Test specification & report: IBAN

# Specification

## SUT

The algorithm under test is an elgorithm to validate IBAN codes.

The IBAN codes are fed to the algorithm as a character string, the output is either

* True: The input string is a valid IBAN code
* False: The input string is not a valid IBAN code.

Validity I determined as conformation to the requirements stated in the “IBAN: International Bank Account Number” document written by the European committee for banking standards. Which can be found here: <http://www.europeanpaymentscouncil.eu/documents/ECBS%20IBAN%20standard%20EBS204_V3.2.pdf>

## Test method

### Hoar triple

We have had problems implementing a test according to hoar logic. If call the iban validation algorithm ‘iban’, the Hoar triplets would simply be

{isIban xs} y = iban xs {y ==True} and

{isNotIban xs} y = iban xs {y ==False}

But then we get to the problem of how to define ‘isIban’ and ‘isNotIban’. Both functions can be easily created by wrapping the ‘iban’ function, but that would render the test useless as the algorithm under test is used for it’s own verification, which would result in positive tests no matter the implementation of the algorithm.

Another way to go about this is to actually implement ‘isIban’ and ‘isNotIban’ from scratch. This would be the best solution if we had not all worked on the ‘iban’ algorithm. Because we have, we would be extremely likely to make the same mistakes in ‘isIban’ and ‘isNotIban’ (which are basically equal to ‘iban’) as we did in ‘iban’, which would result in improper testing.

### Input generation

We decided to go with a different method in which we create a function to randomly generate IBAN codes. These codes can be either valid or invalid based on a function argument. Because generating IBAN codes approaches the problem from the opposite side compared to validating IBAN codes, there should be less risk of repeated mistakes.

Valid IBAN generation

Valid IBANs are generated by:

* Picking a random country code from the list in ISO 3166 (source <https://nl.wikipedia.org/wiki/ISO_3166-1)>
* Generating a random valid BBAN
  + Random length between 0 and 34
  + Random valid characters, e.g. digits and upper case characters
* Calculating the checksum over the combination of the country code and the BBAN

Invalid IBAN generation

Invalid IBAN are generated by first generating a valid IBAN and then randomly mangle this in one of the following ways:

* Change the country code to an invalid country code. The invalid country code is generated by randomly taking a two-character string that is not part of the list of country codes described in ISO 3166
* Changing the checksum to an invalid checksum by taking the valid checksum, adding a random integer in the range 1-96 and taking modulo 97 of it. Thereby changing the ckecsum to anything in the range 0-96 except for the original checksum.
* Changing the BBAN to an invalid BBAN by replacing a random number of characters in the string (up to the length of the supplied BBAN) by a random non-valid BBAN character.

### Test code

### Our test code takes the number of tests to perform as an input. For each test either a valid or an invalid IBAN (again randomly decided) is generated, fed to the ‘iban’ alrothm and the output is checked with the expected output (True for valid IBANs, False for invalid IBANS)

# Result:

The result of running this test 10000 times is:

(…)

"pass on: \"JM473XVOGB5GVLT\""

"pass on: \"LK28-|wd5\*L6<hiS=Z^p5V\_h()1pq7|S[\_hA\""

"pass on: \"GG244UXV86I0OS9RR3KP\""

"pass on: \"ET190KWP0KPBQRVCG37NVAI\""

"pass on: \"NO97TC07WHHWKC6T2\""

"pass on: \"PW036ZVHM2I5O58UDFML08H9T55K2Z\""

"pass on: \"TJFY1O60A\""

"100000 tests passed"