Test Generation for Codec Validation

TTCN-3 User Conference 2005

Business Card

- Thomas Deiß
- Nokia Research Center Bochum
- Principle Scientist
  - TTCN-2, TTCN-3, SDL, ASN.1, C
  - Test System Development
- thomas.deiss@nokia.com

- Dirk Jebing
- University of Dortmund now CAE

Background Situation

- TTCN-3 Test System Development
- Validation of the test system independently of the System Under Test

Overview

- Motivation
- Test Setup
- Test Data Generation
- Conclusion

Overview

- Motivation
- Test Setup
- Test Data Generation
- Conclusion

Motivation

Testsystem

```
alt {
  [] pt.receive( DHCPOffer:? ) {
     ...
     setverdict( pass )
     }
  [] pt.receive {
     setverdict( fail )
     }
  ...
}
```

SUT

DHCPOffer

Find Errors in the SUT,
not in the test system

Automated Testing of Codecs

- Codecs provide rather limited functionality
- Take a value, compute a string
- Take a string and a type, compute a value

- Type $\mathcal{T}$
  Set of (bit, octet, char, ...) strings: $\mathcal{S}$
- encode: $\mathcal{T} \rightarrow \mathcal{S}$
  decode: $\mathcal{S}$ x 2^$\mathcal{T}$ -> $\mathcal{T}$

```
dhcpv4Discover := {
  opcode       := e_request,
  hwAddrType   := 0,
  hwAddrLen    := 127,
  hops         := 0,
  … }
```

```
0x 01007F00
   3FFFFFFF
   ...
```

© NOKIA

Overview

- Motivation
- Test Setup
- Test Data Generation
- Conclusion

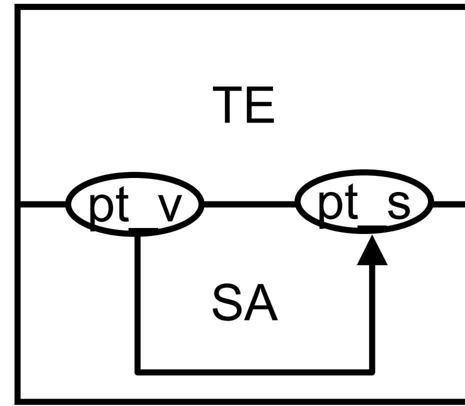
General Approach

- Generate codec test cases automatically
  - Inputs to test case generation must have a formal meaning
- Available inputs
  - Protocol definitions, description of types and encodings
  - TTCN-3 type definitions
  - Codec implementation
- TTCN-3 itself can be used for test case execution
  - When executing test cases the codecs are executed

Use Encoded and Decoded Data

- Testcases consist basically of two values
  - TTCN-3 value (**a_msg**)
  - Encoded value as a string (**a_string**)
- Encoding

```
pt_v.send( a_msg );
alt {
  [] pt_s.receive( a_string ) {
     setverdict( pass ) }
  [] pt_s.receive {
     setverdict( fail )
}
```

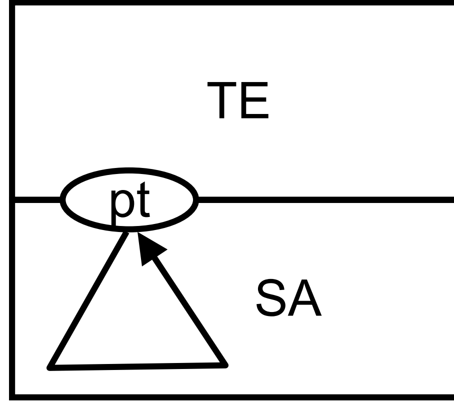


- How to compute the encoded values?

Use Decoded Data Only

- TTCN-3 values only are used
- Encoding followed by decoding
  - decode(encode(**a_msg**), T) == **a_msg**

```
pt.send( a_msg );
alt {
  [] pt.receive( a_msg ) {
     setverdict( pass ) }
  [] pt.receive {
     setverdict( fail )
}
```



