Yet another DSL for Financial Contracts

– and the story behind it –

Jost Berthold*

jberthold@acm.org

Commonwealth Bank

(*formerly ‡University of Copenhagen)

Joint work with Martin Elsman‡ and Patrick Bahr* (now ITU Copenhagen)
published at ICFP 2015[1]

Talk at FP Syd, April 2016

[1] https://github.com/HIPERFIT/contracts

About the Speaker: Jost Berthold




Research: Concepts/Implementation of Parallel Functional Programming

. . . and sometimes other topics.

2008 Dr.rer.nat. – Philipps-Universität Marburg
2008 Research Intern – Microsoft Research (GHC)
PostDoc, SCIEnce – University of St.Andrews
2009 PostDoc, grid.dk – University of Copenhagen
2011 Assistant Professor, HIPERFIT – U.Copenhagen
2015 Commonwealth Bank

Overview

Overview

Sneak Preview: The Contract Language

Contract in natural language

- At any time within the next 90 days,
- party X may decide to
- buy EUR 1000 from party Y,
- for a fixed rate 1.15 of USD.

Translation into our contract language

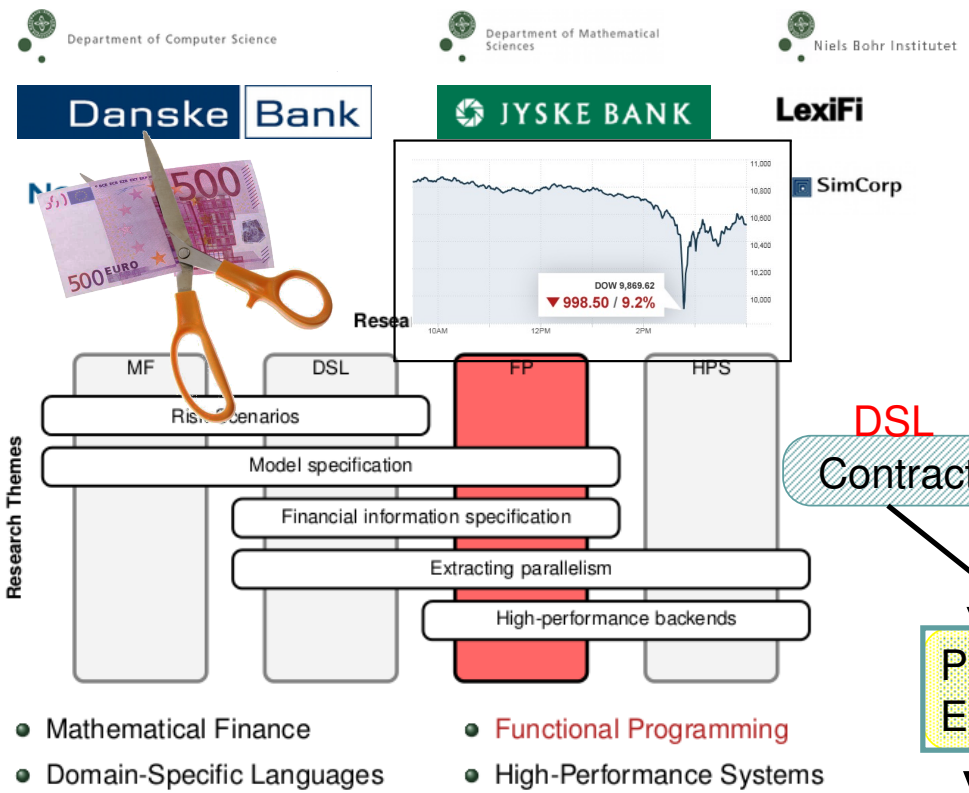
if obs$(X$ exercises option$, 0)$ **within** 90

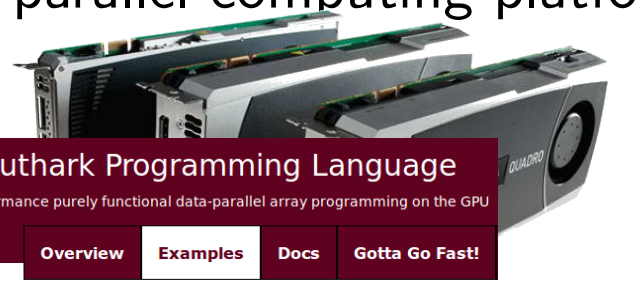
then $1000 \times (\mathrm{EUR}(Y \to X)\,\&\,(1.15 \times \mathrm{USD}(X \to Y)))$

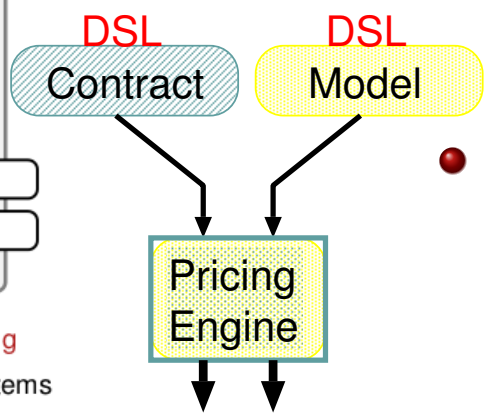
else $\emptyset$

HIPERFIT – Functional High-Performance Financial IT

- state-funded Danish research initiative (centre), 2011 – 2016
- DSLs and FP for math/finance – on parallel computing platforms







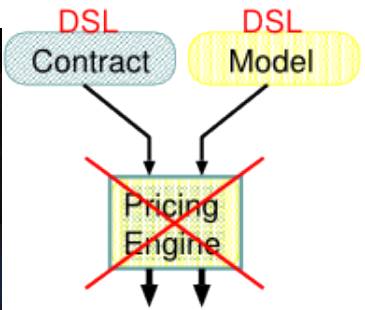
- Math-Finance
- DSLs
- parallel FP

Martin Elsman –
HIPERFIT, DIKU

Patrick Bahr –
ITU, formerly DIKU

A DSL vision...

