Bartosz Wójcik

Haskell Loan Library

Design of loan calculation library

August 20, 2012

Abstract

Loan calculation can be simply understood as providing for given input the amortization schedule. Amortization schedule often happens to bring unexpected problems: works fine until operates in domain of real numbers. Once domain is changed to one accepted by accounting, it stops amortizing or amortizes with some ugly remainder, which is officially ignored and silently removed "somehow". We describe in this paper the general way of loan calculation which operates in domain of integers and doesn't have problem of amortization remainder. We define the properties amortization schedule has to fulfil. Finally we design library and its API.

Contents

1 Introduction

What is a loan from pure calculation viewpoint? It can be understood as a function which takes certain input (amount of loan, number of instalments, interest rate, some additional information like frequency, special periods, special instalments) and provides with certain output (interest amount, principal to be paid, payment dates). This approach allows taking advantage of functional programming languages in the loan implementation. We present an attempt in Haskell. Before first line of code in Haskell is presented, there is some simply theory to be explained.

2 Some definitions

Lets define some useful basic notions. If we need additional ones for particular purposes, they will be defined later.

- Entity which provides with loan to anyone is of course — **bank**.
- Person who takes the loan is — **debtor**.
- Amount debtor owes to the bank is — **principal sum**.
- Act of providing principal to debtor is — **financing**.
- Amount of principal sum debtor pays back to the bank at foreseen day — **repayment**.
- Amount debtor pays to the bank at scheduled day — **instalment**. Instalment consists of repayment amount and interest paid. For practical reasons instalment is often understood as collection of instalment amount, repayment, interest and all input needed to calculate it.
- Date when certain payment has to be done — **maturity date** or **due date**.
- Time between two consecutive due dates — **instalment period**. Often – just period.
- It is assumed that first instalment's due date is one full period after financing. We can also postpone first instalment by certain number of periods. Then we have — **first instalment deferment** or **postponement**.
- For technical reason we need a **technical due date**. It is like due date, just that payment amount equals zero. This allows us representing first instalment deferment in nice way.
- When interest matured is not fully paid on maturity date then we have — **deferred or late interest**. Usually late interest appears when there is first instalment deferment or when interest for period is greater than instalment amount.
- Loan paid off before scheduled date — **early repayment**.
- Principal paid before scheduled date — **partial early repayment**.
- **Effective interest rate** — see for that in Wikipedia under Effective interest rate.
- List of all instalments with all details ordered by their due dates is — **Amortization Schedule** or **Instalment Plan**. Order is important, exact dates are not required.
- Amortization Schedule with exact dates is — **Instantiated Amortization Schedule**.

3 Problem of amortization

In real life amount required by accounting is a number which is rounded to certain number of digits after decimal point or even to certain number of zeroes before decimal point. Both can be represented by integers. Instalment calculation formulas operate on real numbers. How to achieve results which satisfy both: loan calculation formulas and amount limitations?

We present some attempts and one proper solution. Using simple examples we'll explain particular problems one approaches trying solve this task.

We define following example:

- Principal = 1000
- Duration = 12
- Yearly effective interest rate = 10%

Out of this we get instalment amount $= 87,71554472$.

Raw amortization schedule (presented with float numbers) looks like follows.

Instalment amount	Repayment	Interest paid	Principal after instalment
87,71554472	79,74140429	7,974140429	920,2585957
87,71554472	80,37727344	7,338271273	839,8813223
87,71554472	81,01821311	6,697331607	758,8631092
87,71554472	81,66426372	6,051280999	677,1988454
87,71554472	82,31546603	5,400078692	594,8833794
87,71554472	82,97186111	4,743683606	511,9115183
87,71554472	83,63349038	4,082054334	428,2780279
87,71554472	84,30039558	3,415149137	343,9776323
87,71554472	84,97261877	2,742925945	259,0050136
87,71554472	85,65020237	2,06534235	173,3548112
87,71554472	86,33318911	1,382355608	87,02162208
87,71554472	87,02162208	0,693922635	0

Such an amortization schedule although mathematically correct, is usually not acceptable by accounting. Accounting cannot cope with figures which are not of amount data type, whatever this type in particular case is. In order to transform them into accountable form, one has to round in some way two columns: instalment amount and interest paid. The other columns will round accordingly as they are easy sums of the former two. For example purposes we round or truncate to 2nd position behind decimal point.

3.1 Attempt 1

Let assume we **round** numbers in each line of amortization schedule to amounts **before** we calculate next line of amortization. We round instalment amount and interest amount — repayment becomes rounded then per definition. We get following amortization schedule:

```
87,72   79,75   7,97    920,25
87,72   80,38   7,34    839,87
87,72   81,02   6,70    758,85
87,72   81,67   6,05    677,18
87,72   82,32   5,40    594,86
87,72   82,98   4,74    511,88
87,72   83,64   4,08    428,24
87,72   84,31   3,41    343,93
87,72   84,98   2,74    258,95
87,72   85,66   2,06    173,29
87,72   86,34   1,38     86,95
87,72   87,03   0,69     -0,08
```

One can notice following problems:

- We get principal paid of 0.08 too much as consequence of instalment amount increased by 0.0045 (comparing to float number one in previous paragraph).
- Remaining principal after full loan duration (-0.08 in the example) or difference between presented method and theoretical model in terms of total interest paid (52.5865 in raw model and 52.56 in current example) is proportional to the duration. This means that we may expect not acceptable high amount of remaining principal in certain cases. Justification of this statement is following: total sum of instalments differs from raw one proportionally to the duration. Why? Because each instalment amount gets simply rounded, so the difference for single instalment is in range $(-0.005, 0.005)$. And this difference multiplies by number of instalments. This difference must be distributed between repaid principal and paid interest. Hence at least one of both must be different from raw amortization schedule proportionally to the duration.
- Instalment amount is higher than in raw model. This might be a legal problem.
- If initial principal is small enough (or interest rate high enough, or duration long enough) that interest equals instalment amount, then this method will not amortize the loan. We will obtain repayment equal zero for all instalments. **Example**:
  - Principal = 50
  - Duration = 360
  - Yearly effective interest rate = 20%

3.2 Attempt 2

We can try to improve problems of previous attempt. How to address issue with remaining unpaid principal which is caused by changed instalment amount?

We can resolve this problem by recalculating interest rate so it fits to new instalment amount. This may lead to legal problem though, because we increase both instalment amount and interest rate comparing to the original loan. Let's have a look on result.

