Compilation and bytecode execution

Jay Foad

Dy aloc
Sicily 2015
Compiler

• Introduction / recap
• New features in 14.1
• Results
• Future work
Recap

- Compiler *compiles* defined functions (dfn or tradfn) into *bytecode*
- Bytecode executes more efficiently
- Reduces interpreter overhead
- Speeds up "the invisible glue between the tokens" –Nick Nickolov
- Can speed up your code IF it's working on scalars or small arrays
Recap: limitations

- Fundamental restriction: compiler must be able to resolve names
- ... or at least know their nameclass
Recap: limitations

- Fundamental restriction: the compiler must be able to resolve names
- ... or at least know their namses
- syntactic category
  - array (or niladic function)
  - function
  - monadic operator
  - dyadic operator
Recap: limitations

- Fundamental restriction: compiler must be able to resolve names

\[ f \leftarrow \{ t \leftarrow 1.8 \times \omega \diamond 32 + t \} \]
Recap: limitations

- Fundamental restriction: compiler must be able to resolve names

\[ f \leftarrow \{ t \leftarrow 1.8 \times \omega \oplus 32 + t \} \]
Recap: limitations

- Fundamental restriction: compiler must be able to resolve names

\[ f \leftarrow \{ A \ B \ C \} \]
Recap: limitations

- Fundamental restriction: compiler must be able to resolve names

\[ f \leftarrow \{ A \  B \ C \} \]

? ? ? ?
Recap: limitations

\[
f \leftarrow \{ \\
\quad a \leftarrow \alpha \\
\quad b \leftarrow \emptyset \omega \\
\quad a \\
\}\]
Recap: limitations

\[
f \leftarrow \{ \\
\quad a \leftarrow \alpha \\
\quad ? \quad b \leftarrow \phi \omega \\
\quad a
\}
\]
Recap: UI

• Compiler is controlled by an I-beam:

\[
cc \leftarrow 400 I \quad \text{A compiler control}
\]
Compiler in 14.0

cc ← 400I

1 cc'foo' A is foo compiled?
2 cc'foo' A compile foo
3 cc'foo' A uncompile foo
Compiler in 14.1

cc ← 400I

1 cc'foo' A is foo compiled?
2 cc'foo' A compile foo
3 cc'foo' A uncompile foo
4 cc'foo' A show bytecode
# cc'foo' A compile foo with
   A   global names
Compiler in 14.1

```
cc ← 400⍴
iscompiled ← 1 ⋄ cc
compile ← 2 ⋄ cc
uncompile ← 3 ⋄ cc
bytecode ← 4 ⋄ cc
```
New features in 14.1

• Global names
• Bytecode display
• Control structures and tradfns
Global names

• In 14.0, any use of a non-local name caused an error

   bar←{⍵*2}
   foo←{bar 1+⍵}
   compile'foo'

   16 0 0 Undefined name: bar

• In 14.1 a *callback* mechanism lets you overcome this
Global names

- TL;DR

```
quadNC ← ⎕NC
quadAT ← ⎕AT
# cc 'foo'
∧ (success)
```
Global names

- What happened there?
- Left argument of I-beam is a **namespace**
- Namespace contains well-known named **callback functions**
- Compiler uses them to resolve unknown (non-local) names
Global names

Callbacks quadNC and quadAT are called with a single enclosed name:

\[
\text{quadNC} \subset 'bar' \\
\text{quadAT} \subset 'bar'
\]

For quadAT, only the first item of the result (valency) is significant
Why use callbacks?

• To *decouple* the compile-time and run-time environments
  - e.g. when code is loaded dynamically
  - e.g. when you have generated constants

• (To work around some problems)

• If you don't care, use the defaults
Why use callbacks?

\[
\text{quadNC} \leftarrow \{ \\
\quad \omega \in \text{FuncNames}:3.2 \ \land \text{dfn} \\
\quad 0 \\
\}
\]
Why use callbacks?

\[
\text{quadNC} \leftarrow \{
\begin{align*}
'\text{C}' & \mapsto \omega : 2.1 \quad & \text{Constant} \\
'\text{F}' & \mapsto \omega : 3.1 \quad & \text{Function} \\
0 & \end{align*}
\}
\]
Why use callbacks?

quadNC←{  
  ⍵≡⊂'badfunc':0  
  ⍵≡⊂'reallybadfunc':0  
  ⎕NC ⍵  
}
Why use callbacks?

quadNC←⎕NC
Assumptions

• The nameclass of a global name is recorded in the bytecode as an assumption

• Assumptions are checked at run time
Assumptions

- What if the assumption fails?

```
quadNC←{3.1} ⋄ quadAT←{(1 0 0) 0 0 0}
bar←99
foo←{bar ⍵}
# cc'foo'    A compilation succeeds!
foo 3
SYNTAX ERROR: Nameclass of non-local name has changed since compilation
```
Global constants

- Callback function `getValue` can return the `value` of a global constant, enclosed
- ... else θ
- You are promising the compiler that the value won't change
- This assumption is not checked!
Global constants

quadNC←{ }
   'C'≡⊃⍵:2.1 A Constant
   0
}

getValue←{ }
   'C'≡⊃⍵:⊂⍎⍵ A or ⊂⎕OR ⍵
   ∅
}

#dyalog15
Global constants

- Why isn't the assumption checked?
- Because of constant folding

```
C3←1 2 3   ⍝ a constant
foo←{≢C3}
# cc 'foo'
```
Bytecode

