Fun with Semirings

Christopher R. Genovese

Department of Statistics & Data Science

Thu 21 Nov 2024
Session #23

Plan

Semirings

Plan

Semirings

Dynamic Programming Redux

Plan

Semirings

Dynamic Programming Redux

Generic Algorithms

Plan

Semirings

Dynamic Programming Redux

Generic Algorithms

Free Structures

Plan

Semirings

Dynamic Programming Redux

Generic Algorithms

Free Structures

Aside: Hylomorphisms

Announcements

- Natural Joins and `Using`
- Postponing Schema Design until next time
- Poll
- **Reading**: From last time:
  - Interlude F Sec 9.1 180–188 and Sec 9.2
  - Fun with Semrings in `documents/Documents`.
  - Code Design with Types and Concepts (`data` means `type` and other syntax variations, but you'll get it.)
  - Check out SQL practice sites listed last time.
  - Deriving the Z-combinator and Classes with the Z Combinator
- **Homework**: `parser-combinators`, next one of:
  - `classification-tree-basic`
  - `sym-spell`
  - `migit3`
  - `regex-derivatives`
  - Alternates: `laser-tag`, `dominoes`

Plan

Semirings

Dynamic Programming Redux

Generic Algorithms

Free Structures

Aside: Hylomorphisms

Interacting Monoids: Semirings

Definition: Semirings

We say that $\langle \mathcal{S}, \boxplus, \mathbf{0}, \boxdot, \mathbf{1} \rangle$ is a **semiring** when $\mathcal{S}$ is a set with two special elements, denoted by $\mathbf{0}$ and $\mathbf{1}$, and two operators $\boxplus$ and $\boxdot\colon \mathcal{S} \times \mathcal{S} \longrightarrow \mathcal{S}$ that satisfy:

1. $\langle \mathcal{S}, \boxplus, \mathbf{0} \rangle$ is a *commutative* monoid
2. $\langle \mathcal{S}, \boxdot, \mathbf{1} \rangle$ is a monoid
3. $\mathbf{0}$ annihilates: $x \boxdot \mathbf{0} = \mathbf{0} = \mathbf{0} \boxdot x$
4. $\boxdot$ distributes over $\boxplus$:

$$a \boxdot (b \boxplus c) = (a \boxdot b) \boxplus (a \boxdot c)$$
$$(b \boxplus c) \boxdot a = (b \boxdot a) \boxplus (c \boxdot a).$$

The operator $\boxdot$ need not be commutative; if $\boxdot$ is commutative, we call this a **commutative semiring**.

In algorithmic terms, we will think of $\boxplus$ as combining different results and of $\boxdot$ as combining different choices within a result.

Interacting Monoids: Semirings (cont'd)

A **star semiring** is a semiring with an additional operation * defined by

$$a^* = 1 \boxplus (a \boxdot a^*) = 1 \boxplus (a^* \boxdot a).$$

We often are interested in *complete* star semiring, in which

$$a^* = \bigboxplus_{k \geq 0} a^k,$$

where $a^0 = \mathbf{1}$ and $a^k = a \boxdot a^{k-1} = a^{k-1} \boxdot a$.

We'll see how this is useful later.

Example Semirings

- Boolean logic $\langle \{\top, \bot\}, \vee, \bot, \wedge, \top \rangle$
- Natural Sum-Product $\langle \mathbb{N}, +, 0, \cdot, 1 \rangle$
  (Extends to integers and reals and complex numbers.)
- Subsets of $\mathcal{A}$ $\langle 2^{\mathcal{A}}, \cup, \{\}, \cap, \mathcal{A} \rangle$
- Square matrices with matrix sum and product
- Polynomials (and sequences) with coefficients in a semiring, with sum and product.
- Relations $\langle 2^{\mathcal{S} \times \mathcal{S}}, \cup, \{\}, \circ, \Delta \rangle$ where $\Delta = \{\langle s, s \rangle$ such that $s \in \mathcal{S}\}$ is the "diagonal" relation.
- *union*-$\times$ Semiring (up to isomorphism)
  What are the identity elements?
- Regular Languages
- ...

Example Semirings: Numeric Semirings

- *Min-Plus*. The set $\mathbb{R} \cup \{\infty\}$,is a commutative semiring with $\boxplus = \min$, $\boxdot = +$, $\mathbf{0} = \infty$, and $\mathbf{1} = 0$.

