Monoids and Tigers and Folds, oh my!

Christopher R. Genovese

Department of Statistics & Data Science

29 Aug 2024
Session #2

Plan

Follow-up Discussion on SWAG

Plan

Follow-up Discussion on SWAG

Folds

Plan

Follow-up Discussion on SWAG

Folds

Monoids and Semirings

Plan

Follow-up Discussion on SWAG

Folds

Monoids and Semirings

Activity

Announcements

- Please fill out **Experience survey**.
- Please fill out **office hours poll**.
- Special Office Hour times this week will be posted on Canvas.
- Email subject: [750]
- **Please bring your laptop to every class**
- **Reading**:
  - Tuesday: `https://36-750.github.io/course-info/shells/`, `https://git-scm.com/book/en/v2` sections 1.1 and 1.3, and Thinking Languages (almost ready).
- **Homework**: HW #1 (swag) due Thursday 5 Sep.
- Don't forget to send me your Github account names

Plan

Plan

Follow-up Discussion on SWAG

Folds

Monoids and Semirings

Activity

The “Fold” Pattern

In the basic Sliding-Window Aggregation, we repeatedly compute an aggregation over the window with the following pattern (expressed in TL2):

```
1     accumulator = starting_value
2     for item in items
3         accumulator = update_fn(accumulator, item)
```

Line 1 initializes the accumulator. Lines 2–3 successively updates it. Here, `update_fn` is often called the folding function, reducing function, or update rule. In the SWAG problem, it is just our binary operation.

This is a common pattern called a **fold**.

Examples? Variants?

Folds as "Objects"

We define a *type* to describe folds as a unified object. (We can make this a `Object` but it need not be, depending on the language.)

A fold is specified by four parameters:

1. a function that wraps or annotates input values,
2. a folding function that does the updates,
3. an initial value, and
4. a finalizing function that unwraps the final update.

In TL1, this is a type `Fold v w r` with *type variables* `v`, `w`, and `r` where

- `v` is the type of input *values*;
- `w` is the type of the *wrapped* values; and
- `r` is the type of the final *result* produced.

Here, `Fold` is a "type constructor," it takes three types and returns a new type. Indeed, it is a function with type `Type -> Type -> Type -> Type`.

In terms of these types, what are the parts of a `Fold`?

Folds as “Objects” (cont’d)

In TL1, we describe the `Fold` type as

```
1   type Fold v w r = forall a.
2       record Fold where
3           lift : (v -> w)
4           step : (a -> w -> a)
5           init : a
6           done : (a -> r)
```

Think of this like a function definition where the arguments and return values are types.

The `forall a` on Line 1 defines a “local” type variable a (for accumulator) that can be anything. Given four appropriate functions for the fields on Lines 3–6, we can *infer* what type a must be, so we don’t need to specify it as an independent parameter.

If `x` has type `Fold v w r`, then we can use `x.lift`, `x.`, `x.init`, and `x.done` to refer to the fields of `x`, and we can use `Fold.lift`, `Fold.step`, etc. as accessor functions.

Folds as “Objects” (cont’d)

You can represent this type in various ways in various languages. In Python or R, for instance, we might define a class, or we could define a simpler data structure (such as a `namedtuple` or a list with named components).

For example, in Python:

```
A = TypeVar['A']
class Fold(Generic[V, W, R]):
    def __init__(
        self,
        lift: Callable[[V], W],
        step: Callable[[A, W], A],
        init: A,
        done: Callable[[A], R]
    ) -> None:
        ...
    ...
```

It’s also good practice to define function that creates your data structure, if it’s not defined through your language (and sometimes even if it is).

The `fold` operation

Q: What does a `Fold` for mean look like?

The `fold` operation

Q: What does a `Fold` for mean look like?

It is a `Fold Num (Num, Int) Num` where `Num` represents numbers (with subtype `Int`) and `(a, b)` is a pair with component types `a` and `b`.

```
lift x = (x, 1)
step (x, m) (y, n) = (x + y, m + n)
init = 0
done (x, n) = x / n
```

Notice that we made this even more general: the aggregated values can have a different type than the input.

(Ex: counting total number of characters in a sequence of strings.)

The `fold` operation

Q: What does a `Fold` for mean look like?

Q: What is the type of `swag` in terms of the `Fold` type?

The `fold` operation

Q: What does a `Fold` for mean look like?

Q: What is the type of `swag` in terms of the `Fold` type?

```
swag : Fold v a b -> Nat -> [v] -> [b]
```

Notice that we made this even more general: the aggregated values can have a different type than the input.

