

# gauge

A mortgage grokking engine

Scott J. Cederberg

CS 395T Final Project  
Prof. Greg Lavender  
Spring 2005

# A Bit About Mortgages

- Basic idea: some initial amount of principal borrowed, at some interest rate.
- Assuming that the loan will be paid off in a given amount of time, and that each payment is for the same amount, calculate the payment amount.
- This calculation is an amortization.

# About Mortgages 2

- Under this fixed-payment scheme, the first few payments are mostly interest while the last few are mostly principal.

# Adjustable Rate Mortgages (ARMs)

- Interest rate is computed by adding a “margin” to a published “index rate” (e.g. T-Bill rate as published in Wall Street Journal plus 3.5%).
- Remaining balance of loan re-amortized over remaining term of loan under new rate.
- 5/1: Five years at initial fixed rate; rate changes every one year thereafter.

# ARMs 2

- There may be caps: rate will not rise more than  $x\%$  each change, or may never exceed  $x\%$ .
- There may also be caps on payments: payments may not rise more than  $x\%$  each time loan is re-amortized.

# ARMs 3

- If payments are capped and rates rise, it may happen that the payments do not rise enough to cover interest.
- A loan that allows this to happen is a “negative amortization” mortgage. You can end up owing more over time, even though you make your payments!

# How to represent this?

- We want to calculate: payments and amount due.
- Clearly, these payments depend on the interest rates over time in the case of an ARM.

```
> data Mortgage = Mortgage {remainder :: Remainder,  
>                             rates :: Rates,  
>                             update :: Remainder -> Rates -> Double ->  
>                                 Rates}  
  
> data Remainder = Remainder {principal :: Double,  
>                             numPeriods :: Int}  
> deriving Show  
  
> data Rates = Rates {apr :: Double, payment :: Double}  
> deriving Show  
  
> type RateSchedule = [Double]
```

# Calculating Payments

- > payments :: Mortgage -> RateSchedule -> [Double]
- > payments (Mortgage (Remainder \_ 0) \_ \_) \_ = []
- > payments m r = pmt:(payments nm nr)
- >     where (pmt, nm, nr) = advance m r

# Calculating Payments

```
> payments :: Mortgage -> RateSchedule -> [Double]
> payments (Mortgage (Remainder _ 0) _ _) _ = []
> payments m r = pmt:(payments nm nr)
>     where (pmt, nm, nr) = advance m r
```

```
> advance :: Mortgage -> RateSchedule ->
>     (Double, Mortgage, RateSchedule)
> advance (Mortgage (Remainder p np) (Rates apr pmt) ufn)
>     (r:rs) =
>     (pmt, (Mortgage (Remainder newp (np-1)) newr ufn), rs)
>     where newp = p*(1+apr/12) - pmt
>           newr = ufn (Remainder newp (np-1))
>                 (Rates apr pmt) r
```

# Fixed-Rate Mortgage

```
> makeFixed :: Double -> Double -> Int -> Mortgage
> makeFixed p apr np =
>   (Mortgage (Remainder p np) (Rates apr pmt) fixedUpdate)
>   where pmt = calcPayment apr p 12 np
>         fixedUpdate _ (Rates a p) _ = Rates a p
```

# Adjustable-Rate Mortgage

```
> makeARM :: Double -> Double -> Int -> Int -> Int -> Double
>           -> Mortgage
> makeARM p iapr np ip ai margin =
>   (Mortgage (Remainder p np) (Rates iapr pmt) armUpdate)
>   where pmt = calcPayment iapr p 12 np
>           armUpdate (Remainder up unp) oldRates indRt =
>             if ((unp <= np-ip) && (unp `mod` ai == 0))
>               then (Rates (indRt+margin)
>                         (calcPayment (indRt+margin)
>                                       up 12 unp))
>             else
>               oldRates
```

# Design Analysis

- We've abstracted the “tricky part” of calculating payments into a single update function.
- Specifying this function amounts to deciphering all of the tricks. (“Tell me the update function.” == “Show me all your tricks.”)

# Example

- Is it worth it to pay “points”?
- You can buy a lower interest rate; the cost of purchasing a lower rate is expressed as a percentage of the principal (2 points = 2% of principal).

# How to avoid “gotchas”

- Now: browse through websites, books, brochures etc. to learn what the possible “gotchas” are. Read through your mortgage and/or question your broker and loan officer to avoid them.
- Can we do better with gauge?

# Future Work

- Better UI (how?)
- Abstracting common parts of update functions into some sort of library.
- Embedding (e.g. into Gnumeric or Gnucash). (Spreadsheet as providing a certain semantic/visualization environment in which we can embed other things.)

# Future Work

- Develop list of scenarios to “vet” possible mortgages (lint for mortgages).