An Actuarial Programming Language for Life Insurance and Pensions

David R. Christiansen [1] Klaus Grue [2] Henning Niss [2]
Peter Sestoft [1] Kristján S. Sigtryggsson[2]

[1] IT University of Copenhagen

[2] Edlund A/S

- Contracts have a longer lifespan than IT systems

- At any given time, multiple systems must administer a contract

Our Vision

- A **formalized description** of life insurance and pension products
- Supporting **automated** administration and reporting
- **Readable** and **manageable** by humans

Participants

edlund – inside insurance

UNIVERSITY OF COPENHAGEN

IT University of Copenhagen

Supported by

The Danish National Advanced Technology Foundation

Overview

Context

AML Models

Safety

Status and Continuing Work

Overview

Context

AML Models

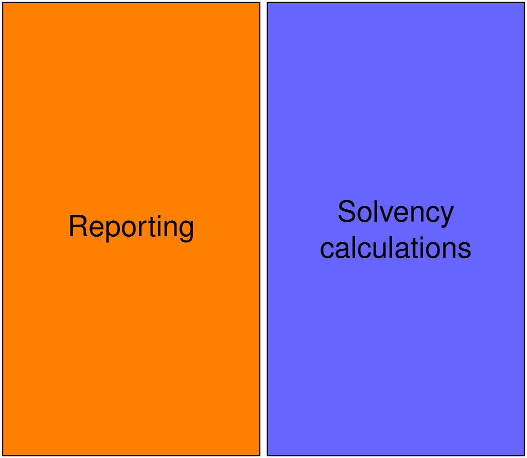
Safety

Status and Continuing Work

Context

- Solvency II: new EU rules require more flexible calculations
- Contracts are held longer than IT systems exist
- Current programming tools are too slow or too difficult

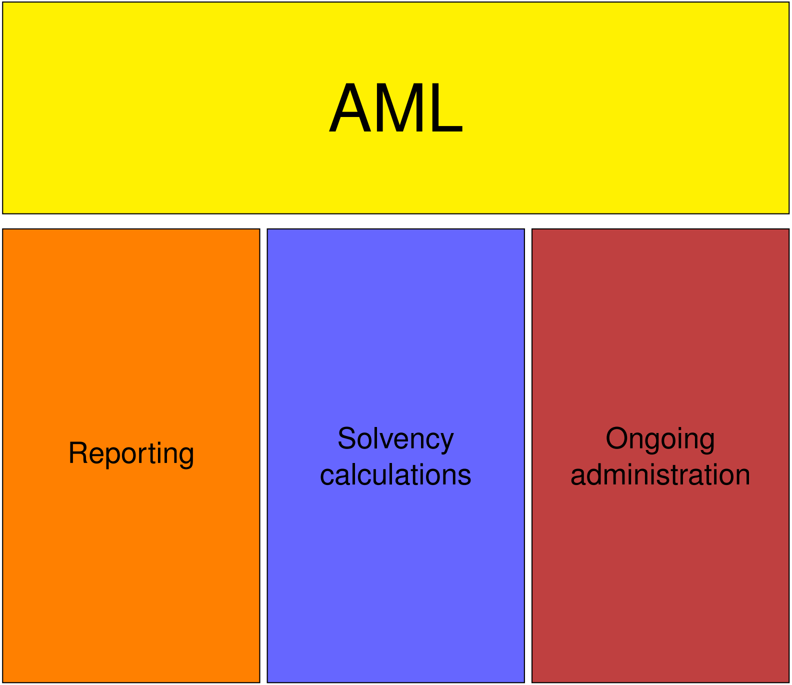
Reporting


Reporting
Solvency calculations

Reporting

Solvency calculations

Ongoing administration


AML
Reporting
Solvency calculations
Ongoing administration

Models

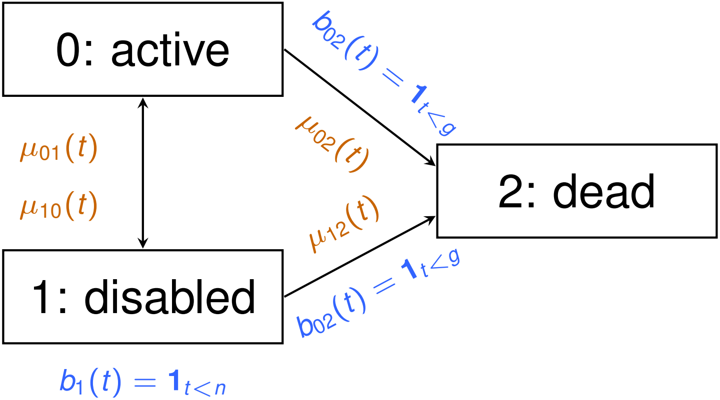
Sample product

- Pay \$1 on death before some time g
- Before some expiry time n, pay \$1 per year while disabled
- Allow an unlimited number of disability diagnoses and reentries to the workforce