(Ex: counting total number of characters in a sequence of strings.)

The `fold` operation

Q: What does a `Fold` for mean look like?

Q: What is the type of `swag` in terms of the `Fold` type?

In general, we have an operation which takes a `Fold` and a "foldable" container and returns the final result.

```
fold : Foldable c => Fold v w r -> c v -> r
```

Here, `c` is a type that represents containers containing that can be folded and to the left of the => is a *constraint*. (What's an example of a Foldable besides an array?)

We'll see soon some of what we get from representing fold operations this way.

We can also define related operations like `scan` (a fold that collects intermediate values), `foldRight` (a fold from the end, where appropriate), and `scanRight`.

Plan

Lists

Let $\mathcal{A}$ be a non-empty set. Then $\mathsf{List}(\mathcal{A})$, commonly denoted $\mathcal{A}^*$, is the set whose elements are *finite*-length tuples of elements from $\mathcal{A}$.

We can endow the set $\mathsf{List}(\mathcal{A})$ with structure by defining a binary operator $:: : \mathsf{List}(\mathcal{A}) \times \mathsf{List}(\mathcal{A}) \longrightarrow \mathsf{List}(\mathcal{A})$ that concatenates two lists. For example, $\langle 0, 0\rangle :: \langle 1, 0, 1, 1\rangle = \langle 0, 0, 1, 0, 1, 1\rangle$. This operator satisfies three **algebraic laws**, which you can confirm for yourself:

$$\begin{gathered}\langle\rangle :: \ell = \ell \\ \ell :: \langle\rangle = \ell \\ \ell_1 :: (\ell_2 :: \ell_3) = (\ell_1 :: \ell_2) :: \ell_3.\end{gathered}$$

where $\ell, \ell_1, \ell_2, \ell_3 \in \mathsf{List}(\mathcal{A})$. So, $\langle\rangle$ is an *identity* element for $::$, and $::$ is *associative*. Notice that $::$ is *not* commutative.

Monoids

A **monoid** $\langle \mathcal{M}, \Diamond, e \rangle$ consists of a set $\mathcal{M}$ equipped with a binary operator $\Diamond \colon \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{M}$ and a special element $e \in \mathcal{M}$ that satisfy:

1. $\Diamond$ is associative: $m_1 \Diamond (m_2 \Diamond m_3) = (m_1 \Diamond m_2) \Diamond m_3$ for every $m_1, m_2, m_3 \in \mathcal{M}$, and
2. e is an *identity element*: $e \Diamond m = e = m \Diamond e$ for every $m \in \mathcal{M}$.

$\Diamond$ need not be commutative, but if it is we call this a *commutative monoid*.

A special case we will use is an ***ordered monoid***, which is a monoid $(\mathcal{M}, \Diamond, e)$ with a partial order $\prec$ such that $x, y \in \mathcal{M}$ with $x \prec y$ implies

$$x \Diamond z \prec y \Diamond z \quad \text{and} \quad z \Diamond x \prec z \Diamond y,$$

for all $z \in \mathcal{M}$.

How might we represent a monoid in a program?

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???
5. Subsets of a set with union or intersection

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???
5. Subsets of a set with union or intersection
6. Multisets

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???
5. Subsets of a set with union or intersection
6. Multisets
7. Functions $\mathcal{X} \longrightarrow \mathcal{X}$ with composition

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???
5. Subsets of a set with union or intersection
6. Multisets
7. Functions $\mathcal{X} \longrightarrow \mathcal{X}$ with composition
8. Dictionaries

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???
5. Subsets of a set with union or intersection
6. Multisets
7. Functions $\mathcal{X} \longrightarrow \mathcal{X}$ with composition
8. Dictionaries
9. Product monoids

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???
5. Subsets of a set with union or intersection
6. Multisets
7. Functions $\mathcal{X} \longrightarrow \mathcal{X}$ with composition
8. Dictionaries
9. Product monoids
10. Dual monoids

Monoid Examples

What are some examples of Monoids? (Remember it's not just the set but the identity and operator as well.)

1. $(\mathbb{R}, +, 0)$
2. $(\mathbb{R}_+, \cdot, 1)$
3. Booleans with and or or. (What are the units?)
4. Natural numbers with max and ???
5. Subsets of a set with union or intersection
6. Multisets
7. Functions $\mathcal{X} \longrightarrow \mathcal{X}$ with composition
8. Dictionaries
9. Product monoids
10. Dual monoids
11. ...