Working with a risk management team from one of our partners...
a vision emerged:



Banks have large portfolios of derivatives

Example: FX derivatives (promise or option to buy, "insurance" against rate moves, choice of currency...)

- Analyse a portfolio symbolically;
- consider the entire portfolio together;

...using an appropriate DSL.

Required/desireds for this DSL:

- what-if scenarios
- ...and likelihoods
- multiple parties
- compositional contracts
- avoid focus on pricing

Existing DSLs for Financial Contracts

MLFi language[2] by Lexifi, for contract management and valuation

- Core product: pricing
- integrated in contract management software

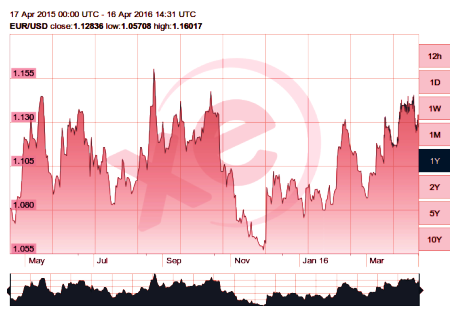
Example: "Zero-coupon bonds"

(MLFi-Haskell paraphrased)

```
eu1 = zcb (date "1 May 2016") 100 EUR :: Contract
us1 = zcb (date "1 May 2016") 115 USD :: Contract
eu2 = zcb (date "31 Jul 2016") 100 EUR :: Contract

c = (eu1 `or` us1) `and` give eu2
```

- How "valuable" is the above opportunity? Contract valuation semantics
- Used today in a number of banks: similar in-house languages



[2] Peyton Jones, Eber, Seward. *Composing contracts: an adventure in financial engineering*. ICFP'00, ACM (2000)

Another example. . . and what we did not like

"American FX Put Option"

(MLFi-Haskell paraphrased)

```
americanPut = anytime (between t1 t2) zero sell
  where t1     = date "01 May 2016"
        t2     = date "31 Jul 2016"
        amount = konst 1000
        strike = konst 1.15
        sell   = scale amount
                 (one EUR `and` give (scale strike (one USD)))
```

- Great for contract management and valuation:
- But: who is buying?
- And difficult to compose with absolute dates in it $t1, t2$
- Observables carry modeling information

(in many implementations)

Overview

Compositional Multi-Party Contracts

American Put again – in our contract language

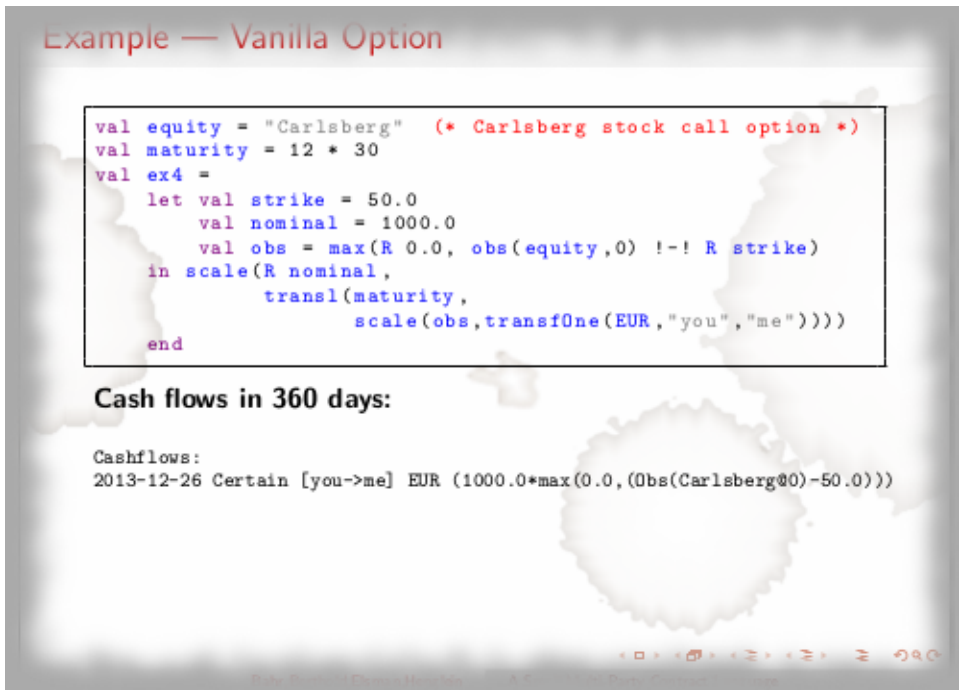
if obs(X exercises option, 0) **within** 90
then 1000×(EUR($Y \to X$)& (1.15×USD($X \to Y$)))
else $\emptyset$

- Atoms: Asset transfer (here: FX, can be any asset)
- scaling and combining similar to MLFi
- Observation (`X exercises option`) explicit, observed with label and time
- Relative time (need to fix a date externally)
- Parties named explicitly for all transfers

$american1 =$ **if** $bobs$ ($Decision\ X$ "exercise" 0) ‘$within$‘ 90
then 1000 # (transfer $Y\ X\ EUR$ & 1.15 # transfer $X\ Y\ USD$)
else zero

In the beginnings...

