The Cost of Sugar

Ryan Culpepper

Early in 2018...

I was working on the `crypto` package.

- FFI wrappers for several cryptographic libraries
- common interfaces, convenience functions, etc

And I happened to take a look at the `.zo` files.

```
$ find -name '*.zo' | xargs ls -lh

...  10K ...  ./private/nettle/compiled/cipher_rkt.zo
... 5.1K ...  ./private/nettle/compiled/digest_rkt.zo
...  10K ...  ./private/nettle/compiled/factory_rkt.zo
... 139K ...  ./private/nettle/compiled/ffi_rkt.zo
... 2.7K ...  ./private/nettle/compiled/kdf_rkt.zo
...  34K ...  ./private/nettle/compiled/pkey_rkt.zo

...  25K ...  ./private/common/compiled/common_rkt.zo
...  31K ...  ./private/common/compiled/pk-asn1_rkt.zo
...  48K ...  ./private/common/compiled/pk-common_rkt.zo

... 200K ...  ./compiled/main_rkt.zo
```

There's a tool for answering exactly this kind of question...

The Macro Profiler

est. 2016

```
$ raco macro-profile crypto/private/nettle/ffi
profiling (lib "crypto/private/nettle/ffi.rkt")
Initial code size: 6189
Final code size  : 47637
========================================
Phase 0
define-cstruct (defined in ffi/unsafe)
  total: 19216, mean: 2402
  direct: 15048, mean: 1881, count: 8, stddev: 459.96
define-cpointer-type (defined in ffi/unsafe)
  total: 3522, mean: 104
  direct: 972, mean: 29, count: 34, stddev: 21.52
provide (defined in racket/private/reqprov)
  total: 2664, mean: 2664
  direct: 8, mean: 8, count: 1, stddev: 0
_fun (defined in ffi/unsafe)
  total: 2658, mean: 28
  direct: 2380, mean: 25, count: 96, stddev: 19.72
provide-trampoline (defined in racket/private/reqprov)
  total: 2656, mean: 2656
  direct: 2656, mean: 2656, count: 1, stddev: 0
define-cpointer-pred (defined in ffi/unsafe)
  total: 2108, mean: 124
  direct: 680, mean: 40, count: 17, stddev: 0
define (defined as new-define in racket/private/kw)
  total: 1768, mean: 3
...
```

```
$ raco macro-profile crypto/main
profiling (lib "crypto/main.rkt")
Initial code size: 6217
Final code size  : 123261
========================================
Phase 0
->* (defined in racket/contract/private/arrow-val-first)
  total: 60696, mean: 1785
  direct: 24746, mean: 728, count: 34, stddev: 541.13
λ (defined as new-λ in racket/private/kw)
  total: 20184, mean: 120
  direct: 10584, mean: 63, count: 168, stddev: 113.98
define (defined as new-define in racket/private/kw)
  total: 13850, mean: 51
  direct: 8112, mean: 30, count: 269, stddev: 91.26
provide (defined in racket/private/reqprov)
  total: 13129, mean: 1313
  direct: 188, mean: 19, count: 10, stddev: 0.6
bad-number-of-results (defined in racket/contract/private/arrow-common)
  total: 12880, mean: 56
  direct: 11500, mean: 50, count: 230, stddev: 50
send (defined in racket/private/class-internal)
  total: 12788, mean: 149
  direct: 11402, mean: 133, count: 86, stddev: 12.97
handle-contract-out (defined in racket/contract/private/out)
  total: 12303, mean: 1367
...
```

The Cost of Macros

Term size

Occurrences

Direct cost

Indirect cost

Term size

The **size** of a term is the number of atoms and pairs in it.

```
(x y z)
(x . (y . (z . ()))) ; 7 nodes

(x (y) z)
(x . ((y . ()) . (z . ()))) ; 9 nodes
```

Macro occurrence

A **macro occurrence** refers to a distinct expansion of a macro.

```
(define-syntax-rule (with-output-to-string e)
  (parameterize ((current-output-port
                  (open-output-string)))
    e
    (get-output-string (current-output-port))))

(with-output-to-string
  (for ([i 3]) (printf "hello\n")))
(with-output-to-string
  (printf "is there anybody in there?"))
```

The **parent** of a macro occurrence is the occurrence that produced it.

An occurrence with no parent is a **surface** occurrence.

Direct Cost

The **direct cost** of a macro occurrence is the difference between the input term size and the output term size. *

* `local-expand` complicates everything.

Direct Cost

The **direct cost** of a macro occurrence is the difference between the input term size and the output term size. *

* **`local-expand`** complicates everything.

```
(with-output-to-string
  (printf "is there anybody in there?"))
```

⇒

```
(parameterize ((current-output-port
                (open-output-string)))
  (printf "is there anybody in there?")
  (get-output-string (current-output-port)))
```

Indirect Cost

The **indirect cost** also includes the direct costs of all child occurrences.

```
(with-output-to-string
  (for ([i 3]) (printf "hello\n")))
```

⇒

```
(parameterize ((current-output-port
                (open-output-string)))
  (for ([i 3]) (printf "hello\n"))
  (get-output-string (current-output-port)))
```

Direct vs Indirect Cost

Indirect cost shows what macros contribute most to compiled size.

Direct cost shows concentrated opportunities for simplification.

```
$ raco macro-profile crypto/private/nettle/ffi
profiling (lib "crypto/private/nettle/ffi.rkt")
Initial code size: 6189
Final code size  : 47637
========================================
Phase 0
define-cstruct (defined in ffi/unsafe)
  total: 19216, mean: 2402
  direct: 15048, mean: 1881, count: 8, stddev: 459.96
define-cpointer-type (defined in ffi/unsafe)
  total: 3522, mean: 104
  direct: 972, mean: 29, count: 34, stddev: 21.52
provide (defined in racket/private/reqprov)
  total: 2664, mean: 2664
  direct: 8, mean: 8, count: 1, stddev: 0
_fun (defined in ffi/unsafe)
  total: 2658, mean: 28
  direct: 2380, mean: 25, count: 96, stddev: 19.72
provide-trampoline (defined in racket/private/reqprov)
  total: 2656, mean: 2656
  direct: 2656, mean: 2656, count: 1, stddev: 0
define-cpointer-pred (defined in ffi/unsafe)
  total: 2108, mean: 124
  direct: 680, mean: 40, count: 17, stddev: 0
define (defined as new-define in racket/private/kw)
  total: 1768, mean: 3
...
```

```
$ raco macro-profile crypto/main
profiling (lib "crypto/main.rkt")
Initial code size: 6217
Final code size  : 123261
========================================
Phase 0
->* (defined in racket/contract/private/arrow-val-first)
  total: 60696, mean: 1785
  direct: 24746, mean: 728, count: 34, stddev: 541.13
λ (defined as new-λ in racket/private/kw)
  total: 20184, mean: 120
  direct: 10584, mean: 63, count: 168, stddev: 113.98
define (defined as new-define in racket/private/kw)
  total: 13850, mean: 51
  direct: 8112, mean: 30, count: 269, stddev: 91.26
provide (defined in racket/private/reqprov)
  total: 13129, mean: 1313
  direct: 188, mean: 19, count: 10, stddev: 0.6
bad-number-of-results (defined in racket/contract/private/arrow-common)
  total: 12880, mean: 56
  direct: 11500, mean: 50, count: 230, stddev: 50
send (defined in racket/private/class-internal)
  total: 12788, mean: 149
  direct: 11402, mean: 133, count: 86, stddev: 12.97
handle-contract-out (defined in racket/contract/private/out)
  total: 12303, mean: 1367
...
```

The fix: use helper functions

Use run-time helper functions

```
(define-syntax-rule (with-output-to-string expr ...)
  (parameterize ((current-output-port (open-output-string)))
    expr ...
    (get-output-string (current-output-port))))
```

⇒

```
(define-syntax-rule (with-output-to-string expr ...)
  (call-with-output-string (lambda () expr ...)))
; call-with-output-string : (-> Any) -> String
(define (call-with-output-string proc)
  (define out (open-output-string))
  (parameterize ((current-output-port out)) (proc))
  (get-output-string out))
```

Use compile-time helper functions

```
(define-syntax-rule (define-ffi-definer define-x lib)
  (define-syntax-rule (define-x name type)
    (define name (get-ffi-obj name lib type))))
```

