10th UCAAT

User Conference on Advanced Automated Testing

ETSI The Standards People

Automatic Test Case Generation from Software Specifications

Thomas Arts

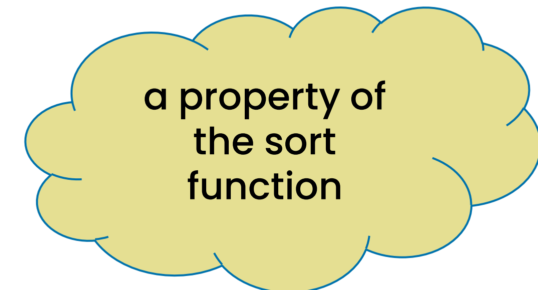
Idea

A specification explains how the software should behave

A (logic) property is a kind of specification that states what should hold for the software

Simple example:

For all lists of integers, the *sort* function should return a list in which the integers occur in order



Idea

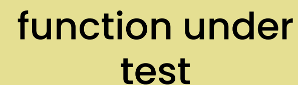
A specification explains what the software should do

A (logic) property is a kind of specification that states what should hold for the software

Simple example:

For all lists of integers, the *sort* function should return a list in which the integers are ordered

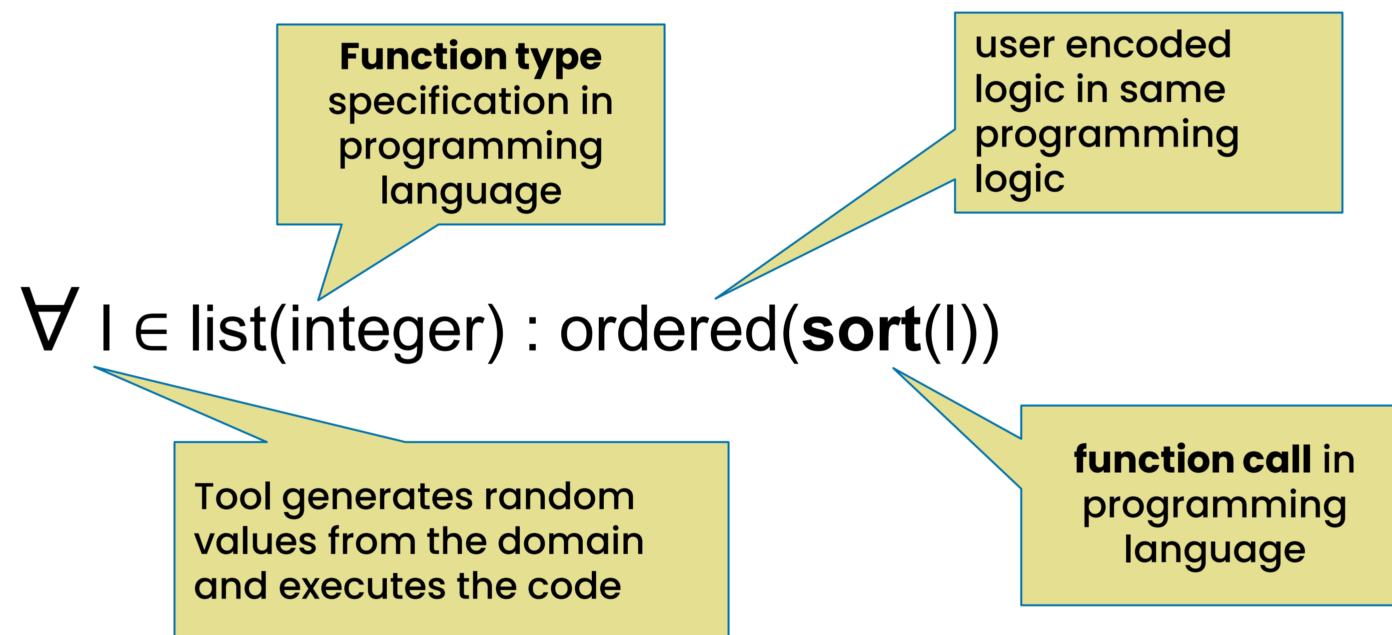
$\forall\, l \in \text{list(integer)} : \text{ordered}(\textbf{sort}(l))$



Idea

Property based testing

Koen Claessen and John Hughes. 2000. QuickCheck: a lightweight tool for random testing of Haskell programs. **SIGPLAN Not. 35**, *9 (Sept. 2000), 268–279. https://doi.org/10.1145/357766.351266*



Example: SMS encoding (ETSI TS 123 042)

Text messages on mobile phones in the early 2000

Use some free bytes in the communication protocol

140 bytes for text

With a little bit of compression,
we can get 160 bytes in there!