The attempt works like follows then: We **round** all instalments of amortization schedule to amounts, then we recalculate interest rate, then we **recalculate and round** interest and repayment for each line.

```
87,72  79,74  7,98  920,26
87,72  80,37  7,35  839,89
87,72  81,02  6,70  758,87
87,72  81,66  6,06  677,21
87,72  82,31  5,41  594,90
87,72  82,97  4,75  511,93
87,72  83,63  4,09  428,30
87,72  84,30  3,42  344,00
87,72  84,97  2,75  259,03
87,72  85,65  2,07  173,38
87,72  86,34  1,38   87,04
87,72  87,03  0,69    0,01
```

Despite hopes the solution is arithmetically not 100% correct – the principal is not paid fully, 0.01 stays unpaid. The reason for this hides in rounding. Since we round interest, discrepancies may cumulate to some visible amount. We come back to this issue in later attempt.

3.3 Attempt 3

This attempt addresses legal issue mentioned in first attempt, keeping improvement achieved in attempt 2. We simply truncate instalment amount instead of rounding it.

The attempt description looks like then: We **truncate** all instalments of amortization schedule to amounts, then we recalculate interest rate, then we **recalculate and round** interest and repayment for each line.

```
87,71  79,75  7,96  920,25
87,71  80,38  7,33  839,87
87,71  81,02  6,69  758,85
87,71  81,67  6,04  677,18
87,71  82,32  5,39  594,86
87,71  82,97  4,74  511,89
87,71  83,63  4,08  428,26
87,71  84,30  3,41  343,96
87,71  84,97  2,74  258,99
87,71  85,65  2,06  173,34
87,71  86,33  1,38   87,01
87,71  87,02  0,69   -0,01
```

Like in previous attempt here also remains not amortized amount of principal. We focus then on question about the size of this discrepancy. Perhaps it is limited and we can accept it. How we can estimate it?

3.3.1 Discrepancy analysis

Like stated in previous paragraph, calculated interest of each instalment has to be rounded before applying it to amortization schedule. This gives maximum discrepancy of 0.005 per instalment. This discrepancy accrues by interest rate every next instalment. We can expect discrepancy being formalized by:

$$\sum_{i=0}^{n-1} \alpha_i (1+r)^i \tag{1}$$

where $\alpha_i \in (-0.005, 0.005]$.

Now we can have two approaches of estimating the size of the discrepancy. First one assumes (incorrectly, but still acceptable for all practical purposes) that αs are probability variables of uniform distribution. Then simulation of the size of discrepancy becomes an analytical task. Second one is practical approach which has been pursued. We have simulated large number of different amortization schedules and collected results per specific duration and interest rate.

Below we present few examples of data collected where original discrepancy is depicted on the X-axe (multiplied by 100 (so 0.01 gives 1)) and % of cases on the Y-axe.

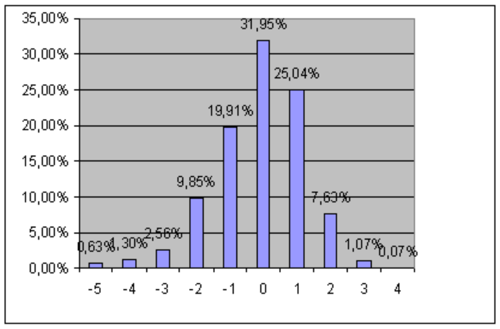

Figure 1: Discrepancy distribution for duration 12 and interest rate 1% per annum.

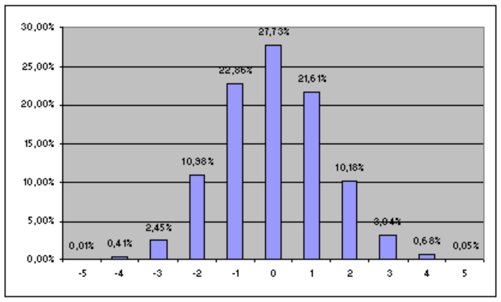

Figure 2: Discrepancy distribution for duration 12 and interest rate 10% per annum.

It is recognizable that discrepancy distribution is normal-like and for not long term loans its volume is limited.

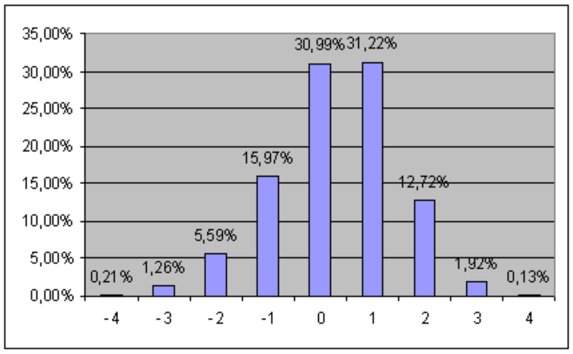

Figure 3: Discrepancy distribution for duration 48 and interest rate 1% per annum.

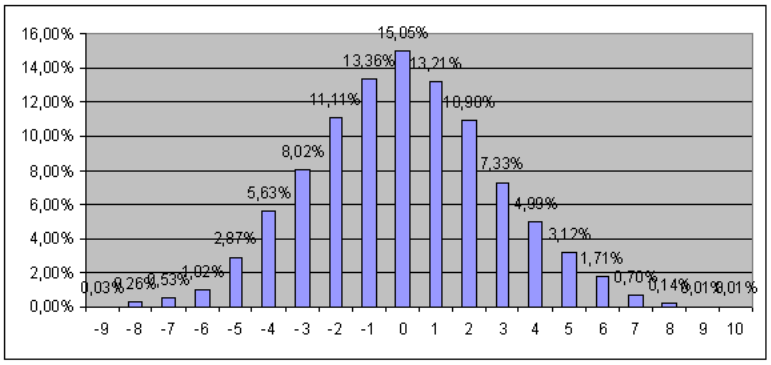

Figure 4: Discrepancy distribution for duration 48 and interest rate 10% per annum.

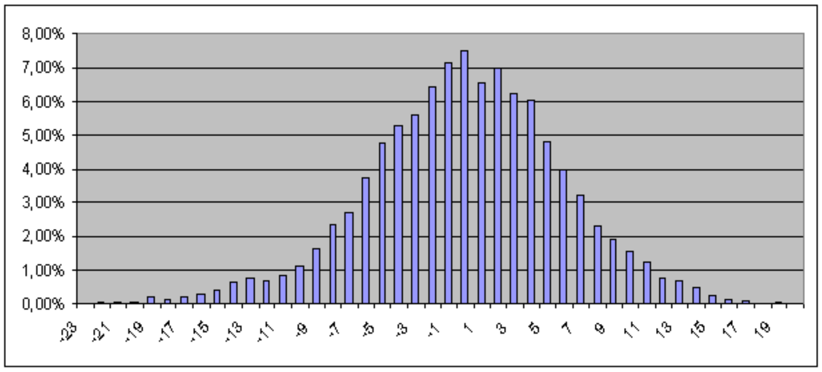

Figure 5: Discrepancy distribution for duration 120 and interest rate 10% per annum.

But it also becomes exploding when duration and interest rate increase. What can we expect for a loan with daily interest calculation over 10 years with 10% per annum?