- *Max-Plus*. The set $\mathbb{R} \cup \{-\infty\}$,is a commutative semiring with $\boxplus = \min$, $\boxdot = +$, $\mathbf{0} = -\infty$, and $\mathbf{1} = 0$.

- *Max-Min*. The set $[-\infty\_\infty]$, which includes $\pm\infty$, is a commutative semiring with $\boxplus = \max$, $\boxdot = \min$, $\mathbf{0} = -\infty$, and $\mathbf{1} = \infty$.

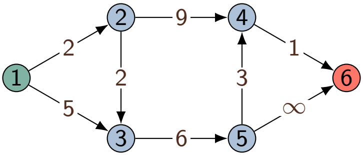
- *Max-Times*. The set $[0\_)$ is a commutative semiring with $\boxplus = \max$ the maximum of two numbers, $\boxdot = \cdot$ regular multiplication, $\mathbf{0} = 0$, and $\mathbf{1} = 1$.

- *Min-Times*. The set $(0\_\infty]$, which includes ∞, is a commutative semiring with $\boxplus = \min$, $\boxdot = \cdot$, $\mathbf{0} = \infty$, and $\mathbf{1} = 1$.

Example Semirings: Numeric Semirings

Min-Plus. The set $\mathbb{R} \cup \{\infty\}$,is a commutative semiring with $\boxplus = \min$, $\boxdot = +$, $\mathbf{0} = \infty$, and $\mathbf{1} = 0$.

Think of these as distances or costs in moving from source to target.



Look at every path from node 1 to node 6. Along each path, we combine distances with $\boxdot$ and *across* paths we aggregate with $\boxplus$. This gives us

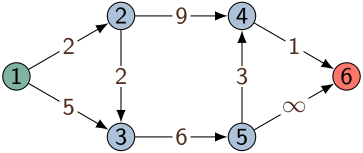
$$\begin{aligned}
&(2 \boxdot 9 \boxdot 1) \boxplus (2 \boxdot 2 \boxdot 6 \boxdot 3 \boxdot 1) \boxplus (2 \boxdot 2 \boxdot 6 \boxdot \infty) \\
&\quad \boxplus (2 \boxdot 5 \boxdot 6 \boxdot 3 \boxdot 1) \boxplus (2 \boxdot 5 \boxdot 6 \boxdot \infty) \\
&\quad = \min(12, 14, \infty, 17, \infty) = 12.
\end{aligned}$$

The Min-Plus semiring operations gives us the minimum distance/cost over all paths from node 1 to node 6.

Example Semirings: Numeric Semirings

Max-Min. The set $[-\infty \,\_\, \infty]$, which includes $\pm\infty$, is a commutative semiring with $\boxplus = \max$, $\boxdot = \min$, $\mathbf{0} = -\infty$, and $\mathbf{1} = \infty$.

Now, edge weights represent the capacity of flow on a channel along that edge.



We combine capacities along the paths from 1 to 6 using the semiring operations:

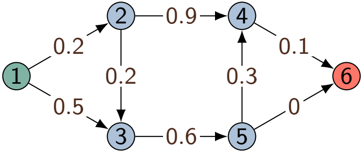
$$
\begin{aligned}
(2 \boxdot 9 \boxdot 1) \boxplus (2 \boxdot 2 \boxdot 6 \boxdot 3 \boxdot 1) &\boxplus (2 \boxdot 2 \boxdot 6 \boxdot \infty) \\
\boxplus (2 \boxdot 5 \boxdot 6 \boxdot 3 \boxdot 1) &\boxplus (2 \boxdot 5 \boxdot 6 \boxdot \infty) \\
= \max(1, 1, 2, 1, 2) &= 2.
\end{aligned}
$$

The Max-Min semiring operations gives us optimal capacity of flow from nodes 1 to 6.

Example Semirings: Numeric Semirings

Max-Times. The set $[0\_)$ is a commutative semiring with
$\boxplus = \max$, $\boxdot = \cdot$, $\mathbf{0} = 0$, and $\mathbf{1} = 1$.

Decorate the edges with the *reliability* of the connection represented by the edge. Here, reliability is measured by a number between 0 and 1 (i.e., a probability). We use a variant of the previous graph.