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 $\boxdot : \mathcal{S} \times \mathcal{S} \longrightarrow \mathcal{S}$ that satisfies

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 it is, we have a *commutative semiring*.

Semirings describe sets that are monoids in two separate ways, with some consistency requirements for how the two monoids relate. They generalize the natural numbers and have a huge range of algorithmic applications. We'll see much more of them later.

Semiring Examples (brief)

1. Booleans
2. Subsets
3. Relations
4. Languages
5. Matrices
6. Polynomials/Sequences
7. Tropical (and other numeric) Semirings (min-plus and max-plus)
8. . . .

Plan

Follow-up Discussion on SWAG

Folds

Monoids and Semirings

Activity

Activity

Find the contiguous segment of cells with the largest absolute difference between the number of red and blue cells. Expect that the number of cells can be very large, e.g., $n = 10^6$. Can you find a method that scales *linearly* in n?

(h/t Bret Yorgey)

Activity

Find the contiguous segment of cells with the largest absolute difference between the number of red and blue cells. Expect that the number of cells can be very large, e.g., $n = 10^6$. Can you find a method that scales *linearly* in n?

- What is a naive approach to this problem? How does its effort scale with n?

(h/t Bret Yorgey)

Activity

Find the contiguous segment of cells with the largest absolute difference between the number of red and blue cells. Expect that the number of cells can be very large, e.g., $n = 10^6$. Can you find a method that scales *linearly* in n?

- What is a naive approach to this problem? How does its effort scale with n?
- Where is there redundancy in the naive approach?

(h/t Bret Yorgey)

Activity

Find the contiguous segment of cells with the largest absolute difference between the number of red and blue cells. Expect that the number of cells can be very large, e.g., $n = 10^6$. Can you find a method that scales *linearly* in n?

- What is a naive approach to this problem? How does its effort scale with n?
- Where is there redundancy in the naive approach?
- How can we identify the maximizing segment as well?

(h/t Bret Yorgey)

Activity

Find the contiguous segment of cells with the largest absolute difference between the number of red and blue cells. Expect that the number of cells can be very large, e.g., $n = 10^6$. Can you find a method that scales *linearly* in n?

- What is a naive approach to this problem? How does its effort scale with n?
- Where is there redundancy in the naive approach?
- How can we identify the maximizing segment as well?
- How might we think about “red” and “blue” in a way that generalizes the problem?

(h/t Bret Yorgey)

Activity

Find the contiguous segment of cells with the largest absolute difference between the number of red and blue cells. Expect that the number of cells can be very large, e.g., $n = 10^6$. Can you find a method that scales *linearly* in n?

- What is a naive approach to this problem? How does its effort scale with n?
- Where is there redundancy in the naive approach?
- How can we identify the maximizing segment as well?
- How might we think about "red" and "blue" in a way that generalizes the problem?
- Should we be restricted to numbers? What are the algebraic properties that we need from a cell's contents?

(h/t Bret Yorgey)

Kadane's Algorithm (mutable version)

```
def kadane(cells: list[int]) -> int:    # any summable
    "Returns the maximum segment sum of cells w/Kadane's alg
    best, current = 0, 0

    for cell in cells:
        current = max(current + cell, 0)
        best = max(best, current)
    return best
```

Why does this work?

Note we could write the loop contents equivalently as

```
current += cell
if current < 0:
    current = 0
if current > best:
    best = current
```

Kadane's Algorithm (towards an immutable version)

Let's express the algorithm in terms of a Fold

```
type State c = record State { total : c, biggest : c }

-- Assume we have agg : c -> c -> c that aggregates cells
-- (with analogue of negative reset) and an value c0 : c
-- to start the aggregation

kadane : Fold c (State c) c
kadane = Fold id step (State c0 c0) State.biggest

step : Ordered c => State c -> c -> State c
step (State cur best) value = State next (max best next)
    where
        next = agg cur value
```

Kadane's Algorithm (TL1 version)

The only operations we use in the original were adding, a neutral element, and comparing. We can apply this to cells containing any *ordered Monoid*.

```
bestSoFar : (Ordered c, Foldable t) => Fold a c c -> t a ->
bestSoFar f as = scan f as |> max

kadane : (Monoid a, Ordered a, Foldable t) => t a -> a
kadane = bestSoFar f
    where
        next s a = max munit (s <> a)
        f = Fold {wrap=id, step=next, init=munit, done=id}
```

Here, `munit : a` is the inferred unit for the Monoid type. It could be a class member or selected by a protocol.

THE END