Distribution of the discrepancy is not normal anymore, but volume of discrepancy is still limited. All in all the situation is not satisfactory, so we continue looking for a better solution.

3.4 Attempt 4

Let's try something very simple. Let assume we round numbers in the amortization schedule to amounts **after** we calculate full amortization schedule. We get following amortization schedule then:

```
87,72   79,74   7,97    920,26
87,72   80,38   7,34    839,88
87,72   81,02   6,70    758,86
87,72   81,66   6,05    677,20
87,72   82,32   5,40    594,88
87,72   82,97   4,74    511,91
87,72   83,63   4,08    428,28
87,72   84,30   3,42    343,98
87,72   84,97   2,74    259,01
87,72   85,65   2,07    173,35
87,72   86,33   1,38     87,02
87,72   87,02   0,69      0,00
```

This is usual approach of many loan calculators that can be found in internet. Its biggest advantage is that it amortizes principal nicely to zero.

But: With this approach following properties are not always satisfied:

- `instalment amount = repayment + interest paid` (ex. first line)
- `principal before instalment = principal after instalment + repayment` (ex. lines 9 and 10)

These properties haven't been mentioned explicit, but they constitute core of amortization schedule. If they are not satisfied, basic accounting and arithmetic on accounts cannot be ensured. Hence we won't come back to this attempt.

3.5 Proper solution of loan amortization

Before we explain how it works, let list open issues of attempt 3:

- There is still some discrepancy in the amortization table, which is annoying due to the fact, that it cannot be properly limited. In consequence one cannot build properties in order to automate validation of amortization schedule.

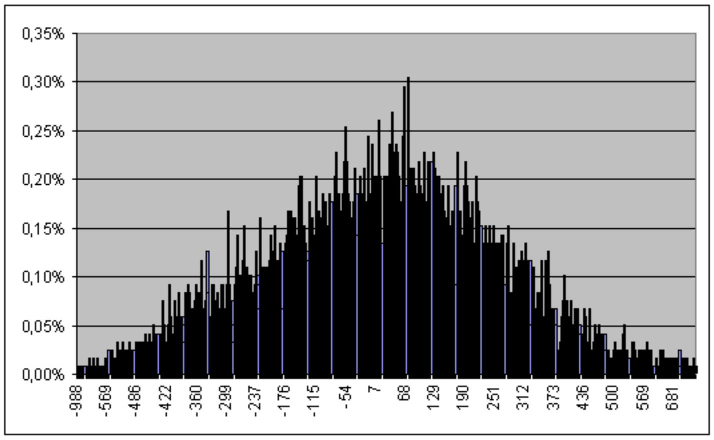

Figure 6: Discrepancy distribution for duration 360 and interest rate 20% per annum.

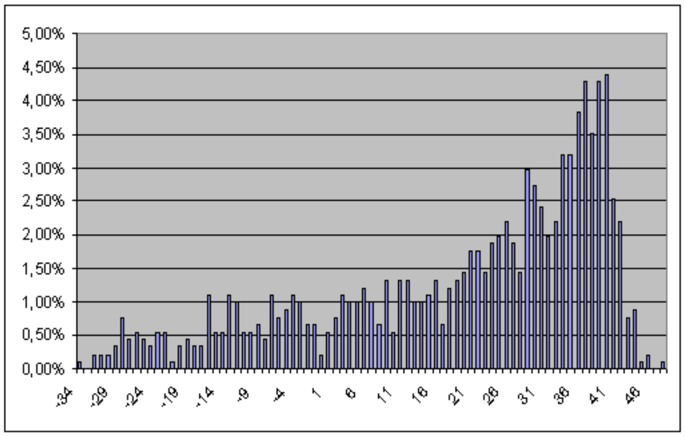

Figure 7: Discrepancy distribution for duration 3600, daily interest calculation and interest rate 10% per anno.

- In some cases loan cannot be amortized at all. These cases are probably beyond real life limitations, still we do not like solutions which are awkward.

Like previously mentioned discrepancy we observe originates in rounding. On the other hand we have to round interest paid due to requirements of accounting. What if we split interest on interest calculated (real number) and interest paid (amount data type)? Could we take difference for calculation of next instalment and possibly get rid of discrepancies?

Before we follow this path, we want to have short look on second open issue. Why attempt 3 doesn't amortize loan for some input? This is again due to interest rounding. When rounded interest equals instalment amount we cannot repay principal. Having interest calculated and not rounded can we possibly fix also this issue?

We define amortization calculation like follows: We **truncate** or **round** instalment amount (important is that instalment amount is of Amount data type), we recalculate interest rate so it fits to truncated (or rounded) instalment amounts, we split interest on interest paid and interest calculated, we keep difference between both for next instalment calculation purpose. Here more formal definition:

Let

- a — instalment amount – already rounded or truncated to expected format,
- r — recalculated interest rate,
- p_i — principal after i-th instalment, p_0 — initial principal (borrowed amount),
- b_i — calculated interest of i-th instalment,
- c_i — paid interest of i-th instalment,
- d_i — calculated and unpaid part of interest due to rounding – of i-th instalment ($d_0 = 0$).
- e_i — repayment of i-th instalment.

Then we define for each $i \in \{1, ..., n\}$, where n is number of instalments:

$$\begin{aligned}
b_i &= (p_{i-1} + d_{i-1})r \\
c_i &= [b_i + d_{i-1}] \\
d_i &= b_i - c_i + d_{i-1} \\
e_i &= a - c_i \\
p_i &= p_{i-1} - e_i
\end{aligned}$$

Following extract from spread sheet provides another view of the same definition:

	A	B	C	D	E	F
3	Instalment	Repayment	Interest	Interest paid	Interest	Principal
4			calculated		rounding	
5					difference	
6					0	Initial Principal
7	truncated instalment	=A7-D7	=(E6+F6)*C$3	=round(C7+E6;2)	=C7+E6-D7	=F6-B7
8	truncated instalment	=A8-D8	=(E7+F7)*C$3	=round(C8+E7;2)	=C8+E7-D8	=F7-B8
...						

Our example loan will amortize as follows:

```
Instalment  Repayment  Interest             Difference Principal
    amount             calculated paid    of int.         after
                                                     instalment
     87.71      79.75  7.964192884  7.96   0.00419288    920.25
     87.71      80.38  7.329081894  7.33   0.00327478    839.87
     87.71      81.02  6.688912758  6.69   0.00218754    758.85
     87.71      81.66  6.043645192  6.05  -0.00416727    677.19
     87.71      82.32  5.393238590  5.39  -0.00092868    594.87
     87.71      82.97  4.737652024  4.74  -0.00327666    511.90
     87.71      83.64  4.076844241  4.07   0.00356758    428.26
     87.71      84.30  3.410773657  3.41   0.00434124    343.96
     87.71      84.97  2.739398359  2.74   0.00373960    258.99
     87.71      85.64  2.062676098  2.07  -0.00358430    173.35
     87.71      86.33  1.380564290  1.38  -0.00302001     87.02
     87.71      87.02  0.693020013  0.69   0.00000000      0.00
```

This example allows us to hope, that we are on proper way. How to proof then that found method is correct one, i.e. that it amortizes loans properly?

