

FunDNA

A Domain Specific Embedded Language
for Functional DNA Programming

Dr. Greg Lavender
Department of Computer Sciences
The University of Texas at Austin

Domain Specific Languages

- A language whose syntax and semantics are tailored to a specific application domain
 - versus a general purpose programming language
 - How do we construct a DSL?
 - just use classes in an OOP language to define new types, overload operators, etc?
 - e.g., for matrix arithmetic
 - embed the DSL into an existing language which has the ability to extend its syntax and semantics?
 - invent a completely new language each time?
 - create “language frameworks” that allow rapid creation of new languages

2

Example DSLs

- Lots of “little languages” are invented all the time for various special purpose tasks:
 - database programming (prolog, sql)
 - network protocols & programming
 - parallel & distributed computing
 - programming embedded devices
 - robotics
 - gaming
 - and many more...

3

DNA Computing

“Molecular biology is a science of complex ideas supported by test tube experiments with molecules. Consequently, the science has remained largely inaccessible to those without a knowledge of chemistry” — Karl Driica in *Understanding DNA and Gene Cloning*

- Encoding computation with DNA
 - exploit biochemical processes
 - massively parallel
 - but hard to control *in vitro* reactions
 - high rates of “error”
 - active area of research
 - see paper by Leonard Adleman
- Can simulate DNA computing *in silico*
 - to explore the idea of domain specific languages
 - to learn more about DNA structure and computation

4

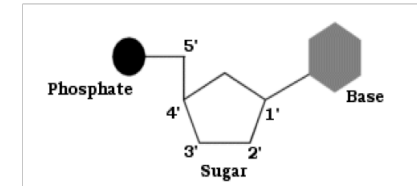
FunDNA

- Functional DNA Programming
 - use an equational language
 - similar to writing chemical equations
 - extended with types and functions for DNA
 - vocabulary familiar to biochemists & biologists
 - capture key DNA structures & processes
 - structures = data types
 - processes = (higher-order) functions
 - exploit natural parallelism

5

DNA Nucleotide Structure

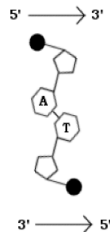
- Base
 - key differentiating component
 - Adenine
 - Guanine
 - Thymine
 - Cytosine
- Deoxyribose Sugar
 - note 5' and 3' "ends"
- Phosphate
 - a 5' phosphate group links to a 3' hydroxyl group in an adjacent nucleotide to form a single-stranded structure



6

Base Pairs

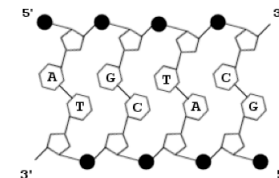
- Weak hydrogen bond between complementary base pairs
 - A<->T
 - C<->G
- Pairing principle is called Watson-Crick Complement
 - a strand in 5'->3' orientation is called the "template" strand
 - the strand in 3'->5' orientation is called the "complement" strand



7

Double Stranded DNA: Helix

- A template strand "anneals" with its complement strand
 - linked bases form the interior of a double helix
- Note orientation of the strands
 - template: 5' -> 3'
 - complement: 3' -> 5'



8

Simple DNA Abstractions

- Base, BasePair, Strand & Helix types
 - using enumeration, pair, and list types

```
module DnaComputing (...) where
data Base = A | C | G | T
    deriving(Eq,Enum,Ord,Bounded,Read,Show)
type BasePair = (Base,Base) -- pair of bases
type Strand = [Base] -- list of bases
type Tube = [Strand] -- list of strands
type Helix = [BasePair] -- list of base pairs
toBase :: Char -> Base
toBase c = read [c]
toStrand :: String -> Strand
toStrand s = [toBase b | b <- s] -- list comprehension 9
```

Useful Builtin Functions

```
DnaComputing> enumFrom A
[A,T,C,G]
DnaComputing A < T
True
DnaComputing> succ A
T
DnaComputing> pred G
C
DnaComputing> (map toEnum [0,1,2,3]) :: [Base]
[A,T,C,G]
DnaComputing> [A,T,C,G] == [A,T,C,G]
True
DnaComputing> [A,T,C] ++ [G,C,A]
[A,T,C,G,C,A]
DnaComputing> length [A,T,C,G]
4
```

10

Simple Conversions

```
DnaComputing> toBase 'A'
A
DnaComputing> toStrand "A"
[A]
DnaComputing> toStrand "ATCG"
[A,T,C,G]
DnaComputing> toStrand "ATTTCCGGCCGGGGCAAAGCGAGCT"
[A,T,T,T,C,C,G,G,C,C,G,G,G,G,C,A,A,A,G,C,G,A,G,C,T]
DnaComputing> toStrand "ACTGF"
[A,C,T,G,
Program error: Prelude.read: no parse
DnaComputing> fromBase A
'A'
DnaComputing> fromStrand [A,C,T,G]
"ACTG"
```

