CHAPTER 5
Sequence and Bag

Chapter Outline

- Extended Z Structure
  - Sequence
  - Bag

Sequence

Sequence

- It is sometimes necessary to record the order in which objects are arranged, e.g.:
  - Data may be indexed by an ordered collection of keys
  - Messages may be stored in order of arrival
  - Tasks may be performed in order of importance
- When we want to model ordered collections of objects:
  - Sets are inadequate because not ordered and do not allow duplicates
  - Tuples are inadequate because fixed length

Sequence

- In many situations the ordering of elements is significant. These are modelled by the sequence.
- A sequence is a kind of function. It is NOT a set.
- Sequence is an ordered collection of objects.
- It is just a finite function whose domain is drawn from a sequence of non-zero natural numbers called indices.
- Sequences can model arrays, lists, queues, and other sequential structures.

Sequence

- Can be viewed as collections with predefined constraints.
- For example:

  weekday = $\{1 \mapsto mon, 2 \mapsto tue, 3 \mapsto wed, 4 \mapsto thu, 5 \mapsto fri\}$

- The terms in a sequence are ordered by their first components. $1 \mapsto X$ always comes immediately before $2 \mapsto Y$ whatever X and Y are.
- The first components of pairs in a sequence are often called indices.

Sequence

- In a sequence order matters. And repetitions are allowed.
- Since sequence is just a function, we can write:

weekday(1) = mon

weekday(5) = fri

and

weekday(6) is undefined

Sequence

- In Z we might define

  DAY ::= mon | tue | wed | thu | fri

- Then declare:

$$\begin{array}{|l} weekday : seq\ DAY \\ \hline weekday == \langle mon, tue, wed, thu, fri \rangle \end{array}$$

  where ***seq*** introduces a sequence.

- Sequences are notated inside angle brackets ⟨ and ⟩.

Types of Sequence

- Empty sequence – no objects in the collection

  $\langle\ \rangle$

- Normal (including empty sequence)

  seq

- Non-empty

  seq_1

- Injective (not containing duplicates)

  iseq

Example of Sequence

- Repetitions can occur in a sequence.

  $\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \langle a, b, a \rangle$

- But an ***iseq*** never contains repetitions.

  $\{ 1 \mapsto a, 2 \mapsto b \} = \langle a, b \rangle$

Example of Sequence

$(\mathrm{seq}\ \mathbb{N})$ is $\{1 \mapsto 3, 2 \mapsto 9, 3 \mapsto 9, 4 \mapsto 11\}$

written as $\langle 3, 9, 9, 11 \rangle$

$(\mathrm{iseq}\ files)$ is $\{1 \mapsto UpdateFile, 2 \mapsto LogFile, 3 \mapsto TaxFile\}$

written as $\langle UpdateFile, LogFile, TaxFile \rangle$

Operations on Sequence

- Operations on non-empty sequences include:

  head s — the first element of s

  head weekday = ⟨mon⟩

  last s — the last element of s

  last weekday = ⟨fri⟩

  front s — s without its last element

  front weekday = ⟨mon, tue, wed, thu⟩

  tail s — s without its head element

  tail weekday = ⟨tue, wed, thu, fri⟩

Operations on Sequence

- The *concatenation operator*, $\frown$, joins two sequences by appending the second sequence onto the end of the first.

$$\langle sun \rangle \frown weekday \frown \langle sat \rangle = \langle sun, mon, tue, wed, thu, fri, sat \rangle$$

Operations on Sequence

- The *filter operator*, $\upharpoonright$, creates a new sequence from an existing sequence.

  $$weekday \upharpoonright \langle tue, wed, anyday \rangle = \langle tue, wed \rangle$$

  - The new sequence contains just those elements of the original sequence, *weekday*, with the order of *weekday* preserved.

- Another example:

  $$\langle a, b, c, d, e, d, c, b, a \rangle \upharpoonright \langle a, d \rangle = \langle a, d, d, a \rangle$$

  - Order and multiplicity of elements preserved

Operations on Sequence

- The *extraction operator*, 1, also creates a new sequence from an existing one.

$$\{ 2, 4, 6 \} \ 1 \ weekday = \langle tue, thu \rangle$$

- The new sequence contains only those elements that appear at the given indices. Order is maintained.

Operations on Sequence

- The *squash operator* compacts a function whose domain is from non-zero positive integers into a sequence.
- For example:

$$\text{squash}\ \{ 4 \mapsto c, 2 \mapsto b, 6 \mapsto d \} = \langle b, c, d \rangle$$

$$\Rightarrow \{1 \mapsto b, 2 \mapsto c, 3 \mapsto d\}$$

Operations on Sequence

- A sequence is a kind of function, and a function is a kind of set. So we can use, with care, the usual set operations (eg. Cardinality).
- For example

$$\#weekday = \#\langle mon, tue, wed, thu, fri \rangle = 5$$

Sequence Exercise

Question

- Consider a scenario concerning aircraft final approach to the runway at an airport.