⇒

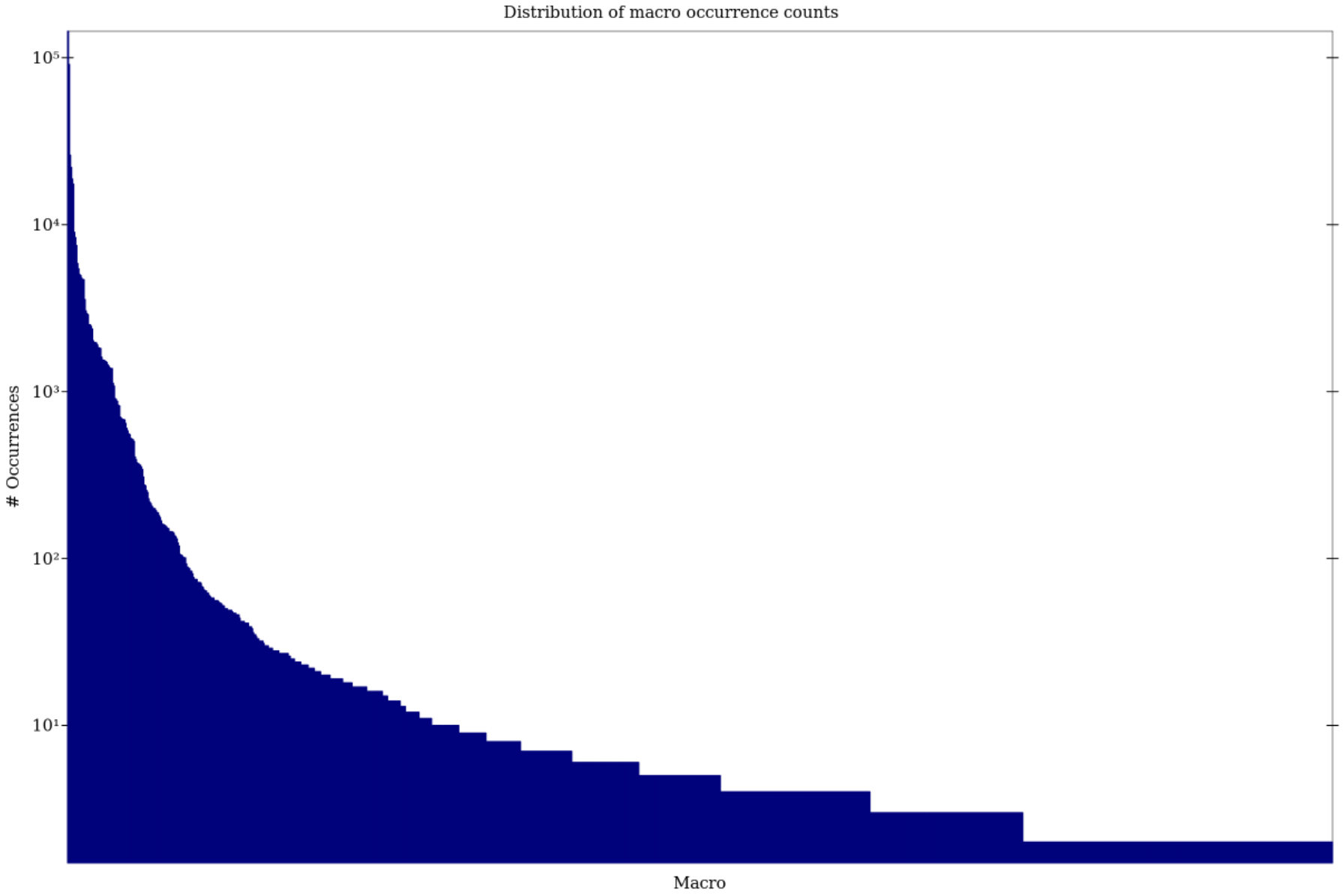
```
(define-syntax-rule (define-ffi-definer define-x lib)
  (define-syntax define-x
    (make-definer-transformer (quote-syntax lib))))
(begin-for-syntax
  ; make-definer-transformer : Syntax -> Syntax -> Syntax
  (define ((make-definer-transformer lib-stx) stx)
    (syntax-case stx ()
      [(_ name type)
       (with-syntax ([lib lib-stx])
         #'(define name (get-ffi-obj name lib type)))])))
```

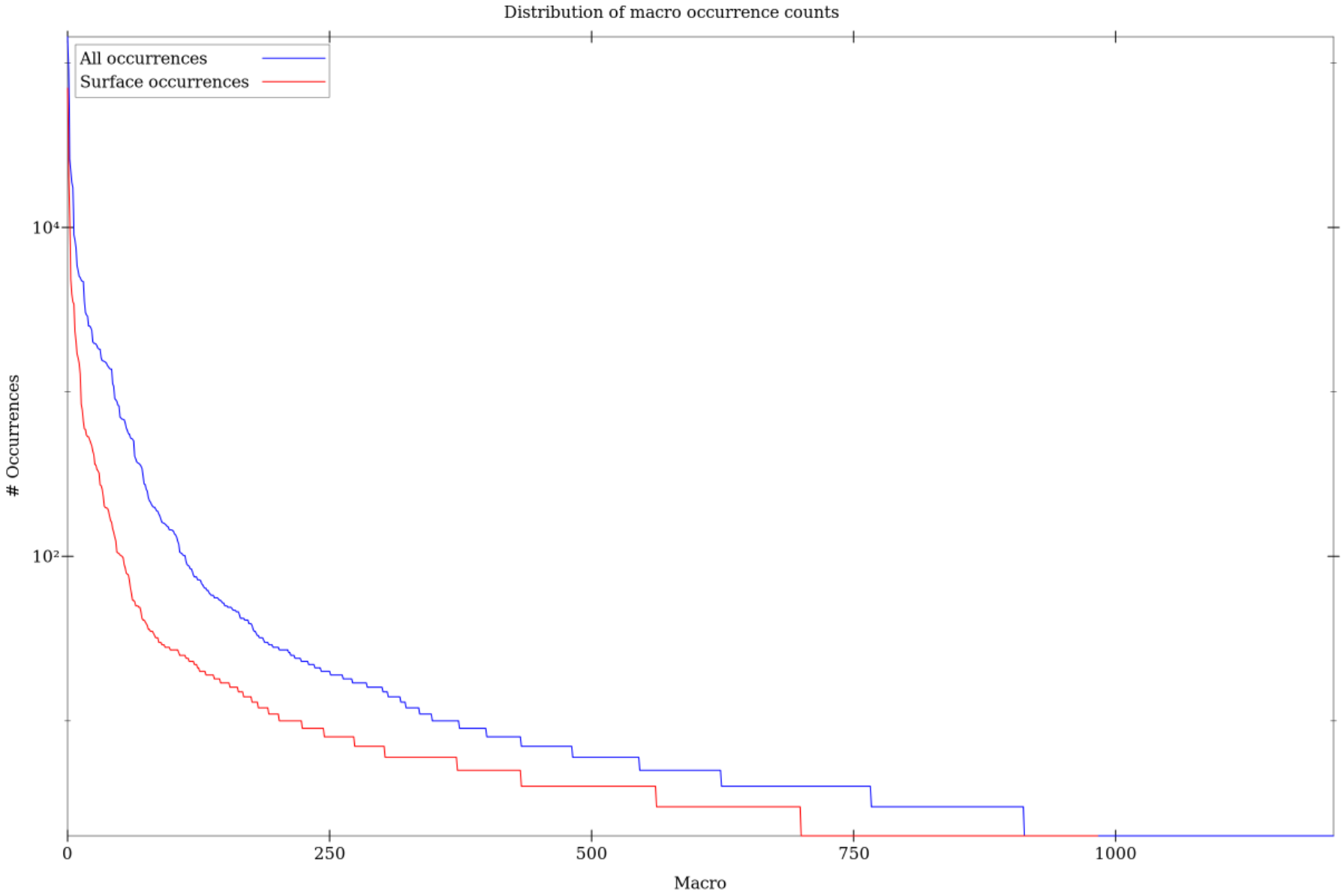
Data

The Racket Macro Ecosystem

In the main distribution

- 28685 distinct macros *
- 8137087 macro occurrences
- 1046512 surface macro occurrences

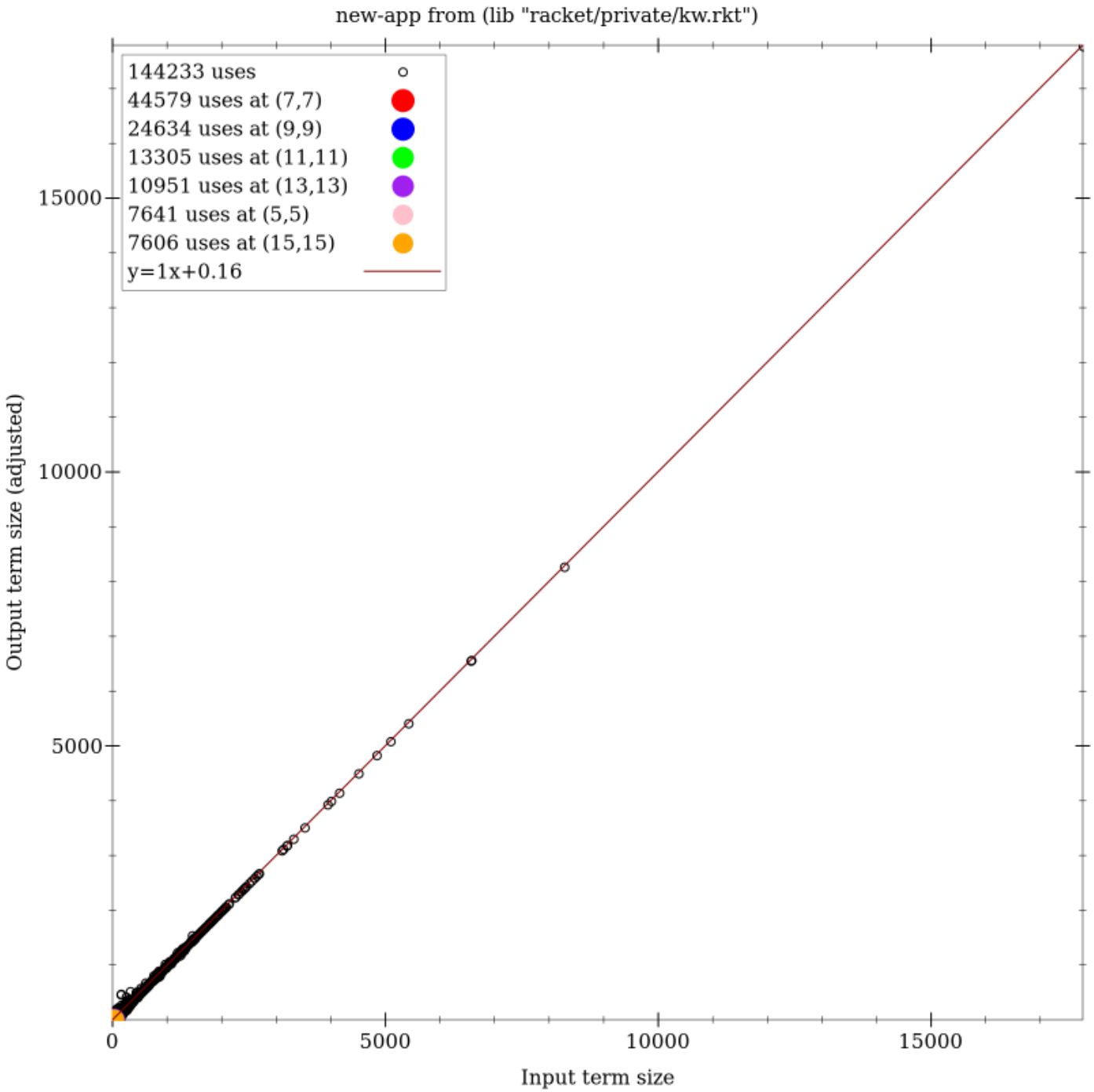

Distribution of macro occurrence counts
Occurrences
10^5
10^4
10^3
10^2
10^1
Macro