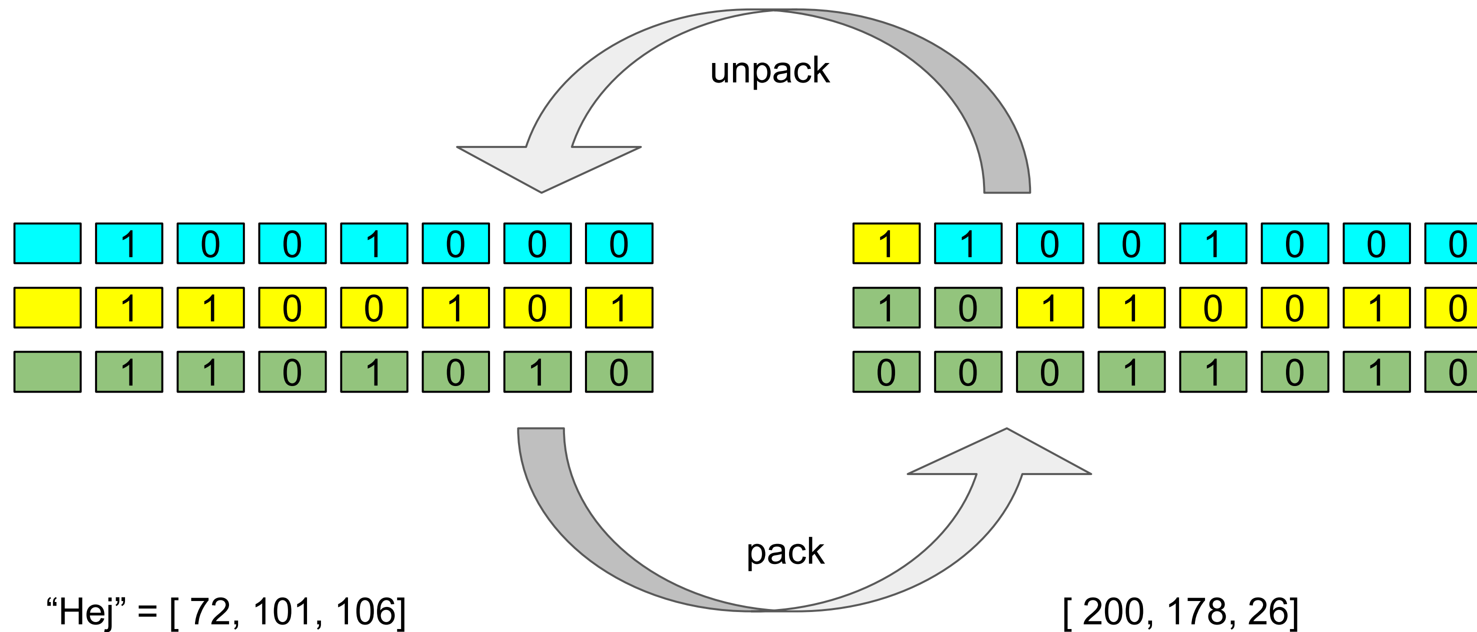
Algorithm: Change UCS2 Row

Example: SMS encoding (ETSI TS 123 042)

3GPP TS 23.038: GSM 7 bit default alphabet (or ASCII)

```
Dec  Char                                Dec  Char        Dec  Char        Dec  Char
---------                                ---------        ---------        ----------
  0  NUL (null)                           32  SPACE        64  @            96  `
  1  SOH (start of heading)               33  !            65  A            97  a
  2  STX (start of text)                  34  "            66  B            98  b
  3  ETX (end of text)                    35  #            67  C            99  c
 …
 22  SYN (synchronous idle)               54  6            86  V           118  v
 23  ETB (end of trans. block)            55  7            87  W           119  w
 24  CAN (cancel)                         56  8            88  X           120  x
 25  EM  (end of medium)                  57  9            89  Y           121  y
 26  SUB (substitute)                     58  :            90  Z           122  z
 27  ESC (escape)                         59  ;            91  [           123  {
 28  FS  (file separator)                 60  <            92  \           124  |
 29  GS  (group separator)                61  =            93  ]           125  }
 30  RS  (record separator)               62  >            94  ^           126  ~
 31  US  (unit separator)                 63  ?            95  _           127  DEL
```

Example: SMS encoding (ETSI TS 123 042)



Example: SMS encoding (ETSI TS 123 042)

10th UCAAT

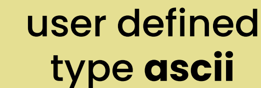
Instead of 3 tests with “random” input,

```
test(X) -> assertEqual(X, unpack(pack(X))).

test(“HEJ”).
test(“1234567890”).
test(“this is a message … of 160 characters long”).
```

Arbitrary many tests with randomly generated inputs

```
property() ->
  ?FORALL(Len, choose(0, 160),
     ?FORALL(Msg, vector(Len, ascii()),
            Msg == unpack(pack(Msg)))).
```



Example: SMS encoding (ETSI TS 123 042)

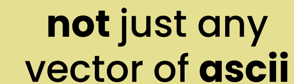
Instead of 3 tests with “random” input,

```
test(X) -> assertEqual(X, unpack(pack(X))).

test(“HEJ”).
test(“1234567890”).
test(“this is a message … of 160 characters long”).
```

Arbitrary many tests with randomly generated inputs

```
property() ->
   ?FORALL(Len, choose(0, 160),
      ?FORALL(Msg, vector(Len, ascii()),
              Msg == unpack(pack(Msg)))).
```



Example: SMS encoding (ETSI TS 123 042)

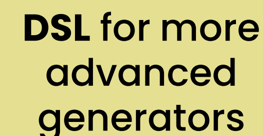
Instead of 3 tests with “random” input,

```
test(X) -> assertEqual(X, unpack(pack(X))).

test(“HEJ”).
test(“1234567890”).
test(“this is a message … of 160 characters long”).
```

Arbitrary many tests with randomly generated input

```
property() ->
  ?FORALL(Len, choose(0, 160),
     ?FORALL(Msg, vector(Len, ascii()),
            Msg == unpack(pack(Msg)))).
```



Example: SMS encoding (ETSI TS 123 042)

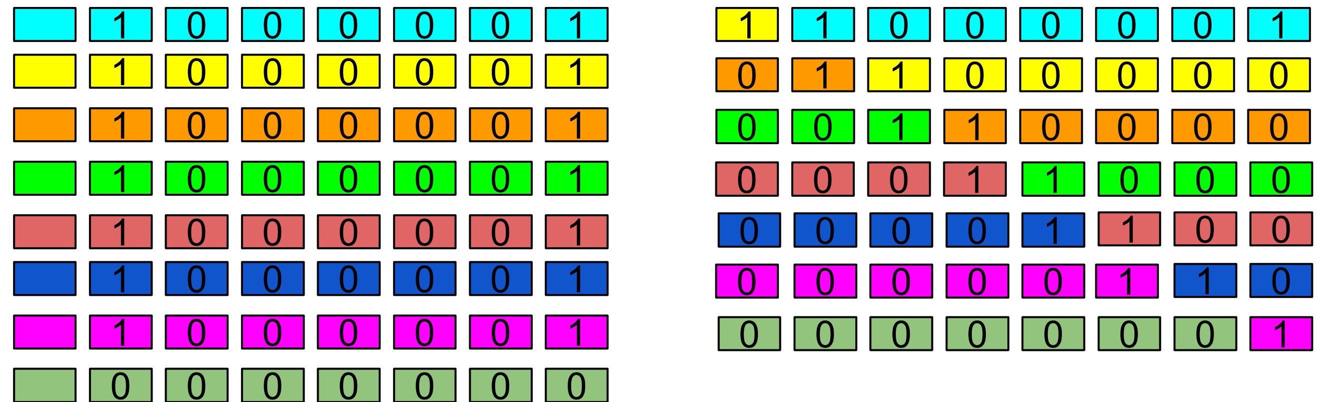
eqc:quickcheck(eqc:testing_time(10, sms_eqc:property())).
....(x10)...(x1)...
Failed! After 993 tests.

