COMP1216
Lists and queues

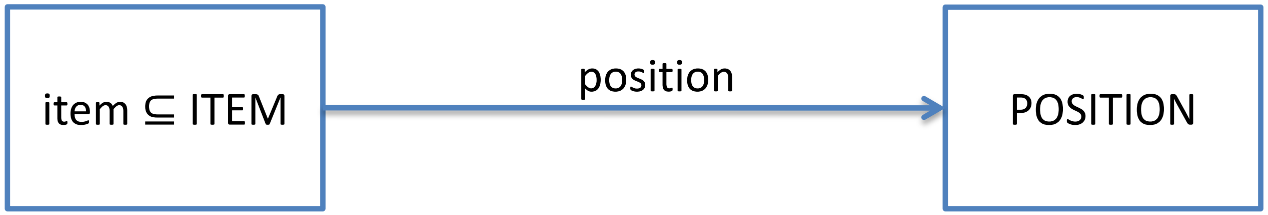
Michael Butler

27 April 2020

Ordered Collections

- Set: unordered collection
- List: ordered collection
  - Items have a position in a list
  - Positions are ordered
  - We can use an ordered set to define positions, e.g., natural numbers

Modelling a list of items



@axm1 $\quad$ POSITION = ℕ

Position as natural number:

first item has lowest position number

@inv1 $\quad$ item ⊆ ITEM

@inv2 $\quad$ position ∈ item ⟶ POSITION

Functional mapping: each item has a single position

BUT: this allows two different items to have the same position ☹

Injective Functions

One-to-one function: different domain elements are mapped to different range elements.
In other words, inverse is also a function.

To declare f as an injective function:

$$\boxed{f \in X \rightarrowtail\!\!\!\!\!\!+\; Y}$$

This is defined in terms of the inverse of f as follows:

Predicate	Definition
$f \in X \rightarrowtail\!\!\!\!\!\!+\; Y$	$f \in X \nrightarrow Y \land f^{-1} \in Y \nrightarrow X$

Total Injective Functions

Just as for standard total functions, we can declare an injective funciton to be total on some set.

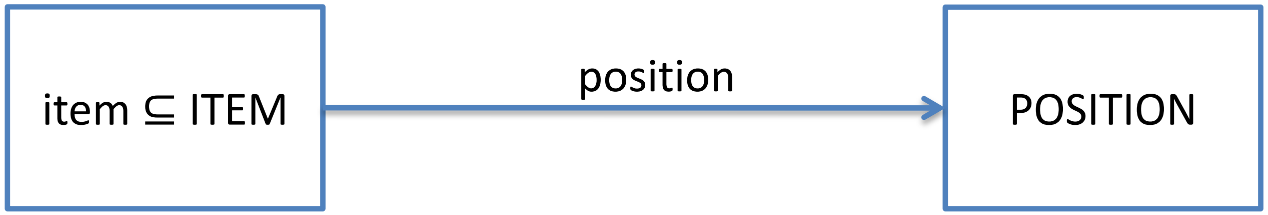
To declare f as a total injective function:

$$\boxed{f \in S \rightarrowtail Y}$$

This is defined i as follows:

Predicate	Definition
$f \in S \rightarrowtail Y$	$f \in S \mapsto\!\!\!\!\!+ Y \wedge dom(f) = S$

List positions are injective



@inv1 item $\subseteq$ ITEM

@inv2 position $\in$ item $\rightarrowtail$ POSITION

Injective mapping: each item has a single position and different jobs cannot be at the same position 🙂

Queues

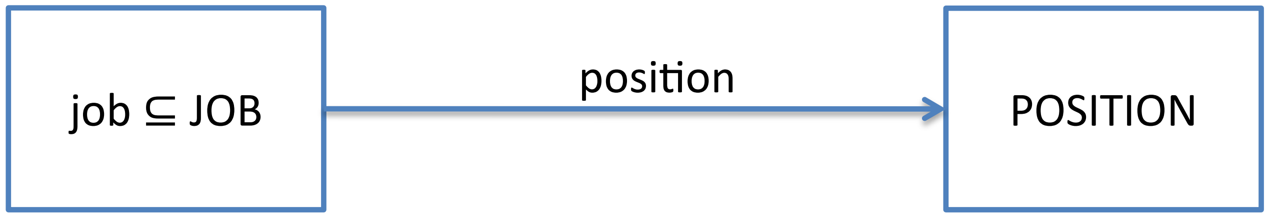
- Queue’s are useful for managing access to shared resources in a fair way
- Physical queue, e.g., queue for check-in desk at airport
- Virtual queue, e.g., queue for aircraft landing slot
- Queue can be viewed as a list
- Queues are very common in computing to manage access to shared resources such as a CPU, memory, disk, communications channel

Modelling a Printer Queue

- Queue: used to control access to some resource, e.g., printer

- First-in first-out (FIFO): jobs should be serviced in the order in which they are placed on a queue

