DSLs in finance, an overview

Vittorio Zaccaria

January 30, 2018

Outline

Introduction

Contract languages

Certified symbolic management cite:Bahr:2015hm

- Contracts are expressed in relative times
- Can describe stipulation between multiple parties (& operator)
- Can describe *observable external decisions*:

$$\text{if } \underbrace{\text{Obs(X exercises option}, 0)}_{\text{eval'd at the beginning of each day}} \text{ in } \overbrace{90}^{\text{days' range}} \text{ then } 100 * \underbrace{\text{DKK}(Y \rightarrow X)}_{\text{atomic exchange contract}} \text{ else } 0$$

Findel cite:Biryukov:2017ip

- Absolute times;
- Syntax similar to others but no intuitive reference to parties involved as contracts change over time:

$$\underbrace{Give(10 * One(USD))}_{\text{Bob pays 10 now}} \wedge \overbrace{At(now + 1\ years, 11 * One(USD))}^{\text{Bob expects to receive 11 in a year}}$$

Semantics

Certified symbolic management cite:Bahr:2015hm

Subdivides semantics into contract and expression (denotational) semantics.

Contract semantics maps expressions into a cash-flow trace:

$$C : [\![\Gamma]\!] \rightarrow \overbrace{\mathbb{N} \rightarrow \underbrace{\mathit{Party} \times \mathit{Party} \times \mathit{Asset} \rightarrow \mathbb{R}}_{\text{transaction}}}^{\text{trace}}$$

for example (note the delay $\uparrow$ and *unit transfer* $\rightarrow$ operators)[1]:

$$\begin{aligned}
\mathcal{C}[\![0]\!] &= \lambda n.\lambda t.0 \\
\mathcal{C}[\![c1 \& c2]\!] &= \mathcal{C}[\![c1]\!] + \mathcal{C}[\![c2]\!] \\
\mathcal{C}[\![d \uparrow c]\!] &= \lambda n.\mathcal{C}[\![c]\!](n - d) \\
\mathcal{C}[\![a(p_1 \rightarrow p_2)]\!] &= \lambda n.\lambda t.\delta_{0,(p_1,p_2,a)}(n,t) - \delta_{0,(p_2,p_1,a)}(n,t)
\end{aligned} \tag{1}$$

[1] δ = Kronecker's delta

Certified symbolic management

Contract transforms consist in specialisation and advancement, i.e., instantiation of a contract to a concrete starting time or simplification. Consider the following contract:

$$\mathrm{DKK}(Y \to Z) \;\&\; \mathrm{if}\ \mathrm{Obs}(\mathrm{X\ defaults}, 0)\ \mathrm{in}\ 30\ \mathrm{then}\ \mathrm{DKK}(Z \to Y)\ \mathrm{else}\ 0$$

and assume that

$$\mathit{default}(X, i) = \top \ \mathit{if}\ i = 15,\ \perp \mathit{otherwise}$$

Then, at time $i = 16$, the contract can be transformed into:

$$\mathrm{DKK}(Y \to Z) \;\&\; \mathrm{DKK}(Z \to Y) \sim 0$$

Software verification and certified software

Type systems

Certified symbolic management

- Problem: Simple expressions could involve non-causality, e.g.:

$$\underbrace{\mathrm{obs}(FX(USD, DKK), 1)}_{\text{tomorrow's observation}} \times \underbrace{DKK(X \to Y)}_{\text{pay today}}$$

- Solution: time-indexed types;

Certified symbolic management

Examples of typing rules using time-indexed types:

- an observation at time t is available at all times t' after t:

$$\frac{t \leq t'}{\Gamma \vdash \mathsf{Obs}(l, t) : \tau^{t'}}$$

- an expression e can only meaningfully scale a contract c if e is available at some time t' and c makes no stipulations strictly before t':

$$\frac{\Gamma \vdash e : \mathit{Real}^{t'} \qquad \Gamma \vdash c : \mathit{Contr}^{t'} \qquad t \leq t'}{\Gamma \vdash e \times c : \mathit{Contr}^{t}}$$

Functional Reactive programs

Most of the typing for financial DSLs is based on *functional reactive programming*, i.e., pure functions over signals. This includes loops as well.

Linear-time Temporal Logic (LTL) cite:Jeffrey:2012fh, cite:Pnueli:1977wy is a natural extension of the type system for FRP, which constrains the temporal behaviour of reactive programs.

Functional reactive programs

- LTL can be defined in a dependently typed functional language, and reactive programs form proofs of constructive LTL properties. Types are indexed over time[2]:

$$RSet = Time \to \mathcal{U}_0, \quad RSet \in \mathcal{U}_1$$

- One of the inhabitant of *RSet* is the function *Past*:

[2] we use $\mathcal{U}_i$ to represent the universe of types i.

and temporal logic cite:Pnueli:1977wy

Correctness over time can be reduced to

- Invariance: a property holding throughout the execution of a program
- Eventuality: temporal implication, or a property Q eventually follows from P

References

Papers

bibliographystyle:unsrt bibliography:biblio.bib