[106, 53, 43, 0, 109, 27, 44] /= [106, 53, 43, 0, 109, 27, 44, 0]

Shrinking .xxxxxxxxxx.......xxxxxxxxx(8 times)

[65, 65, 65, 65, 65, 65, 65] /= [65, 65, 65, 65, 65, 65, 65, 0]

SUT drops the last zero

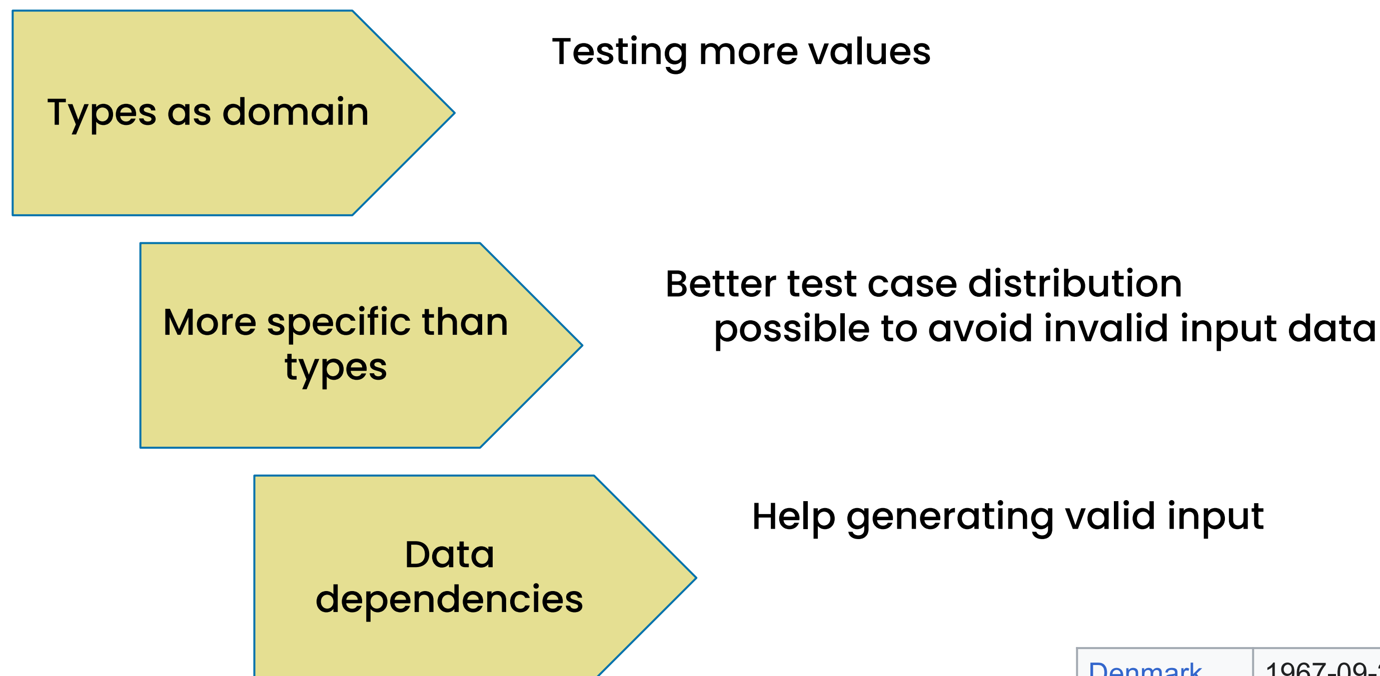


SMS example

With very little effort

- better testing than manually crafted tests
- find border case that fails

DSL for generators



Testing more values

Better test case distribution
possible to avoid invalid input data

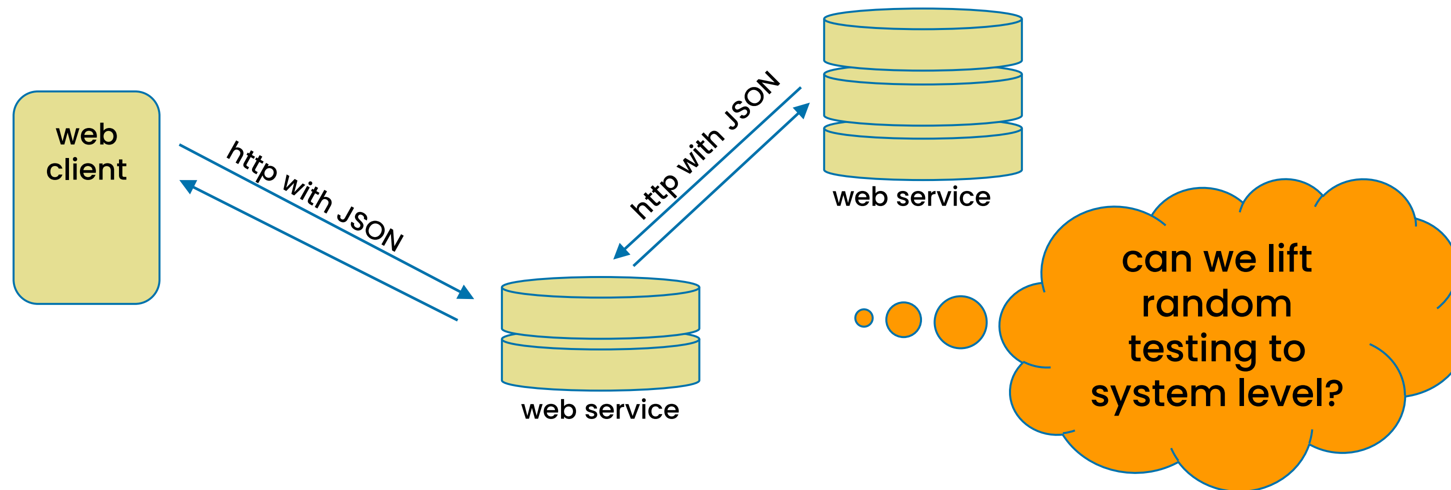
Help generating valid input

Denmark	1967-09-20	DK	NNNN
Japan	1968-07-01	JP	NNN-NNNN
Netherlands	1977-12-31	NL	NNNN AA

Generalize to all API specifications ?

For example

for web services using JSON schema to specify valid request and response data




```yaml
MessageTextCreate:
  type: object
  properties:
    chatId:
      type: string
      format: uuid
    senderId:
      type: string
      format: uuid
    message:
      maxLength: 65535
      minLength: 1
      type: string
    version:
      minimum: 1
      type: integer
      format: int32
```

Generalize to all API specifications ?

Two observations

1) **Controlled random generation**

   Need the DSL, many type notations cannot express dependencies, distributions, etc

1) **Real software has state**

   Just sending rubbish over http won't give good tests

   (even generating an object may require several API calls to create object)

Stateful generators: generation of test sequences

Thomas Arts, John Hughes, Joakim Johansson, and Ulf Wiger. 2006.

Testing telecoms software with quviq QuickCheck. In Proceedings of the 2006 ACM SIGPLAN workshop on Erlang (ERLANG '06). Association for Computing Machinery, New York, NY, USA, 2–10. https://doi.org/10.1145/1159789.1159792

Media proxy implements H.248 protocol

Characteristics of the protocol:

Text based + binary format

Complex internal state

ITU specification 212 pages (Too much freedom)

Interwork description 183 pages (Bind freedom to product)

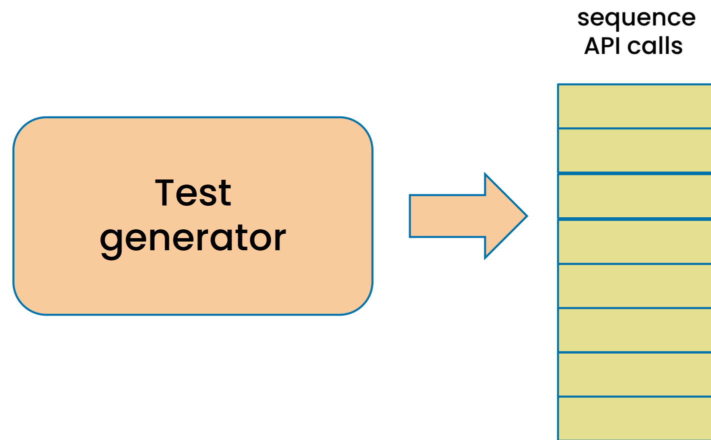
approx 150,000 lines of code in control part

Very typical
ITU specification far too broad
Interwork description defines what is actually implemented

