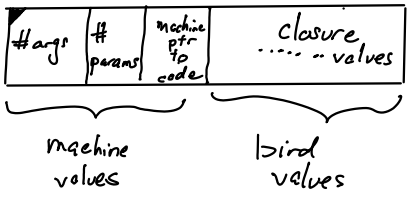
Falcon

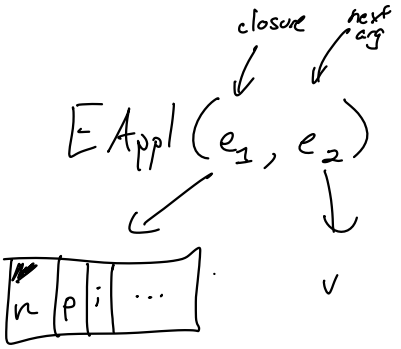
- ★ Partial application
- ★ First-class functions

- • (Anonymous functions)

(Finch)

Closures on heap



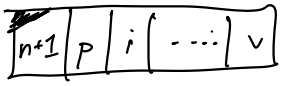


Application algorithm

If $n+1 = p$ Then

copy args from closure into stack
add v into stack
call i

Else



Initial Closures

```
def f x y z =
  x*y+z
end
def g x = x end

... f ...
.....
  let f=2 in
  f+1
```

f ↦ closure_of_f+1

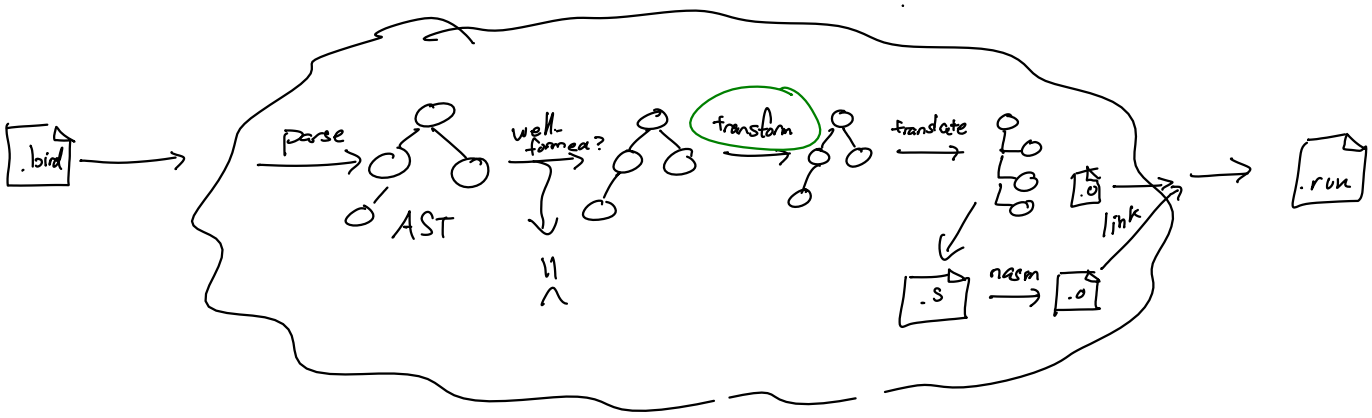
```
section .text
  bird_main:
    .....
    mov rax, closure_of_f+1
    .....
  fn_f:
    .....

section .data
  align 8
  heap_cursor:
    dq 0
  closure_of_f:
    dq 0x8000000000000000, 3, fn_f
  closure_of_g:
    dq 0x8000000000000000, 1, fn_g
```

Anonymous Functions

Finch

```
<expr> ::= ....
         | 'fun' <param_list> '->' <expr>
```

```
def map f lst =
  ....
end
....
  map (fun x->x+1) mylist
....
```



Finch:

```
def twice f x =
  f (f x)
end
twice (fun n->n+1) 4
```

Falcon:

```
def $0 n =
  n+1
end
def twice f x =
  f (f x)
end
twice $0 4
```

```
let rec closure_convert_expr (e:expr) : expr * declaration list =

;;
let closure_convert (p:program) : program =
  ...
;;
```

Finch

```
def twice f n =
  f (f n)
end
let z = false in
let q = 5 in
twice (fun n -> n * q) 4
```

Non-locals are captured in closure if they appear free in the body of the function and are not parameters.

Falcon

```
def $0 q n =
  n * q
end
def twice f n =
  f (f n)
end
let q = 5 in
twice ($0 q) 4
```

let a=2 in
a+b