3.5.1 Proof of correctness

Notice that we take raw model then round instalment and recalculate interest, then it will still amortize loan properly. This is due to fact, that interest rate has been re-calculated after instalment got rounded. In order to proof the thesis, we proof firstly following lemma.

Lemma Let p'_i be principal after i-th instalment in the raw amortization with rounded instalment amount and recalculated interest rate. Then for any $i \in \{1, ..., n\}$ $p_i + d_i = p'_i$.

Proof of lemma Let e'_i be repayment, b'_i — calculated interest, both of of i-th instalment in the raw amortization with rounded instalment amount. We notice that $p_0 = p'_0$. We also remind that:

$$\begin{aligned} b'_i &= p'_{i-1} r \\ e'_i &= a - b'_i \\ p'_i &= p'_{i-1} - e'_i \end{aligned}$$

We use mathematical induction. For $i = 1$ we get from raw amortization:

$$p'_1 = p'_0 - a + b'_1 = p'_0 - a + p'_0 r = p'_0(1 + r) - a$$

and from thesis:

$$\begin{aligned} p_1 = p_0 - a + c_1 = p_0 - a + [b_1 + d_0] = p_0 - a + [p_0 r] \\ d_1 = p_0 r - [p_0 r] \end{aligned}$$

hence

$$p_1 + d_1 = p_0 - a + [p_0 r] + p_0 r - [p_0 r] = p_0(1 + r) - a$$

This proves lemma for $i = 1$.
Now we assume that $p'_i = p_i + d_i$ for any $i > 1$.
We are supposed to prove that $p'_{i+1} = p_{i+1} + d_{i+1}$. We calculate then:

$$\begin{array}{rrcl}
 & p'_{i+1} & = & p_{i+1} + d_{i+1} \\
\Leftrightarrow & p'_i - e'_{i+1} & = & p_i - e_{i+1} + d_{i+1} \\
\Leftrightarrow & p'_i - a + b'_{i+1} & = & p_i - a + c_{i+1} + b_{i+1} - c_{i+1} + d_i \\
\Leftrightarrow & p'_i + p'_i r & = & p_i + (p_i + d_i)r + d_i \\
\Leftrightarrow & p'_i(1 + r) & = & p_i(1 + r) + d_i(r + 1) \\
\Leftrightarrow & p'_i & = & p_i + d_i
\end{array}$$

What proves the thesis.

Having proved lemma and knowing that raw model amortizes, we conclude that in the last instalment ($i = n$) we have: $p'_n = 0$ and $p'_n = p_n + d_n$. Hence $p_n = -d_n$. Now we notice that p_n is rounded value and d_n is remainder of rounding: the only value which allows fulfilling this equality is 0.

4 Data types

Having theory done, we know how to amortize loans. So we switch to library design. First we define some simply data types, then we present properties.

4.1 Basic data types

```
type Amount    = Integer
type Interest  = Double
type Rate      = Double
type Duration  = Int
```

Alternatively would be possible to have `type Amount = Int` but then we'd have to limit and validate size of amount.

4.2 Instalment

We define then `Instalment`:

```
data Instalment = I { iAmt       :: Amount   -- ^ instalment amount
                    , iRepayment :: Amount   -- ^ repayment
                    , iInterest  :: Interest -- ^ interest calculated over the full period
                    , iIntPaid   :: Amount   -- ^ interest paid with instalment
                    }
```

where

- `iInterest` – interest calculated for period since last instalment. This is not necessarily interest which is paid within instalment.
- `iIntPaid` – interest paid with instalment. Difference between both is deferred or late interest to be paid in the next instalment and is base for interest calculation.
- Following condition is always satisfied:

$$\texttt{iAmt} = \texttt{iRepayment} + \texttt{iIntPaid} \tag{2}$$

- Instalment has to pay off firstly interest of the period, then deferred interest and then principal.

4.2.1 Instalment validation

We can then check whether instalment satisfies given condition.

```
instalmentCheckM :: Instalment -> ValidMonad ()
instalmentCheckM i | iAmt i == iRepayment i + iIntPaid i = return ()
                   | otherwise                           = throwError $ InstalmentDiscrepancy
                                                                        (iAmt i)
                                                                        (iRepayment i)
                                                                        (iIntPaid i)
```

where

```
type ValidMonad = Either ValidationError
data ValidationError =
     -- | Instalment discrepancy
     InstalmentDiscrepancy Amount      -- ^ Instalment amount
                           Amount      -- ^ Repayment amount
                           Amount      -- ^ Interest paid
```

4.3 Instalment Plan

Instalment Plan is a general, "unscheduled" amortization schedule – unscheduled, because without dates. An amortization schedule is a list of amortization schedule lines. Consequently instalment plan is a list of instalment plan lines. We define then:

```
data InstalmentPlanLine = IPL { iplInst      :: Instalment  -- ^ 'Instalment' itself
                              , iplPrincipal :: Amount      -- ^ Principal after
                              , iplIntLate   :: Interest    -- ^ Sum of not paid late interest
                                                            --   after current instalment
                              , iplRate      :: Rate        -- ^ Nominal interest rate
                              }
```

For single instalment plan line the only validation is one of `Instalment`. We define additional controls for each two consecutive instalment plan lines:

- Interest calculation (interest amount is `round`ed calculated interest):

```
checkIntCalcul ipl1 ipl2 =
    abs ((fromIntegral (iplPrincipal ipl1 + iplIntLate ipl1) *
           iplRate ipl)
    - (iInterest . iplInst) ipl) < 1e-4
```

- Deferred interest continuation:

```
checkLateInterest ipl1 ipl2 =
   abs (iplIntLate ipl1 - (iIntLate . iplInst) ipl2
                        - iplIntLate ipl2) < 1e-2
```

- Principal after instalment:

```
checkPrincipalAfter ipl1 ipl2 = iplPrincipal ipl1 - (iRepayment . iplInst) ipl2
                              == iplPrincipal ipl2
```

One can notice that we have to bear in mind rounding errors and construct properties accordingly.

