PRECOG™

Frontier Developers

6:00: Pizza & Networking

6:20: Presentation

Fun with Automata!

John A. De Goes
@jdegoes

http://precog.com

Automata

Abstract machines that transition between states based on reading of input.

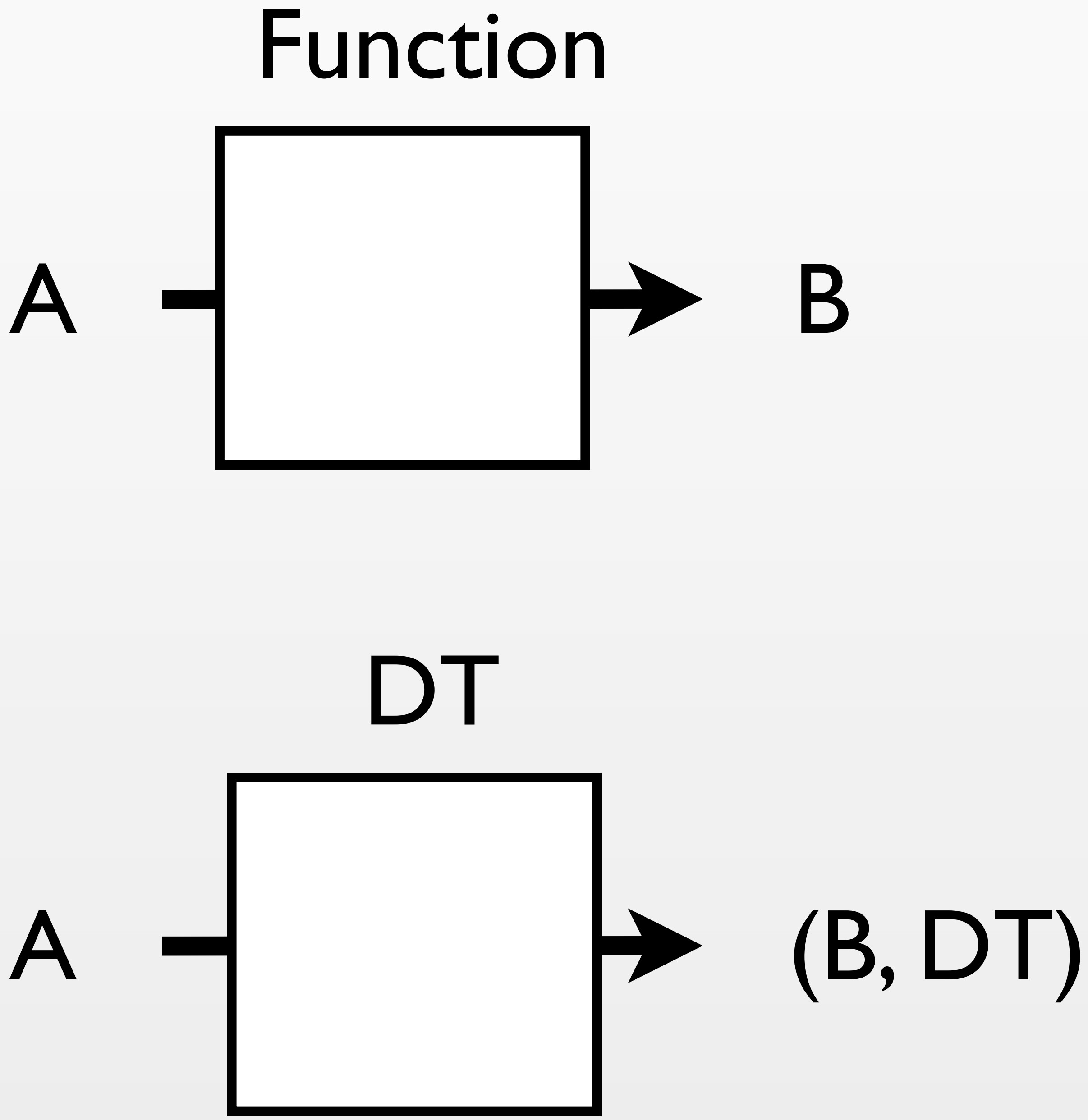
Transducers

Automata that read and write.

Deterministic Transducers

Output deterministically depends on current state and prior input.

DTs vs Functions?



Function

A

B

DT

A

(B, DT)

“Functions with a memory.”

Benefits

- Purely-functional yet stateful
- Incredibly composable
- Safely concurrent
- Recoverable

```
trait DT[A, B] {
  def ! (input: A): (B, DT[A, B])
}
object DT {
  def apply[A, B](f: A => (B, DT[A, B])) =
    new DT[A, B] { def ! (v: A) = f(v) }
}
```

```
val d1 = {
  def f(b: Boolean): DT[Int, Int] =
  DT { v =>
    val output = if (b) v * 2 else v
    (output, f(!b))
  }
  f(false)
}
```

```
val (v1, d2) = d1 ! 1

// v1 = 1

val (v2, d3) = d2 ! 1

// v2 = 2

...
```

Handy-Dandy

`def !! (inputs: A*): (List[B], DT[A, B])`	←	read many
`def >>> [C] (that: DT[B, C]): DT[A, C]`	←	compose
`def map[C](f: B => C): DT[A, C]`	←	output transform
`def contramap[C](f: C => A): DT[C, B]`	←	input transform

`def & [C, D](that: DT[C, D]):`
`    DT[(A, C), (B, D)]`
	←	parallelize

PRECOG™

Handy-Dandy

```
def identity[A]: DT[A, A]

def constant[A, B](b: B): DT[A, B]

def rep[A]: DT[A, (A, A)]
def rep3[A]: DT[A, ((A, A), A)]

def merge[A, B, C](f: (A, B) => C): DT[(A, B), C]
def merge3[A, B, C, D](f: (A, B, C) => D): DT[((A, B), C), D]
```

Exercise 1

Develop DT and some of its helper methods in a language of your choice.

PRECOG™

Exercise 2

Develop a simple streaming analytics framework, with aggregation and windowing.

Exercise 3

Create a recognizer for the language defined by the grammar ab* (that is, one ‘a’ character followed by zero or more ‘b’ characters). The recognizer emits true’s until a character breaks the pattern, then emits false’s continuously.