- 4(400⅛) dumps the bytecode of a compiled function

Health warning:
- This is for interest and amusement only!
- The bytecode display can and will change at any time!
Bytecode

f←{⍺+⍵}
compile'f'
bytecode'f'

Dump of bytecode for f:
0000: 0000000F // version 15
0001: 00000000 // localised system variables: none
0002: 00000001 // 0 slots
0003: 00000002 // 0 uslots
0004: 00000224 eval 0x02 // +
0005: 00003A12 tokoff 003A
0006: 00000003 ret
Bytecode: slots

g←{(1+ω)/(1−ω)}
compile 'g'
bytecode 'g'

Dump of bytecode for g:
0000: 0000000F // version 15
0001: 00000000 // localised system variables: none
0002: 00000201 // 2 slots
0003: 00000002 // 0 uslots
0005: 00003125 cpy Larg, lst[1]
0006: 00000F46 cpy slot[0], Rarg
0007: 00000324 eval 0x03 // -
0009: 00003125 cpy Larg, lst[1]
000A: 00000E45 mov Rarg, slot[0]
000B: 00002E66 mov slot[1], Rslt
000C: 00000224 eval 0x02 // +
000E: 00006025 mov Larg, Rslt
000F: 00002E45 mov Rarg, slot[1]
0010: 00000524 eval 0x05 // ÷
0012: 00000003 ret
Bytecode: recursion

gcd←{⍵=0:α ⋄ ⍵ ∇ ⍵|⍺}
compile'gcd'
bytecode'gcd'

Dump of bytecode for gcd:
0000: 0000000F // version 15
0001: 00000000 // localised system variables: none
0002: 00000201 // 2 slots
0003: 00000002 // 0 uslots
0004: 0000E26 mov slot[0], Larg
0006: 0004125 cpy Larg, Rarg
0007: 0002E46 mov slot[1], Rarg
0008: 0003145 cpy Rarg, lst[1]
0009: 0001524 eval 0x15 // =
000B: 0006045 mov Rarg, Rslt
000C: 000100F jumpfalse 0010
000D: 0000F65 cpy Rslt, slot[0]
000E: 0000003 ret
0010: 0000F65 cpy Rslt, slot[0]
0011: 0002F25 cpy Larg, slot[1]
0012: 0006045 mov Rarg, Rslt
0013: 0000A24 eval 0xOA // |
0015: 0002E25 mov Larg, slot[1]
0016: 0006045 mov Rarg, Rslt
0017: 0000411 tailrecurse 0004
Bytecode

- Values are moved around in *registers*
  Larg, Rarg, Rslt
- Constants loaded from lst[n]
- Temporaries stored in slot[n]
- Functions executed with eval
Control structures

- In 14.0 the compiler (mostly) just targeted dfns
- In 14.1 both branch (→) and all normal control flow structures are supported
  - →label is special-cased
  - →expression is less efficient
Control structures

\[ n \leftarrow \text{loop } n \]
[1] : Repeat
[2] \[ n \leftarrow n - 1 \]
[3] : Until \[ n = 0 \]

\[ \nabla \]

\begin{align*}
0005: & \quad 00000E46 \text{ mov } \text{slot[0]}, \text{Rarg} \\
0006: & \quad 00000F25 \text{ cpy } \text{Larg, slot[0]} \\
0007: & \quad 00005145 \text{ cpy } \text{Rarg, lst[2]} \\
0008: & \quad 00000324 \text{ eval } 0x03 \quad // - \\
000B: & \quad 00000F66 \text{ cpy } \text{slot[0]}, \text{Rslt} \\
000C: & \quad 00006025 \text{ mov } \text{Larg, Rslt} \\
000D: & \quad 00007145 \text{ cpy } \text{Rarg, lst[3]} \\
000E: & \quad 00001524 \text{ eval } 0x15 \quad // = \\
0010: & \quad 00006045 \text{ mov } \text{Rarg, Rslt} \\
0011: & \quad 0000060F \text{ jumpfalse } 0006 \\
0012: & \quad 00000F65 \text{ cpy } \text{Rslt, slot[0]} \\
0013: & \quad 00000003 \text{ ret}
\end{align*}
Control structures

∇ n←osc n;cond;t
[1] :Repeat
[2] cond←2|n ◆ :If cond
[3] t←1+3×n
[5] t←n÷2
[6] :End ◆ n←t
[7] :Until n=1
∇

Factor of 2.5 speed-up
Coverage

On a large application with 64501 defined functions (0.1% dfns)

- < 1% in 14.0
- 59.15% in 14.1
- 79.05% with indexed assignment
- 83.47% with selective assignment
Coverage

The next top priorities (for this code base)

5428 Non-local assignment
4209 Dotted namespace reference
1003 Execute
876 Unsupported system function: ⎕WG
594 Unsupported system function: ⎕WS
576 Unsupported system function: ⎕NC
396 Unsupported keyword: :With
365 Unsupported keyword: :Trap
...

#dyalog15
Coverage

• But this sample is biased!
• 99.9% tradfns
• For example, no use of right argument namelists:

∇ r←foo(x y z)
[1] r←x+y×z
∇
Future work: coverage

- Indexed assignment
- Selective assignment
- Right argument namelist
- Better support for namespaces
Future work: performance

- Inline single-line dfns
- Better constant folding
- Better support for ⎕ML ⎕IO etc
- More idioms: 0=N| (for all N)
- Scalar loop fusion: A+B×C
Feedback

Please:
- Try it out
- Report failures
- Report successes
- Send us your code!