Modeling risk

- 3-state continuous-time Markov model
- States: 0 active, 1 disabled, 2 dead
- Transition intensities: $\mu_{ij}(t)$ at time t

Models

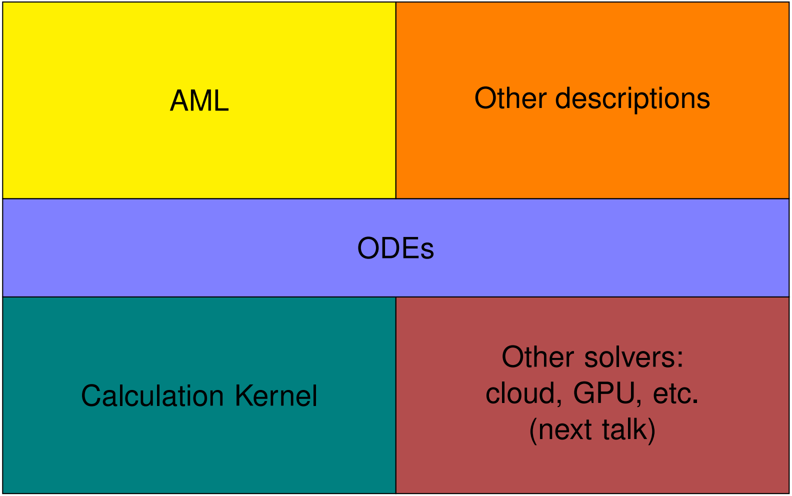


0: active
1: disabled
2: dead
$\mu_{01}(t)$
$\mu_{10}(t)$
$\mu_{02}(t)$
$\mu_{12}(t)$
$b_{02}(t) = \mathbf{1}_{t<g}$
$b_{02}(t) = \mathbf{1}_{t<g}$
$b_1(t) = \mathbf{1}_{t<n}$

Thiele's Differential Equations

$$
\begin{aligned}
\frac{d}{dt} V_j(t) = & r(t) V_j(t) - b_j(t) \\
& - \sum_{k \neq j} \mu_{jk}(t) \left(b_{jk}(t) + V_k(t) - V_j(t) \right)
\end{aligned}
$$

Overall Architecture



Overview

Context

AML Models

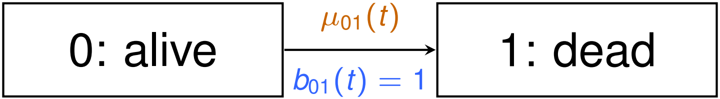
Safety

Status and Continuing Work

AML Models

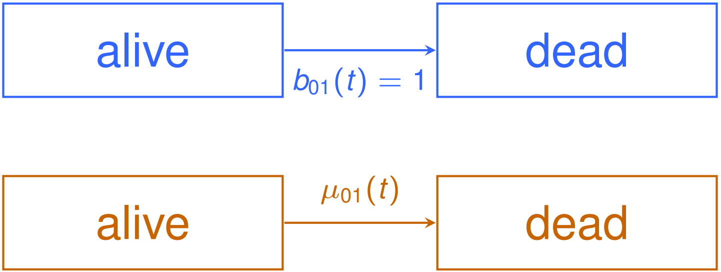
- Separate *risk models* from *product definitions*
- Define transformations on products and risk models
- Generate ODEs from the flexible, readable models
- Allow fast experimentation with new products

Whole-Life Insurance



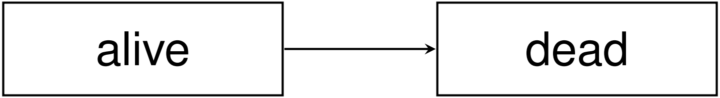
Upon death of insured, pay \$1. Intensity of mortality is $\mu_{01}(t)$.

AML : Whole-Life Insurance



Separate payment from risk information and name the states.