Relating jobs and queue positions



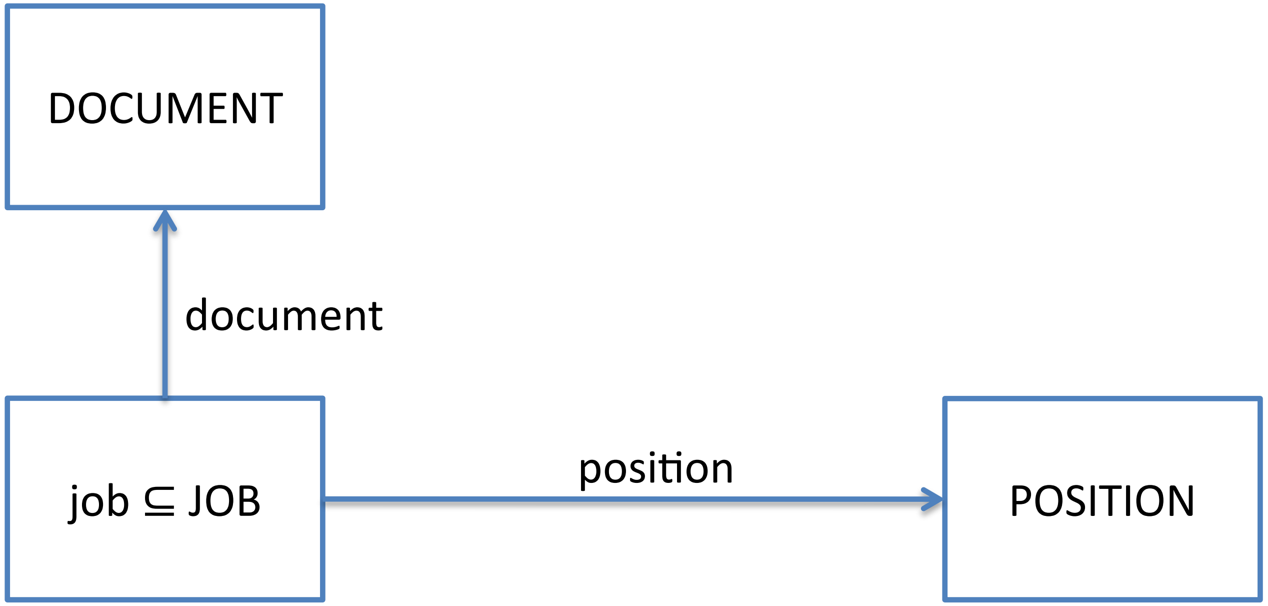
@axm1 POSITION = ℕ

Position as integer: more recent items have higher position number

@inv1 job ⊆ JOB
@inv2 position ∈ job ↣ POSITION

Injective mapping: each job has a single position and different jobs cannot be at the same position

Associating documents with jobs



@inv1 $job \subseteq JOB$

@inv2 $position \in job \rightarrowtail POSITION$

@inv3 $document \in job \rightarrow DOCUMENT$

Adding a job to the queue

```
event QueueJob
  any j d p
  where
    @grd1  j ∈ JOB \ job
    @grd2  d ∈ DOCUMENT
    @grd3  p ∈ POSITION
    @grd4  p is greater than all current positions in the queue
  then
    @act1  job ≔ job ∪ { j }
    @act2  document(j) ≔ d
    @act3  position(j) ≔ p
  end
```

Adding a job to the queue

```
event QueueJob
  any j d p
  where
    @grd1  j ∈ JOB \ job
    @grd2  d ∈ DOCUMENT
    @grd3  p ∈ POSITION
    @grd4  ∀k· k∈job   ⇒  p > position(k)
  then
    @act1  job ≔ job ∪ { j }
    @act2  document(j) ≔ d
    @act3  position(j) ≔ p
  end
```

FIFO removal of job from queue

```
event FifoRemove
  any j
  where
    @grd1  $j \in job$
    @grd2  j is at the lowest position in the queue
  then
    @act1  $job := job \setminus \{ j \}$
    @act2  $document := \{ j \} \lhd document$
    @act3  $position := \{ j \} \lhd position$
  end
```

FIFO removal of job from queue

```
event FifoRemove
  any j
  where
    @grd1  j ∈ job
    @grd2  ∀k· k∈job  ⇒  position(j) ≤ position(k)
  then
    @act1  job ≔  job \ { j }
    @act2  document ≔  { j } ⩤ document
    @act3  position ≔  { j } ⩤ position
  end
```

FIFO and LIFO

- First-in first-out (FIFO): items that arrive earlier in the queue are removed earlier

- Last-in first-out (LIFO): items that arrive later in the queue are removed earlier
  - a LIFO queue is also referred to as a *stack*

LIFO removal of job from queue

```
event LifoRemove
  any j
  where
    @grd1  j ∈ job
    @grd2  j is at the highest position in the queue
  then
    @act1  job ≔  job \ { j }
    @act2  document ≔  { j } ⩤ document
    @act3  position ≔  { j } ⩤ position
  end
```

LIFO removal of job from queue

```
event LifoRemove
  any j
  where
    @grd1  j ∈ job
    @grd2  ∀k· k∈job  ⇒  position(j) ≥ position(k)
  then
    @act1  job ≔  job \ { j }
    @act2  document ≔  { j } ⩤ document
    @act3  position ≔  { j } ⩤ position
  end
```

Recap

- Lists can be modelled as an injective functions between items and integers
  - representing the position of the items

- Position can be used to select the highest or lowest position