Now we can wrap them in into `ValidMonad`:

- Interest calculation:

```
checkIntCalculM ipl1 ipl2
    | checkIntCalcul ipl1 ipl2 = return ipl2
    | otherwise                = throwError $ IPLInterest ...
```

- Deferred interest continuation:

```
checkLateInterestM ipl1 ipl2
    | checkLateInterest ipl1 ipl2 = return ipl2
    | otherwise                   = throwError $ FollowUpError ...
```

- Principal after instalment:

```
checkPrincipalAfterM ipl1 ipl2
    | checkPrincipalAfter ipl1 ipl2 = return ipl2
    | otherwise                     = throwError $ FollowUpError ...
```

and chain:

```
instalmentPlanLineCheckM :: InstalmentPlanLine
                         -> InstalmentPlanLine
                         -> ValidMonad InstalmentPlanLine
instalmentPlanLineCheckM ipl1 ipl2 =
    (instalmentCheckM . iplInst) ipl2 >>    -- Instalment validation
    checkIntCalculM ipl1 ipl2 >>            -- Interest calculation validation
    checkLateInterestM ipl1 ipl2 >>         -- Deferred interest amount check
    checkPrincipalAfterM ipl1 ipl2          -- Principal after instalment check
```

Finally we validate the whole Instalment Plan which is list of `InstalmentPlanLine`:

```
foldM instalmentPlanLineCheckM initIPL ip
```

where

- `ip` — `type InstalmentPlan = [InstalmentPlanLine]`
- `initIPL = (IPL (I 0 0 0 0) cap iL 0)`
- `cap` — initial principal
- `iL` — initial deferred interest. We can assume at this point this is always 0.

5 `InstalmentPlan` constructors

Now we can define some handful of constructors. We want them to be a type of `a -> ValidMonad InstalmentPlan`. All constructors assume that:

- Instalments are always of the same frequency. This frequency is defined explicit when necessary.
- First instalment lasts exactly same time like any other – one period of frequency.
- Any gaps between instalments as well as deferment of the first instalment are represented by instalment of amount 0.

5.1 `newLoanR`

To wrm up we will construct an `InstalmentPlan` having following information:

- Principal
- List of instalment amounts
- Optionally initial deferred interest. This can be useful for features we'll describe later.

We can define then:

```
newLoanRIL  :: Amount     -- ^ late interest
                          --   amount of interest to be taken before principal.
            -> Amount     -- ^ principal
            -> [Amount]   -- ^ list of instalment amounts
            -> ValidMonad InstalmentPlan
newLoanRIL iL c is = liftM (newLoanR' c is iL) $ rateIrr is (c+iL)
    where newLoanR' _ [] _ _       = []
          newLoanR' c (i:is) iL' r = newIPL
                                     : newLoanR' (iplPrincipal newIPL) is (iplIntLate newIPL) r
              where newIPL = newInstalmentPlanLine c iL' r i
```

and further

```
newInstalmentPlanLine :: Amount              -- ^ capital before excluded late interest
                      -> Interest            -- ^ total late interest before instalment
                      -> Rate                -- ^ rate in frequency units
                      -> Amount              -- ^ instalment amount
                      -> InstalmentPlanLine  -- ^ instalment plan one row
newInstalmentPlanLine c iL r i = IPL inst
                                     (c - iRepayment inst)
                                     (iL + iInterest inst         -- late interest can be negative
                                         - fromIntegral (iIntPaid inst))
                                     r
    where inst = newInstalment c r i iL
```

and one step more

```
-- | Instalment details have some assumptions:
--   late interest accrue interest
--   instalment is always in full instalment duration
--   there are no gaps between instalments, though instalment amount can be = 0
newInstalment :: Amount        -- ^ capital before instalment (excluded late interest)
              -> Rate          -- ^ rate in frequency unit
              -> Amount        -- ^ instalment amount; when < 0 will be forced to interest + late inter
              -> Interest      -- ^ late interest to be paid by the instalment
              -> Instalment    -- ^ full details for instalment
newInstalment c r i lateInterest
   | i < 0 = I interestPaid'
               0
               interest
               interestPaid'
   | otherwise = I i
                   capitalPaid
                   interest
                   interestPaid
    where interest = (fromIntegral c + lateInterest) * r
          interestPaid' = round $ interest + lateInterest
          interestPaid = min i $ round $ interest + lateInterest
          capitalPaid = i - interestPaid
```

At this point it is also necessary to explain, that `rateIrr` implements the Newton-Raphson method and is of following type:

```
rateIrr :: [Amount]         -- ^ list of instalment amounts
        -> Amount           -- ^ Principal
        -> ValidMonad Rate
```

Now we define one more convenient constructor: `newLoanR = newLoanRIL 0`.

5.2 `newLoanI`

Next constructor we define is more type of loan specific. There are many types of loans. We'll define some of them but we want also to leave possibility for further definition of loan types on any stage. Therefore we define a type class. Before we do this we spend some time on boiler plating. New constructor requires two parameters which usually are defined once and don't change too often. They are `RoundingType` and `Freq`. In order to not feed them every time to the function separately we define:

```
data InstalmentPlanParam = IPP
     {freq        :: Freq
     ,rounding    :: RoundingType
     }

paramMT = IPP Monthly Truncated
```

then we make use of *Reader* monad:

```
type IPPMonad = ReaderT InstalmentPlanParam ValidMonad

runWithIPP param loan = runReaderT loan param
```

finally we can define our type class:

```
class ClassicLoan a where
      newLoanI :: a              -- ^ type of loan
               -> IPPMonad InstalmentPlan
      extract :: a -> InstalmentLoanData

-- | Data type for retriving loan data of particular products.
data InstalmentLoanData = InstalmentLoanData {principal  :: Amount
                                             ,duration   :: Duration
                                             ,deferment  :: Duration
                                             ,rate       :: Rate
                                             }
```

Now we can define some loan types and their constructors. Please note that interest rate although given, will be recalculated. Due to instalment amount truncation or rounding recalculated interest rate will be usually not equal to the given one.

5.3 Classical

The very basic loan type with all instalments equal and interest rate constant over whole its duration.

```
data Classical = Classical !Amount        -- ^ principal
                           !Duration      -- ^ duration in freq units
                           !Duration      -- ^ 1st instalment postponement in duration units
                           !Rate          -- ^ yearly effective interest rate

instance ClassicLoan Classical where
    newLoanI (Classical c n d rE) =
        ask >>= \param ->
        let   r = cE2N (freq param) rE
```

```
              i = myRound (rounding param) $ rawCalcInstCl (fromIntegral c) n r d
        in lift $ newLoanR c $ replicate d 0 ++ replicate n i

    extract (Classical p n d r) = InstalmentLoanData p n d r

cE2N fr r = (1+r)**(1/n) - 1
     where n = freqPerYear fr

freqPerYear Daily   = 365
freqPerYear Monthly = 12
freqPerYear Yearly  = 1
```

where `myRound` rounds or truncates and

$$rawCalcInstCl(C,n,r,d) = \begin{cases} 0 & \text{if } n = 0; \\ C/n & \text{if } n \neq 0 \text{ and } r = 0; \\ Cr\frac{(r+1)^{n+d}}{(r+1)^n - 1} & \text{otherwise.} \end{cases} \qquad (3)$$