Aggregating again over paths from node 1 to node 6, we get

$$
\begin{aligned}
(0.2 \boxdot 0.9 \boxdot 0.1) \boxplus (0.2 \boxdot 0.2 \boxdot 0.6 \boxdot 0.3 \boxdot 0.1) \boxplus (0.2 \boxdot 0.2 \boxdot 0.6 \boxdot 0) \\
\boxplus (0.2 \boxdot 0.5 \boxdot 0.6 \boxdot 0.3 \boxdot 0.1) \boxplus (0.2 \boxdot 0.5 \boxdot 0.6 \boxdot 0) \\
= \max(0.018, 0.00072, 0, 0.0018, 0) = 0.018.
\end{aligned}
$$

We get the reliability of the most reliable path from node 1 to 6.

Plan

Semirings

Dynamic Programming Redux

Generic Algorithms

Free Structures

Aside: Hylomorphisms

Unifying Dynamic Programming

We will start by reconsidering dynamic programming algorithms and how we can unify them quite beautifully with a few concepts and some semirings.

The key idea is that the Bellman equations can be expressed as applying $\boxdot$ along paths through the DAG to get a result and then combining results with $\boxplus$.

For instance, in the edit-distance problem we have

$$E_{ij} = \min(1 + E_{i-1,j}, 1 + E_{i,j-1}, d_{ij} + E_{i-1,j-1}),$$

which should be viewed in the Min-Plus semiring.

Unifying Dynamic Programming

We will start by reconsidering dynamic programming algorithms and how we can unify them quite beautifully with a few concepts and some semirings.

We can define Semirings as a trait as we did for Monoids.

```
trait Semiring s where
  zero : s
  one  : s

  plus : s -> s -> s
  times : s -> s -> s

  (<+>) : s -> s -> s
  (<+>) = plus
  infix_left 60 <+>

  (<.>) : s -> s -> s
  (<.>) = times
  infix_left 70 <.>
```

We will assume several instances with this trait below.

Unifying Dynamic Programming

We will start by reconsidering dynamic programming algorithms and how we can unify them quite beautifully with a few concepts and some semirings.

We'll start by expressing a general dynamic programming problem and solver in TL1. (Based on Llorens and Vilar 2019.)

A problem has three parts:

```
type DPProblem problem score
  = record DPProblem where
      initial     : problem
      isTrivial   : problem -> Boolean
      subproblems : problem -> List (Pair score problem)
```

Unifying DP Example: Edit distance

```
type alias EDistProblem = Pair String String
type alias Distance = Int

edp : EDistProblem -> DPProblem EDistProblem (MinPlus Distance)
edp words = DPProblem words is_trivial subprobs
  where
    is_trivial : EDistProblem -> Bool
    is_trivial words = words == ("", "")

    subprobs : EDistProblem -> List (Pair Distance EDistProblem)
    subprobs (a :: as, "") = [ (1, (as, "")) ]
    subprobs ("", b :: bs) = [ (1, ("", bs)) ]
    subprobs (a :: as, b :: bs) = [ (1, (a :: as, bs))
                                  , (1, (as, b :: bs))
                                  , (if a == b then 0 else 1, (as, bs))
                                  ]
```

Unifying DP Example: Knapsack Problem

```
type alias Capacity = Int
type alias Value = Int
type alias Weight = Int
type Item = record Item { value : Value, weight : Weight }
type KnapsackProblem = record KnP where
                         capacity : Capacity
                         items    : List Item

knp : KnapsackProblem -> DPProblem KnapsackProblem (MaxPlus Value)
knp sack = DPProblem sack is_trivial subprobs
  where
    is_trivial : KnapsackProblem -> Bool
    is_trivial (KnP _ items) = empty items

    subprobs : KnapsackProblem -> List (Pair Value KnapsackProblem)
    subprobs (KnP cap (Cons item rest)) =
      | item.weight <= cap = [ (0, KnP cap rest)
                             , (item.value, KnP (cap - item.weight) rest)
                             ]
      | otherwise          = [ (0, KnP cap rest) ]
```