When we started our language experiments we used Standard ML.

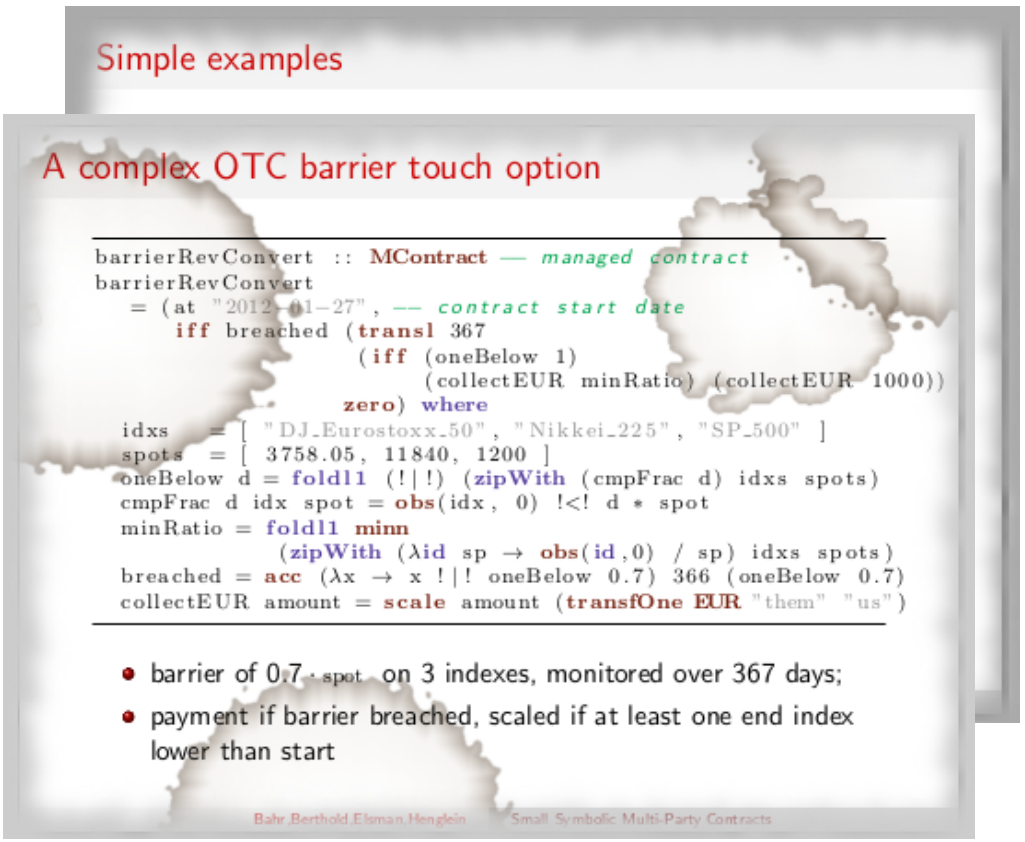


- working on simple stock options and FX derivatives
- First concentrating on potential applications
  - Expected cash flows (management)
  - Simplification, execution of contracts

This is a European, or "Vanilla" option.

. . . a bit later

Switching to Haskell allowed for better expression types.



- Refining the language constructs
- Trying to cover more option types (introducing new features)
- . . . not always with the prettiest surface syntax

This is a complex OTC barrier contract with 3 underlyings.

More Options

Asian Option After 90 days, party X may decide to buy USD 100; paying the average of the exchange rate USD to DKK observed over the last 30 days.

$$\begin{aligned} \mathit{asian} = 90 \uparrow\ & \textbf{if } \textbf{obs}(X \text{ exercises option}, 0) \textbf{ within } 0 \\ & \textbf{then } 100 \times (\mathrm{USD}(Y \to X)\ \&\ (\mathit{rate} \times \mathrm{DKK}(X \to Y))) \\ & \textbf{else } \emptyset \\ \text{where} \quad & \mathit{rate} = \textbf{acc}(\lambda r.\ r + \textbf{obs}(\mathrm{FX}(\mathrm{USD}, \mathrm{DKK}), 0), 30, 0)/30 \end{aligned}$$

- Note the strike price is an average (market price during the last 30 days)
- $rate$ is a metavariable.

$$\begin{aligned} \mathit{asian} = 90\ !\ & \textbf{if } \mathrm{bObs}\ (\mathit{Decision}\ X\ \text{"exercise"})\ 0 \\ & \textbf{then } 100\ \#\ (\mathrm{transfer}\ Y\ X\ \mathit{USD}\ \&\ (\mathit{rate}\ \#\ \mathrm{transfer}\ X\ Y\ \mathit{DKK})) \\ & \textbf{else } \mathrm{zero} \\ \textbf{where } \mathit{rate} & = (\mathrm{acc}\ (\lambda r \to r + \mathrm{rObs}\ (\mathit{FX}\ \mathit{USD}\ \mathit{DKK})\ 0)\ \ 30\ 0)\ /\ 30 \end{aligned}$$