All examples below should be run then with:

```
runWithParam paramMT $ <here the example>
```

Example of `InstalmentPlan` of Classical loan with following input:

```
newLoanI $ Classical 100000 6 1 0.1

 Instalment  Repayment Interest   Interest            Principal  Interest              Monthly interest
 amount                    paid   calculated              after  deferred              rate

(      0.00=      0.00+    0.00; 7.967105604980594)     1000.00  7.967105604980594     7.967105604980594e-3
(    172.71=    156.71+   16.00; 8.030580376701508)      843.29 -2.314018317897535e-3  7.967105604980594e-3
(    172.71=    165.99+    6.72; 6.718562049595774)      677.30 -3.751968722123138e-3  7.967105604980594e-3
(    172.71=    167.32+    5.39; 5.39609073392232)       509.98  2.338765200197486e-3  7.967105604980594e-3
(    172.71=    168.64+    4.07; 4.063083149617339)      341.34 -4.578085182463951e-3  7.967105604980594e-3
(    172.71=    170.00+    2.71; 2.719455353115959)      171.34  4.8772679334950905e-3 7.967105604980594e-3
(    172.71=    171.34+    1.37; 1.3651227320660646)       0.00 -4.402522790769581e-13 7.967105604980594e-3
```

At the end of amortization principal equals zero and deferred interest differs from zero by rounding error.

5.4 Balloon

Loan type of all instalments equal but last one (which is given), with interest rate constant over whole its duration.

```
data Balloon = Balloon !Amount        -- ^ principal
                       !Duration      -- ^ duration in freq units
                       !Duration      -- ^ 1st instalment postponement in duration units
                       !Rate          -- ^ yearly effective interest rate
                       !Amount        -- ^ balloon amount

instance ClassicLoan Balloon where
```

```
    newLoanI (Balloon c n d rE b) =
        ask >>= \param ->
        let   r = cE2N (freq param) rE
              i = myRound (rounding param) $ rawCalcInstBal (fromIntegral c) n r d (fromIntegral b)
        in lift $ newLoanR c $ replicate d 0 ++ replicate (n-1) i ++ [b]

    extract (Balloon p n d r _) = InstalmentLoanData p n d r
```

where

$$rawCalcInstBal(C,n,r,d,b) = \begin{cases} 0 & \text{if } n = 0; \\ \frac{C-b}{n-1} & \text{if } n \neq 0 \text{ and } r = 0; \\ r\frac{C(r+1)^{n+d-1} - \frac{b}{r+1}}{(r+1)^{n-1}-1} & \text{otherwise.} \end{cases} \tag{4}$$

Example of `InstalmentPlan` of Balloon loan:

```
newLoanI $ Balloon 100000 6 1 0.1 50000

 Instalment  Repayment Interest   Interest            Principal  Interest             Monthly interest
 amount                    paid   calculated              after  deferred             rate

(      0.00=      0.00+    0.00; 7.970756081132196)    1000.00  7.970756081132196    7.970756081132195e-3
(    108.80=     92.79+   16.01; 8.034289033637101)     907.21 -4.954885230704349e-3  7.970756081132195e-3
(    108.80=    101.57+    7.23; 7.231110130182356)     805.64 -3.844755048348816e-3  7.970756081132195e-3
(    108.80=    102.38+    6.42; 6.42152928359866)      703.26 -2.3154714496888573e-3 7.970756081132195e-3
(    108.80=    103.20+    5.60; 5.60549546555889)      600.06  3.179994109200379e-3  7.970756081132195e-3
(    108.80=    104.01+    4.79; 4.7829572410015695)    496.05 -3.862764889230448e-3  7.970756081132195e-3
(    500.00=    496.05+    3.95; 3.953862764888895)       0.00 -3.353761712787673e-13 7.970756081132195e-3
```

5.5 Balloon Plus

Loan type of all instalments but last equal, with interest rate constant over whole its duration. When all instalments equal i then the last one equals $i + b$ where b is given.

```
data BalloonPlus = BalloonPlus !Amount        -- ^ principal
                               !Duration      -- ^ duration in freq units
                               !Duration      -- ^ 1st instalment postponement in duration units
                               !Rate          -- ^ yearly effective interest rate
                               !Amount        -- ^ balloon amount

instance ClassicLoan BalloonPlus where

    newLoanI (BalloonPlus c n d rE b) =
        ask >>= \param ->
        let   r = cE2N (freq param) rE
              i = myRound (rounding param) $ rawCalcInstBalPlus (fromIntegral c) n r d
                                                                     (fromIntegral b)
        in lift $ newLoanR c $ replicate d 0 ++ replicate (n-1) i  ++ [b + i]

    extract (BalloonPlus p n d r _) = InstalmentLoanData p n d r
```

where

$$rawCalcInstBalPlus(C,n,r,d,b) = \begin{cases} 0 & \text{if } n = 0; \\ \frac{C-b}{n} & \text{if } n \neq 0 \text{ and } r = 0; \\ r\frac{C(r+1)^{n+d}-b}{(r+1)^n-1} & \text{otherwise.} \end{cases} \quad (5)$$

Example of `InstalmentPlan` of Balloon Plus loan:

```
newLoanI $ BalloonPlus 100000 6 1 0.1 50000

 Instalment  Repayment Interest   Interest             Principal  Interest                Monthly interest
 amount                    paid   calculated               after  deferred                rate

(      0.00=      0.00+    0.00; 7.965958832601708)     1000.00  7.965958832601708       7.965958832601708e-3
(     91.02=     75.02+   16.00; 8.029415332724414)      924.98 -4.625834673879581e-3   7.965958832601708e-3
(     91.02=     83.66+    7.36; 7.368315751771349)      841.32  3.6899170974697883e-3  7.965958832601708e-3
(     91.02=     84.31+    6.71; 6.701949878772163)      757.01 -4.3602041303665824e-3  7.965958832601708e-3
(     91.02=     84.99+    6.03; 6.030275762661216)      672.02 -4.084441469151443e-3   7.965958832601708e-3
(     91.02=     85.67+    5.35; 5.353251118192403)      586.35 -8.333232767483878e-4   7.965958832601708e-3
(    591.02=    586.35+    4.67; 4.670833323277095)        0.00  3.4674485505092887e-13 7.965958832601708e-3
```

