Introducing Arachne VM (仮)

@raviqqe

Progress from the last meetup

- Function calls
- Tail call optimization
- Miscellaneous bug fixes and optimization

Benchmark

Fibonacci

```
Benchmark 1: target/release/arachne bench/fibonacci/main.arc
  Time (mean ± σ):      4.421 s ±  0.020 s    [User: 4.387 s, System: 0.032 s]
  Range (min … max):    4.390 s …  4.460 s    10 runs

Benchmark 2: python3 bench/fibonacci/main.py
  Time (mean ± σ):      4.589 s ±  0.096 s    [User: 4.536 s, System: 0.040 s]
  Range (min … max):    4.518 s …  4.795 s    10 runs

Relative speed comparison
        1.00          target/release/arachne bench/fibonacci/main.arc
        1.04 ±  0.02  python3 bench/fibonacci/main.py
```

Benchmark

Tak

```
Benchmark 1: target/release/arachne bench/tak/main.arc
  Time (mean ± σ):      1.390 s ±  0.014 s    [User: 1.379 s, System: 0.010 s]
  Range (min … max):    1.370 s …  1.415 s    10 runs

Benchmark 2: python3 bench/tak/main.py
  Time (mean ± σ):      1.428 s ±  0.007 s    [User: 1.414 s, System: 0.011 s]
  Range (min … max):    1.419 s …  1.439 s    10 runs

Relative speed comparison
        1.00          target/release/arachne bench/tak/main.arc
        1.03 ±  0.01  python3 bench/tak/main.py
```

Benchmark

Sum

```
Benchmark 1: target/release/arachne bench/sum/main.arc
  Time (mean ± σ):      3.050 s ±  0.034 s    [User: 3.009 s, System: 0.024 s]
  Range (min … max):    2.993 s …  3.099 s    10 runs

Benchmark 2: python3 bench/sum/main.py
  Time (mean ± σ):      2.343 s ±  0.026 s    [User: 2.322 s, System: 0.017 s]
  Range (min … max):    2.308 s …  2.384 s    10 runs

Relative speed comparison
        1.30 ±  0.02  target/release/arachne bench/sum/main.arc
        1.00          python3 bench/sum/main.py
```

- Tail call elimination?
  - Not eligible in general (e.g. Scheme)

Arachne VM

- Stack machine
- For functional programming
  - Immutable values
- Dynamically sized frame
  - Starts from 0
- No global variable
  - The top level is another function scope.
  - Pushing a value onto a stack defines a variable.

Instructions

`Add`, `And`, `Branch`, `Call`, `Close`, `Divide`, `Drop`, `Dump`, `Environment`, `Equal`, `Float64`, `Get`, `GreaterThan`, `GreaterThanOrEqual`, `Integer32`, `Jump`, `Length`,

Instructions

- Pushing constants: `Nil`, `Float64`, `Integer32`, `Symbol`
- Control: `Jump`, `Branch`
- Functions: `Call`, `TailCall`, `Return`
- Closure environment: `Environment`
- Local/global variables: `Peek`
- Array operations: `Get`, `Set`
- Arithmetic operations: `Add`, `Subtract`, `Multiply`, `Divide`
- Boolean operations: `Not`, `And`, `Or`
- Comparison operations: `Equal`, `NotEqual`, `LessThan`, `LessThanOrEqual`, ...

Examples

Closure call

Source

```
(let x 42)
(let f (fn () x))

(f)
```

Examples

Closure call

Bytecodes

```
float64 42
jump 3 # -> A
environment 0 # B
return
peek 0 # A
close c 0 1
peek 0
call 0 # -> B
dump
drop
```

Examples

Tail call

Source

```
(let-rec f (fn () (if 1 42 (f)))

(f)
```

Examples

Tail call

Bytecodes

```
jump 1a # -> A
float64 1 # B
branch a # -> C
float64 42
return
peek 0 # C
tail_call 0 # -> B
close 3 0 0 # A
peek 0
call 0 # -> B
dump
drop
```

Questions or feedback?

Next tasks...

- May implement delimited continuations
  - This requires some cleanup and design.
  - `SubContinuation` type vs functions as sub-continuations
    - `prompt` and `control0` as primitives?
  - Single stack for everything vs multiple stacks (temporary values, frame info, prompts?)

Summary

- Building a VM is fun.