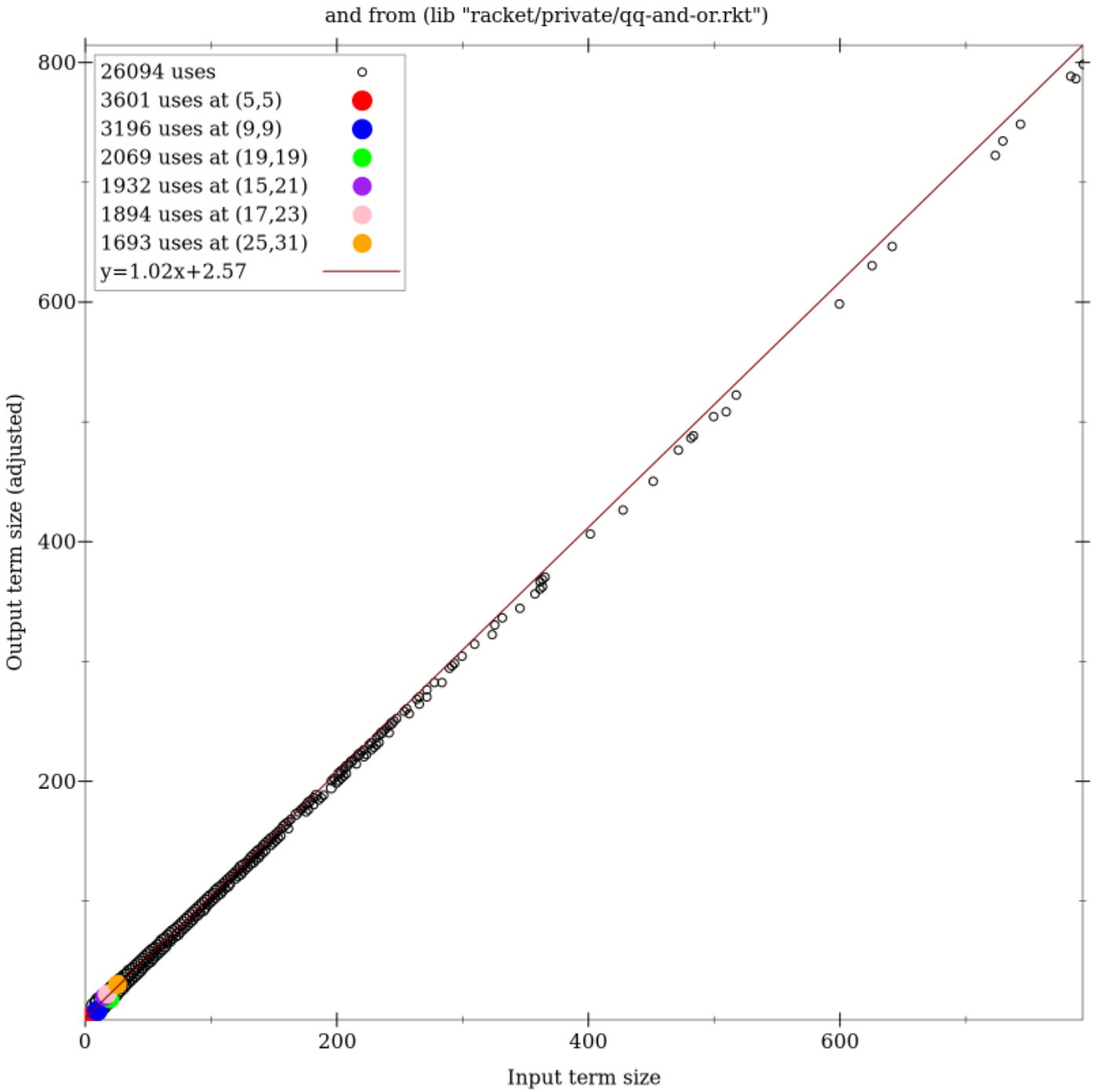

Distribution of macro occurrence counts
All occurrences
Surface occurrences
Occurrences
10⁴
10²
0
250
500
750
1000
Macro

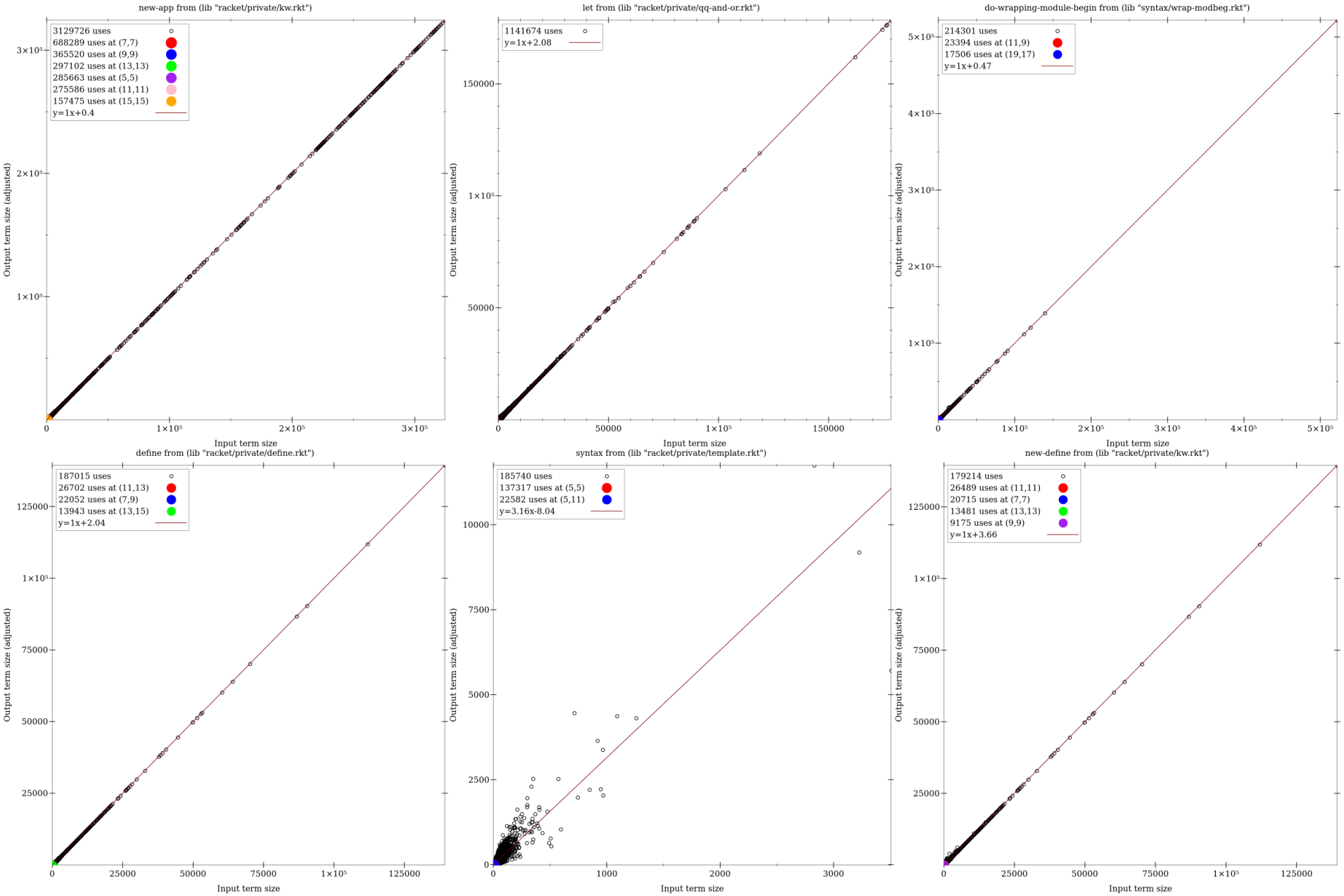
What are the most popular macros?

