interactive code checking
with Cobra

a Tutorial

Gerard Holzmann
Nimble Research
gholzmann@acm.org
of topics covered

1. **background** and principle of operation
   - installation and configuration
   - guide to online documentation

2. **pattern** queries and regular expressions
   - exercises

3. **interactive** queries (in two parts)
   - token attributes
   - sets and ranges
   - functions
   - reading files, libraries
   - exercises

4. **scripted** queries
   - recursive functions
   - associative arrays
   - the query libraries
   - using concurrency
   - exercises

5. **standalone** checkers
   - using concurrency: multi-threaded checkers

6. use of Cobra for *runtime verification*
   - using live data or event-logs
the query language

overview

there are about 40 query commands in all, but 4 or 5 suffice to handle most types of queries. they can be used for:

A. Setting, Moving, or Removing Marks
B. Setting Ranges
C. Output
D. Meta Commands
E. Defining Sets of Marks

examples:

A: mark, next, back, jump, contains, extend, undo, reset
B: stretch
C: display, list, pre, =, help
D: history (h), browse (B), files (F), system (!), cfg, fcg, fcts
E: save (>), restore (<)

try:
$ cobra –c help /dev/null
or type “?” or “help” in an interactive session

note that the output is different from:
$ cobra -help
an example
finding switch statements without a default clause

$ cobra *.ch  # start an interactive session on the cobra 3.0 sources
1 core, 13 files, 58381 tokens
: mark switch  # mark all switch statements
29 matches
: next {       # move mark to the start of the body
29 matches
: contains no default  # check the range from { to }, no is a qualifier
6 matches
: display 2 +8  # display the 2nd match plus the following 8 lines
cobra_lib.c:538:
  2: >  538   {   switch (*s) {
  2:  539       case '&':
  2:  540       case '|':
  2:  541       case '^':
  2:  542           tmp = t;
  2:  543           t = s;
  2:  544           s = tmp;
  2:  545           break;
  2:  546       }   }
: quit
$

$
the query language

convenient shorthands

instead of writing:

```
: mark switch (  
: next {  
: contains no default  
: display
```

we can also use shorthands:

```
: m switch (  
: n {  
: c no default  
: d 
```

and we can combine commands on a single line, using semi-colons to separate commands:

```
: m switch (; n {; c no default; d 
```

or execute everything from *the command line* with the `–c` flag:

```
$ cobra –c ‘m switch (; n {; c no default; d’ *.c
```

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m[ark]</td>
<td>defines a set of matches</td>
</tr>
<tr>
<td>n[ext]</td>
<td>moves all current marks forward</td>
</tr>
<tr>
<td>d[isplay]</td>
<td>displays the current marks</td>
</tr>
<tr>
<td>c[ontains]</td>
<td>checks a token range</td>
</tr>
<tr>
<td>{ ... }</td>
<td>defines a token range, as do:</td>
</tr>
<tr>
<td>[ ... ]</td>
<td></td>
</tr>
<tr>
<td>( ... )</td>
<td></td>
</tr>
</tbody>
</table>
the query language

command-line use

$ cobra –c “m switch (; n {; c top no default; d” *.c

`switch (f->n->ntyp) {
    case UNLESS:
        attach_escape(f->sub->this, e);
        break;
    case IF:
        case DO:
            for (z = f->sub; z; z = z->nxt)
                attach_escape(z->this, e);
            break;
        case D_STEP:
           /* attach only to the guard stmt */
            escape_el(f->n->sl->this->frst, e);
            break;
    case ATOMIC:
    case NON_ATOMIC:
        /* attach to all stmts */
        attach_escape(f->n->sl->this, e);
        break;
} ` 

top is another query **qualifier** to restrict the check to the **top level** of nesting

# with a –runtimes flag (and without the ‘d’):

$ cobra -runtimes -c ‘m switch; n {; c top no default’ *.c

(0.0404 sec)
(0.00338 sec)
(0.000523 sec)
(0.000344 sec)
(0.0005 sec)

$
we’ve so far mentioned *five* commands: *mark*, *next*, *contains*, *display*, and *quit*
three other useful commands are *stretch*, *list*, and *pre*:

```bash
$ cobra *.c
1 core, 10 files, 56450 tokens
: mark for (
206 matches
: next \;
206 matches
: stretch \;
206 matches
: contains ->
45 matches
: pre 1
# show the first matched range with pre (or p)
```

```c
for ( cur = ( Prim * ) n ; rval && cur && cur -> seq <= n -> jmp -> seq ; cur = cur -> nxt )
```

```
^ ~~~~ ~~ ~~~ ~~~ ~~~ ~~~ ~~~ ~~~ ~~~ ~~~~`
```
the query language
using command qualifiers

we’ve so far mentioned two query qualifiers: top and no
there are two more: & and ir

- **top** # restrict to matching at the same nesting level as the mark (contains and stretch)
- **no**  # to find non-matches (mark and contains)
- **ir**  # mark all matching tokens inside the current range (mark)
- **&**   # restrict to *marks* that also match a new pattern (mark)
- **&**   # restrict to *ranges* that also match a new pattern, and move the mark to *the first* (contains)

to see how these work, at the end of the last example, we can type:

```
: c & seq
: p 1
```

```
cobra_cfg.c:38:
  1:    38  for ( cur = ( Prim * ) n ; rval && cur && cur->seq <= n->jmp->seq ; cur = cur->nxt )
  1:         ^^^
```

**Note:** only the first match was marked
token attributes
using expressions in query commands

• every lexical token in the input sequence is tagged with a number of attributes that can be queries in interactive commands (see table)

• for instance, to find for-loops or switch statements longer than 100 lines we can say:

  : mark for ( # mark all for statements
  : mark switch ( # and also all switch statements
  : next { # move mark forward to open curly brace
  : mark & (.curly > 5) # restrict to those nested deeper than 5 levels
  : mark & (.range > 100) # restrict to blocks longer than 100 lines)
  : mark & (.lnr < 50) # that appear in the first 50 lines of a file

• or, we can combine the last three downselects in one expression as:

  : mark & (.curly > 5 && .range > 100 && .lnr < 50)
the query language

some exercises

0: example: find recursive functions, using $$
1: find global variables with fewer than 3 characters
2: find loops that contain gotos but no labels
4: find goto statements immediately followed by the label

<table>
<thead>
<tr>
<th>answer 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>answer 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>answer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>answer 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

| .range | # nr of lines in a range |
| .fnm   | # source filename (a string) |
| .lnr   | # source line-number |
| .curly | # level of {...} nesting |
| .round | # level of (…) nesting |
| .bracket | # level of […] nesting |
| .len   | # length of token text |
| .typ   | # token type (a string) |
| .txt   | # token text (a string) |
| .seq   | # token sequence number |
| .mark  | # marked value |