11

Watson-Crick Complements

- A <-> T, C <-> G

```
watsonCrickComp :: Base -> Base -> Bool
watsonCrickComp b b' = b == (wcc b')

wcc :: Base -> Base
wcc A = T
wcc T = A
wcc C = G
wcc G = C

wccStrand :: Strand -> Strand
wccStrand [] = []
wccStrand s = [ wcc base | base <- s ]

wccHelix :: Strand -> Helix
wccHelix s = anneal s (wccStrand s)
```

12

Strand Examples

```
DnaComputing> wcc A
T
DnaComputing> wccStrand [A,T,G,C]
[T,A,C,G]
DnaComputing> wccStrand (enumFrom A)
[T,G,C,A]
DnaComputing> wccStrand (toStrand "AATTCGG")
[T,T,A,A,G,G,C,C]
DnaComputing> zip [A,T,G,C] (wccStrand [A,T,G,C])
[(A,T),(G,C),(C,G),(T,A)]
DnaComputing> cycle [A,T,C,G]
[A,T,C,G,A,T,C,G,A,T,C,G,A,...]
DnaComputing> replicate 2 (zip [A,T,C,G] [T,A,G,C])
[[ (A,T), (T,A), (C,G), (G,C) ], [ (A,T), (T,A), (C,G), (G,C) ]]
DnaComputing> unzip [(A,T), (C,G), (T,A), (G,C)]
([A,C,T,G],[T,G,A,C])
```

13

Strand Orientation

- With strands, 5'→3' orientation is important for various processes, so we need to have a convention.
 - assume left-to-right ordering in a single-strand is 5'→3'
 - assume first base of a pair is in the 5'→3' template strand
 - second base of a pair is in the 3'→5' complement strand
- Haskell list comprehensions are just the right thing

```
templateStrand :: [BasePair] -> Strand
templateStrand bp = [b | (b,b') <- bp] -- result is 5'→3' strand

complementStrand :: [BasePair] -> Strand
complementStrand bp = [b' | (b,b') <- bp] -- result is 3'→5' strand

orientComplementStrand :: Strand -> Strand
orientComplementStrand s = reverse s -- convert 3'→5' to 5'→3'
```

14

Annealing

- Given two complementary strands, annealing forms a helix structure represented as a list of base pairs

```
anneal :: Strand -> Strand -> Helix
anneal [] [] = []
anneal s s'
  | willAnneal s s' = zip s s'
  | otherwise = []

willAnneal :: Strand -> Strand -> Bool
willAnneal [] [] = True
willAnneal [] (_:_) = False
willAnneal (_:_) [] = False
willAnneal (x:xs) (y:ys)
  | x == wcc y && willAnneal xs ys = True
  | otherwise = False
```

15

More Strand Examples

```
DnaComputing> templateStrand [(A,T),(C,G),(G,C),(T,A)]
[A,C,G,T]
DnaComputing> complementStrand [(A,T),(C,G),(G,C),(T,A)]
[T,G,C,A]
DnaComputing> orientComplementStrand [T,G,C,A]
[A,C,G,T]
DnaComputing> willAnneal [A,C,G,T] [T,G,C,A]
False
DnaComputing> anneal [A,C,G,T] [T,G,C,A]
[(A,T),(C,G),(G,C),(T,A)]
```

16

Random Sequences

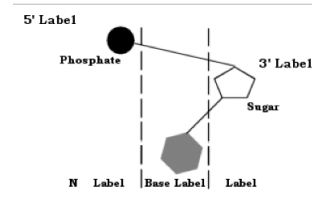
- Define a random base sequence generator
 - takes a “seed” value for random generator
 - and the length of the desired sequence
 - also define an “infinite” probabilistic sequence
 - based on a probability of occurrence of a base

```
DnaComputing> randomBaseSeq 1023 20
[C,G,G,A,T,G,G,G,C,A,G,A,G,T,A,A,G,T]
DnaComputing> take 20 (randomBaseProbSeq
 [(A,0.25),(T,0.45),(C,0.25),(G,0.05)])
[T,C,A,C,T,A,T,C,C,T,T,T,A,T,T,C,A,T,T,A]
```

17

Nucleotide Labeling

- To fully represent a DNA nucleotide we have to take into consideration each of the following properties of nucleotides which occur in practical molecular biology experiments:
 - type of the base
 - methyl group attached to the base, if any
 - 3' site attachments, if any
 - 5' site attachments, if any



18

Nucleotide Type

- We introduce a “tag” as a place holder for attaching another type of object to a nucleotide

```
data Nucleotide = Nucl Base Tag Tag Tag
  deriving (Eq, Ord, Read)
```

- We construct a nucleotide data object using a function as a constructor

```
makeNucl :: Base->Tag->Tag->Tag->Nucleotide
makeNucl b t t5' t3' = Nucl b t t5' t3'
```