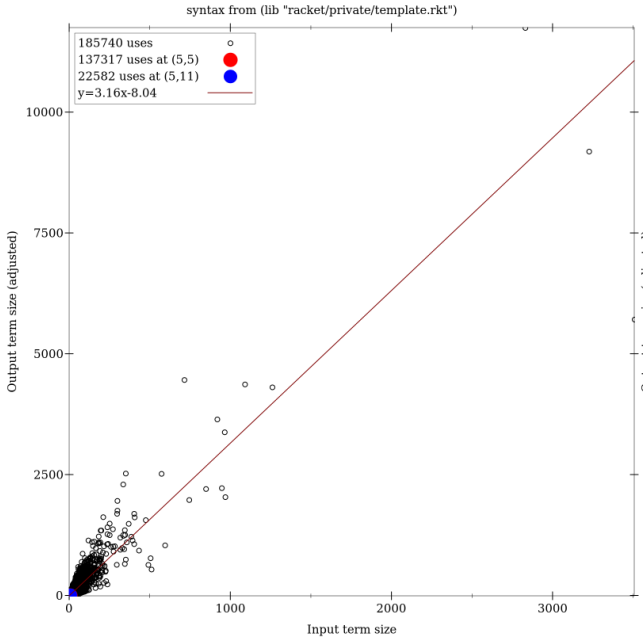
Let's go to the data!

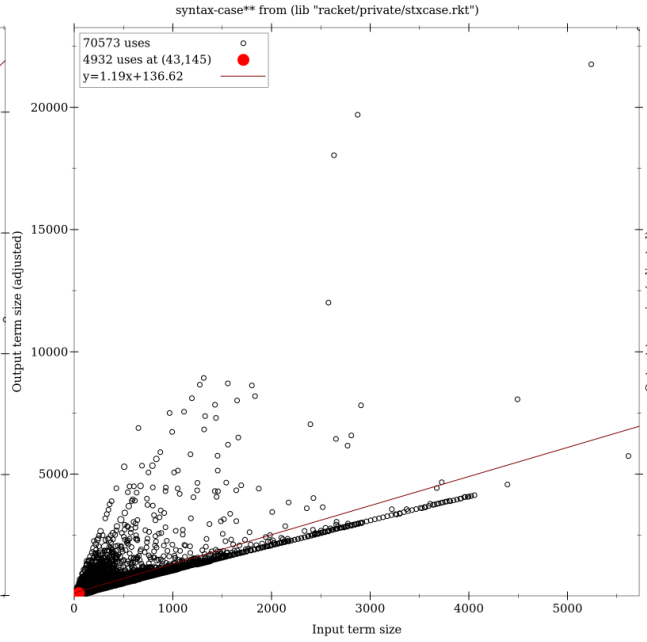
What do macros look like?

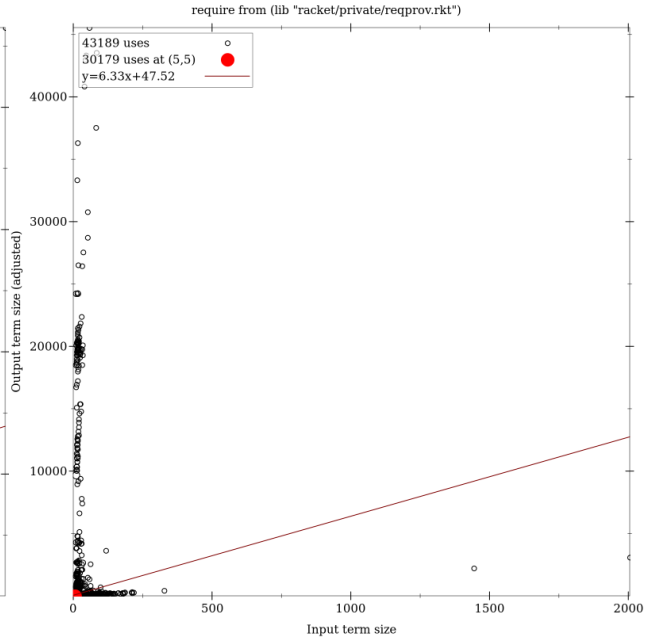

new-app from (lib "racket/private/kw.rkt")
144233 uses
44579 uses at (7,7)
24634 uses at (9,9)
13305 uses at (11,11)
10951 uses at (13,13)
7641 uses at (5,5)
7606 uses at (15,15)
y=1x+0.16
Output term size (adjusted)
Input term size
0
5000
10000
15000
5000
10000
15000


and from (lib "racket/private/qq-and-or.rkt")
26094 uses
3601 uses at (5,5)
3196 uses at (9,9)
2069 uses at (19,19)
1932 uses at (15,21)
1894 uses at (17,23)
1693 uses at (25,31)
y=1.02x+2.57
800
600
400
200
0
200
400
600
Output term size (adjusted)
Input term size

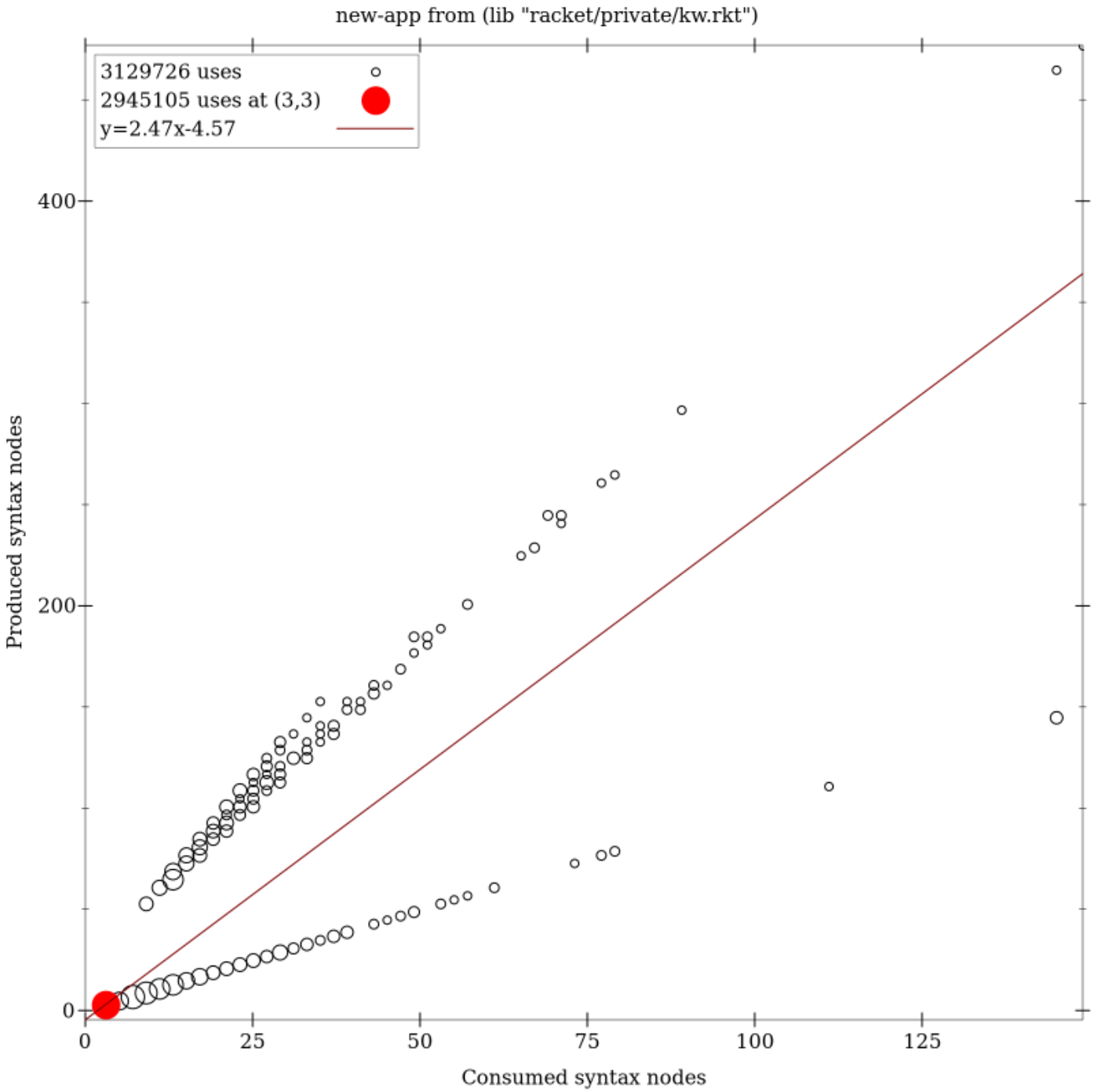

new-app from (lib "racket/private/kw.rkt")
3129726 uses
688289 uses at (7,7)
365520 uses at (9,9)
297102 uses at (13,13)
285663 uses at (5,5)
275586 uses at (11,11)
157475 uses at (15,15)
y=1x+0.4
Output term size (adjusted)
Input term size
let from (lib "racket/private/qq-and-or.rkt")
1141674 uses
y=1x+2.08
Output term size (adjusted)
Input term size
do-wrapping-module-begin from (lib "syntax/wrap-modbeg.rkt")
214301 uses
23394 uses at (11,9)
17506 uses at (19,17)
y=1x+0.47
Output term size (adjusted)
Input term size
define from (lib "racket/private/define.rkt")
187015 uses
26702 uses at (11,13)
22052 uses at (7,9)
13943 uses at (13,15)
y=1x+2.04
Output term size (adjusted)
Input term size
syntax from (lib "racket/private/template.rkt")
185740 uses
137317 uses at (5,5)
22582 uses at (5,11)
y=3.16x-8.04
Output term size (adjusted)
Input term size
new-define from (lib "racket/private/kw.rkt")
179214 uses
26489 uses at (11,11)
20715 uses at (7,7)
13481 uses at (13,13)
9175 uses at (9,9)
y=1x+3.66
Output term size (adjusted)
Input term size