Unifying DP Example: Dirichlet Problem

Here, we do a 1-dimensional version: what is probability that symmetric walk state $\geq s$ in t steps or fewer? This generalizes easily to general Dirichlet problems.

```
type alias Probability = Real
type alias Position = Int
type alias Step = Int
type DirichletProblem = record DirP where
                          start : Position
                          final : Position
                          limit : Steps

dirp : DirichletProblem -> DPProblem DirichletProblem Probability
dirp state = DPProblem state is_trivial subprobs
  where
    is_trivial : DirichletProblem -> Bool
    is_trivial d = d.start >= d.final

    subprobs : DirichletProblem -> List (Pair Probability DirichletProblem)
    subprobs d =
      | d.limit == 0 = []
      | otherwise    = let d_up = d { start=d.start + 1, limit=d.limit - 1 }
                           d_dn = d { start=d.start - 1, limit=d.limit - 1 }
                       in
                         [ (0.5, d_up), (0.5, d_dn) ]
```

Unifying DP (cont'd)

We can solve these problems with the same simple code:

```
solveDP : Semiring score => DPProblem problem score -> score
solveDP dpp = go (initial dpp)
  where
    go : problem -> score
    go p
      | dpp.isTrivial p = one
      | otherwise       =
          let next = [sc <.> go subp | (sc, subp) <- dpp.subproblems p]
          in
            fold (<+>) zero next

editDistance : EDistProblem -> MinPlus Distance
editDistance = solveDP . edp

knapsack : KnapsackProblem -> MinPlus Distance
knapsack = solveDP . knp

dirichlet : DirichletProblem -> Probability
dirichlet = solveDP . dirp
```

Unifying DP (cont'd)

One issue: we have not yet memoized.

The solution is easy: make one change in `solveDP`:

```
solveDP : Semiring score => DPProblem problem score -> score
solveDP dpp = mem_go (initial dpp)
  where
    mem_go : problem -> score
    mem_go = memo go

    go : problem -> score
    go p
      ...
```

Getting the Solution Paths

In practice, for DP problems, we want to find not just the best score but the best solution (or all the best solutions or all solutions or how many solutions ...).

We can do this by constructing new semirings to capture what we want. For instance, for the best solution this looks like

```
type BestSolution decisions score =
  BestSolution (List (Maybe decisions)) score

instance Semiring (BestSolution ds sc) where
  BestSolution ds1 sc1 <+> BestSolution ds2 sc2
    | sc1 == optimum sc1 sc2 = BestSolution ds1 sc1
    | otherwise              = BestSolution ds2 sc2

  BestSolution ds1 sc1 <.> BestSolution ds2 sc2 =
    BestSolution (ap concat ds1 ds2) (sc1 <.> sc2)
```

With some small changes to our original code, we now get the best solution with the best score.

With tweaks to the semiring, we can get All Best Solutions, All Solutions, Count of Solutions, and more.

Plan

Semirings

Dynamic Programming Redux

Generic Algorithms

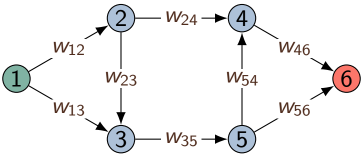
Free Structures

Aside: Hylomorphisms

Different Problems, Same Algorithm

A wide variety of problems can be solved with generic algorithms – the same code – that is intepreted in ***different semirings***

Path Problems



- Shortest paths: edge weights are distances, Min-Plus Semiring
- Connectivity: edge weights are Booleans, Boolean Logic Semiring
- Capacity: edge weights are capacities, Max-Min Semiring
- Reliability: edge weights are probabilities, Max-Plus Semiring
- Language Accepted: edge labels are transitions, $\cup-$concat Semiring
- ...

All of these problems use the **exact same code**.

Path Problems (cont'd)

General structure

- We have a matrix of edge weights W
- Compute
$$P = \bigboxplus_{k \geq 0} W^k = I \boxplus W \boxplus (W \boxdot W) \boxplus \cdots .$$
- If this limit exists, it solves a ***fixed-point equation***
$$X = WX + I.$$

One algorithm to rule them all. Same implementation, same complexity.

Other Problems

- Satisfiability Problems

  Example: Map coloring, See Knuth vol 4, more later if time allows

- Database Queries and Joins

  Example: See (Olteanu 2022)

- Bayesian Networks

  Factor probabilities into lower dimensional marginalized terms. Same implementation with Max-Times instead of Plus-Times gives max posterior.

Plan

Semirings

Dynamic Programming Redux

Generic Algorithms

Free Structures

Aside: Hylomorphisms

Advent of Code Challenge: Part 1

You've managed to sneak in to the prototype suit manufacturing lab. The Elves are making decent progress, but are still struggling with the suit's size reduction capabilities.