Overview of the Contract Language

Contract combinators

- $\emptyset$
- $a(p \to q)$
- $d \uparrow c$
- $e \times c$
- $c_1 \,\&\, c_2$
- **let** $x = e$ **in** c
- **if** e **within** d **then** c_1 **else** c_2

Expression Language

Real-valued and Boolean-valued expressions, extended by

obs(l, d) observe the value of l at time d

acc(f, d, e) accumulation over the last d days

Implementation in Coq, extracting Haskell code (types replaced by Haskell types)

Overview

Example: Credit Default Swap

Bond

$$C_{\text{bond}} = \textbf{if obs}(X \text{ defaults}, 0) \textbf{ within } 30 \textbf{ then } \emptyset \textbf{ else } 1000 \times \text{EUR}(X \to Y)$$

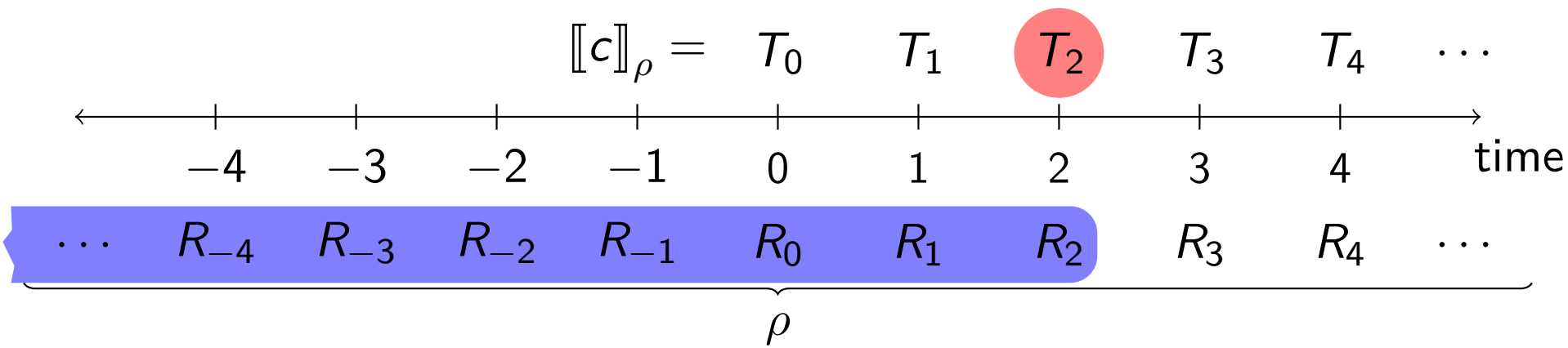
Credit Default Swap

$$C_{\text{CDS}} = (10 \times \text{EUR}(Y \to Z)) \,\&\, \textbf{if obs}(X \text{ defaults}, 0) \textbf{ within } 30 \textbf{ then } 900 \times EUR(Z \to Y) \textbf{ else } \emptyset$$

$$C_{\text{bond}} \,\&\, C_{\text{CDS}} \equiv (10 \times \text{EUR}(Y \to Z)) \,\&\, \textbf{if obs}(X \text{ defaults}, 0) \textbf{ within } 30 \textbf{ then } 900 \times EUR(Z \to Y) \textbf{ else } 1000 \times \text{EUR}(X \to Y)$$

Denotational Semantics

$$[\![\cdot]\!]_{\cdot} : \text{Contr} \times \text{Env} \to \text{CashFlow}$$



$$[\![c]\!]_\rho \in \quad \text{CashFlow} = \mathbb{N} \to \text{Transactions}$$

$$T_i \in \quad \text{Transactions} = \text{Party} \times \text{Party} \times \text{Asset} \to \mathbb{R}$$

$$\rho \in \quad \text{Env} = \text{Label} \times \mathbb{Z} \to \mathbb{B} \cup \mathbb{R}$$

Denotational Semantics: Some Details

$$\boxed{\mathcal{E}\,[\![e]\!] : [\![\Gamma]\!] \times \mathrm{Env} \to [\![\tau]\!]}$$

- Accumulator steps forward in time

$$\mathcal{E}\,[\![\mathbf{acc}(\lambda x.\ e_1, d, e_2)]\!]_{\gamma,\rho} = \begin{cases} \mathcal{E}\,[\![e_2]\!]_{\gamma,\rho} & \text{if } d = 0 \\ \mathcal{E}\,[\![e_1]\!]_{\gamma[x \mapsto v],\rho} & \text{if } d > 0 \end{cases}$$

$$\text{where} \qquad v = \mathcal{E}\,[\![\mathbf{acc}(e_1, d-1, e_2)]\!]_{\gamma,\rho/-1}$$

$$\boxed{[\![c]\!] : [\![\Gamma]\!] \times \mathrm{Env} \to \mathbb{N} \to \mathrm{Party} \times \mathrm{Party} \times \mathrm{Asset} \to \mathbb{R}}$$

