From Computation to Programming Languages

Ken Q Pu, Faculty of Science, Ontario Tech University

Turing Machine Computation and Programming

Universal TM allows us to express a TM as a program.

So programming is reduced to describing the control logic of the *program TM*.

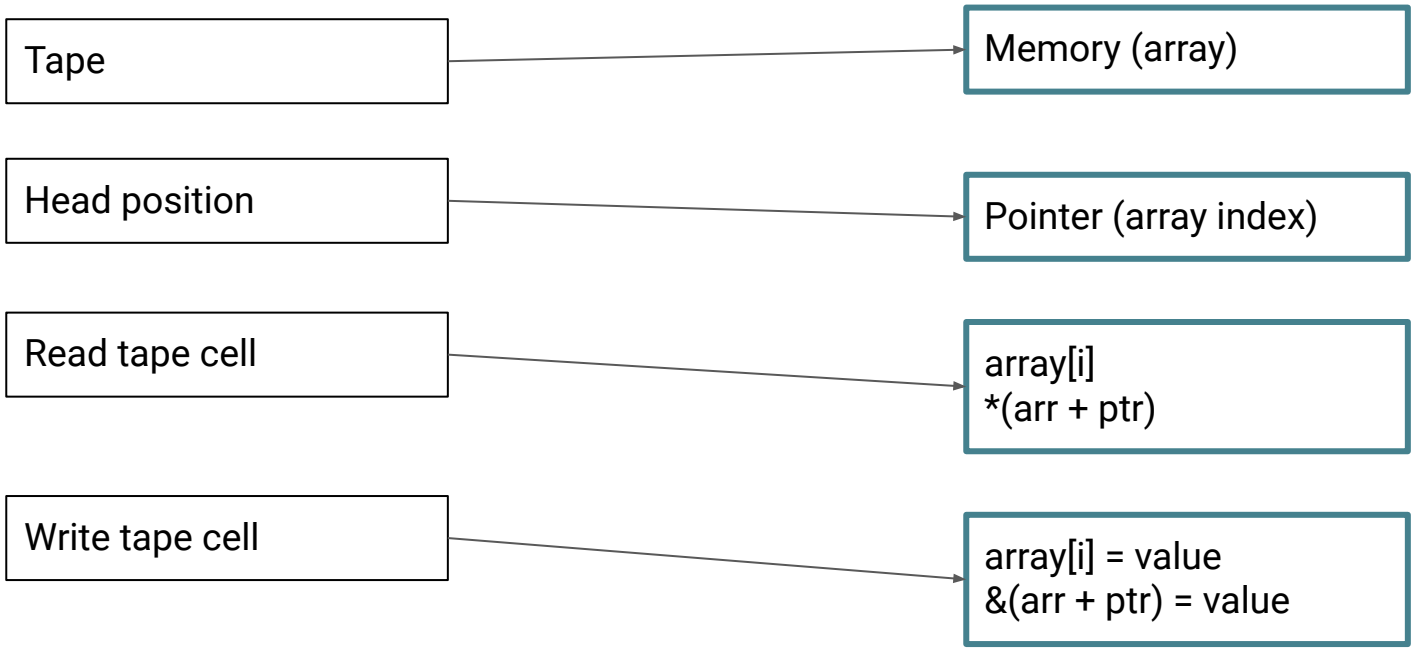
Basic elements of the control logic of any TM:

- States: initial state, halting states
- Transitions:
  - Triggers from the tape content read by the head
  - Command to execute:
    - symbol to write back to the tape
    - movement of the head
  - Next state

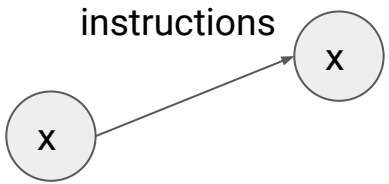
Humanly intractable to author in TM primitives.

