Network Packet Header Generation Using Graph Based Techniques combined with Software Testing Strategies

Sridevi Navulur
Satheesh Parasumanna
Rama Chaganti
Agenda

• Application Overview
• Goals and Main Idea
• Strategy and Approach
• Implementation
• Evidence and Results
• Conclusion
• Limitations & Future Scope
• Q&A
Application Overview

- Simulation-based Network Processor Verification presents unique challenges due to the large number of packet constructions and requirements to model and measure those that are legal.
Motivation & Challenges

• Motivation:
  – Simplify generation of complex packet or frame header information and exhaustively test all random permutations and combinations of legal network protocol headers.

• Challenges
  – Huge space of legal combinations to cover
  – Computational expensive simulation cycles
  – Cumbersome hand written cover groups and hence no metrics for what has been covered.
Goals & Main Idea

**Stimulus Generation Goals & Modeling Approaches tried**

<table>
<thead>
<tr>
<th>Stimulus Generation Goals &amp; Approaches tried</th>
<th>Functional Coverage Generation Approaches tried</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rand-sequence:</strong></td>
<td><strong>Hand-coding:</strong></td>
</tr>
<tr>
<td>o Rand-sequence to cover all the combinations</td>
<td>o A huge task in identifying and coding to cover all the protocols header types</td>
</tr>
<tr>
<td>o Multiple hand-coded tests with different rand-sequences for path coverage</td>
<td>o ~800 lines of hand-written Cover point code for SV Transition coverage</td>
</tr>
<tr>
<td>o No automatic functional coverage generation support</td>
<td>o Difficult to handle illegal and ignore bins for SV cover points and cross points</td>
</tr>
</tbody>
</table>

**Recursive Graph:** *Initial solution*

*Cons:*
- Lack of recursion depth control
- No automatic functional coverage generation support

**Pairwise Graph:**
Final solution, addressed recursion issues and enabled functional coverage generation

**Graph Based:**
- Graph tool supports automatic creation of pairwise SV cover points & crosses.
- Generates cross of previous header type and index of next header type
- Automatic generation of previous & next SV illegal bins restrict solutions to legal subset
Strategy and Approach

- Uses a Portable Stimulus solver
  - Constraint-based stimulus description
  - Goal-based value generation
- User developed portable stimulus rules
- Analyzes variables for reachability
- Identifies strategy-specific (pair-wise) goals
- Tool creates
  - Stimulus-generator class
  - Strategy-specific SV cover group
- Why pair-wise testing?
  - Proven approach from the software testing domain
  - Strategize to select pairs, triples, quads of variable to exercise
  - Pairwise strategy sufficient for this application
- Pairwise Graph Structure
  - Previous/Next State options tracked
  - Scalable to large sequential headers
  - Tool generates SV Coverage classes

Pair-wise Strategy Explained
For a Test System S with 3 inputs, X, Y, Z
Where
X can be \{A,B\}
Y Can be \{C,D\}
Z can be \{E,F\}

There can be 2**3 = 8 combinations of test inputs to try out.

<table>
<thead>
<tr>
<th>Test id</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>T2</td>
<td>A</td>
<td>D</td>
<td>F</td>
</tr>
<tr>
<td>T3</td>
<td>A</td>
<td>C</td>
<td>F</td>
</tr>
<tr>
<td>T4</td>
<td>A</td>
<td>D</td>
<td>F</td>
</tr>
<tr>
<td>T5</td>
<td>B</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>T6</td>
<td>B</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>T7</td>
<td>B</td>
<td>C</td>
<td>F</td>
</tr>
<tr>
<td>T8</td>
<td>B</td>
<td>D</td>
<td>F</td>
</tr>
</tbody>
</table>

With Pair-wise strategy we can test with minimum combinations.
\{(A, , E)\} is covered in T1
\{(A, , D)\} is covered in T4
\{(D, , E)\} is covered in T6
Thus T2 can be removed/eliminated etc.

Number of combinations can be reduced to achieve coverage by using 4 tests. This is the main idea of PairWise Strategy.
Implementation: Stimulus Generator

Portable Stimulus Rule file
Example Application Rules:

- Every Packet in this example would start with Ethernet Header and can traverse through multiple types of headers of various combinations and construct a packet.