- Each cash flows appears twice:

$$[\![a(p \to q)]\!]_{\gamma,\rho} = \begin{cases} \lambda n.\lambda t.0 & \text{if } p = q \\ unit_{a,p,q} & \text{otherwise, where} \end{cases}$$

$$unit_{a,p,q}(n)(p', q', b) = \begin{cases} 1 & \text{if } b = a, p = p', q = q', n = 0 \\ -1 & \text{if } b = a, p = q', q = p', n = 0 \\ 0 & \text{otherwise} \end{cases}$$

- Variables invade everything: $[\![\mathbf{let}\ x = e\ \mathbf{in}\ c]\!]_{\gamma,\rho} = [\![c]\!]_{\gamma[x \mapsto v],\rho}$, where $v = \mathcal{E}\,[\![e]\!]_{\gamma,\rho}$

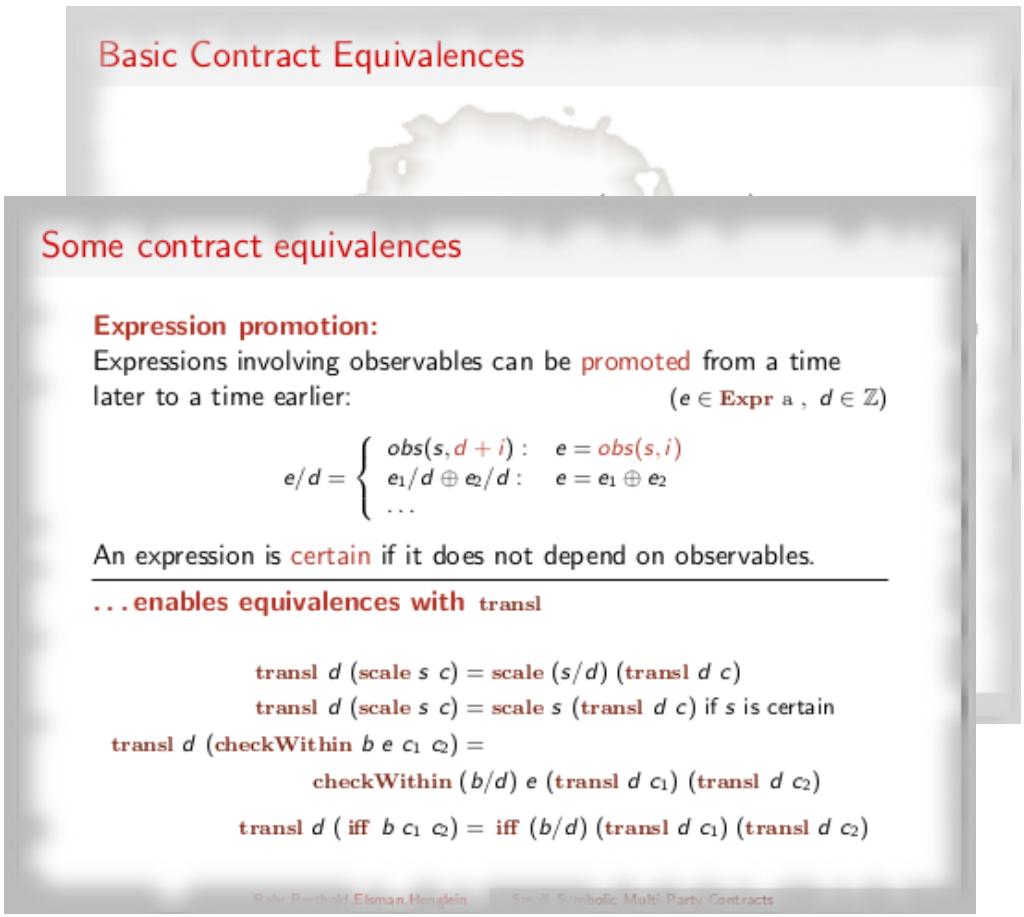
Denotational Semantics

$$[\![\cdot]\!]_{\cdot} : \text{Contr} \times \text{Env} \rightarrow \text{CashFlow}$$

- Contract and external environment naturally separated
- Scenarios: selected environments
- Simulation: statistical model of environments
- Symbolic analysis: on contracts, not environments

Contract Equivalences

We were contemplating contract equivalences very early in this work.



- They enable simplification of large portfolios.
- . . . especially useful when contracts are also transformed.

- Some are very simple,
- some involve promoting expressions in time.

. . . but the CDS equivalence is not proven by any of our rules :-)

Another Tricky Detail: Contract Causality

A funny contract. . . (X pays tomorrow's price of one EUR to Y today ???)

$$\mathit{funny} = \mathbf{obs}(\mathrm{FX}(\mathrm{EUR}, \mathrm{USD}), 1) \times \mathrm{USD}(X \to Y)$$

Contracts may be non-causal.

- No reference to observables in the future: obvious causality.

But some transformations may introduce future observable values.
Some contracts are typically expressed using references to future.