  " Aircraft approach an airport in a random pattern. When an airport is busy, arriving aircraft are queued before being instructed to make their final approach. All aircraft must join at the end of the queue on final approach to the runway"

- You are required to model the queue of aircraft on their final approach to the runway.

Question

- Suppose you are given one basic type,

  [FLIGHTID] - the set of flight ID that uniquely identify each aircraft

  a limit to the number of aircraft that can be queued for landing on the final approach:

  $| \ limit : \mathbb{N}$

  and a state space schema, ***Queue***:

  $$\begin{array}{|l}\hline \multicolumn{1}{l}{\textit{Queue}} \\ \hline queue : iseq\ FLIGHTID \\ \hline \end{array}$$

Question

- Initially there are no aircraft in the queue for landing on the final approach

$$
\begin{array}{|l}
\hline
\textit{InitQueue} \\
\hline
\textit{Queue} \\
\hline
\textit{queue} = \langle\rangle \\
\hline
\end{array}
$$

Question

- Write a schema to allow the aircraft joining the rear of the queue for the final approach.

Question

- Write a schema to allow the aircraft leaves the front of the queue when on the final approach.

Question

- Assume that the schemas must be refined to deal with error scenarios to produce robust schema for the system. Given the response for the error handling for the aircraft as below:

RESPONSE ::= success | exceedLimit | notExist | alreadyExist | queueNotEmpty | queueIsEmpty | notTheFirst

Question

- Write an error schema for every error in the schema **JoinQueue**. Then, prepare a complete function which caters for all possible violations of the precondition in the schema.

Question

- Write an error schema for every error in the schema **LeaveQueue**. Then, prepare a complete function which caters for all possible violations of the precondition in the schema.

Bag

Bag

- We have seen that sets are unordered collections of items, which do not contain duplicates (repetitions).
- A sequence is an ordered collection of items, that **may** contain duplicates.
- A ***bag*** is an unordered collection of items that may contain duplicates.
- Bags are sometimes called ***multi-sets***.

	Ordered?	Duplicates?
Set	N	N
Sequence	Y	Y
Bag	N	Y

Bag

- Example: The set of all bags over type T:

  bag T

- *bag* my be enumerated, by listing their contents using ⟦⟧.
- For example:

  num : bag ℕ

  then num == ⟦ 1, 1, 2, 3 ⟧

  not the same as the set {1, 2, 3}

- However, ⟦ 1, 1, 2, 3 ⟧ = ⟦ 1, 2, 3, 1 ⟧

Bag

- We write ⟦ **a, a, b, b, c, c** ⟧ to denote the bag containing two copies of **a**, two copies of **b**, and two copies of **c**.
- The order in which elements are written is not important.
- The expression

  ⟦ a, a, b, b, c, c ⟧ equals ⟦ a, b, b, a, c, c ⟧

Bag

- If B is a bag of elements from set X, then B may be regarded as a partial function from X to $\mathbb{N}$.
- Any element of X in B is associated with natural number, recoding number of instances in it.
- Example : $[\![a, a, b, b, c, c, c]\!] = \{a \mapsto 2, b \mapsto 2, c \mapsto 3\}$
  - Can be denoted as:

    $$\text{bag } X == X \nrightarrow \mathbb{N}_1$$

  - "==" is read as 'is defined to be'; it gives us a method for naming sets, so that we can subsequently use them.

Bag

- Let PRODUCT be a set of products sold on a store, where [PRODUCT] is a type.
- A set of all bags of products can be denoted as:

$$\text{bag PRODUCT} == \text{PRODUCT} \nrightarrow \mathbb{N}_1$$

- Now, stock is variable representing a bag of product in a store

stock : bag PRODUCT

⇨ stock = { glass $\mapsto$ 100, cup $\mapsto$ 200, plate $\mapsto$ 300 }

Types of Bag

- Example type of bags:

bag	- Bags
count, #	- Multiplicity
$\otimes$	- Bag scaling
$[\![\,]\!]$	- Empty bag

Counting Bag

- Suppose we want to know how many times a value *x* occurs in *bag B*, we use **#**.