While the very latest in 1518 alchemical technology might have solved their problem eventually, you can do better. You scan the chemical composition of the suit's material and discover that it is formed by extremely long polymers (one of which is available as your puzzle input).

The polymer is formed by smaller units which, when triggered, react with each other such that two adjacent units of the same type and opposite polarity are destroyed. Units' types are represented by letters; units' polarity is represented by capitalization. For instance, r and R are units with the same type but opposite polarity, whereas r and s are entirely different types and do not react.

For example:

- In `aA`, `a` and `A` react, leaving nothing behind.
- In `abBA`, `bB` destroys itself, leaving `aA`. As above, this then destroys itself, leaving nothing.
- In `abAB`, no two adjacent units are of the same type, and so nothing happens.
- In `aabAAB`, even though `aa` and `AA` are of the same type, their polarities match, and so nothing happens.

Now, consider a larger example, `dabAcCaCBAcCcaDA`.

```
dabAcCaCBAcCcaDA  The first 'cC' is removed.
dabAaCBAcCcaDA    This creates 'Aa', which is removed.
dabCBAcCcaDA      Either 'cC' or 'Cc' are removed (the result is the same).
dabCBAcaDA        No further actions can be taken.
```

After all possible reactions, the resulting polymer contains 10 units.

How many units remain after fully reacting the polymer you scanned?

(h/t Justin Le and Advent of Code)

Part 1

What are the data here? What is the algebraic structure?

```
represent : Data -> Structure
represent = foldM inject

inject : Component -> Structure
inject c
    | isAlpha c and isLowerCase c = ???
    | isAlpha c and isUpperCase c = ???
    | otherwise                   = munit
```

Part 1

What are the data here? What is the algebraic structure?

```
represent : Data -> Structure
represent = foldM inject

inject : Component -> Structure
inject c
    | isAlpha c and isLowerCase c = free c
    | isAlpha c and isUpperCase c = invert (free c)
    | otherwise                   = munit

solve : Data -> Int
solve = length . toList . represent
```

Advent of Code Challenge: Part 2

Time to improve the polymer.

One of the unit types is causing problems; it's preventing the polymer from collapsing as much as it should. Your goal is to figure out which unit type is causing the most problems, remove all instances of it (regardless of polarity), fully react the remaining polymer, and measure its length.

For example, again using the polymer `dabAcCaCBAcCcaDA` from above:

- Removing all A/a units produces `dbcCCBcCcD`. Fully reacting this polymer produces `dbCBcD`, which has length 6.
- Removing all B/b units produces `daAcCaCAcCcaDA`. Fully reacting this polymer produces `daCAcaDA`, which has length 8.
- Removing all C/c units produces `dabAaBAaDA`. Fully reacting this polymer produces `daDA`, which has length 4.
- Removing all D/d units produces `abAcCaCBAcCcaA`. Fully reacting this polymer produces `abCBAc`, which has length 6.

In this example, removing all `C/c` units was best, producing the answer 4.

What is the length of the shortest polymer you can produce by removing all units of exactly one type and fully reacting the result?

Part 2

Here, we are *mapping* between algebraic structures.

What is the mapping?

Part 2

Here, we are *mapping* between algebraic structures.

What is the mapping?

We construct a group homomorphism $\phi\colon F_{26} \longrightarrow F_{25}$ for each target character, that simply replaces the target character with the identity.

We then minimize over target characters.

And we get an efficient algorithm from this.

Plan

Hylomorphisms

```
type alias Algebra f a = f a -> a
type alias Coalgebra f b = b -> f b

hylo : Functor f => Algebra f p -> Coalgebra f s -> p -> s
hylo alg coalg = h
  where h = alg . map h . coalg

-- Memoized version
hyloM : Functor f => Algebra f p -> Coalgebra f s -> p -> s
hyloM alg coalg = h
  where h = memo (alg . map h . coalg)
```

Hylomorphisms

```
-- Example
type Fib a = Base | Next a a

instance Functor Fib where
  map _ Base = Base
  map f (Next a b) = Next (f a) (f b)

fibonacci : Int -> Integer
fibonacci = hylo alg coalg
  where
    alg Base = 1
    alg (Next a b) = a + b

    coalg n | n <= 2     = Base
            | otherwise = Next (n - 1) (n - 2)
```

THE END