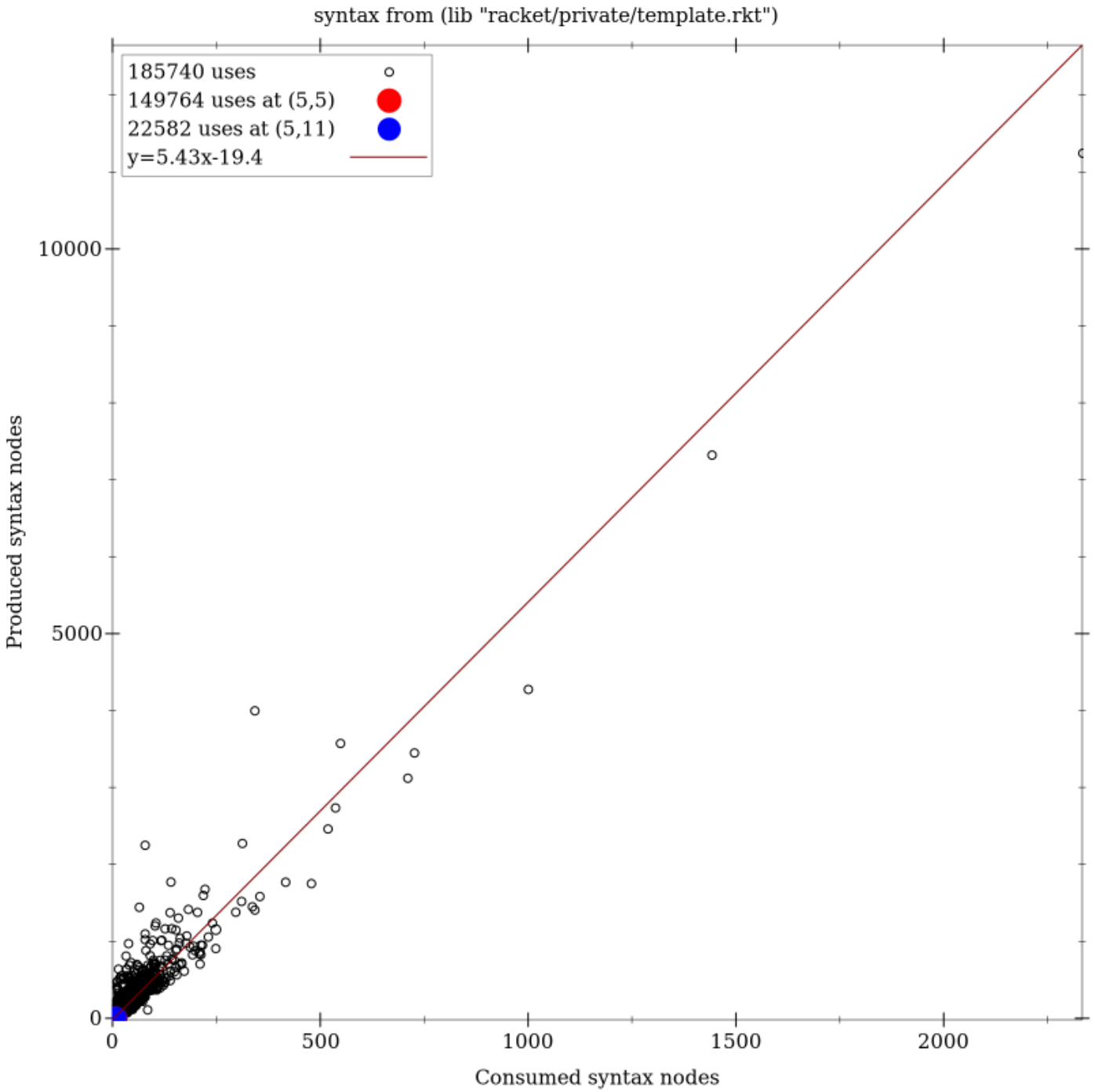

syntax from (lib "racket/private/template.rkt")
185740 uses
137317 uses at (5,5)
22582 uses at (5,11)
y=3.16x-8.04
Output term size (adjusted)
Input term size
0
2500
5000
7500
10000
1000
2000
3000

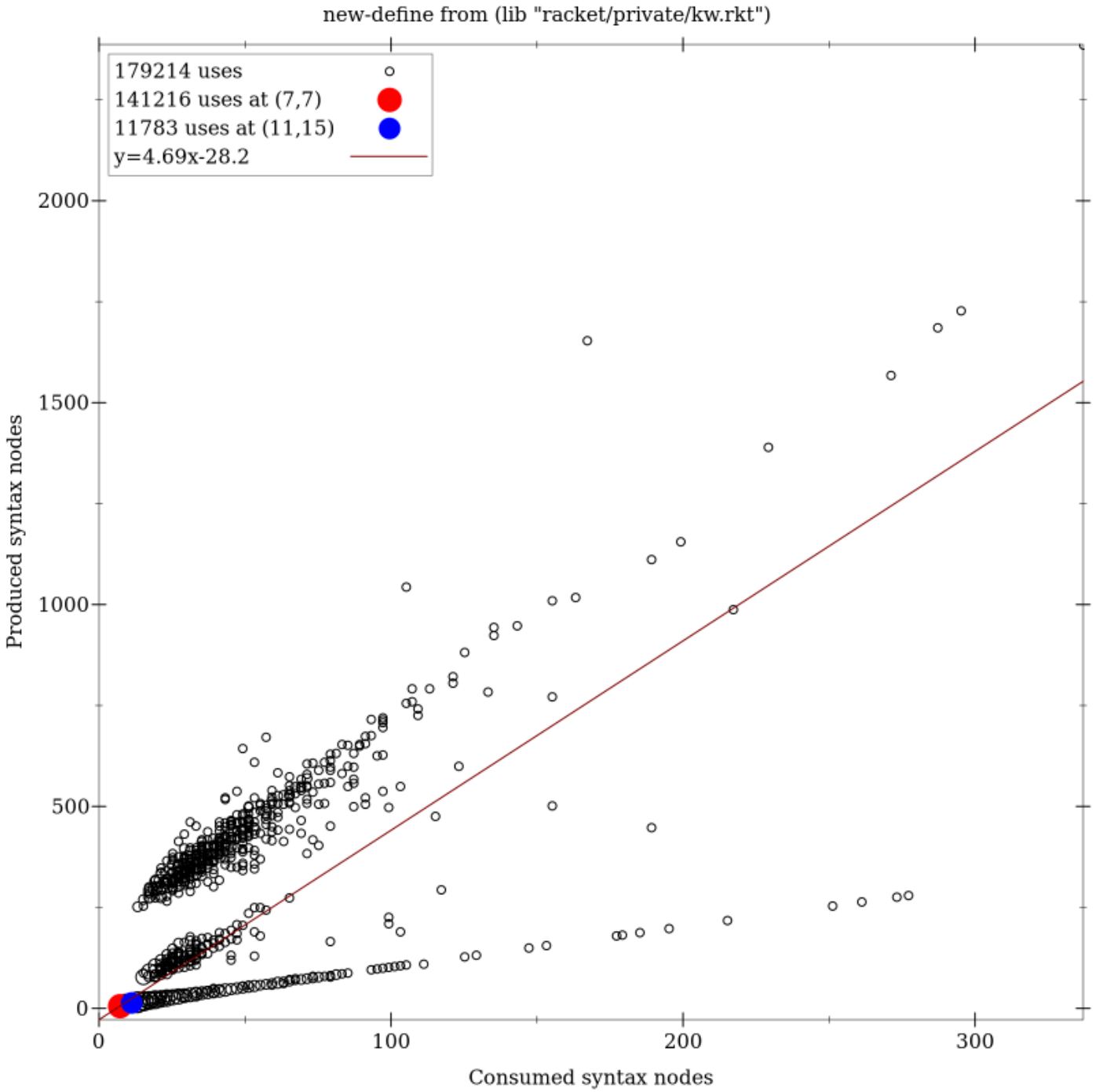

syntax-case** from (lib "racket/private/stxcase.rkt")
70573 uses
4932 uses at (43,145)
y=1.19x+136.62
Output term size (adjusted)
Input term size
5000
10000
15000
20000
1000
2000
3000
4000
5000