- Advanced time-indexed type system to infer contract causality.
- Causality not compositional (non-causal transfers can cancel out)

Type System

Time-Indexed Types

- $e : \mathrm{Real}^t$, $e : \mathrm{Bool}^t$ — value of e available at time t (and later)
- $c : \mathrm{Contr}^t$ — no obligations strictly before t

Essential Typing Rules

$$\frac{\Gamma \vdash e : \mathrm{Real}^s \quad \Gamma \vdash c : \mathrm{Contr}^s \quad t \leq s}{\Gamma \vdash e \times c : \mathrm{Contr}^t}$$

$$\frac{l \in \mathrm{Label}_\tau \quad t \leq s}{\Gamma \vdash \mathbf{obs}(l, t) : \tau^s} \qquad \frac{t \leq 0}{\Gamma \vdash a(p \to q) : \mathrm{Contr}^t}$$

$$\vdots$$

Contract Analysis: Horizon

Time until a contract is guaranteed to be $\emptyset$:
Approximated by a simple syntax-directed analysis

Appendix: Contract Horizon

d is the horizon of a contract: $c \dashv d$: $(c \in \text{contr},\ d \in \mathbb{Z}_0^+)$

$$\frac{}{\textbf{zero} \dashv 0}\ \textit{(H-Zero)} \qquad \frac{}{\textbf{transfOne}\ (C, p_1, p_2) \dashv 0}\ \textit{(H-TO)}$$

$$\frac{b_1 \vdash e \quad c \dashv b_2 \quad b_1 \leq b_2}{\textbf{scale}\ (e, c) \dashv b_2}\ \textit{(H-Sc)}$$

$$\frac{c \dashv i}{\textbf{transl}\ (d, c) \dashv (i + d)}\ \textit{(H-TL)} \qquad \frac{c_1 \dashv b_1 \quad c_2 \dashv b_2}{\textbf{both}\ (c_1, c_2) \dashv max(b_1, b_2)}\ \textit{(H-Both)}$$

$$\frac{0 \vdash e \quad c_1 \dashv b_1 \quad c_2 \dashv b_2}{\textbf{checkWithin}\ (e, d, c_1, c_2) \dashv (d + max(b_1, b_2))}\ \textit{(H-CW)}$$

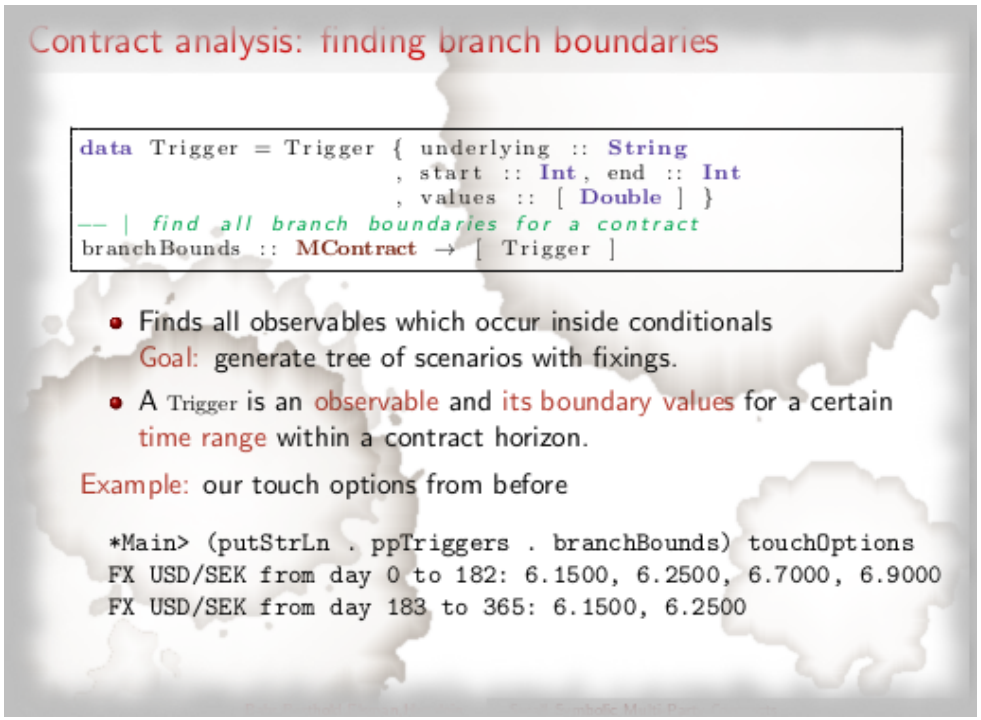
Intuition: $c \dashv i$ means the last transfer may happen after no more than i days (*H-TL* and *max*). c may or may not be causal (*H-SC*).

$$\text{HOR}(a($$
$$\text{HOR}(\textbf{let}\ x =$$
$$\text{HOR}($$
$$\text{HOR}\left(\begin{array}{l}\textbf{if}\ e\ \textbf{with}\\ \textbf{then}\ c_1\ \textbf{e}\end{array}\right.$$

where

Contract Analysis: More (older) experiments...

Initial purpose: Risk analysis. Some ideas we had...