5.6 Unfolded Balloon

Loan type based on Balloon type. It differs from Balloon that is unfolds balloon instalment so that all instalments are equal, except the last one which is less or equal.

```
data UnfdBalloon = UnfdBalloon
                   !Amount     -- principal
                   !Duration   -- duration till balloon (inclusive) in freq units
                   !Duration   -- 1st instalment postponement in duration units
                   !Rate       -- yearly effective interest rate
                   !Amount     -- residual balloon amount (RBA).
                   !Duration   -- maximal duration of unfolded RBA

instance ClassicLoan UnfdBalloon where
    newLoanI (UnfdBalloon c n d rE b x) =
        ask >>= \param ->
        let r = cE2N (freq param) rE
            i = myRound (rounding param) $ rawCalcInstBal (fromIntegral c) n r d (fromIntegral b)
            durOfBal :: Integer
            durOfBal  = calcDurCl (fromIntegral $ calcCapBeforeBal b r) (fromIntegral i) r 0 truncate
            durOfBal' :: Int
            durOfBal' = fromIntegral durOfBal
            lastInst = round $ rawCalcInstCl (fromIntegral $ capBeforeLast) 1 r 0
            capBeforeLast = calcCapAfterN c i (n - 1 + durOfBal') d r
            unfoldedBalloon | durOfBal > fromIntegral x = replicate x newI
                            | otherwise                 = replicate durOfBal' i ++ [lastInst]
                where newI = myRound (rounding param) $
                             rawCalcInstCl (fromIntegral $ calcCapBeforeBal b r) x r 0
        in lift $ newLoanR c $ replicate d 0 ++ replicate (n-1) i ++ unfoldedBalloon

    extract (UnfdBalloon p n d r _ _) = InstalmentLoanData p n d r
```

where

$$calcDurCl(C,i,r,d) = \begin{cases} 0 & \text{if } i = 0; \\ \lfloor \frac{C}{i} \rfloor & \text{if } i \neq 0 \text{ and } r = 0; \\ \lfloor \frac{\log \frac{i}{Cr(1+r)^d + i}}{\log r+1} \rfloor & \text{otherwise.} \end{cases} \tag{6}$$

$$calcCapAfterN(C,i,n,d,r) = \begin{cases} C - i \times n & \text{if } r = 0; \\ \lfloor C(1+r)^{n+d} - \frac{i((1+r)^n - 1}{r} \rfloor & \text{if } r \neq 0. \end{cases} \tag{7}$$

Examples of `InstalmentPlan` of UnfdBalloon loan with following input:

```
newLoanI $ UnfdBalloon 100000 6 1 0.1 50000 6

 Instalment  Repayment Interest   Interest             Principal  Interest                Monthly interest
 amount                    paid   calculated               after  deferred                rate

(      0.00=      0.00+    0.00; 7.973470083063039)     1000.00  7.973470083063039     7.973470083063039e-3
(    108.80=     92.79+   16.01; 8.03704630822854)       907.21  5.163912915804759e-4  7.973470083063039e-3
(    108.80=    101.57+    7.23; 7.233615911486135)      805.64  4.13230277771504e-3   7.973470083063039e-3
(    108.80=    102.37+    6.43; 6.423779386511479)      703.27 -2.088310710805672e-3  7.973470083063039e-3
(    108.80=    103.19+    5.61; 5.607485654232766)      600.08 -4.60265647803908e-3   7.973470083063039e-3
(    108.80=    104.02+    4.78; 4.7846832283007386)     496.06  8.05718226990848e-5   7.973470083063039e-3
(    108.80=    104.84+    3.96; 3.955320211841269)      391.22 -4.599216336031873e-3  7.973470083063039e-3
(    108.80=    105.69+    3.11; 3.119344294182061)      285.53  4.745077846029062e-3  7.973470083063039e-3
(    108.80=    106.52+    2.28; 2.2767027475532364)     179.01  1.4478253992655254e-3 7.973470083063039e-3
(    108.80=    107.37+    1.43; 1.4273424237616212)      71.64 -1.209750839113326e-3  7.973470083063039e-3
(     72.21=     71.64+    0.57; 0.5712097508385124)       0.00 -6.008349373587407e-13 7.973470083063039e-3


newLoanI $ UnfdBalloon 100000 6 1 0.1 50000 3

(      0.00=      0.00+    0.00; 7.96998569690742)     1000.00  7.96998569690742       7.96998569690742e-3
(    108.80=     92.80+   16.00; 8.033506368916328)      907.20  3.4920658237479075e-3  7.96998569690742e-3
(    108.80=    101.57+    7.23; 7.230398855949079)      805.63  3.890921772826914e-3   7.96998569690742e-3
(    108.80=    102.38+    6.42; 6.420890587590402)      703.25  4.781509363228906e-3   7.96998569690742e-3
(    108.80=    103.19+    5.61; 5.604930549911378)      600.06 -2.8794072539312766e-4  7.96998569690742e-3
(    108.80=    104.02+    4.78; 4.782467322402804)      496.04  2.179381677410106e-3   7.96998569690742e-3
(    167.99=    164.03+    3.96; 3.9534490747347535)     332.01 -4.3715435878363e-3     7.96998569690742e-3
(    167.99=    165.35+    2.64; 2.646080110090364)      166.66  1.7085665025274466e-3  7.96998569690742e-3
(    167.99=    166.66+    1.33; 1.3282914334971778)       0.00 -2.9473312679328953e-13 7.96998569690742e-3
```

5.7 Unfolded Balloon Plus

Loan type based on Balloon Plus type. It differs from Balloon Plus that is unfolds balloon instalment so that all instalments are equal, except the last one which is less or equal.

```
data UnfdBalloonPlus = UnfdBalloonPlus
                        !Amount       -- principal
                        !Duration     -- duration till balloon (inclusive) in freq units
                        !Duration     -- 1st instalment postponement in duration units
                        !Rate         -- yearly effective interest rate
                        !Amount       -- residual balloon amount (RBA).
                        !Duration     -- maximal duration of unfolded RBA
```

```
instance ClassicLoan UnfdBalloonPlus where
    newLoanI (UnfdBalloonPlus c n d rE b x) =
        ask >>= \param ->
        let   r = cE2N (freq param) rE
              i = myRound (rounding param) $
                  rawCalcInstBalPlus (fromIntegral c) n r d (fromIntegral b)
              durOfBal :: Integer
              durOfBal  = calcDurCl (fromIntegral $ calcCapBeforeBal b r)
                                    (fromIntegral  i) r 0 truncate
              durOfBal' :: Int
              durOfBal' = fromIntegral durOfBal
              lastInst = round $ rawCalcInstCl (fromIntegral $ capBeforeLast) 1 r 0
              capBeforeLast = calcCapAfterN c i (n +  durOfBal') d r
              unfoldedBalloon | durOfBal > fromIntegral x = replicate x newI
                              | otherwise                 = replicate durOfBal' i ++ [lastInst]
                  where newI = myRound (rounding param) $
                               rawCalcInstCl (fromIntegral $ calcCapBeforeBal b r) x r 0
        in lift $ newLoanR c $ replicate d 0 ++ replicate n i ++ unfoldedBalloon

    extract (UnfdBalloonPlus p n d r _ _) = InstalmentLoanData p n d r
```