- Maximum depth of the header types can also be changed. In this application example we limited the MAX Depth of the header combinations to be 8 at any given time.

```c
// next constraint keeps hdr sequences confined to be legal within MAXPKT=8 reg.
// developed these by inspecting the unique sequences generated running graph random
constraint maxpkt_c {
    prevH_idx >= 6 } -> {nextH_hdr_type outside [vlan_e, ipv4_e, ipv6_e]};
(constraint maxpkt_c {
    prevH_idx >= 5 } -> {nextH_hdr_type outside [vlan_e]};
    // following creates functional coverage exclusion logic that assures MAXPKT compliance regardless of why
    // recommend changing the generated f-cov code ignore_bins to illegal_bins to trap these cases as errors
    if (prevH_idx==7) {nextH_hdr_type outside [vlan_e, ipv4_e, ipv6_e, tcp_e, sctp_e, udp_e]}; // exclude sec
};

constraint cov_restrictions_c_coverage {
    prevH_hdr_type inside [eth_e, vlan_e, ipv4_e, ipv6_e, tcp_e, udp_e, sctp_e]; // you can reduce this set !
    nextH_hdr_type inside [eth_e, vlan_e, ipv4_e, ipv6_e, tcp_e, udp_e, sctp_e, done_e, nop_e]; // .reduce
    prevH_hdr_type != nop_e; // dont tabulate nop cases
    prevH_hdr_type != done_e; // dont tabulate anything after done (ie, nop)
};
```

```c
release4 test = init repeat {
    init_hdr
    get_first_hdr(firstH)
    repeat RPTCNT {
        start_cov
        n_eth n_vlan n_ipv4 n_ipv6 n_tcp n_udp n_sctp n_done n_nop
        prevH
        get_next_hdr(nextH)
        end_cov
    }

    configure hdrs // action will implement p.cfg_hdrs in final imp
    infact_checkcov
}
```
Implementation of Graph

• **Stimulus graph traversal**

Example PX file

```cpp
class Ethernet :: Section(1) {
    struct {
        dmac : 48,
        smac : 48,
        tpid : 16
    }
    map types {
        (VLAN_TYPE, VLAN),
        (IPv4_TYPE, IPv4),
        (IPv6_TYPE, IPv6),
        done(ERR_ETHERNET)
    }
    method increment_offset = ParserTuple.offset + 48 + 48 + 16;
    method move_to_section = types(tpid);
}
```
Implementation of test

Portable Stimulus UVM test

```plaintext
hdr_seq = {}; // new pktdlh
p = new();
// populate an infact header sequence
i_comp_inst.ifc_get_first_hdr(ihdr_seq.hdr_arr[0]);
for (int i=1; i<MAXPKT; i++) begin
    i_comp_inst.ifc_get_next_hdr(ihdr_seq.hdr_arr[i]);
    if (ihdr_seq.hdr_arr[i].hdr_type == infact_test_pkg::done_e) ihdr_seq_len = i;
    else if ((ihdr_seq.hdr_arr[i].hdr_type == infact_test_pkg::tcp_e) || (ihdr_seq.hdr;
        //display("HDR Seq length \%d", ihdr_seq_len);
end
//assert(ihdr_seq_len > 0 && ihdr_seq_len <= `NUM_PKTS);

for (int i=0; i<MAXPKT; i++) begin
    case (ihdr_seq.hdr_arr[i].hdr_type)
        eth_e : hdr_seq.push_back(p.eth[n_eth++]);
        vlan_e : hdr_seq.push_back(p.dot1q[n_vlan++]); // verify this is actually vlan
        ipv4_e : hdr_seq.push_back(p.ipv4[n_ipv4++]);
        ipv6_e : hdr_seq.push_back(p.ipv6[n_ipv6++]);
        tcp_e : hdr_seq.push_back(p.tcp[n_tcp++]);
        sctp_e : hdr_seq.push_back(p.sctp[n_sctp++]);
        udp_e : hdr_seq.push_back(p.udp[n_udp++]);
        done_e : hdr_seq.push_back(p.data[n_data++]);
        nop_e : ; // occurs for short hdr sequences, ignore
    default : // error
        begin
```
Evidence & Results

- Compare Pairwise-driven generation and Rand-Sequence
  - Use Ethernet packet
  - Apply pairwise strategy
- Pairwise-driven stimulus easily hits all cases
- Replace a suite of random+directed tests with 1 portable stimulus test
  - 12 directed+CR tests (~1800 lines of code) replaced with 1 portable stimulus test (220 lines of code) attaining 95% code coverage and 100% functional coverage. Coverage snippet below.
Conclusion

- **Pairwise-driven** generation targets high-value cases
  - Value ranges across variable domains
  - Constraint-implied corner cases
  - All-pairs value combinations
- Automatically hits key cases and all legal combinations of headers
- Automate cover group generation and thus add metrics on which cases are being exercised.
- Graph-based Portable Stimulus very effective for this application
Limitation & Future Scope

- **Limitations**: Every time design spec changes, the stimulus rule graph needs to be updated.
- **Future Scope**: Automate the ability to choose various rule graphs with respective strategies

References

- http://www.pairwise.org/
Q&A