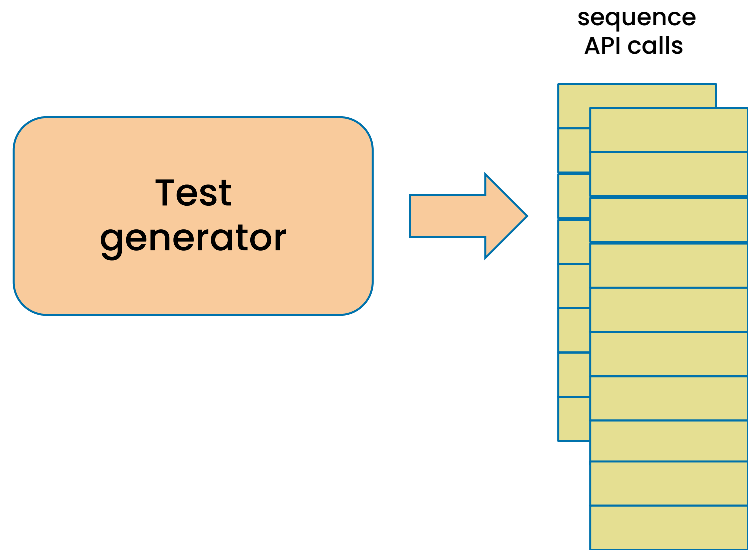
Generation of test sequences

Test generator

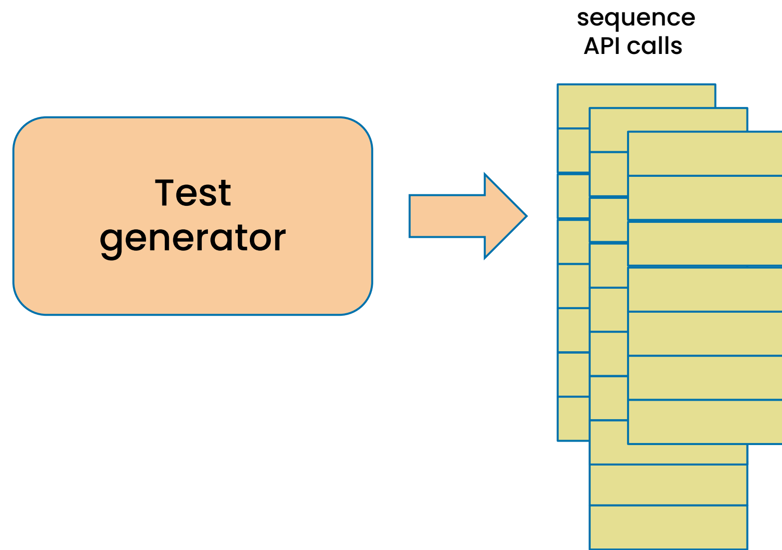
Generation of test sequences



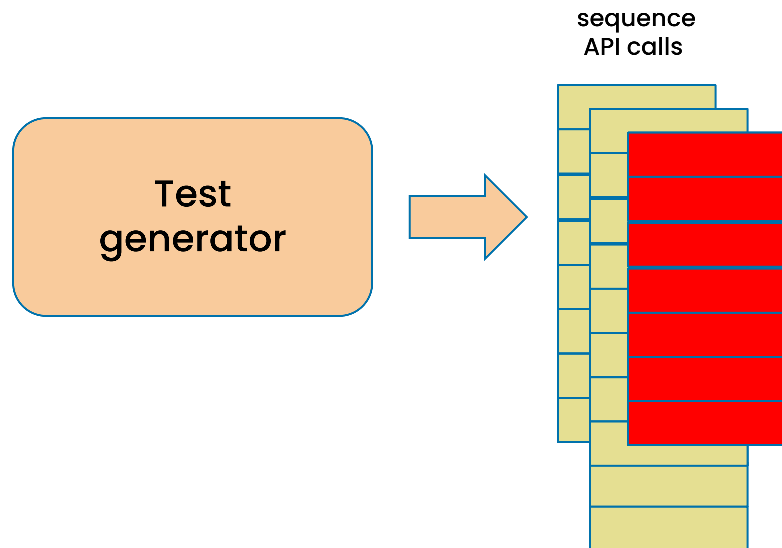
Generation of test sequences



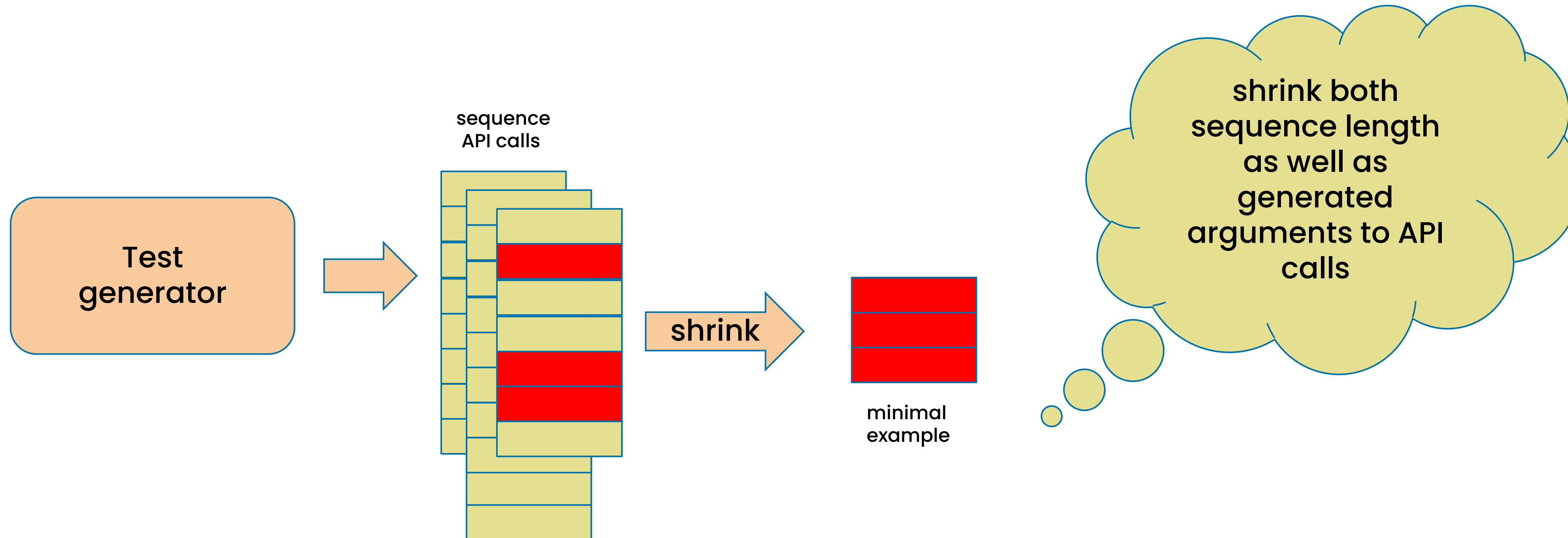
Generation of test sequences



Generation of test sequences



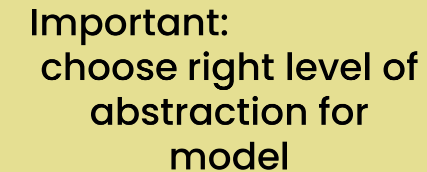
Generation of test sequences

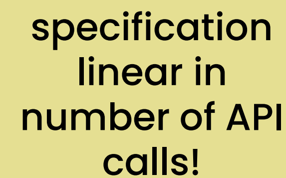


Generation of test sequences

Specification is stateful model for API

initial state
for each API
precondition: possible in this state?
generate arguments for the API call
next state: update the model state given the call
postcondition: is SUT result comptable with model state





Generation of test sequences

Example H.248

initial state: no calls

ADD args: random choose call ID, or none if first
parameters for the call

ADD pre: less than 2 call ID in call

ADD: adapter to call the SUT with given arguments

ADD next: add new call (first) with returned caller ID
or new caller to existing call in state

ADD post: Check result of ADD is compatible with model
(returns the right thing, e.g. not an error)

Generation of test sequences

Example H.248

SUB pre: is there an ongoing call?

