Verifying the generation of payoff-language expressions

Danil Annenkov, PhD student, DIKU

Supervisor: Martin Elsman, Associate Professor, DIKU

The HIPERFIT Prototype Project

- Integrates two HIPERFIT projects:
  - Contract DSL: certified domain-specific language;
  - FINPAR: parallel high-performance contract valuation implementation
- Features
  - OpenCL payoff function code generation from contract DSL;
  - high performance contract valuation using OpenCL pricer implementation form FINPAR;
  - Web-interface with automatic web form generation based on Haskell data types

Contract DSL[*]

- allows to express a large variety of financial contracts;
- supports multi-party contracts (we will focus only on more classic two-parties contracts for valuation purposes);
- has a formal semantics;
- contract management and transformations are proven correct wrt. specified semantics;
- contract DSL semantics along with all proofs are formalized in Coq proof assistant.

*) Patrick Bahr, Jost Berthold, Martin Elsman. Certified Symbolic Management of Financial Multi-Party Contracts, ICFP’2015.

Contract DSL Semantics

Contract semantics is represented as a partial function:

```
C⟦c⟧ : Env -> ExtEnv -> option Trace
```

where `Trace` is a mapping from time (days) to transfer of some amount between parties:

```
Trace = ℕ -> Trans
```

```
Trans = Party -> Party -> Asset -> ℝ
```

Patrick Bahr, Jost Berthold, Martin Elsman. Certified Symbolic Management of Financial Multi-Party Contracts, ICFP’2015.

Payoff Language

We define intermediate language (IL) inspired by traditional approaches to payoff languages:

```
Inductive ILExpr : Set :=
| ILIf      : ILExpr -> ILExpr -> ILExpr -> ILExpr
| FloatV    : ℝ -> ILExpr
| Model     : ObsLabel -> ℤ -> ILExpr
| ILUnExpr  : ILUnOp -> ILExpr -> ILExpr
| ILBinExpr : ILBinOp -> ILExpr -> ILExpr -> ILExpr
| Payoff    : ℕ -> Party -> Party -> ILExpr.
```

Payoff Language

- It’s easier to translate payoff IL to various target languages
- Certified translation from a contract DSL to payoff IL
  - define payoff IL semantics;
  - implement translation of the contract DSL to payoff IL;
  - prove correctness wrt the contract cashflow semantics.

Payoff Language Semantics

```
IL⟦e⟧: ILExtEnv -> Disc -> Party -> Party -> option ILVal
```

where `Disc` represents a discounting function from day offset to discount rate

```
    Disc = ℕ -> ℝ
```

`ILVal is` defined as

```
    Inductive ILVal : Set :=
    | ILBVal : 𝔹 -> ILVal
    | ILRVal : ℝ -> ILVal.
```

Translating Contracts to Payoffs

Two “sublanguages” in contract DSL:

- expressions
- contracts

They are both translated to the single intermediate expression language:

$\tau[\![e]\!] : \mathbb{N}$ `-> ILExpr`

$\tau[\![c]\!] : \mathbb{N}$ `-> ILExpr`

Translation functions take care of aggregation of contract cashflows, adding relative time shifts etc.

Translation correctness*

If $\texttt{C}[\![\texttt{c}]\!]\ \texttt{env extC} = \texttt{trace}$ and

$\boldsymbol{\tau}[\![\mathbf{c}]\!] = \mathbf{e}_{\mathbf{IL}}$ and

$\texttt{IL}[\![\mathrm{e}_{\mathrm{IL}}]\!]\ \texttt{extIL} = \texttt{v}$ and

assuming that environments extIL and extC agree at all points

then

$$\sum_{t=0}^{h} \texttt{disc(t)*trace(t)} = \texttt{v}$$

where h is contract horizon, disc - discount function, $\tau[\![\mathrm{c}]\!]$ - translation from contract DSL to IL.

** Some details are deliberately omitted.*

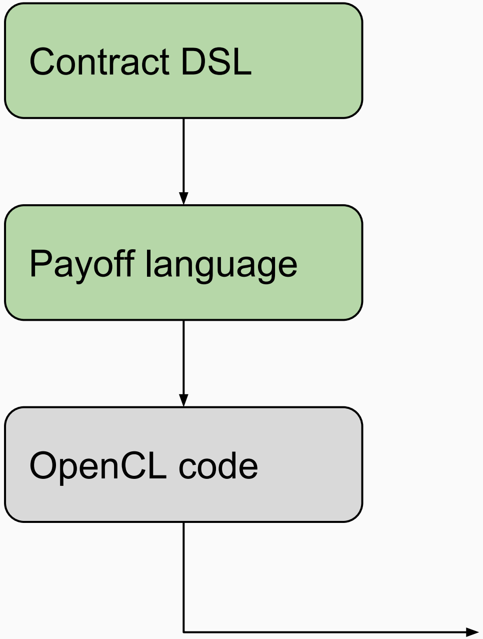
Code extraction

- Translation functions $\boldsymbol{\tau}[\![\mathbf{c}]\!]$ and $\boldsymbol{\tau}[\![\mathbf{e}]\!]$ are “extracted” from Coq as Haskell code.
- Extracted translation code works nicely with certified code for contract analysis and transformation.
- Payoff IL is mapped relatively straightforwardly to a subset of language constructs in other languages, such as OpenCL, Haskell, and Futhark.

Valuation engine

- various valuation engines can be used to calculate contract’s price using extracted payoff function;
- an example: hand-tuned high-performance OpenCL implementation from FINPAR project;
- map payoff language expressions to the subset of OpenCL -> generate OpenCL code -> “fuse” generated code into a valuation engine.

“Fusing” code into the valuation engine



Contract DSL

Payoff language

OpenCL code

```
inline
void trajectory_inner(
      const       UINT  num_cash_flows, // number of discounts
      const       UINT  model_ind,  // the model index
      const       UINT  disct_index,// index of the discount
      const       REAL  amount,     // update with amount
      __constant REAL*  md_discts,
      __local     REAL* local_vhat
) {
      // some code ...
}
inline
void payoffFunction(
      const       UINT  model_num,  // the index of the current model
      const       UINT  num_under,  // the number of underlyings
      const       UINT  num_cash_flows, // the number of discounts
      const       UINT  num_pricers,// the number of deterministic pricers
      __constant REAL*  md_discts,  // [num_models][num_cash_flows] discounts
      __constant REAL*  md_detvals, // [num_models, num_det_pricers]  pricers
      const       REAL* inst_traj,  // [num_dates, num_under] current trajectory
      __local     REAL* vhat        // [model_num] Accumulated per-model price
) {
{|CODE|} // code placeholder
}
```

Future work

- implement and prove correctness of translation from payoff language to target languages (OpenCL, Futhark etc.);
- integrate certified translation code with Prototype;
- add support for more features of the contract DSL
    - add loop-like constructs to the payoff language (for now, `IfWithin` is compiled into nested ifs);
    - add support for accumulators.

Thank you!

Questions?