Fundamentals (and Lisp Wrap Up)

CSCI 334

Stephen Freund

Garbage Collection

```
~] java -verbose:gc Garbage

[GC 17024K->3633K(83008K), 0.0067267 secs]
[GC 20657K->6988K(83008K), 0.0073014 secs]
[GC 24012K->10505K(83008K), 0.0059666 secs]
...
[GC 121496K->108035K(126912K), 0.0077921 secs]
[Full GC 125059K->110934K(126912K), 0.1330559 secs]
[Full GC 126911K->114224K(126912K), 0.1077395 secs]
[GC 114543K(126912K), 0.0021219 secs]
...
```


Programs As Data and Eval

```
; substitute "to" for "from" in "term"
(defun substitute (to from term)
  (cond ((atom term)
             (cond ((eq term from) to)
                   (t term)))
        (t  (cons (substitute to from (car term))
                  (substitute to from (cdr term))))))


(substitute 3 'w '(+ 1 w))
```

is (+ 1 3)

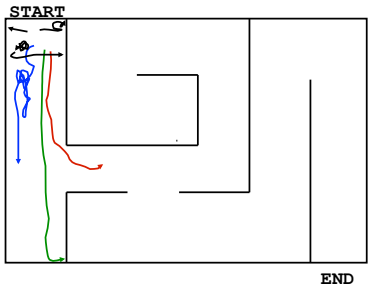
Programs As Data and Eval

```
(defun substitute-and-eval (to from term)
   (eval (substitute to from term)))

(substitute-and-eval '* '+ '(+ 10 2 3))
```

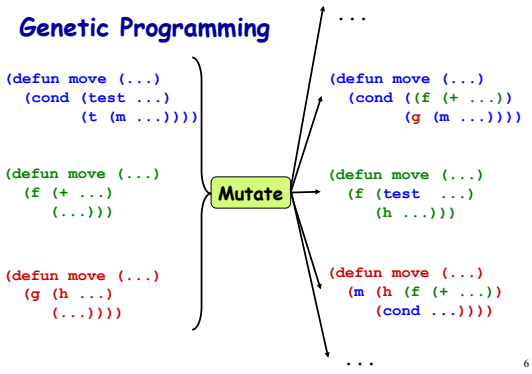
evaluates to 60

```
(derivative '(* 3 x x))  ->  '(* 6 x)

(substitute-and-eval 6
                     'x
                     (derivative '(* 3 x x))) -> 36
```


Genetic Programming



Genetic Programming



Rule Based Systems

```
(rule symptom-predicate diagnosis treatment confidence)

(rule (and (> temp 99) (headache) (cough))
      (flu)
      (take tylenol)
      0.75))

(rule (and (williams-student) (sleeping-in-class))
      (African Trypanosomiasis)
      (prescribe pentamidine)
      1.0))

for rule X:
(if (and (symptoms X) (> (confidence X) 0.5))
    (print (diagnosis X) "-->" (treatment X))
```

Another Example: Yahoo Store Front (Paul Graham)

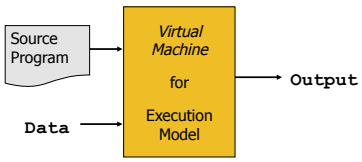
7

Summary

- Successful language
  - symbolic computation, experimental programming
- Specific language ideas
  - expression-oriented: functions and recursion
  - lists as basic data structures
  - programs as data, with universal function **`eval`**
  - idea of garbage collection

8

Running A Program



- VM provides abstract view of hardware:
  - define / use types of data
  - define / use computations over data

9

Level of Abstraction

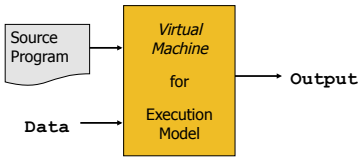
- Concrete (assembly,C,C++,...)

```
movf 0x1233, fp2
mulf #60.0, fp2
movf $8(sp), fp1
addf fp2, fp1
movf fp1, $12(sp)
```

- More abstract (Python,Java,Lisp,...)
  - Lisp: lists, mapcar, higher-order functions
- Very Abstract (LIM,Dylan,FP)
  - FP: M3 = T(M1) x M2

10

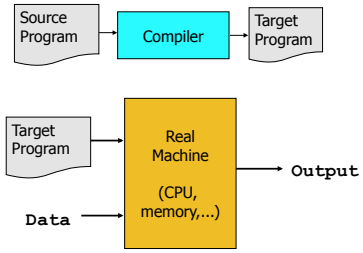
Running A Program



- Syntax is text of program
- Semantics is effect of running the program

11

Program Translation



12

Compiler

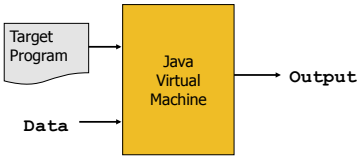

```
if (x == 0) {
  x = x + 1;
}
...
```

```
cmp (1000), $0
bne L
add (1000), $1
L:
...
```