SUB args: random choose call ID with callers
parameters for subtract

SUB: adapter to call the SUT with given arguments

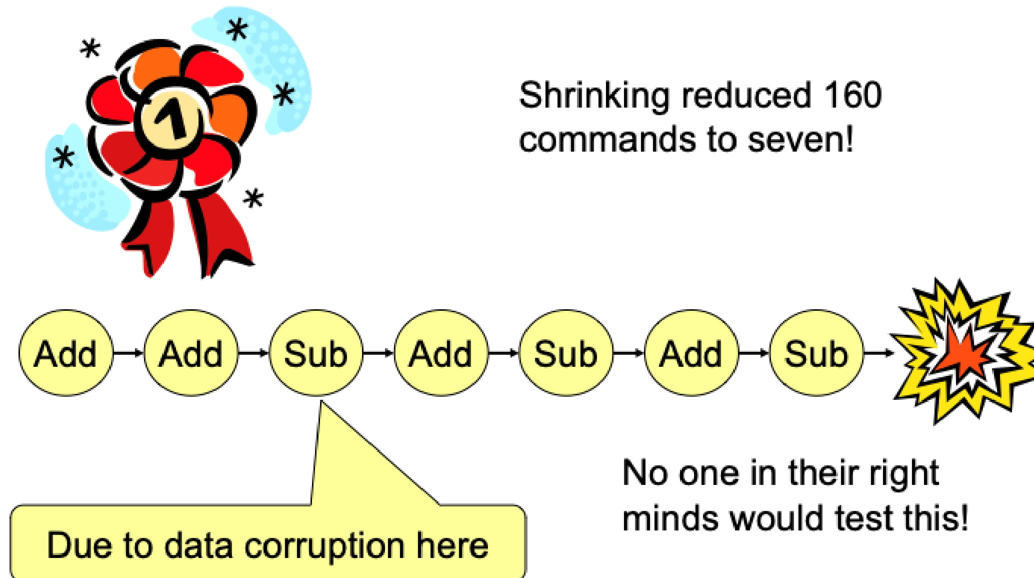
SUB next: remove caller from calls in state

SUB post: Check result of SUB is compatible with model
(returns the right thing, e.g. not an error)

Generation of test sequences

Shrinking of utmost value!