Example of `InstalmentPlan` of Unfolded Balloon Plus loan:

```
newLoanI $ UnfdBalloonPlus 100000 6 1 0.1 50000 6

 Instalment  Repayment Interest   Interest           Principal  Interest                Monthly interest
 amount                    paid   calculated             after  deferred                rate

(      0.00=      0.00+    0.00; 7.9739003626848435)    1000.00  7.9739003626848435      7.973900362684843e-3
(     91.02=     75.01+   16.01; 8.037483449678868)      924.99  1.3838123637106036e-3  7.973900362684843e-3
(     91.02=     83.64+    7.38; 7.3757891308617625)     841.35 -2.8270567745266816e-3  7.973900362684843e-3
(     91.02=     84.31+    6.71; 6.708818527475853)      757.04 -4.008529298673693e-3   7.973900362684843e-3
(     91.02=     84.99+    6.03; 6.036529566953704)      672.05  2.5210376550307955e-3  7.973900362684843e-3
(     91.02=     85.66+    5.36; 5.358879841245421)      586.39  1.4008789004515165e-3  7.973900362684843e-3
(     91.02=     86.34+    4.68; 4.675826604143538)      500.05 -2.7725169560108044e-3  7.973900362684843e-3
(     91.02=     87.04+    3.98; 3.9873267685865943)     413.01  4.554251630583508e-3   7.973900362684843e-3
(     91.02=     87.72+    3.30; 3.293336903941196)      325.29 -2.108844428220209e-3   7.973900362684843e-3
(     91.02=     88.43+    2.59; 2.5938132332624013)     236.86  1.7043888341811452e-3  7.973900362684843e-3
(     91.02=     89.13+    1.89; 1.8887116305322749)     147.73  4.160193664560552e-4   7.973900362684843e-3
(     91.02=     89.84+    1.18; 1.177987617876409)       57.89 -1.5963627571349548e-3  7.973900362684843e-3
(     58.35=     57.89+    0.46; 0.4615963627582575)       0.00  1.1225154139538062e-12 7.973900362684843e-3
```

5.8 Reversal Balloon

Loan type Balloon like. It differs from Balloon that its regular instalment is given and balloon amount is to be calculated.

```
data ReversBalloon = ReversBalloon !Amount        -- ^ principal
                                   !Duration      -- ^ duration till balloon (inclusive) in freq units
                                   !Duration      -- ^ 1st instalment postponement in duration units
                                   !Rate          -- ^ yearly effective interest rate
                                   !Amount        -- ^ regular instalment amount.
```

```
instance ClassicLoan ReversBalloon where
    newLoanI (ReversBalloon c n d rE i) =
        ask >>= \param ->
        let   r = cE2N (freq param) rE
              b = myRound (rounding param) $
                  rawCalcBalBal (fromIntegral c) n r d (fromIntegral i)
        in lift $ newLoanR c (replicate d 0 ++ replicate (n-1) i ++ [b])

    extract (ReversBalloon p n d r _) = InstalmentLoanData p n d r
```

where

$$rawCalcBalBal(C,n,r,d,i) = \begin{cases} C - i(n-1) & \text{if } r = 0; \\ C(r+1)^d - i\frac{((r+1)^{n-1}-1)(r+1)}{r} & \text{if } r \neq 0. \end{cases} \quad (8)$$

Example of `InstalmentPlan` of Reversal Balloon loan:

```
newLoanI $ ReversBalloon 100000 6 1 0.1 5000

 Instalment  Repayment Interest   Interest            Principal  Interest                 Monthly interest
 amount                    paid   calculated              after  deferred                 rate

(      0.00=      0.00+    0.00; 7.972981062269007)     1000.00  7.972981062269007       7.972981062269006e-3
(     50.00=     33.99+   16.01; 8.036549489288307)      966.01 -4.69448442686371e-4     7.972981062269006e-3
(     50.00=     42.30+    7.70; 7.701975693058939)      923.71  1.5062446162528431e-3  7.972981062269006e-3
(     50.00=     42.63+    7.37; 7.364734346288305)      881.08 -3.759409095441697e-3   7.972981062269006e-3
(     50.00=     42.98+    7.02; 7.024804180646453)      838.10  1.044771551011081e-3   7.972981062269006e-3
(     50.00=     43.32+    6.68; 6.682163758231444)      794.78  3.2085297824551162e-3  7.972981062269006e-3
(    801.12=    794.78+    6.34; 6.336791470217354)        0.00 -1.9099388737231492e-13 7.972981062269006e-3
```

5.9 Bullet

Loan type Balloon like. Its all instalments equal zero except the last one which equals principal and the first one which contains all interest.

```
data Bullet = Bullet !Amount        -- ^ principal
                     !Duration      -- ^ duration till balloon (inclusive) in freq units
                     !Duration      -- ^ 1st instalment postponement in duration units
                     !Rate          -- ^ yearly effective interest rate

instance ClassicLoan Bullet where
    newLoanI (Bullet c n d rE) =
        ask >>= \param ->
        let   r = cE2N (freq param) rE
              fstInst = myRound (rounding param) $
                        rawCalcMaxFstInst (fromIntegral c) n r d
        in lift $ newLoanRIL (fromIntegral fstInst) c $
                  replicate d 0 ++ [fstInst] ++ replicate (n-2) 0 ++ [c]

    extract (Bullet p n d r) = InstalmentLoanData p n d r
```

where

$$rawCalcMaxFstInst(C,n,r,d) = \begin{cases} 0 & \text{if } r = 0; \\ C(r+1)^d \frac{(r+1)^n - 1)}{(r+1)^{n-1}} & \text{if } r \neq 0. \end{cases} \tag{9}$$

Example of `InstalmentPlan` of Flat Balloon loan with following input:

```
newLoanI $ Bullet 100000 6 1 0.1

 Instalment  Repayment Interest   Interest   Principal Interest  Monthly interest
 amount                    paid   calculalted    after deferred  rate

(       0.00=       0.00+    0.00;  0.0)         1000.00 47.28     0.0
(      47.28=       0.00+   47.28;  0.0)         1000.00  0.0      0.0
(       0.00=       0.00+    0.00;  0.0)         1000.00  0.0      0.0
(       0.00=       0.00+    0.00;  0.0)         1000.00  0.0      0.0
(       0.00=       0.00+    0.00;  0.0)         1000.00  0.0      0.0
(       0.00=       0.00+    0.00;  0.0)         1000.00  0.0      0.0
(    1000.00=    1000.00+    0.00;  0.0)            0.00  0.0      0.0
```

On this example one can notice that it is possible to input any interest amount on the loan. This given interest will be kept as deferred interest and go under same rules like deferred interest: they are to be paid off before principal gets paid.

6 Future work

Described library provides only with basic functionality of features. Further work in this scope can contain:

- Financing and amortization schedule instantiated with dates.
- Loan charges.
- Early repayment and partial early repayment.