Ken Q Pu, Faculty of Science, Ontario Tech University

Abstractions from Turing Machine

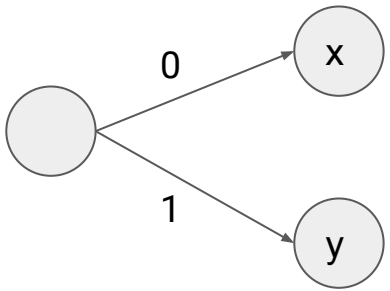


Abstractions from Turing Machine



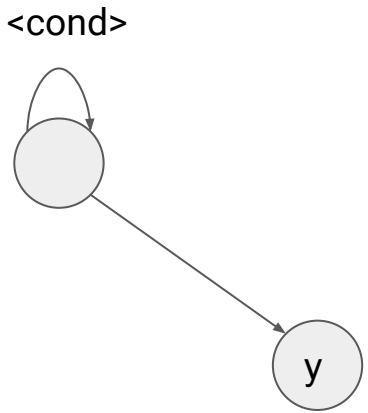

```
<instructions>

GOTO #y
```

Abstractions from Turing Machine



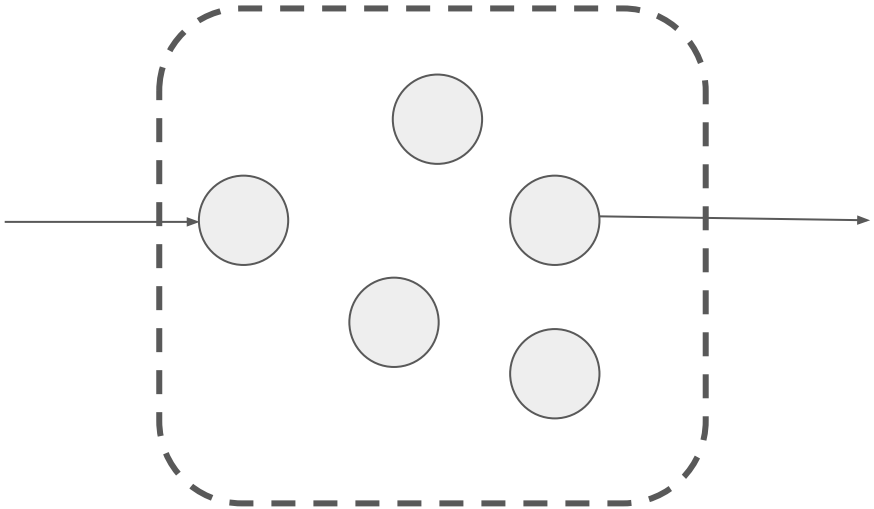

```
if( <condition> ) {
    ...
    GOTO #x
} else {
    ...
    GOTO #y
}
```

