Rufous

Ben Simner

How to choose?

Queue

```
empty :: Q a
snoc :: Q a -> a -> Q a
head :: Q a -> a
tail :: Q a -> Q a
```

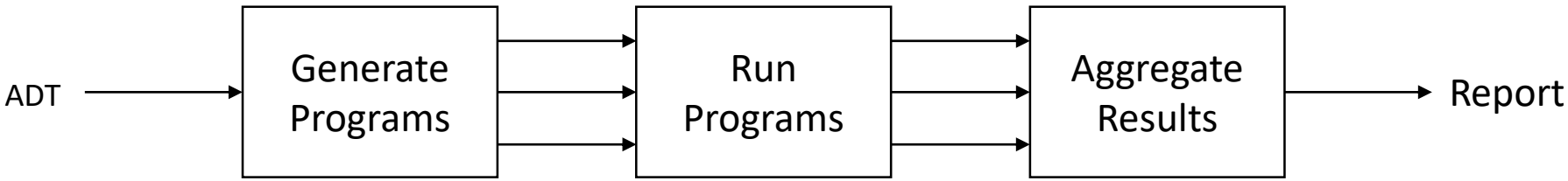
Possible Implementations…

- Linked List
- Banker's Queue
- Physicist's Queue

How to choose?

- Pick easiest to implement …
- Pick most complicated …
- Pick best complexity …
- Write benchmarks …

Rufous!


ADT
Generate Programs
Run Programs
Aggregate Results
Report

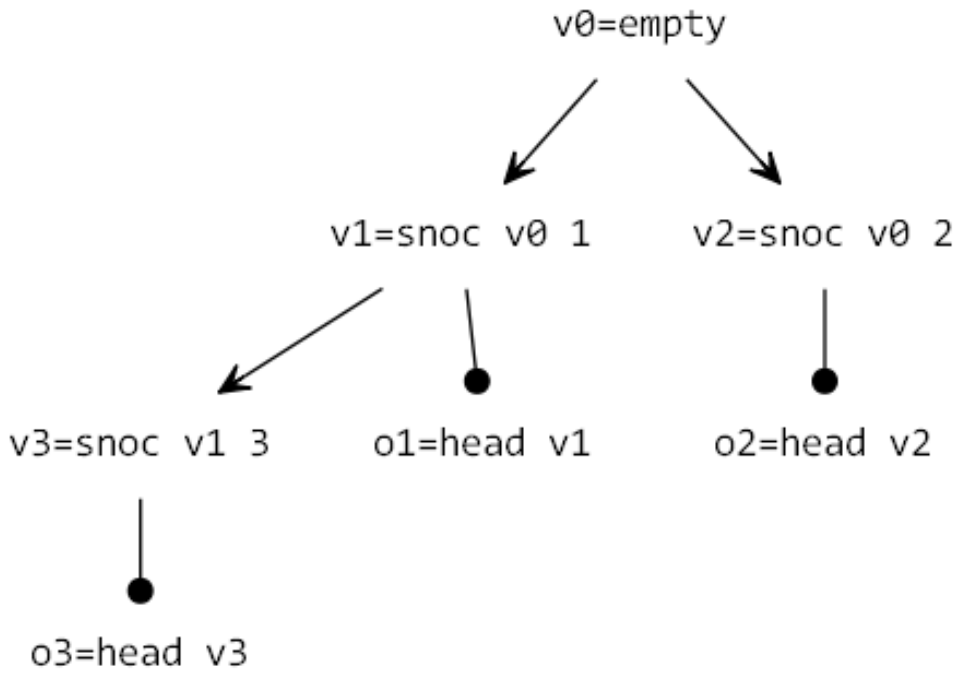
A Program

```
v0 = empty
v1 = snoc v0 1
v2 = snoc v0 2
v3 = snoc v1 3
o1 = head v1
o2 = head v2
o3 = head v3

main = print (o1 + o2 + o3)
```

A Program

```
v0 = empty
v1 = snoc v0 1
v2 = snoc v0 2
v3 = snoc v1 3
o1 = head v1
o2 = head v2
o3 = head v3

main = print (o1 + o2 + o3)
```

The DUG:



Rufous API

Defining the ADT

```
class Queue q where
   empty :: q a
   snoc :: q a -> a -> q a
   head :: q a -> a
   tail :: q a -> q a
```

Rufous API

Defining the ADT

```
class Queue q where
   empty :: q a
   snoc :: q a -> a -> q a
   head :: q a -> a
   tail :: q a -> q a
```

Defining the Implementation

```
instance Queue [] where
   empty       = []
   snoc xs x   = xs ++ [x]
   head (x:_)  = x
   tail (_:xs) = xs
```

Preconditions

Undefined Applications

- `head empty`
- `tail empty`

Preconditions

Undefined Applications

- `head empty`
- `tail empty`

Shadow Implementations

```
newtype ShadowQueue a = S Int


instance Queue ShadowQueue where
   empty = S 0
   snoc (S n) _ = S (n + 1)
   tail (S n) | n > 0 = S (n – 1)
   tail (S n) | n == 0 = guardFailed
   head (S n) | n > 0 = S n
   head (S n) | n == 0 = guardFailed
```

Running Rufous

	empty	head	snoc	tail	mortality	pmf	pof	ListQueue	BQueue	RQueue
A	2823	751	1283	593	0.636	0.153	0.421	9.005ms	7.159ms	**6.936ms**
B	504	90	204	82	0.694	0.152	0.379	0.685ms	0.598ms	**0.528ms**
C	10237	732	786	239	0.927	0.22	0.613	13.553ms	**12.243ms**	12.455ms
D	4863	1691	4471	1662	0.48	0.069	0.265	**22.809ms**	41.630ms	59.152ms
E	77	23	57	21	0.571	0.072	0.172	0.128ms	0.135ms	**0.108ms**

Tabular output

- *pmf, pof* denote sharing (breadth)
- *mortality* denotes lifespan (depth)
- RQueue wins (as predicted by literature!)

Evaluation

Good

- Class-based API
- Fair generation of test programs
- Portable, single language implementation.

Bad

- Extracting DUGs from programs difficult
- Implementation inefficient
- Hard to use tabular output

Related Work

Auburn (2000) defined the DUG

- Very similar to Rufous.
- Older, no longer compiles.
- Not user-friendly.

The Future

Towards Full Automation!

- Easier Extraction of DUGs from Programs
- Profile and Select best structure as code is running.

References

- Queue implementations from “Chris Okasaki. *Purely functional data structures*. Cambridge University Press, 1999”.
- Auburn described in “Graeme E Moss. *Benchmarking Purely Functional Data Structures*, PhD thesis, University of York, 2000”