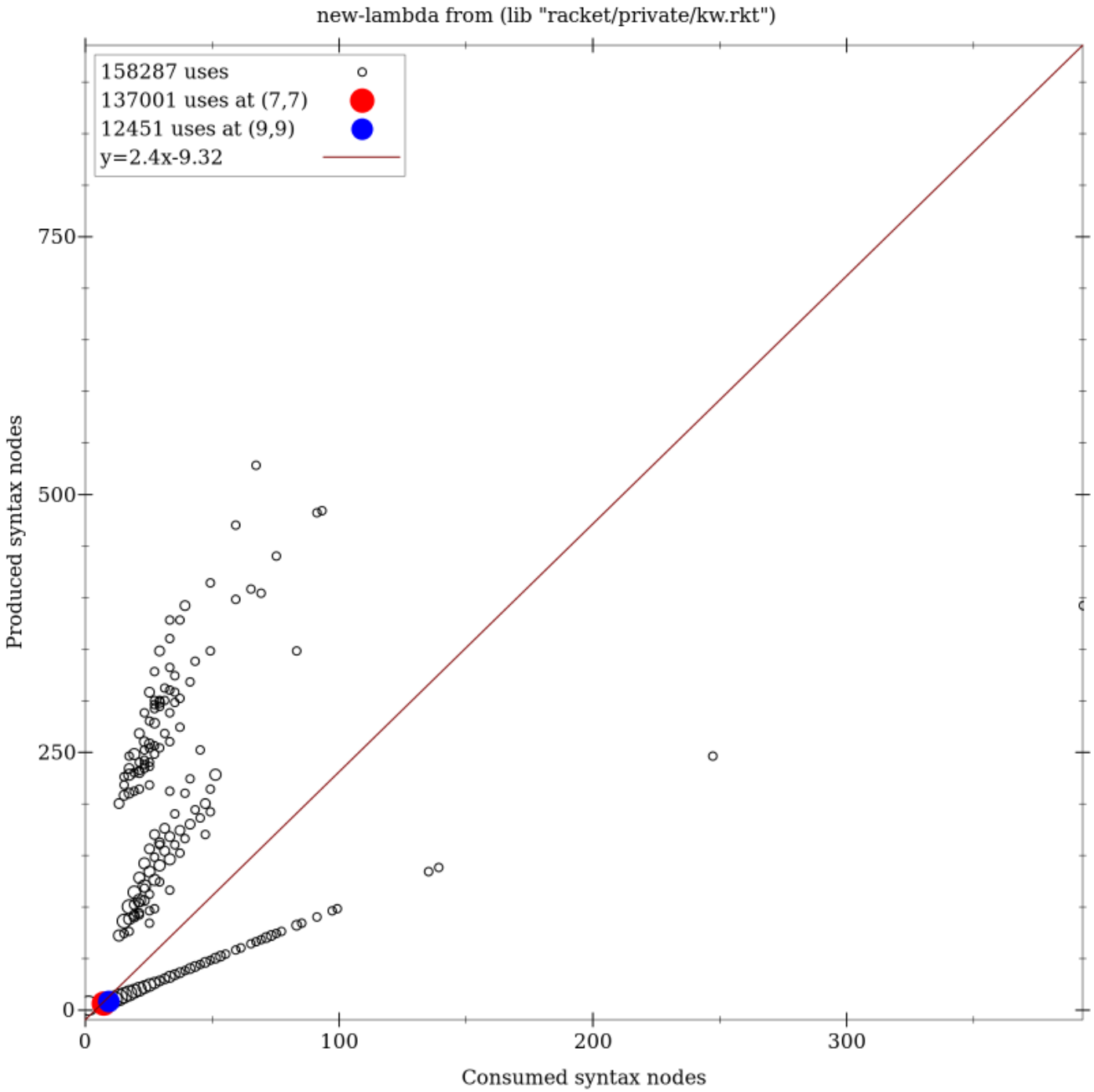

require from (lib "racket/private/reqprov.rkt")
43189 uses
30179 uses at (5,5)
y=6.33x+47.52
Output term size (adjusted)
Input term size
10000
20000
30000
40000
500
1000
1500
2000

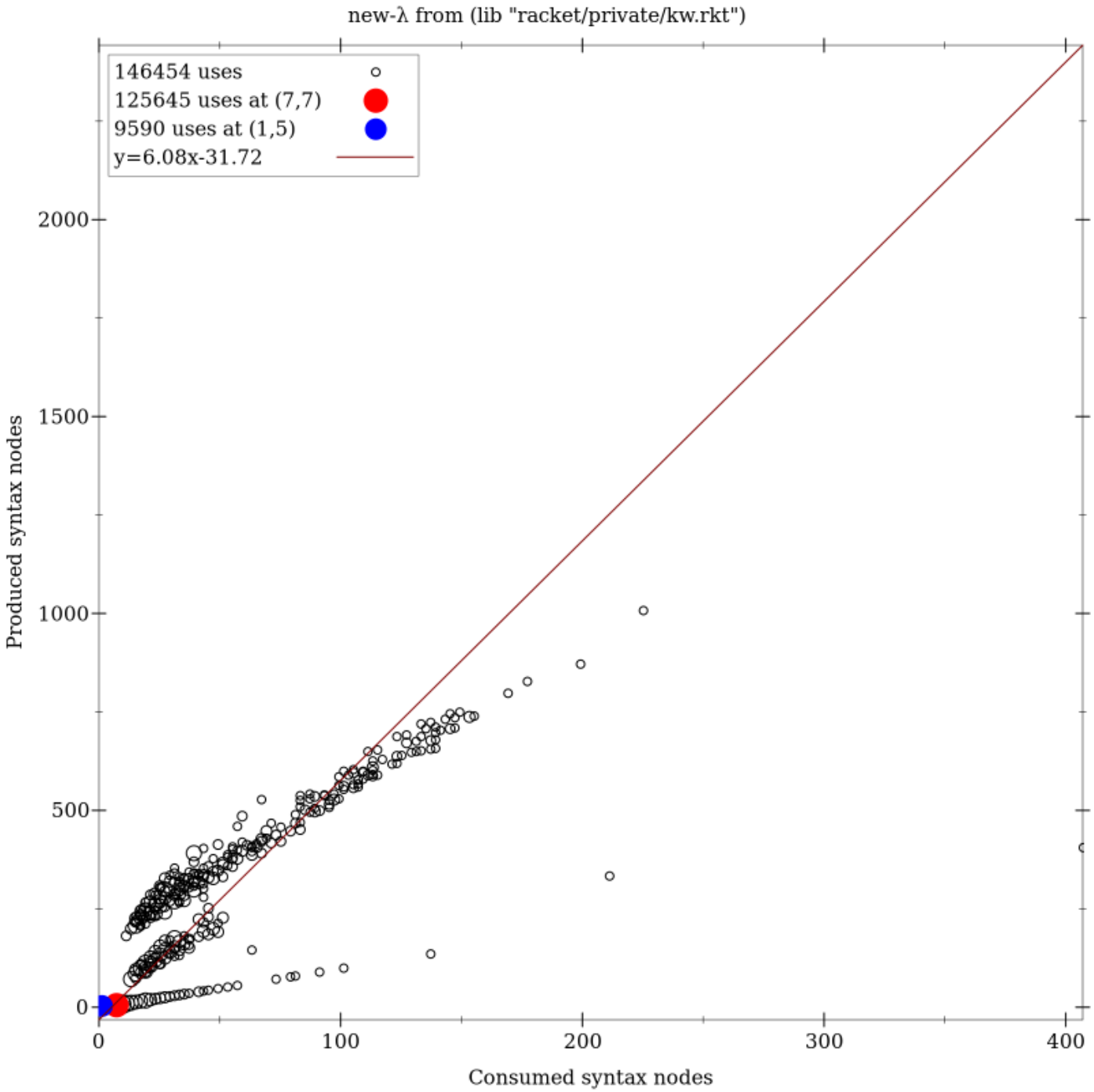
Let's remove common nodes from the counts.

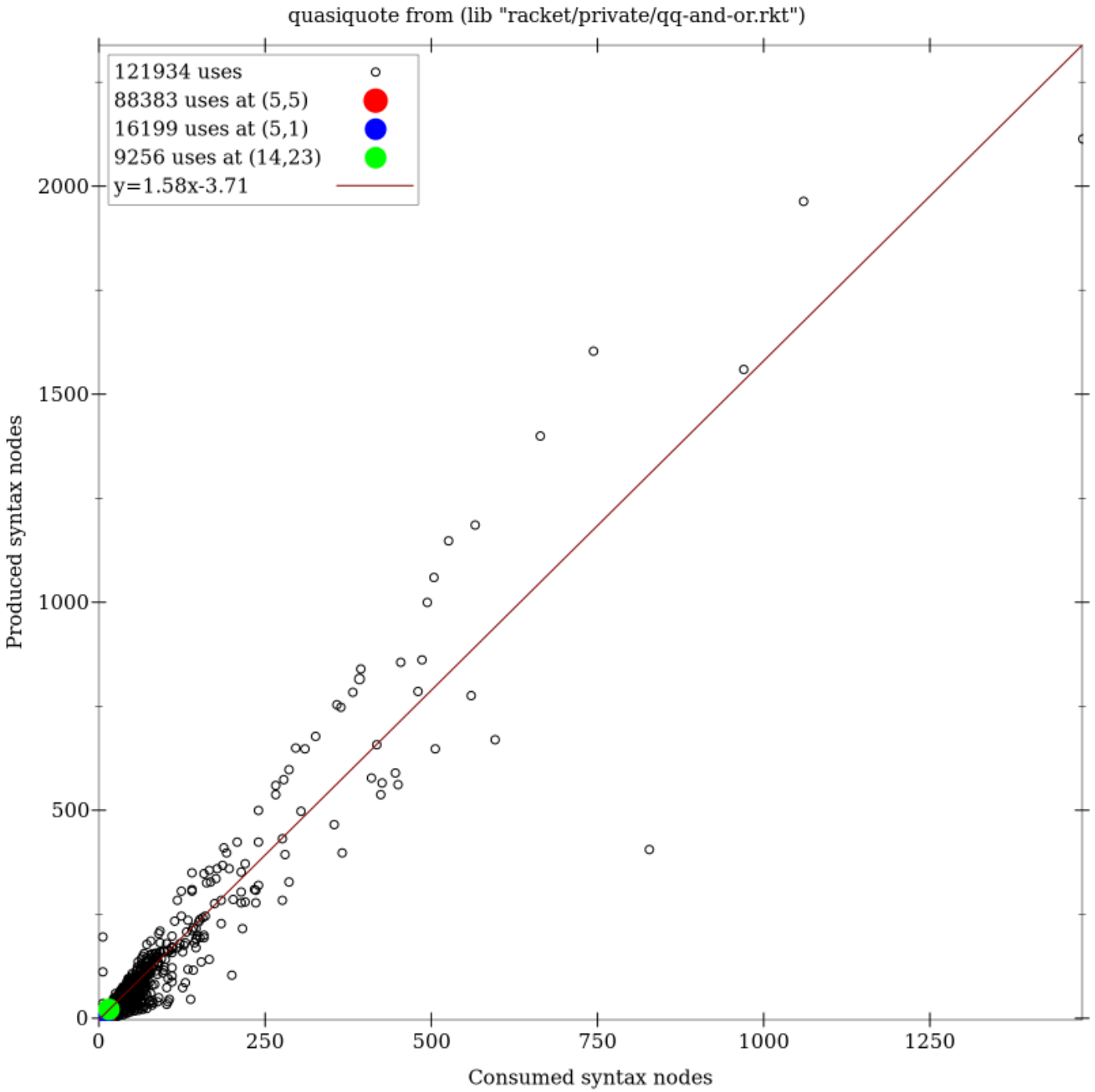

new-app from (lib "racket/private/kw.rkt")
3129726 uses
2945105 uses at (3,3)
y=2.47x-4.57
Produced syntax nodes
Consumed syntax nodes
0
200
400
0
25
50
75
100
125