- Target has same semantics as source

13

Program Translation



14

Java Compiler

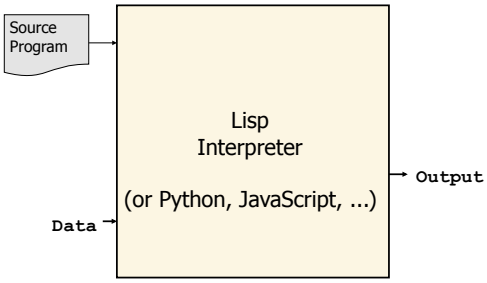

```
if (x == 0) {
  x = x + 1;
}
...
```

```
load 0
ifne L
load 0
inc
store 0
L:
...
```

(compare compiled C to compiled Java)

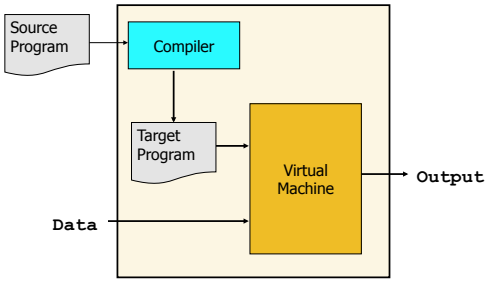
15

Interpreted Languages



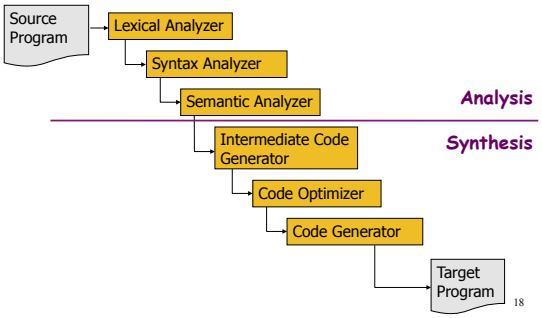
16

Interpreted Languages



17

Typical Compiler



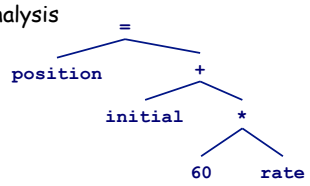
18

Compiler "Front End" Stages

- Input: `"position= initial + rate*60"`
- Lexical Analysis

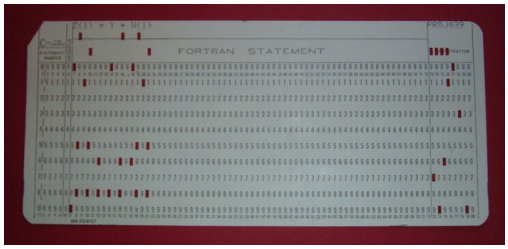
```
position, =, initial, +, rate, *, 60
```

- Syntax Analysis



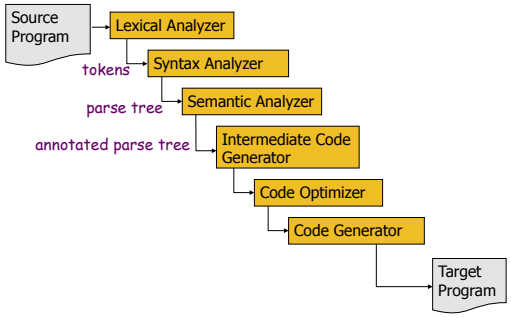
- Semantic Analysis

19



20

Typical Compiler



21

Compiler "Back End" Stages

- Intermediate Code:

```
temp1 = convert_int_to_double(60)
temp2 = mult(rate, temp1)
temp3 = add(initial, temp2)
position = temp3
```

- Optimized Code:

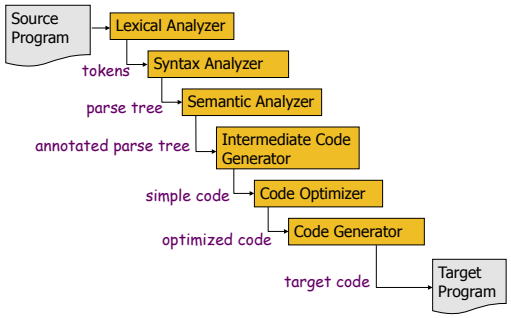
```
temp1 = mult(rate, 60.0)
position = add(initial, temp1)
```

- Generated Machine Code:

```
movf rate, fp2
mulf #60.0, fp2
movf initial, fp1
addf fp2, fp1
movf fp1, position
```

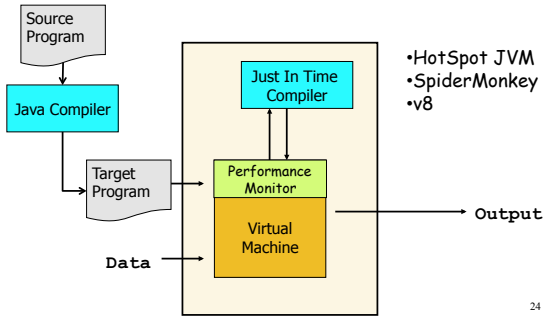
22

Typical Compiler



23

JIT Compilers and Optimization



24