Testing H.248 media proxy implementation



Scaled to industrial examples

20 years of R&D to adapt to industrial needs

3,000 pages of specifications

20,000 lines of QuickCheck

1,000,000 LOC, 6 suppliers

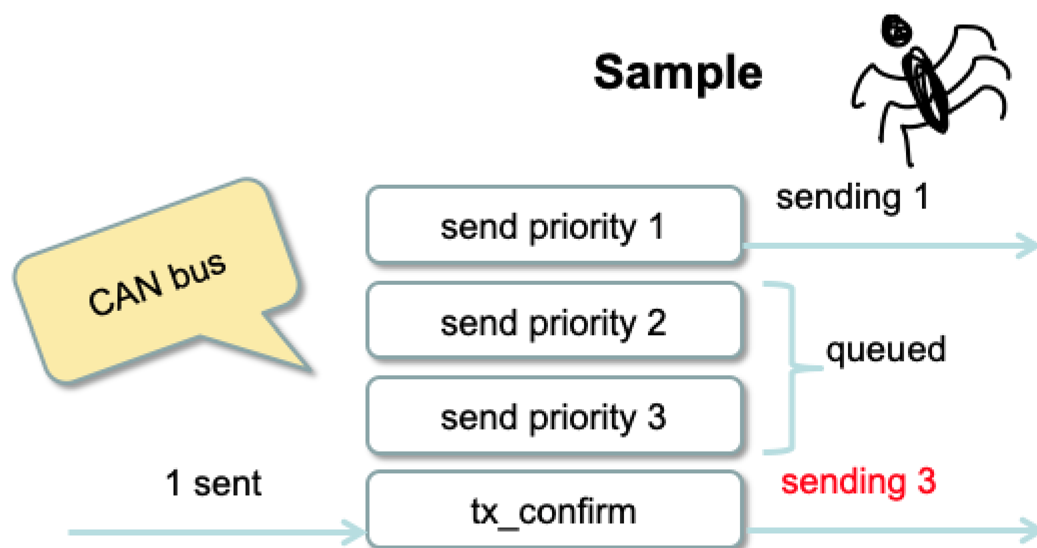
200 problems

100 problems in the standard

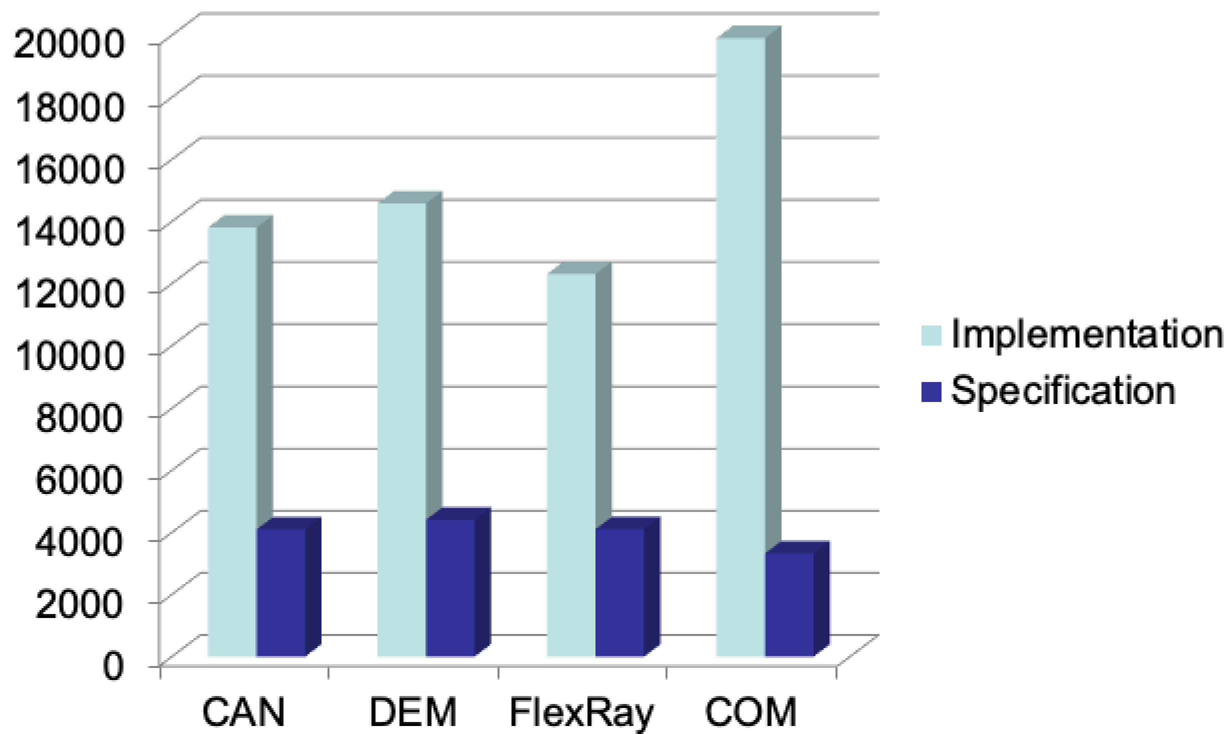
10x shorter test code

Scaled to industrial examples

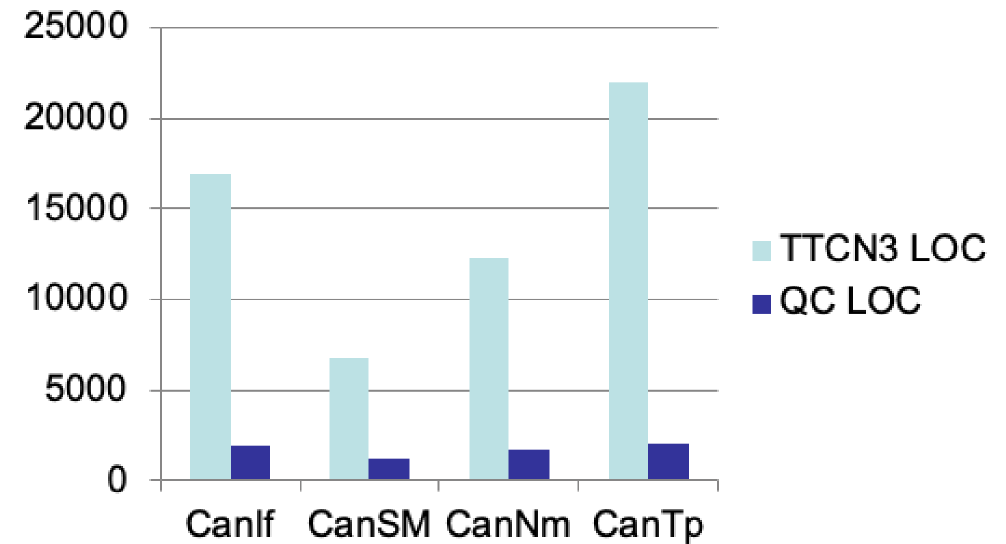
Sequences reveal faults



Scaled to industrial examples



TTCN-3 test suite for CAN modules:
245 test cases, 58KLOC



Scaled to industrial examples

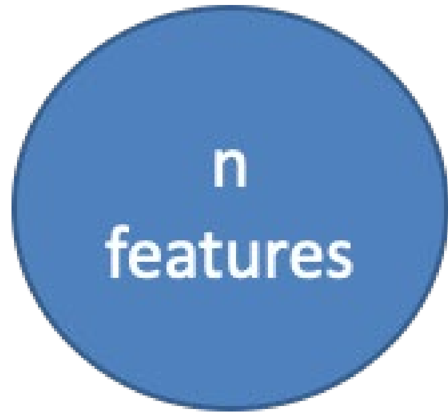
Techniques that make this method scale

Scaled to industrial examples

Linear in size of API

Manually written test cases do not scale!

Why is testing hard?



$O(n^3)$ test cases

triples of features

features	tests	additional tests when adding 1 feature
20	80 + 380 + 6840	4 + 20 + 380

Scaled to industrial examples

Linear in size of API

Manually written test cases do not scale!

Property based testing needs linear amount of code per API call…
…in theory all combinations of interactions can be generated

Scaled to industrial examples

Distribution of commands

If SUT has 192 API calls, then reaching subsequence like

ADD ADD SUB ADD

is difficult with uniform distribution

- create longer sequences and shrink to shorter faulty sequences
- guide distribution

  model encodes weight depending on state and command
  (more ADD than SUB, for example, to get full calls)

Scaled to industrial examples

10th UCAAT

Statistics: what has been tested

Use the state during testing to record whether a specific requirement has been tested

Thomas Arts, John Hughes, "How Well are Your Requirements Tested?", *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp.244-254, 2016.

Scaled to industrial examples

Positive and negative testing

Re-use same model
preconditions that prevent a command to be executed skipped
postcondition changed to expect an error if precondition was invalid

Run either with preconditions filtering failure cases
… or with always executing command and validate that it fails