State Models




```
statemodel LifeDeath(p : Person) where
  states =
    alive
    dead
  transitions =
    alive -> dead
```

Risk Models

```
riskmodel RiskLifeDeath(p : Person) : LifeDeath(p) where
  intensities =
    alive -> dead by gompertzMakehamDeath(p)
```

- Risk models give transition intensities
- Here information about the insured is used to calculate the intensities
- `RiskLifeDeath` is defined *inside* `LifeDeath`

Whole-Life Insurance in AML

```
product WholeLifeInsurance(p : Person) : LifeDeath(p) where
  obligations =
    pay $1 when(alive -> dead)
```

- Products consist of payment specifications
- Payment specifications determine who will pay what when, and under which circumstances

Calculation Bases

```
basis BasisLifeDeath(p : Person) : LifeDeath(p) where
  riskModel = RiskLifeDeath(p)
  interestRate = constant(0.05)
  maxtime = p.BirthDate + 120
```

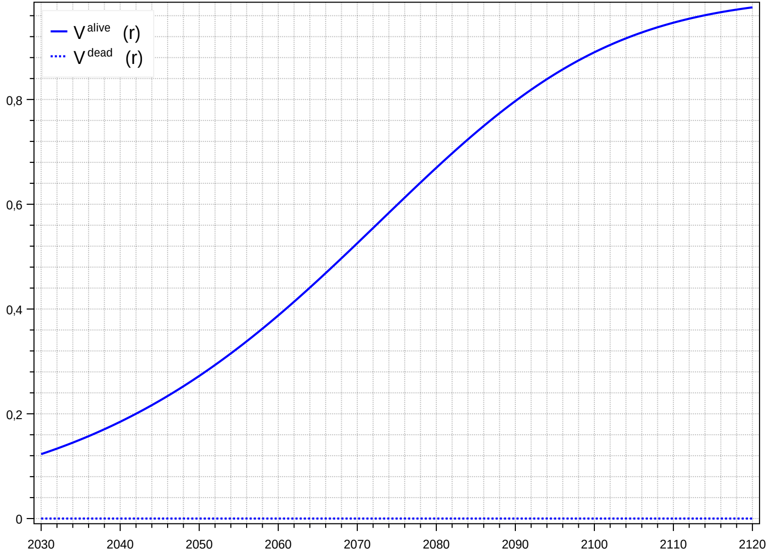
- A basis contains everything needed to compute a reserve
- The interest rate is an arbitrary function, and the `constant` operator creates a constant function
- Some bases will have more information for products that need additional phenomena modeled

Computing Reserves

```
value jane : Person = Person("Jane",
                             TimePoint(2000,1,1),
                             Female)
value r : Money =
  reserve(TimePoint(2030, 1, 1), alive,
          WholeLifeInsurance(jane),
          BasisLifeDeath(jane))
```

- `jane` represents a customer: name, birthdate, and sex
- `reserve` calculates a reserve at some time for some state, from a product and a basis

Reserves for Whole Life Insurance



Real-World Payments

```
statemodel Riskless where
  states = noMatterWhen
```

Real-World Payments

```
statemodel Riskless where
  states = noMatterWhen

product RDA(start: TimePoint,
            expiry: TimePoint) : Riskless where
  obligations =
    at t pay $1 per year
      provided(start < t < expiry)
```

Real-World Payments

```
statemodel Riskless where
  states = noMatterWhen

product RDA(start: TimePoint,
            expiry: TimePoint) : Riskless where
  obligations =
    at t pay $1 per year
      provided(start < t < expiry)


product RDA'(start: TimePoint,
             expiry: TimePoint) : Riskless where
  obligations =
    at t pay payments((expiry-start)*2,
                      $1,
                      1/2 years,
                      start, t)
```

Two-Life Insurance

```
statemodel TwoLife where
  states      = alive_alive
              | alive_dead
              | dead_alive
              | dead_dead
  transitions = alive_alive -> alive_dead
              | alive_alive -> dead_alive
              | alive_dead  -> dead_dead
              | dead_alive  -> dead_dead

product TwoLifeSum : TwoLife where
  obligations = pay $1 when(alive_alive -> alive_dead)
                pay $1 when(alive_alive -> dead_alive)
```

Expressive Power

- AML can represent all standard Danish reference pension products — even those with unknown beneficiaries

