal such that if Causal(c), then c is indeed causal. This is not unlike type checking, which provides a conservative approximation of type safety.

3 Contract Management and Transformation

Apart from a variety of analyses our framework provides functionality to transform contracts in meaningful ways. The most basic form of such transformations are provided by algebraic laws. These laws of the form $c_1 \equiv c_2$ state when it is safe to replace a contract c_1 by an equivalent contract c_2. Using our denotational semantics, these algebraic laws can be proved in a straightforward manner: we have $c_1 \equiv c_2$ iff $\mathcal{C}[\![c_1]\!]_\rho = \mathcal{C}[\![c_2]\!]_\rho$ for all $\rho \in \mathsf{Env}$.

More interesting are transformations that are based on knowledge about observables and external decisions. That is, we transform a contract c based on an environment $\rho \in \mathsf{Env}$ that encodes the knowledge that we already have. We consider two examples, specialisation and reduction.

3.1 Specialisation

A specialisation function f performs a partial evaluation of a contract c under a given environment ρ. The resulting contract $f(c, \rho)$ is equivalent to c under the environment ρ. More generally, we have that $\mathcal{C}[\![f(c,\rho)]\!]_{\rho'} = \mathcal{C}[\![c]\!]_\rho$ for any environment $\rho' \subseteq \rho$, including the empty environment.

3.2 Reduction Semantics

Apart from the denotational semantics our contract language is also equipped with a reduction semantics [1], which advances a contract by one time unit. We write $c \overset{\tau}{\Rightarrow}_\rho c'$, to denote that c is advanced to c' in the environment ρ, where $\tau \in \mathsf{Trans}$ indicates the transfers that are necessary (and sufficient) in order to advance c to c'.

The reduction semantics can be implemented as a recursive function of type

$$f_\Rightarrow : \mathsf{Contr} \times \mathsf{Env} \rightharpoonup \mathsf{Contr} \times \mathsf{Trans}$$

$f_\Rightarrow$ takes a contract c and an environment ρ, and returns the residual contract c' and the transfers τ such that $c \overset{\tau}{\Rightarrow}_\rho c'$. The argument ρ typically contains the knowledge that we have about the observables up to the present time, i.e. for time points ≤ 0.

We can show that the reduction semantics is sound and complete w.r.t. the denotational semantics:

Theorem 1. *If $c \overset{\tau}{\Rightarrow}_\rho c'$, then $\mathcal{C}[\![c]\!]_\rho(0) = \tau$ and $\mathcal{C}[\![c]\!]_\rho(i+1) = \mathcal{C}[\![c']\!]_{\rho\uparrow}(i)$ for all $i \in \mathbb{N}$, where $\rho\uparrow(i) = \rho(i+1)$. If $\mathcal{C}[\![c]\!]_\rho(0) = \tau$ then there is some c' with $c \overset{\tau}{\Rightarrow}_\rho c'$.*

4 Future Work

For future work we plan to implement and certify more extensive analyses and transformations, e.g. scenario generation and "zooming" (changing the granularity of time). Moreover, an important goal is to generate from a contract efficient code to calculate its payoff.

At the moment the Haskell implementation is translated by hand into Coq definitions, which are the basis for the certification. This approach is beneficial for rapid prototyping, but our goal is to turn this process around and automatically extract Haskell code from the Coq definitions.

References

[1] J. Andersen, E. Elsborg, F. Henglein, J. G. Simonsen, and C. Stefansen. Compositional specification of commercial contracts. *International Journal on Software Tools for Technology Transfer*, 8(6):485–516, 2006.

[2] J. Berthold, A. Filinski, F. Henglein, K. Larsen, M. Steffensen, and B. Vinter. Functional High Performance Financial IT – The HIPERFIT Research Center in Copenhagen. In *TFP'11 – Revised Selected Papers*, 2012.

[3] LexiFi. Contract description language (MLFi). Web page and white paper. `http://www.lexifi.com/technology/contract-description-language`.

[4] S. Peyton Jones, J.-M. Eber, and J. Seward. Composing contracts: an adventure in financial engineering (functional pearl). In *ICFP*, 2000.