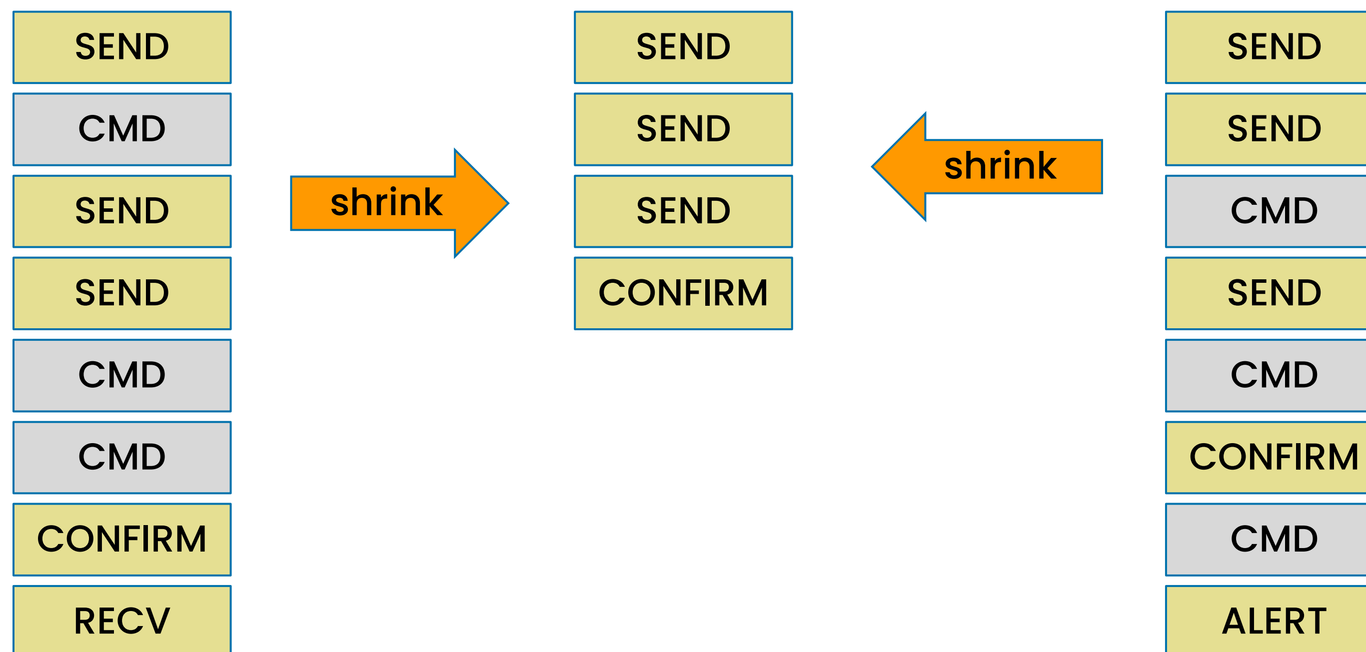
Combinators to steer fault injection distribution (negative tests not too often)

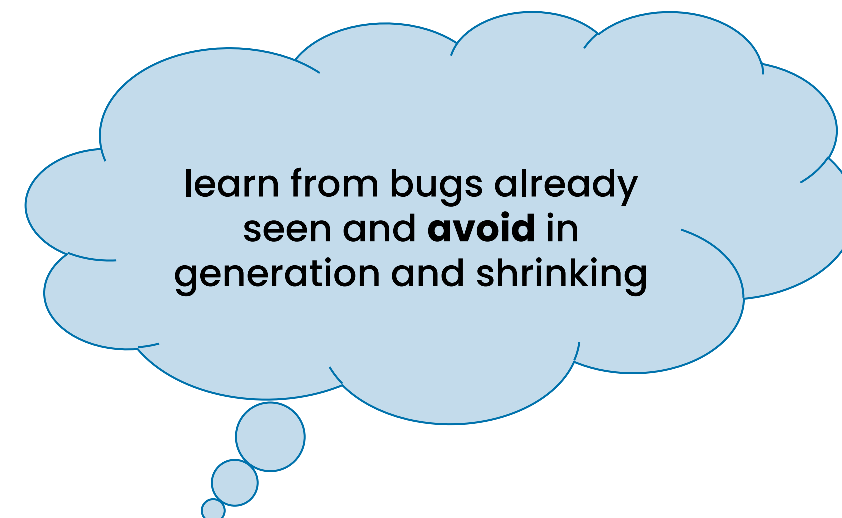
See also:
Vedder, B., Arts, T., Vinter, J., & Jonsson, M. (2013, November). Combining fault-injection with property-based testing. In *Proceedings of International Workshop on Engineering Simulations for Cyber-Physical Systems* (pp. 1-8).

Scaled to industrial examples

Avoid known bugs

Many different sequences of commands may shrink to the same minimal failing case.





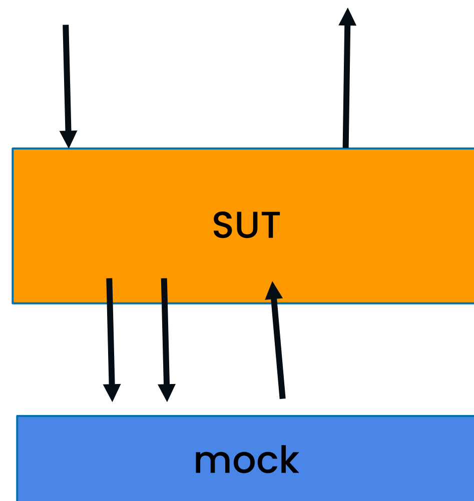
John Hughes, Ulf Norell, Nick Smallbone, Thomas Arts, "Find More Bugs with QuickCheck!", *2016 IEEE/ACM 11th International Workshop in Automation of Software Test (AST)*, pp.71-77, 2016.

Scaled to industrial examples

Mocking

How to do mocking when you generate a random test?

- A language to express mocked response on given model state
- Compute the mocked responses before each command execution



Svenningsson, J., Svensson, H., Smallbone, N., Arts, T., Norell, U., Hughes, J. (2014). An Expressive Semantics of Mocking. In: Gnesi, S., Rensink, A. (eds) Fundamental Approaches to Software Engineering. FASE 2014. Lecture Notes in Computer Science, vol 8411. Springer, Berlin, Heidelberg.

Scaled to industrial examples

Testing for race conditions

- If actions are considered atomic, run them in parallel and check that results can be explained with model
- Take control over scheduler… generate random schedules and shrink them to scheduled with minimal context switches

See also
John Hughes, Benjamin C. Pierce, Thomas Arts, Ulf Norell, "Mysteries of DropBox: Property-Based Testing of a Distributed Synchronization Service", *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp.135-145, 2016.

Questions?