Progress report in Pen programming language

September 3, 2022

@raviqqe

Agenda

- Progress report
  - (Full) lambda lifting
  - Snake game
- Next plans

Progress report

(Full) lambda lifting

- Flatten nested functions into global functions.
- Pen supports lifting closures with free variables.
  - So far, it supported only the cases where no free variable exists.
- MIR normalization to the A-normal(-ish) form is also introduced for this change.
  - In the future, it'll be easier to write passes at the MIR level.

References

- Lambda lifting | Wikipedia

(Full) lambda lifting

Algorithm

Before:

```
f = \(x number) number {
  g = \(y string) number {
    # ...
  }

  g(y)
}
```

(Full) lambda lifting

Algorithm

After:

```
f = \(x number) number {
  lifted_g(y, x)
}

lifted_g = \(y string, x number) number {
  # ...
}
```

Benchmark

	Speed up	Heap allocation decrease
Hash map insert	6%	37.7%
Hash map update	5%	38.9%

- Probably, heap allocation is not a bottle neck in those cases...
- The bottle neck might be redundant hash calculation?

Snake game

> Demo

Missing language features

- Pretty printing for debugging
- String concatenation operator
- General list pattern match
    - Currently, Pen can match only a head and a tail.

Next plans

- More applications?
  - Web services
  - Games
- Language features

Summary

- Progress
  - (Full) lambda lifting
  - Snake game
- Next plans

Appendix

Benchmark results

```
> hyperfine -w 3 ./insert-*
Benchmark 1: ./insert-new
  Time (mean ± σ):     248.7 ms ±   2.5 ms    [User: 180.5 ms, System: 17.9 ms]
  Range (min … max):   245.4 ms … 252.8 ms    12 runs

Benchmark 2: ./insert-old
  Time (mean ± σ):     261.1 ms ±   3.5 ms    [User: 193.0 ms, System: 17.6 ms]
  Range (min … max):   256.9 ms … 269.2 ms    11 runs

Summary
  './insert-new' ran
    1.05 ± 0.02 times faster than './insert-old'
```

```
> hyperfine -w 3 ./update-*
Benchmark 1: ./update-new
  Time (mean ± σ):     405.6 ms ±   3.2 ms    [User: 338.5 ms, System: 16.1 ms]
  Range (min … max):   401.8 ms … 410.9 ms    10 runs

Benchmark 2: ./update-old
  Time (mean ± σ):     431.1 ms ±   4.6 ms    [User: 360.9 ms, System: 19.1 ms]
  Range (min … max):   422.6 ms … 438.2 ms    10 runs

Summary
  './update-new' ran
    1.06 ± 0.01 times faster than './update-old'
```

```
> valgrind ./insert-old
==595278== Memcheck, a memory error detector
==595278== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==595278== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==595278== Command: ./insert-old
==595278==
==595278==
==595278== HEAP SUMMARY:
==595278==     in use at exit: 10,784 bytes in 74 blocks
==595278==   total heap usage: 1,073,769 allocs, 1,073,695 frees, 50,359,170 bytes allocated
==595278==
==595278== LEAK SUMMARY:
==595278==    definitely lost: 0 bytes in 0 blocks
==595278==    indirectly lost: 0 bytes in 0 blocks
==595278==      possibly lost: 320 bytes in 3 blocks
==595278==    still reachable: 10,464 bytes in 71 blocks
==595278==                       of which reachable via heuristic:
==595278==                         newarray           : 536 bytes in 2 blocks
==595278==         suppressed: 0 bytes in 0 blocks
==595278== Rerun with --leak-check=full to see details of leaked memory
==595278==
==595278== For lists of detected and suppressed errors, rerun with: -s
==595278== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
> valgrind ./insert-new
==597282== Memcheck, a memory error detector
==597282== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==597282== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==597282== Command: ./insert-new
==597282==
==597282==
==597282== HEAP SUMMARY:
==597282==     in use at exit: 10,784 bytes in 74 blocks
==597282==   total heap usage: 669,140 allocs, 669,066 frees, 34,174,010 bytes allocated
==597282==
==597282== LEAK SUMMARY:
==597282==    definitely lost: 0 bytes in 0 blocks
==597282==    indirectly lost: 0 bytes in 0 blocks
==597282==      possibly lost: 320 bytes in 3 blocks
==597282==    still reachable: 10,464 bytes in 71 blocks
==597282==                       of which reachable via heuristic:
==597282==                         newarray           : 536 bytes in 2 blocks
==597282==         suppressed: 0 bytes in 0 blocks
==597282== Rerun with --leak-check=full to see details of leaked memory
==597282==
==597282== For lists of detected and suppressed errors, rerun with: -s
==597282== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
> valgrind ./update-old
==599735== Memcheck, a memory error detector
==599735== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==599735== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==599735== Command: ./update-old
==599735==
==599735==
==599735== HEAP SUMMARY:
==599735==     in use at exit: 10,784 bytes in 74 blocks
==599735==   total heap usage: 2,083,027 allocs, 2,082,953 frees, 82,655,426 bytes allocated
==599735==
==599735== LEAK SUMMARY:
==599735==    definitely lost: 0 bytes in 0 blocks
==599735==    indirectly lost: 0 bytes in 0 blocks
==599735==      possibly lost: 320 bytes in 3 blocks
==599735==    still reachable: 10,464 bytes in 71 blocks
==599735==                       of which reachable via heuristic:
==599735==                         newarray           : 536 bytes in 2 blocks
==599735==         suppressed: 0 bytes in 0 blocks
==599735== Rerun with --leak-check=full to see details of leaked memory
==599735==
==599735== For lists of detected and suppressed errors, rerun with: -s
==599735== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
> valgrind ./update-new
==600256== Memcheck, a memory error detector
==600256== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==600256== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==600256== Command: ./update-new
==600256==
==600256==
==600256== HEAP SUMMARY:
==600256==     in use at exit: 10,784 bytes in 74 blocks
==600256==   total heap usage: 1,273,769 allocs, 1,273,695 frees, 50,285,106 bytes allocated
==600256==
==600256== LEAK SUMMARY:
==600256==    definitely lost: 0 bytes in 0 blocks
==600256==    indirectly lost: 0 bytes in 0 blocks
==600256==      possibly lost: 320 bytes in 3 blocks
==600256==    still reachable: 10,464 bytes in 71 blocks
==600256==                       of which reachable via heuristic:
==600256==                         newarray           : 536 bytes in 2 blocks
==600256==         suppressed: 0 bytes in 0 blocks
==600256== Rerun with --leak-check=full to see details of leaked memory
==600256==
==600256== For lists of detected and suppressed errors, rerun with: -s
==600256== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```