Abstractions from Turing Machine



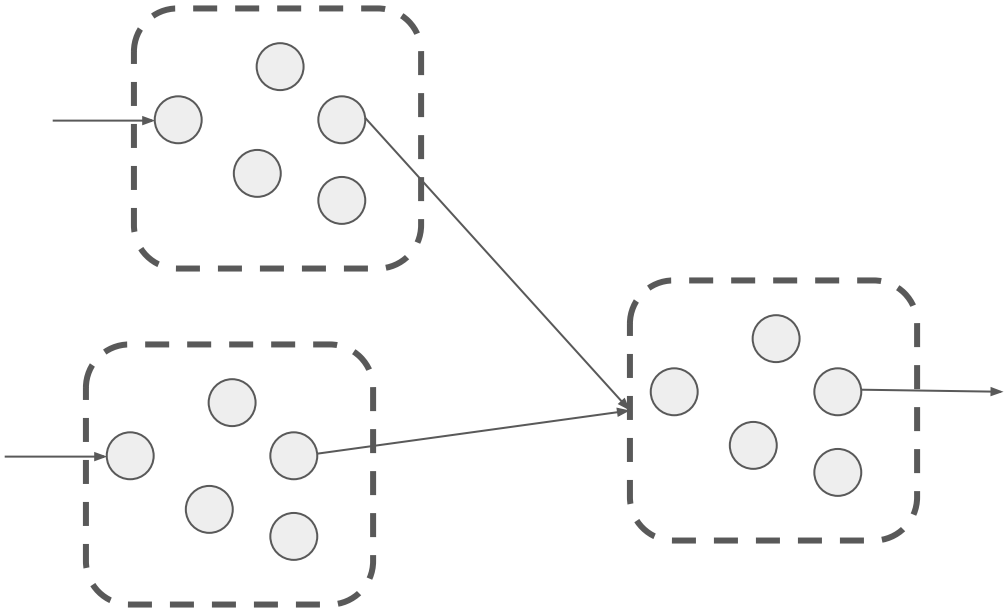

```
while(<cond>) {
    ...
}
```

Abstractions from Turing Machine



```
void f() {
    ...
}
```

Abstractions from Turing Machine



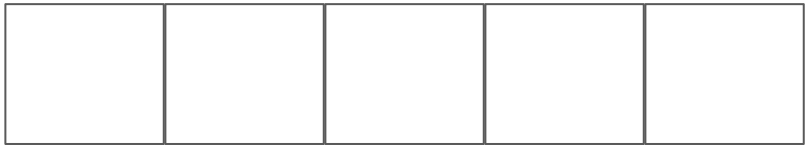
```
g()
f()
f()
g()
```

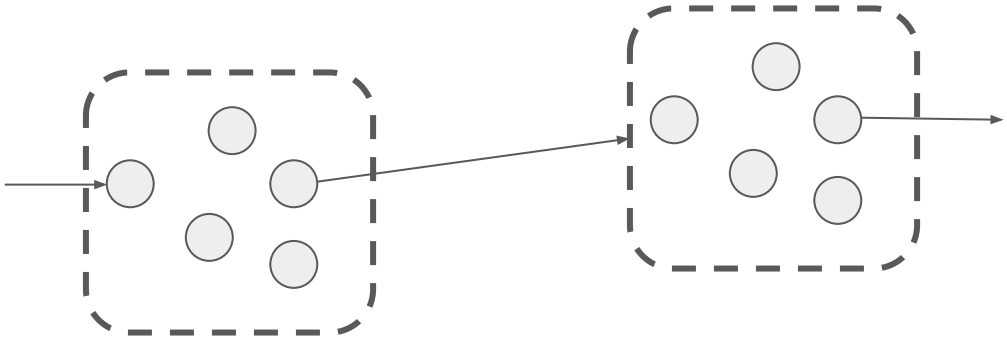
Parameters are data written on the heap portion of the memory.

Return value is data written back on the heap of the caller function.

Ken Q Pu, Faculty of Science, Ontario Tech University

Abstractions from Turing Machine





What's the difference between between values and functions for TM inspired programming languages?

Values are data in tape cells. This is called the **heap region** of the main memory.

Functions are descriptions of control logic structure. They belong to the **code region** of the main memory.

Procedural Programming - Abstraction of TM

So, TM has brought to life various programming constructs that help developers to express a TM.

- memory model with pointers
- mutable variables
- branching
- loops
- function declarations
- function invocations

Lambda Calculus do not have these items...

From Lambda Calculus To Programming

Ken Q Pu, Faculty of Science, Ontario Tech University

Lambda Calculus Programming

LC only has three elements:

- names
- function abstraction
- function application

But LC encodings are very inefficient and humanly impossible to generate error free and scalable code base.

Functional programming is a programming style based on the principles of Lambda Calculus:

- No values are mutable.
- Functions are first-class citizens

Programming Constructs

Succ = \n.\f.\x. f (n f x)

```
def succ(n, f, x) = f(n(f, x))
```

Mult = \m.\n. (m (\x. Succ x) 0)

```
def mult(m, n) = m(\x.Succ(x), 0)
```

Functional programs support:

- Names that are **not** parameters. These names (aka symbols) represent functions (or values).
- Function declarations can have arbitrary parameters.

Programming Constructs

\f.\x. x

\f.\x. f x

\f.\x. f (f x)

...

Built-in constants:
0, 1, 2, ...
3.1415
"Hello world"

Built-in functions:
+ , -, /, *, pow, sin, cos, substr, ...

Functional programming languages come with

- infinitely many constants, aka literals, that represent data without the need of LC expression encodings.
- finitely many built-in functions that perform fast computation without LC expressions.

Convenient consequence of symbols

In pure LC, to implement factorial, we need:

- Define factorial in the form of:
  factorial = H factorial
  where H is a pure LC expression.
- Encode factorial using the Y-combinator:
  (\f. (\x. f (x x)) (\x. f (x x))) H

Since, we have symbol binding in practical functional programming languages, we can use the symbol **factorial** in the body of **factorial**:

```
def factorial (n) =
   if (n == 0) 1 else factorial(n-1) * n
```

History of Lisp

Ken Q Pu, Faculty of Science, Ontario Tech University

Lisp: LISt Processing

John McCarthy developed Listp at MIT in 1958.
"Recursive Functions of Symbolic Expressions and Their Computations by Machine", 1960.

Steve Russell had the first implementation using punched cards on IBM mainframe in 1960.

Garbage collection was added by MIT graduate student (Edwards) in 1962. This feature will not be available to other mainstream languages until Java in 1995.

Lisp is the first **homoiconic** language which is a key property to its macro features.

Lisp through the history

- Scheme in 1975 - now. It's been renamed to Racket.
- Common Lisp, 1984
- Clojure, 2007 - now: Clojure is Lisp running on Java virtual machine (JVM)

Commercial Success:

- iRobot (Scheme)
- Grammarly (Common Lisp, Clojure)
- Walmart Lab (Clojure)
- Boeing (Clojure)
- Cisco (Clojure)
- Netflix (Clojure)

Ken Q Pu, Faculty of Science, Ontario Tech University