part 2: using sets

example: is there any dynamic memory allocation after initialization is completed?

$ cobra *.c

: fcts # mark all function names
: next { # move to function body
: contains malloc # restrict to those containing malloc calls
: back ( # back to the function parameter list
: back # back to the function name
: >1 # store these names in set 1
: reset # clear all marks
: fcg init_run * # mark all functions reachable from init_run
: <&1 # check the intersection with set 1
: list # list them (e.g., it shows “badfct” as a match)
: fcg init_run badfct # show call graph connecting init_run and badfct
the query language

operations on sets

>`n  # save all current marks and ranges in set n
<br>`n  # restore current marks and ranges from set n
<br>|n  # add marks from set n to current (set union)
<br>&n  # keep only marks also in set n (set intersection)
<br>^n  # keep only marks not in set n (set difference)

where n is 1..3
the query language
defining query functions

```
: def p10_rule4(rn, nr)
    fncts  # mark function names
    n {  # move to fct body
        m & (.range > nr)  # restrict to fcts longer than nr lines
        b (  # back to start of parameter list
            b  # back to function name
            = "===} rn: functions exceeding nr physical source lines:"
    d
end
: p10_rule4(R4, 75)
```

a simple textual replacement of parameters

remember: .range is a predefined token attribute
the query language

pattern matches and .mark

by default all tokens in a matched pattern are marked with the value 1

there are two exceptions:
• the first token in each pattern is marked with value 2
• any bound variables in the pattern is marked with value 3

that makes it possible to quickly simplify long pattern matches to just their start, or to just the effectively bound variables from the pattern

$ cobra *.c
: pat for ( x:@ident .* ) { .* :x = .* }$

bound variables matched:
  1: cobra_te.c:1579: q_now
  2: cobra_te.c:1358: m
  3: cobra_te.c:881: b
  4: cobra_sym.c:105: r
  5: cobra_sym.c:51: r
  6: cobra_prep.c:339: c
  7: cobra_lib.c:2492: r
  8: cobra_lib.c:2122: z
  9: cobra_lib.c:1202: s
 10: cobra_lib.c:782: r
 11: cobra_fcg.c:156: r
 12: cobra_cfg.c:171: cur
 13: cobra_cfg.c:67: cur

13 patterns matched
3929 matches
: m & (.mark == 2)
12 matches
: undo
: m & (.mark == 3)
14 matches
the query language
reading commands from files

• to read a query function from the “play” library we can use the dot command :

  $ cobra *.c
  : . play/declarations.cobra

• we can read query files also from the command line:

  $ cobra --f play/declarations.cobra *.c

• cobra query files stored in rules/main can be read without the directory prefix, for instance, try:

  $ cobra -terse --f basic *.c
  $ cobra --terse --f stats *.c
  $ cobra --terse --f metric *.c
  $ cobra --terse --f cwe *.c
the query language

the context command

$cobra *.c
1 core, 10 files, 56546 tokens:
: context process_line
cobra_prim.c:626-673
calls:
cobra_prim.c:671: triple()
cobra_prim.c:669: single()
cobra_prim.c:662: strchr()
cobra_prim.c:649: strcmp()
cobra_prim.c:647: sscanf()
cobra_prim.c:646:strncmp()
cobra_prim.c:644: assert()
cobra_prim.c:640: strlen()
cobra_prim.c:638: strstr()
cobra_prim.c:631: printf()
is called by:
cobra_lex.c:809: line()
cobra_lex.c:279: show1()
cobra_lex.c:288: show2()

the text version on the left is the default, the graph version requires the graphviz/dot or dotty command to be installed on your system
the query language
or full or partial function call graphs (also requires graphviz/dot)

$ cobra *.c
1 core, 10 files, 56546 tokens
: fcg
wrote: /tmp/cobra_dot_Sf8RAv (269 nodes, 499 edges)
view with: !dotty /tmp/cobra_dot_Sf8RAv &
: !dotty /tmp/cobra_dot_Sf8RAv &

be warned: dot/dotty can be very slow on large graphs, and may even crash.
cobra, though, can generate the input file very quickly
the query language

browsing code

: B cobra_te.c 100  # show file starting at line 100
: B                   # browse forward in same file
: B 90                # browse same file from line 90

other sometimes useful short-hands:

: ff par_scan          # find function definition
: cpp on               # processes the header files
: ft Prim              # find a type definition

or with tcl/tk installed:

: V cobra_te.c 100     # like B, but in a popup window
: window               # enable automatic window popups
: m while              # mark something
: d 1                  # now pops up a tcl/tk window with the source text
: : nowindow           # disable window popups

similarly, with graphviz (dotty) installed, you can see how patterns are translated:

$ cobra –view –pat “....” *.c  # pop up graph showing FSA for the pattern
the query language
defining new token categories with the map command

$ cat prepositions.map  # map token text to new user-defined token types
of  preposition
    preposition
in   preposition
for  preposition
on   preposition
with preposition
by   preposition
but  preposition
at   preposition
from preposition
about preposition
like preposition
into preposition
...

$ cobra *.txt  # some random English prose
: map prepositions.map
: m @preposition .
: = “a sentence should not end with a preposition:”
: d
the query language

now we’re getting into the woods:
track_start, track_stop, and shell escapes

```
$ cobra *.ch
1 core, 15 files, 90063 tokens
: m while
127 matches
: track_start file1  # redirect output to file1
: list
: track_stop        # end redirection
: !wc file1         # shell escape to check the size of the file
: !sort file1       # or to display it after sorting
: q
$```

the query language

scaling behavior

18,633,817 Lines of Code of the Linux 4.3 distribution, with 39,144 .c and .h files

checking 2 types of queries:

• find empty else stmts
• find all switch stmts without default clause

using 1..32 CPU cores

with 4 or more cores we obtain truly interactive query processing times < 1 sec per query

![Graph showing query processing times vs number of cores]