  B : bag T   x : T

  then the number of times x occurs in B

  B # x

- Example: rainMonth == ⟦jan, jan, feb, dec⟧

  then rainMonth # jan = 2   rainMonth # apr = 0

Counting Bag

- We can use the **count** keyword to count the number of items appear in the bag:
- Example: rainMonth == ⟦jan, jan, feb, dec⟧

  count rainMonth jan = 2

Scaling Bag

- Scale bags means that we want to multiply the contents of the bag. Use the scaling operator $\otimes$.
- Example:

$$rainMonth == [\![jan, jan, feb]\!]$$

then

$$2 \otimes rainMonth = [\![jan, jan, jan, jan, feb, feb]\!]$$

Scaling Bag

- Some theorems about scaling:

$$n \otimes [\![\ ]\!] = [\![\ ]\!]$$

$$0 \otimes B = [\![\ ]\!]$$

$$1 \otimes B = B$$

$$(n * m) \otimes B = n \otimes (m \otimes B)$$

Bag Membership

- The equivalent of the set membership predicate ∈ is **in** or ⋿.

  item : bag T

  x : T

  then the predicate

  x **in** item   OR   x ⋿ item

  is true iff x appears in item **at least once**
- **x in item** means x is a member of bag item

Sub-bag

- The equivalent of the subset predicate ⊂ or ⊆ is ⊑.

  B_1, B_2 : bag T

  then

  $B_1 \sqsubseteq B_2$

  is true iff each element that occurs in B_1 occurs in B_1 no more often than it occurs in B_2

- $\mathbf{B_1 \sqsubseteq B_2}$ means B_1 is a sub-bag of B_2

Bag Union

- Bag union operator, $\uplus$
- Example:

$$rainMonth == [\![jan, jan, feb]\!]$$

then

$$rainMonth \uplus [\![mar]\!] = [\![jan, jan, feb, mar]\!]$$

$$rainMonth \uplus [\![jan]\!] = [\![jan, jan, jan, feb]\!]$$

Bag Difference/Subtraction

- Bag difference operator, $\uminus$
- Example:

$$rainMonth == [\![jan, jan, feb]\!]$$

then

$$rainMonth \uminus [\![jan]\!] = [\![jan, feb]\!]$$

Making Bag out of Sequence

- Make a bag out of a sequence is by counting up all numbers of times in a sequence using **items**
- Example:

$$\text{items } \langle a, b, a, b, c \rangle = [\![a, a, b, b, c]\!]$$

$$\text{items } \langle a, c, d, a, a \rangle = [\![a, a, a, c, d]\!]$$

Bag Exercise

Question

- Consider a scenario regarding a vending machine where a vending machine will have products and cash in Ringgit Malaysia. Each product can be bought if sufficient cash has been entered and the product is available. The vending machine also can be maintained, so cash and products can be added and removed.

- You are required to model the inventory of the products in the vending machine.

Question

- Given the [PRODUCT] as the set for all products in the vending machine
- And a state space schema called ***Inventory***:

$$
\begin{array}{|l}
\hline
\textit{Inventory} \\
\hline
product : \mathrm{bag}\ PRODUCT \\
price : PRODUCT \rightarrow\!\!\!\!\!+ \ \mathbb{N} \\
\hline
\mathrm{dom}\ product = \mathrm{dom}\ price \\
\hline
\end{array}
$$

Question

- Design a schema called ***InitInventory*** to indicate the vending machine is empty.

$$
\begin{array}{|l}
\hline
\textit{InitInventory} \\
\hline
\textit{Inventory} \\
\hline
\textit{product} = [\!] \\
\textit{price} = \varnothing \\
\hline
\end{array}
$$

Question

- Design a schema where the product will be removed from the vending machine when a customer buys a product.

Question

- Design a schema where the product will need to restocked. The inventory is topped by a new bag of products.

Question

- Design a schema to change the price of the product.

Question

- Design a schema that will display the total of a product in the vending machine.

Question

- Assume that the schemas must be refined to deal with error scenarios to produce robust schema for the system. Given the response for the error handling for the inventory as below:

MESSAGE ::= success | productExist | productNotExist | itemExist | itemNotExist | priceExist | priceNotExist

Question

- Write an error schema for every error in the schema **Restock**. Then, prepare a complete function which caters for all possible violations of the precondition in the schema.

Question

- Write an error schema for every error in the schema **UpdatePrice**. Then, prepare a complete function which caters for all possible violations of the precondition in the schema.

Question

- Write an error schema for every error in the schema **CountProduct**. Then, prepare a complete function which caters for all possible violations of the precondition in the schema.

Question

- Write an error schema for every error in the schema **RemoveProduct**. Then, prepare a complete function which caters for all possible violations of the precondition in the schema.

Summary

- This chapter discussed in details about the extended of Z structure using sequence and bag.
- At the end of each structure, exercises are given and explained to grasp the concepts of the structures.

THANK YOU!!