- Symbolic analysis: branch boundaries depending on observable values
- Purpose: targeted simulation for risk purposes
- Incomplete, worked on only some Boolean expressions

- Hedging advice based on simulation
- Level-of-detail adjustment, to simplify a portfolio

Overview

Contract Equivalences for Simplification

Can use equivalences to rewrite contracts

$$e_1 \times (e_2 \times c) \equiv (e_1 \cdot e_2) \times c$$
$$d_1 \uparrow (d_2 \uparrow c) \equiv (d_1 + d_2) \uparrow c$$
$$d \uparrow (c_1 \,\&\, c_2) \equiv (d \uparrow c_1) \,\&\, (d \uparrow c_2)$$
$$e \times (c_1 \,\&\, c_2) \equiv (e \times c_1) \,\&\, (e \times c_2)$$
$$d \uparrow (e \times c) \equiv (d \Uparrow e) \times (d \uparrow c)$$
$$(e_1 \times c) \,\&\, (e_2 \times c) \equiv (e_1 + e_2) \times c$$

$$d \uparrow \emptyset \equiv \emptyset$$
$$r \times \emptyset \equiv \emptyset$$
$$0 \times c \equiv \emptyset$$
$$c \,\&\, \emptyset \equiv c$$
$$c_1 \,\&\, c_2 \equiv c_2 \,\&\, c_1$$

$$d \uparrow \textbf{if } b \textbf{ within } e \textbf{ then } c_1 \textbf{ else } c_2 \equiv$$
$$\textbf{if } d \Uparrow b \textbf{ within } e \textbf{ then } d \uparrow c_1 \textbf{ else } d \uparrow c_2$$

- . . . assuming a preference of one form over the other

Contract Specialisation

Goal: simplify contracts based on available external information

- Known decisions ⇒ Eliminate branches
- Known observable values ⇒ Compute scaling, eliminate branches

Specialised contract: considers information from partial environment.

$$\begin{aligned} \mathit{spec} &: \mathsf{Env_P} \times \mathsf{Contr} \to \mathsf{Contr} \\ & \quad c \equiv_\rho \mathit{spec}(\rho, c) \text{ for } \rho \in \mathsf{Env_P} \\ \mathsf{Env_P} &: \mathsf{Label}_\tau \times \mathbb{Z} \rightharpoonup [\![\tau]\!] \text{ (partial)} \end{aligned}$$

- The contract is modified, no general equivalence.
- Equivalence $\equiv_\rho$ ($\rho \in \mathsf{Env_P}$) limited to ρ' : Env that extend ρ, i.e. $\rho' \in$ Env with $\rho(l,t) = \rho'(l,t)$ for all$(l,t) \in \mathit{dom}(\rho)$.
- Implementation using smart constructors: earlier equivalences.

Contract Reduction

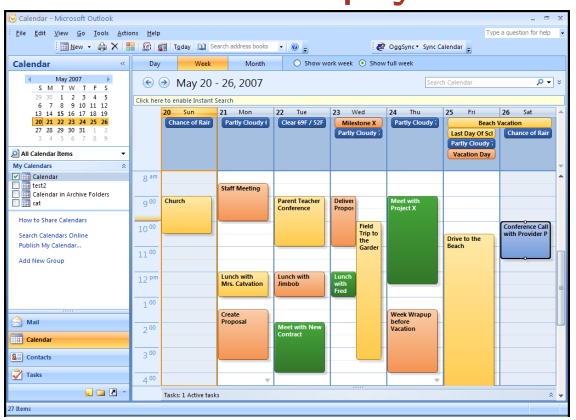
Transformations so far: based on equivalences and partial knowledge.
No actual contract execution (deciding, fulfilling transfers, etc.)

Contract transformation in time:

- As time passes, contracts gradually evolve into empty contracts:

Advancing a contract in time.

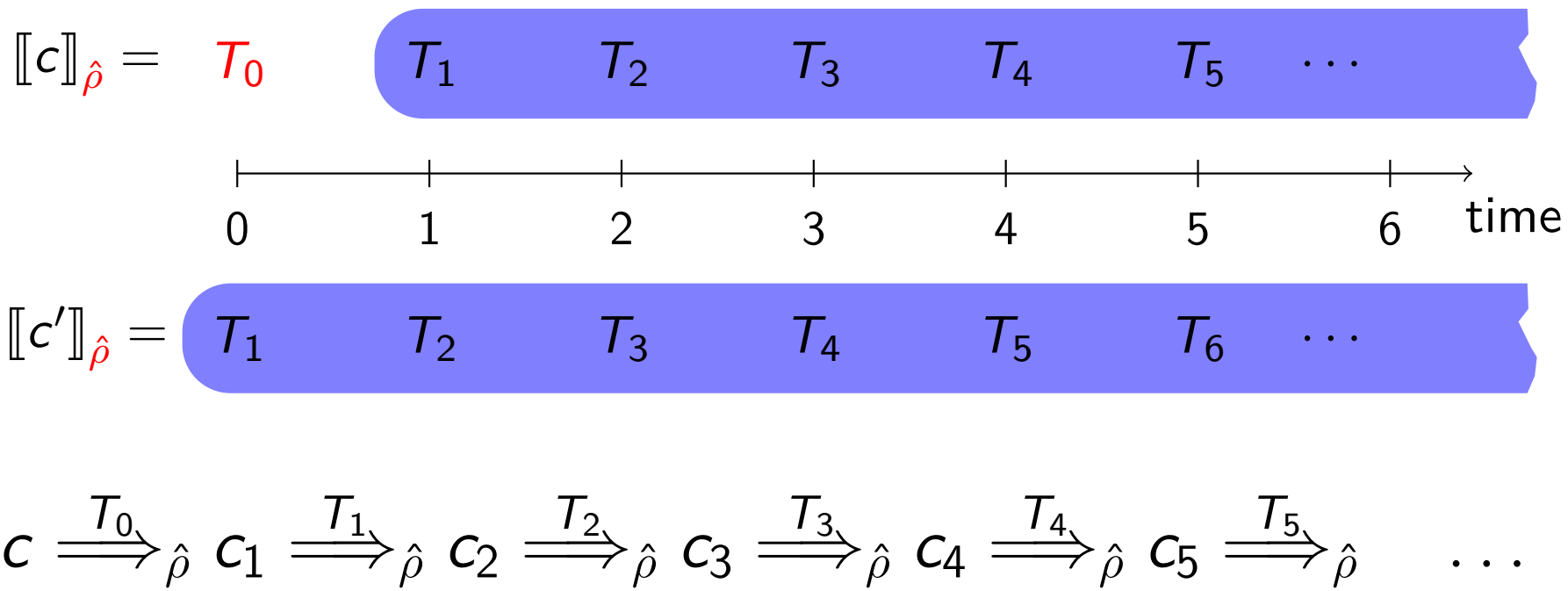
(a major motivation for MLFi and many in-house languages).