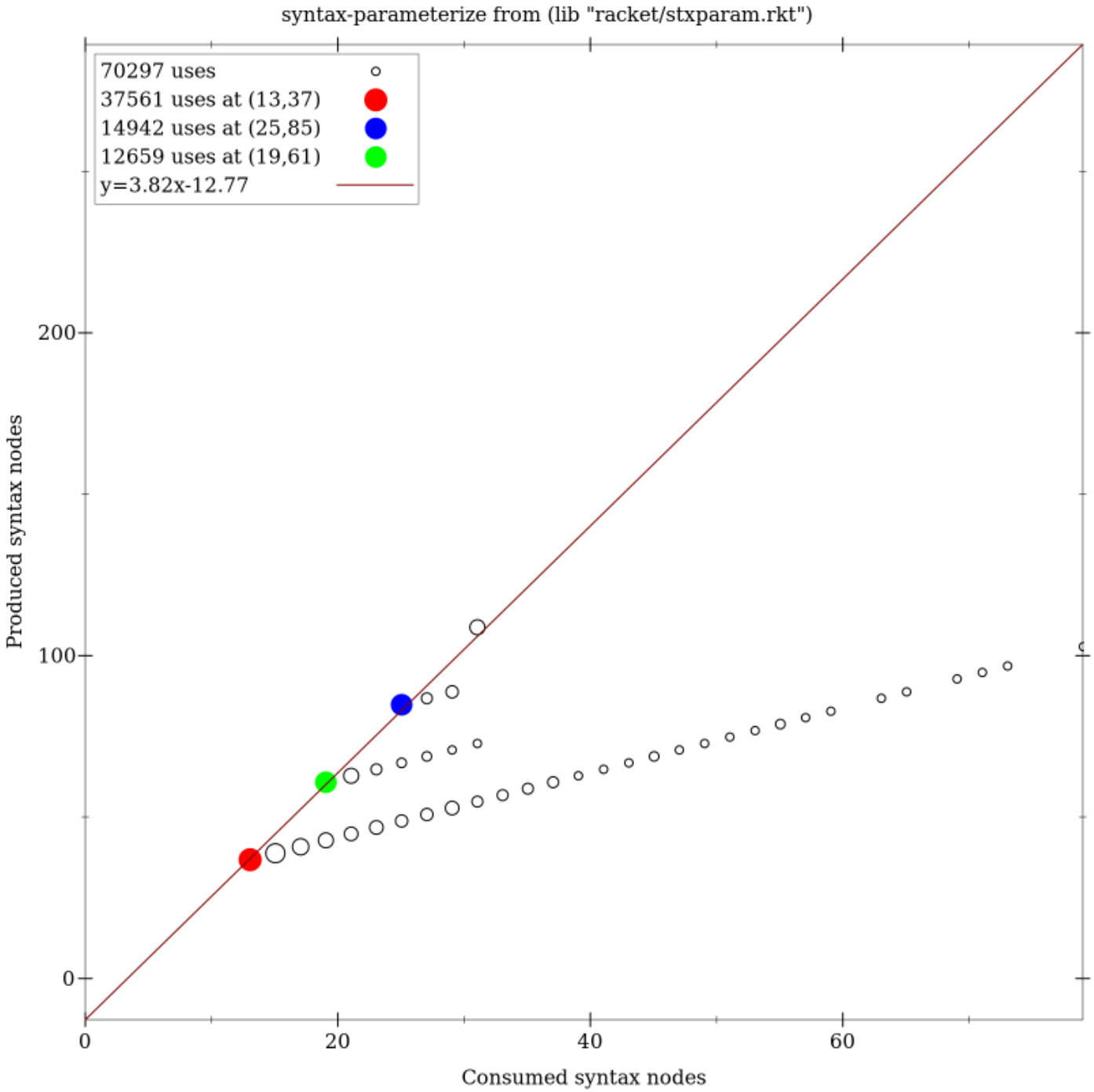

syntax from (lib "racket/private/template.rkt")
185740 uses
149764 uses at (5,5)
22582 uses at (5,11)
y=5.43x-19.4
Produced syntax nodes
Consumed syntax nodes
0
5000
10000
0
500
1000
1500
2000

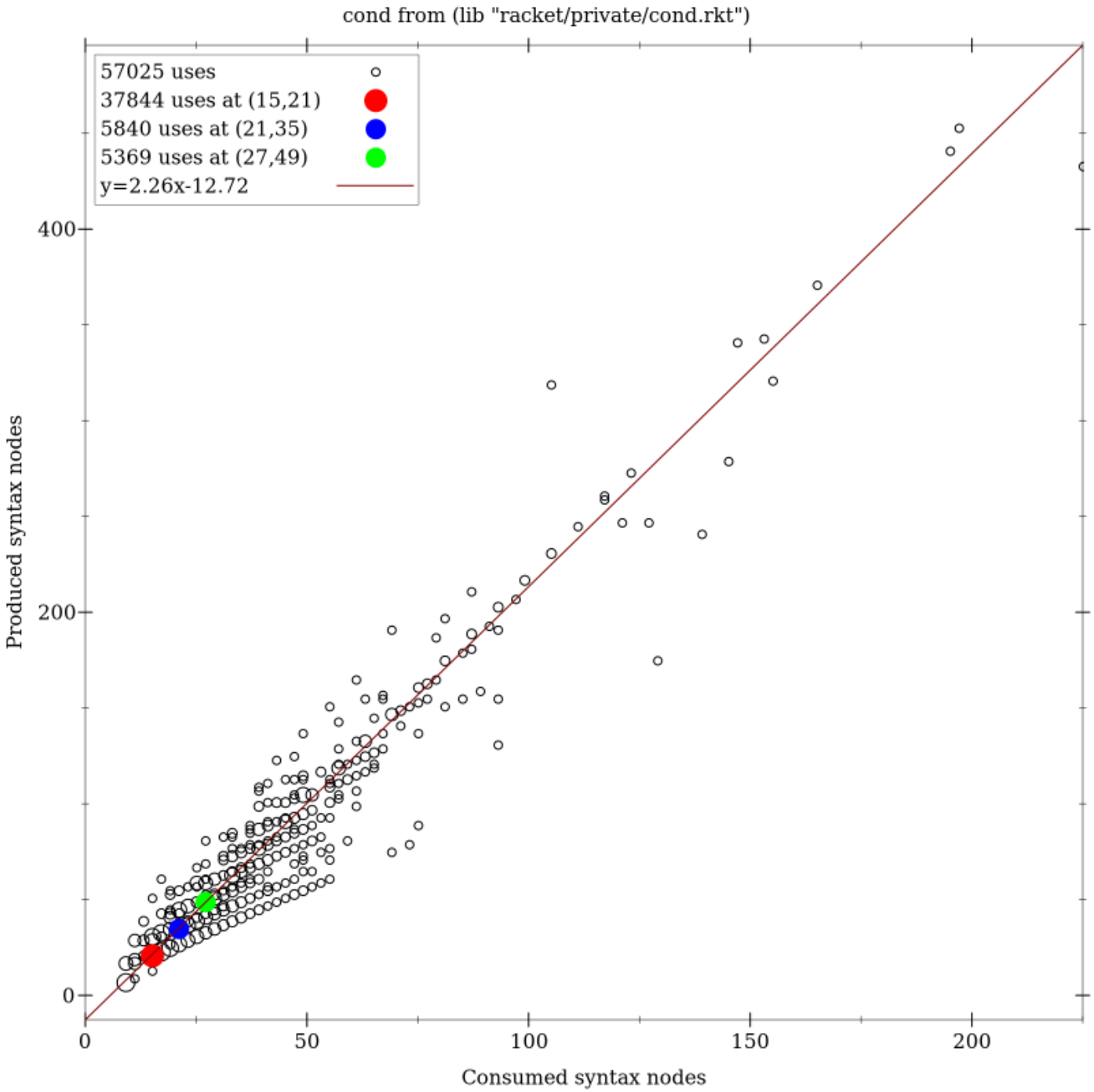

new-define from (lib "racket/private/kw.rkt")
179214 uses
141216 uses at (7,7)
11783 uses at (11,15)
y=4.69x-28.2
Produced syntax nodes
Consumed syntax nodes
0
500
1000
1500
2000
0
100
200
300