19

Tag Type

- The tag is defined as an enumerated type
 - each element represents a type of structure that can be attached to a nucleotide
 - E.g., in the DNA of higher plants and animals cytosine often has a methyl group attached to the 5-carbon atom of the pyrimidine ring making a “methylated” base
 - In laboratory DNA processing tags to allow us to select particular nucleotides from solution, for example, by attaching a metal bead a magnet can be used to separate out nucleotides that have a bead attached

```
data Tag = Nil | Block | Phosphate | Probe |
  MetalBead | MethylGroup |
  Phosphodiester
  deriving (Eq, Enum, Ord, Bounded, Read, Show)
```

20

Other Nucleotide Functions

- WCC depends only on the base

```
watsonCrickNucl :: Nucleotide->Nucleotide->Bool
watsonCrickNucl (Nucl b _ _ _) (Nucl b' _ _ _)
    = watsonCrickComp b b'

makeWCNucl :: Nucleotide->Tag->Tag->Tag->Nucleotide
makeWCNucleotide (Nucl b t t5' t3')
    = (Nucl (wcc b) newb newt5' newt3')
```

21

Extended Base Pair Type

- Base pair
 - A base pair is either a pair of complementary nucleotides, or a “don’t care” (X) and a nucleotide

```
data Nucl = X | Nc Nucleotide
    deriving (Eq, Ord, Read)
type BasePair = (Nucl, Nucl)
```

22

In-silica Interpretation of DNA

- Implement functional simulation of *in vitro* biochemical processes
 - polymerase
 - ligase
 - nuclease
 - anneal
 - denature
 - gel electrophoresis
 - filtering

23

Polymerase

- Synthesizes DNA using template strand
 - works in 5'->3' orientation (template strand)
 - uses a primer to find starting position
 - generates watson-crick complement for each nucleotide and zips together a new double-strand
 - basis for DNA replication

24

Polymerase Function

- Uses list comprehension

```
polymerase :: [Strand]->[Tag]->Tag->Tag->Tag->[Strand]
polymerase strands nonBlockTags t t5' t3' = poly'
where
  poly53 = [ polymeraseStrand strand nonBlockTags t t5' t3'
            | strand<-strands ]
  firstOrientation = [ orientStrand strand | strand<-poly53 ]
  poly35 = [ polymeraseStrand strand nonBlockTags t t5' t3'
            | strand<-firstOrientation ]
  poly' = [ orientStrand strand | strand<-poly35 ]
```

25

Ligase

- Links two strands together
 - links 5' phosphate and 3' hydroxyl
 - used to form new longer strands
 - during DNA replication, gaps may occur along the complement strand, which are 'fixed' by ligase
 - it is possible to attach a "block" to one end of a strand to prevent ligase

26

Nuclease

- Traverses a strand until a specific base pair sequence is matched then cleaves the strand
 - "breaks" the phosphodiester bond
 - result is two separated strands
- Use incremental pattern matching in Haskell to match sequence and then split list

27

Annealing

- Two sequences or subsequences can "stick" together if they are complements
 - annealing looks for stick nucleotides in a sequence and tries to pair them up to form DNA molecules
 - perfect annealing occurs when one strand is the perfect complement of another strand
- Haskell higher-order list functions do the trick

28

Denature

- Inverse of the annealing process
 - “boil” DNA solution to separate double-stranded molecule into single strands
- Haskell’s unzip function and fst/snd pair functions simulate this process

29

Gel Electrophoresis

- Separates DNA molecules by length
 - uses negative charge and movement of molecules towards a positive anode through a gel as a way to determine length
 - Shorter molecules move faster than longer ones
 - results in “sorting” by length
 - easy in Haskell to determine length and sort lists

30

Filtering

- Several types of filtering processes
 - used to select strands with particular properties
 - Might attach a magnetic “probe” as a tag to a strand so that a magnet could be used to extract specific strands from a solution
 - Haskell higher-order filter function is ideal for applying a filter to a tube of strands

31

Further Reading

- *Understanding DNA and Gene Cloning*, 3rd Ed., Karl Drlica, John-Wiley, 1997.
- *Recombinant DNA*, 2nd Ed., James D. Watson, et al., Scientific American Books, 1998
- “Computing with DNA,” by Leonard Adleman, *Scientific American*, August, 1998
- *DNA Computing: New Computing Paradigms*, Paun, Rozenberg & Salomaa, Springer-Verlag, 1998
- “On the Computational Power of DNA Annealing and Ligation,” DIMACS Workshop on DNA Based Computers, Erick Winfree, American Mathematical Society, 1996
- “FunDNA: A Domain Specific Language for Functional DNA Programming,” Lillie A. Anderson & R. Greg Lavender, available upon request

32