Revision: category-partition method — solutions

(2007–8 exam, question 1)

Conrad Hughes

March 10, 2010

1. Black-box (functional) testing, because we have no access to code. Access to source code would allow white-box (structural) testing. Risky because focussing on structure might cause you to lose track of intended/missing functionality, but good because by applying code coverage criteria you get better indications that the implementation has been fully exercised.

2. Category-partition method applied:

ITF:
- That we get the correct fragments out when the expression is in s.
- [That we get [s] when the expression isn't in s.]
- [That we get an appropriate exception when regex is null.]

Parameters:
- s string to be split
- regex regular expression
- limit max #fragments

Environment: JRE version

Partitions:
- s:
  - empty
  - nonempty
    - doesn't contain regex
    - equal to regex
    - contains regex
- regex:
  - null
  - empty [spec unclear]
  - nonempty
- limit:
  - negative
  - zero
  - positive
    - less than #matches
    - equal to #matches
    - greater than #matches
- Env:
  - pre 1.4
  - 1.4 or later

Constraints:
- $4 \times 3 \times 5 \times 2 = 120$ combinations

- s empty ⇒ regex/limit irrelevant
- s doesn't contain regex: limit values irrelevant
- Pre 1.4 JRE: no testing
- regex null ⇒ s/limit irrelevant
- post-constraint combos:
  - s empty × anything × anything → 1 case
  - s without regex × anything × anything → 1
  - s=regex × regex × (1|2) → 2
  - s with regex × empty regex × $(-ve| < | = | >)$ → 4
  - s with regex × some regex at end × 0 → 1
  - anything × null × anything → 1
  - So, about 10 cases in all.

Value classes:
s: "hello"
regex: "", "a", "l", "hello"
limit: -1, 0, 1, 4

Specification:
- "".("a",0) → ["]
- "hello".("a",0) → ["hello"]
- "hello".("",-1) → ["", "h", "e", "l", "l", "o", ""]
- "hello".("hello",0) → []
- "hello".("hello",-1) → ["", ""]
- "hello".("l",1) → ["hello"]
- "hello".("l",3) → ["he", "", "o"]
- "hello".("l",4) → ["he", "", "o"]
- "hello".(null,4) → exception

- Possible sources of ambiguity:
  - Behaviour when regex is ""
  - Behaviour when regex is null

3. Two techniques are constraints (regex being null should cause an error no matter what the other inputs are, so only one test is necessary here instead of all possible combinations of the other inputs), and combinatorial reduction — where, for example, instead of covering all possible combinations of three inputs (N^3), one might cover all possible pairwise combinations (N^2) in the hope that all errors will be triggered solely by one input or by pairwise interactions between two inputs, rather than pathological interactions among all inputs. Models can also be used to reduce the number of tests.