- Semantic model: Contract reduction with associated transfers
  - Assumes given (complete) environment and causal contract
  - Proved sound and adequate with these assumptions

Reduction Semantics

$$c \xRightarrow{T_0}_{\rho} c'$$



$$c \xRightarrow{T_0}_{\hat{\rho}} c_1 \xRightarrow{T_1}_{\hat{\rho}} c_2 \xRightarrow{T_2}_{\hat{\rho}} c_3 \xRightarrow{T_3}_{\hat{\rho}} c_4 \xRightarrow{T_4}_{\hat{\rho}} c_5 \xRightarrow{T_5}_{\hat{\rho}} \quad \cdots$$

Overview

Embedding in Coq, Generating Haskell

Coq formalisation

Denotational semantics is the starting point

- Adequacy of reduction semantics
- Type safety (well-typed $\rightsquigarrow$ causal)
- Soundness & completeness of type inference
- Soundness of partial evaluation & horizon inference

Extraction of executable Haskell code

- efficient Haskell implementation
- embedded domain-specific language for contracts
- contract analyses and contract management

Contracts in Haskell – Example

```
{-# LANGUAGE RankNTypes, RebindableSyntax #-}

module Options where
import RebindableEDSL
import Prelude hiding (ifThenElse, (>>=), (>>), min, max)

asian :: Contr -- asian option
asian = 90 ! if bObs (Decision X "exercise") 0
              then 100 # (transfer Y X USD & (rate # transfer X Y DKK))
              else zero
    where rate  = (acc (λr → r + rObs (FX USD DKK) 0)  30 0) / 30

chooser :: Contr -- chooser option, looks non-causal
chooser = do price ← rObs (FX DKK USD) 60
             payout ← ife (bObs (Decision X "call option") 30)
                          (max (price - strike) 0)
                          (max (strike - price) 0)
             60 ! (payout # transfer Y X DKK)
    where strike = 6.5 -- :: Exp R

weird :: Contr --non-causal example
weird = rObs (FX EUR USD) 1 # transfer X Y USD
```

Expression implementations in contrast

SML:

```
type 'a exp                          (* Expressions          *)
type bexp = bool exp                 (* Boolean expressions  *)
type 'a num
type 'a nexp = 'a num exp            (* Numeric expressions  *)
type rexp = real nexp                (*  real expressions    *)
type iexp = int nexp                 (*  integer expressions *)

val I   : int → iexp
val R   : real → rexp
val B   : bool → bexp
val ⊕   : 'a nexp * 'a nexp → 'a nexp   (* max, +, ... *)
val obs : string * int → rexp
```

Haskell:

```
data Expr a where
    I :: Int    → Expr Int    — Int
    R :: Double → Expr Double — Double
    B :: Bool   → Expr Bool   — Bool
    V :: Var    → Expr a      — Variable
-- | arithmetic operations: + - * / max min
⊕ :: Num a ⇒ Expr a → Expr a → Expr a
-- | logical operations: < = ! |
!<! :: Ord a ⇒ Expr a → Expr a → Expr Bool
!=! :: Eq a  ⇒ Expr a → Expr a → Expr Bool
not :: Expr Bool → Expr Bool
!|! :: Expr Bool → Expr Bool → Expr Bool

type RealE = Expr Double; type BoolE = Expr Bool
```

Coq/Haskell:

```
Inductive Exp : Set := OpE (op : Op) (args : list Exp)
                     | Obs (l:ObsLabel) (i: Z)
                     | VarE (v:Var)
                     | Acc (f : Exp) (d : nat) (e : Exp).
```

```
data Exp =
   OpE Op (List Exp)
 | Obs ObsLabel Int
 | VarE Var
 | Acc Exp Int Exp
```

Overview

Summary of our Journey today

Thanks for joining me on this journey...

- Starting from some simple coding experiments in SML
- ...with reasonably clear goals (risk managemnt, symbolic computation),
- via some embedding experiments in Haskell
- ...and gradually introducing more and more formal treatment, **(† leaving the original bank partner on the track, sorry †)**
- to the final result implemented in Coq:

  Certified analysis and transformation of well-typed causal contracts

  (which attracted interest of research community and Lexifi)