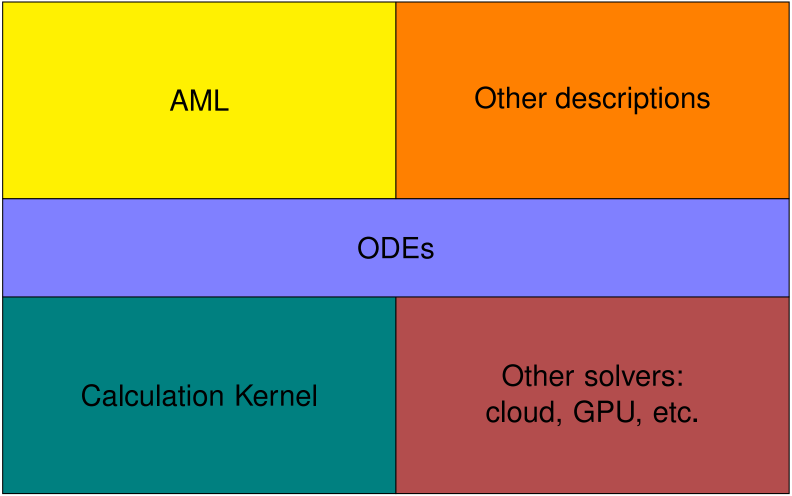
```
product SpouseBenefits(p : Person) : LifeDeath(p) where
  obligations = at t pay $1
                when(alive -> dead)
                provided(married)
                given(married ~ basis.marriageProb(p, t))
```

Expressive Power

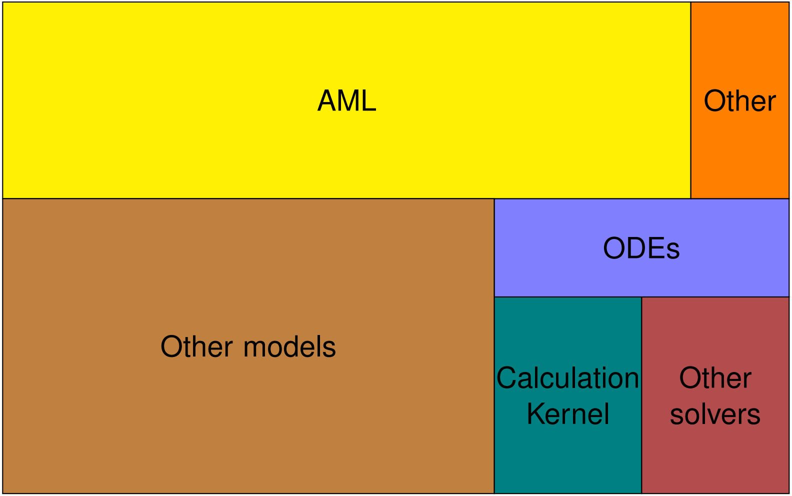
```
function marriage(p : Person,
                  t : TimePoint) : Dist(Bool) =
  boolDist(if p.Gender == Male then 0.8 else 0.55)

basis SpouseBasis(p : Person) where
  riskModel = RiskLifeDeath(p)
  interestRate = constant(0.02)
  maxtime = p.BirthDate + 120
  marriageProb = marriage
```

Perspective



Perspective



Overview

Context

AML Models

Safety

Status and Continuing Work

Domain-Specific Languages

- "Little languages" that support one area very well and others not at all
- Can be safer and faster due to special knowledge
- Examples: SQL, R, $\TeX$

AML Properties

- Type system — prevent errors before the code is run
- Termination — no infinite loops
- Functions are mathematical functions

Preventing Mistakes

Catch errors such as multiplying a date by a dollar:

```
value hourly : Money = $5
value hours : TimePoint = TimePoint(2014,4,3)
value wage = hourly * hours
```

Preventing Mistakes

Catch errors such as multiplying a date by a dollar:

```
value hourly : Money = $5
value hours : TimePoint = TimePoint(2014,4,3)
value wage = hourly * hours
```

Catch mismatches between products, state models, and bases:

```
product DisabilityInsurance(p : Person) : LifeDeath(p) where
  obligations = pay $1 provided(disabled)
```

Preventing Mistakes

Catch errors such as multiplying a date by a dollar:

```
value hourly : Money = $5
value hours : TimePoint = TimePoint(2014,4,3)
value wage = hourly * hours
```

Catch mismatches between products, state models, and bases:

```
product DisabilityInsurance(p : Person) : LifeDeath(p) where
  obligations = pay $1 provided(disabled)
```

Even catch the wrong person:

```
value r : Money =
  reserve(TimePoint(2030, 1, 1), alive,
          WholeLifeInsurance(joe),
          BasisLifeDeath(jane))
```

Overview

- Context
- AML Models
- Safety
- Status and Continuing Work

Status

- The Actulus calculation kernel is part of a product available from Edlund
- AML has been implemented, but the type checker is not yet ready
- Alternate calculation kernels from ITU and Edlund demonstrate the flexibility of the approach

Continuing Work

- Continued development and implementation of the AML type system
- Express calculations in AML directly: accounting, solvency, prognosis, etc.
- Long-term administration of AML-defined contracts
- Find ways to make it go faster — see next talk

Come talk with us!

We appreciate your feedback! Try out AML programming at Booth 10, or drop by for questions or dicsussion.