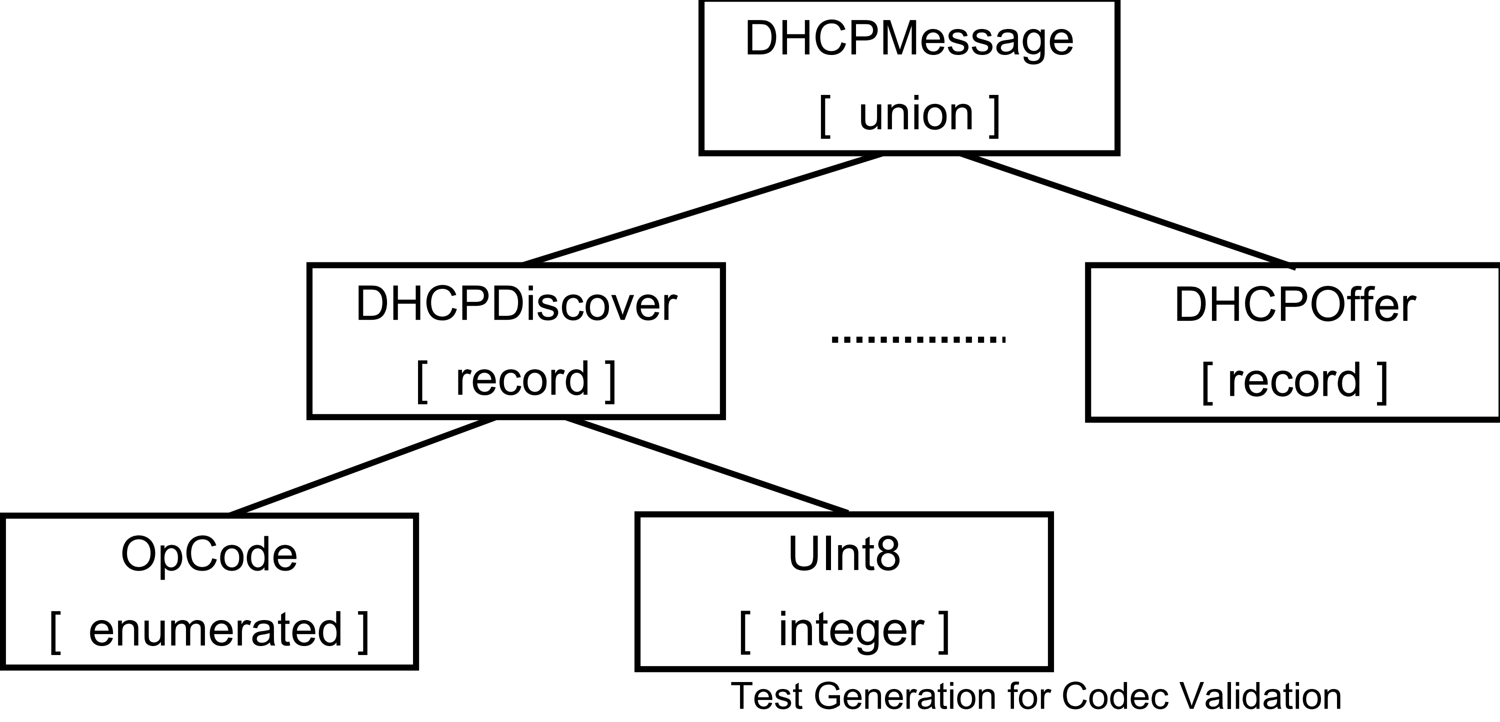
- Conceptual errors might be masked
  - E.g. usage of host byte order instead of network byte order

Overview

- Motivation
- Test Setup
- Test Data Generation
- Conclusion

TTCN-3 Types as Trees

- Basically, generate values for a given TTCN-3 type
  - Additional code for testcases, control part, etc. needed
- Generation of all values is infeasible



Random Strategy

- Generate a requested number of values randomly
- Choice points
  - Alternative in a union
  - Presence of optional fields
  - Length of strings and values of string like types
  - Values of basic types
- Subtyping is observed

Values of Basic Types

- Apply boundary value analysis
  - `type integer UInt16 (0 .. 65535)`
  - `0, 1, 65534, 65535`
  - Valid values only: -1, 65536 are not generated
- Use experience from codec implementation
  - E.g. byte boundaries
  - `127, 128`
- Randomly generate values up to the given limit
- String and string like types
  - `type charstring Name length ( 0 .. 10 )`
  - `"", "a", "abcdefghij"`

Number of Testdata at Leaves

- Set a bound on the number of test data for leaves
  - Stop generating when number reached
  - Supply randomly generated values if not enough values can be generated systematically
- Reduce this bound with the depth of the tree
- Reduction of number of test data

© NOKIA Originator: Thomas Deiß / 11-May-05 / my/ Page:16

Values of Complex Types union, record of

- `union`
  - Take values for each alternative
- `record of`
  - Similar to strings
  - use different values for each element
  - Consider length restrictions similar to strings
  - n^2 values of length 2 if there are n values for the element type sequences of length 2 are not generated

Values of Complex Types
Record

- `record`:
  - Combining all test data for fields is not feasible
    Combinatorial explosion
  - All optional fields set present
  - All optional fields set absent
  - No combinations of values
  - Fields with the same – non-basic – type: iterate over only one field
    `type record Pair { Element f1, Element f2 }`
    Assume that fields are encoded in the same way
- Amount of test data linear in the number of nodes in the type tree

© NOKIA Originator: Thomas Deiß / 11-May-05 / my/ Page:18

Generated Code

- Effect on the size of generated code
  - Test data as constants or as templates
  - Factor 2 in size of generated C-code
  - Very much tool dependent
- Very large amounts of TTCN-3 code can be generated

Overview

- Motivation
- Test Setup
- Test Data Generation
- Conclusion

Experiments and Results
Amount of Testdata

- DHCPv4 used for experiments
  - TTCN-3 type and C codecs taken from real test system
- Size of type tree
  - Maximum height: 12
  - Number of nodes: 241
- Linear-Growth Strategy
  - #leave values #test cases

#leave values	#test cases
2	350
4	650
6	854
8	1010

Experiments and Results
Comparison of Strategies

- Few empirical data
  DHCP and SIP
- Random strategy is surprisingly strong
  - Fault discovery rate
  - Further investigation needed

Limitations and Problems

- Masking of conceptual errors
- Incomplete codecs
  - User can specify to ignore certain parts of the type definition
- Insufficiently defined data types
  - `type charstring IPv4Address; // sd%h#68.9§`
  - User can specify values for certain types
  - Not all restrictions can be expressed
- Partial codecs
  - E.g. codec checks whether conditions between optional fields hold and refuses to decode or encode if not
  - Methodological problem: what is the task of a codec?
- Dependent fields
  - E.g. length fields in the type definition

NOKIA

CONNECTING PEOPLE

Thank you for both your participation and for your attention