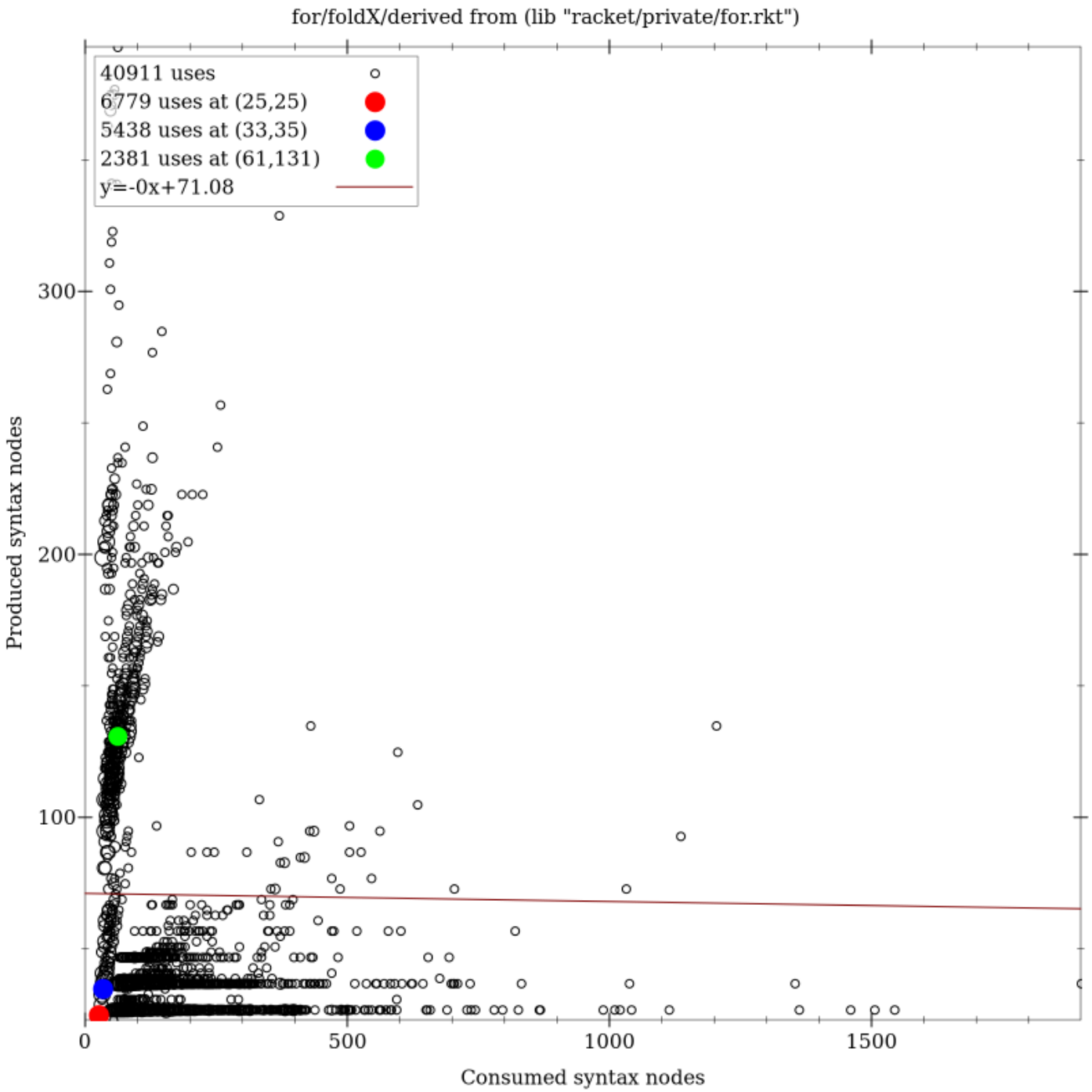

new-lambda from (lib "racket/private/kw.rkt")
158287 uses
137001 uses at (7,7)
12451 uses at (9,9)
y=2.4x-9.32
Produced syntax nodes
Consumed syntax nodes
0
250
500
750
0
100
200
300

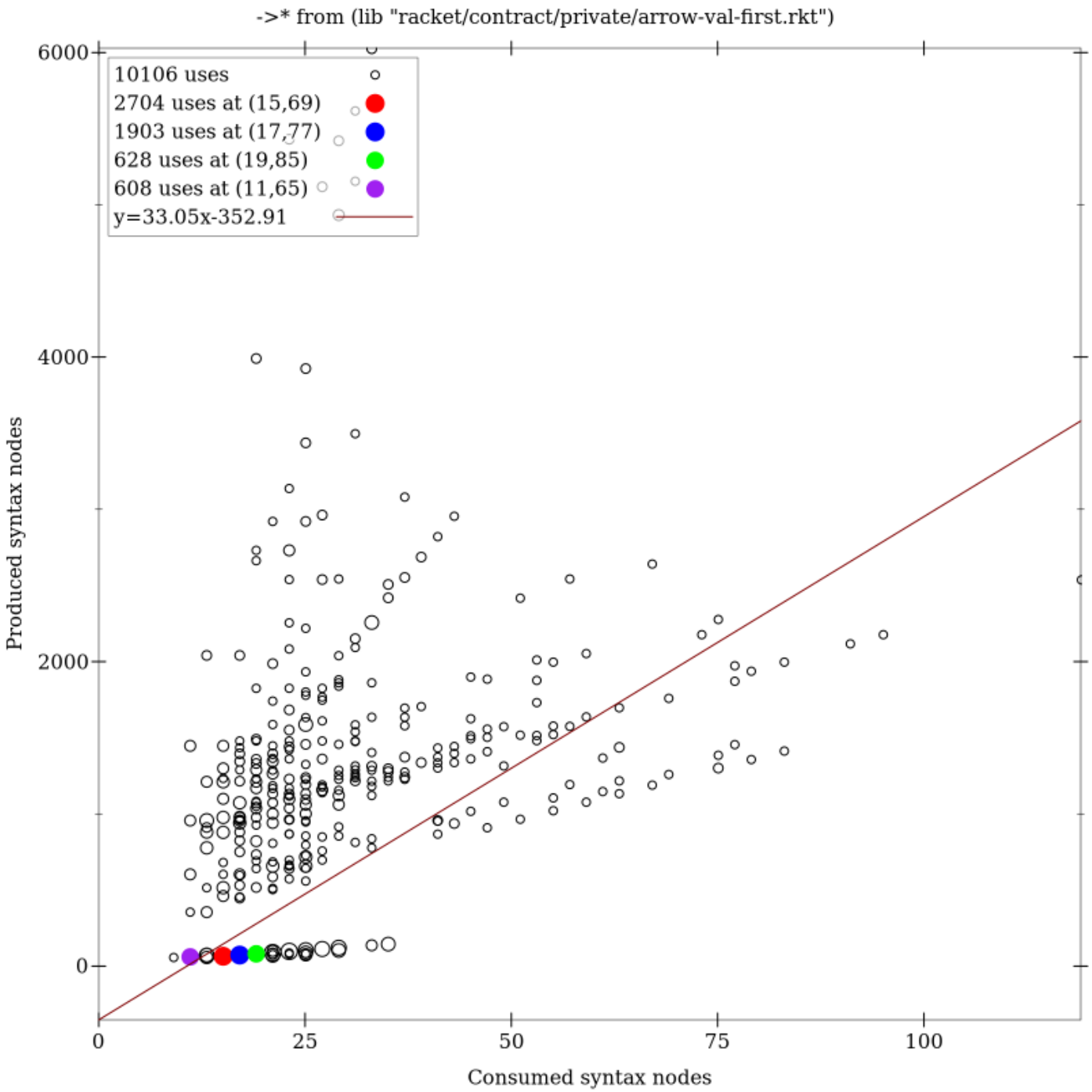

new-λ from (lib "racket/private/kw.rkt")
146454 uses
125645 uses at (7,7)
9590 uses at (1,5)
y=6.08x-31.72
Produced syntax nodes
Consumed syntax nodes
0
500
1000
1500
2000
0
100
200
300
400


quasiquote from (lib "racket/private/qq-and-or.rkt")
121934 uses
88383 uses at (5,5)
16199 uses at (5,1)
9256 uses at (14,23)
y=1.58x-3.71
Produced syntax nodes
Consumed syntax nodes
0
500
1000
1500
2000
0
250
500
750
1000
1250


syntax-parameterize from (lib "racket/stxparam.rkt")
70297 uses
37561 uses at (13,37)
14942 uses at (25,85)
12659 uses at (19,61)
y=3.82x-12.77
Produced syntax nodes
Consumed syntax nodes
0
100
200
0
20
40
60


cond from (lib "racket/private/cond.rkt")
57025 uses
37844 uses at (15,21)
5840 uses at (21,35)
5369 uses at (27,49)
y=2.26x-12.72
Produced syntax nodes
Consumed syntax nodes
0
200
400
0
50
100
150
200


for/foldX/derived from (lib "racket/private/for.rkt")
40911 uses
6779 uses at (25,25)
5438 uses at (33,35)
2381 uses at (61,131)
y=-0x+71.08
Produced syntax nodes
Consumed syntax nodes
0
500
1000
1500
100
200
300


->* from (lib "racket/contract/private/arrow-val-first.rkt")
10106 uses
2704 uses at (15,69)
1903 uses at (17,77)
628 uses at (19,85)
608 uses at (11,65)
y=33.05x-352.91
Produced syntax nodes
Consumed syntax nodes
0
2000
4000
6000
0
25
50
75
100

The End