Towards Example-guided Network Synthesis

Haoxian Chen, University of Pennsylvania
Anduo Wang, Temple University
Boon Thau Loo, University of Pennsylvania
Network management is challenging

• Low-level, vendor-specific configurations
  - complex (~1000 lines in a Cisco router)
  - error-prone (AWS outage 2017)

• Alternative: Software-defined networking (SDN)
  - mitigates distributed complexity by centralized view
  - but controller programs are still complicated to implement
  - high-level Domain-Specific Languages (DSL) reduce lines of codes, but have steep learning curve ([Frenetic], [Pyretic], [FlowLog])
Our solution: networking by input-output examples

1. Network operator provides some input-output (I/O) pairs
   - this work focus on I/O of the controller program in SDN

2. Computer automatically synthesizes a program
Example: stateful firewall

The underlying network

Synthesizer interface

Input
PacketIn
Controller

Output
Fwd

One packet from 10.0.0.1 to 172.217.11.46 is allowed.

Two packets to 172.217.11.46 are blocked.

A packet from 119.212.8.8 to 10.0.0.2 is blocked.

A packet from 172.217.11.46 to 10.0.0.1 is allowed.

A packet from 172.217.11.46 to 10.0.0.2 is blocked.

A packet from 172.217.11.46 to 10.0.0.3 is blocked.
Design space

Synthesis target: controller programs v.s. data plane configurations
Design space

Synthesis target: controller programs

• Understandable to human
• Verifiable
• Compose with other programs to form complex features [Frenetic]
• Reuse in other settings
Synthesize NDLog program

Leverage the compactness of NDLog programs

Smaller search space for program synthesis

NDLog program
Synthesize NDLog program

NDLog evaluates each rule independently

so that we can synthesize one rule at a time
Background: NDLog

- One of the Logic-programming family.
- Inputs and Outputs are organized as structured tables.
- Program consists of a set of rules.
- Rules transform input to output

<table>
<thead>
<tr>
<th>SrcIP</th>
<th>DstIP</th>
<th>InPort</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>1</td>
</tr>
<tr>
<td>10.0.0.3</td>
<td>10.0.0.2</td>
<td>2</td>
</tr>
<tr>
<td>10.0.0.4</td>
<td>10.0.0.5</td>
<td>1</td>
</tr>
</tbody>
</table>

**Input: packetIn**

**Output: fwd**

\[
\text{fwd}(\text{IP}, \text{Port}) \leftarrow \text{packetIn}(\text{SrcIP}, \text{DstIP}, \text{InPort}), \\
\text{IP} = \text{DstIP}, \text{InPort} = \text{Port}.
\]

<table>
<thead>
<tr>
<th>IP</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.2</td>
<td>1</td>
</tr>
<tr>
<td>10.0.0.2</td>
<td>2</td>
</tr>
<tr>
<td>10.0.0.5</td>
<td>1</td>
</tr>
</tbody>
</table>
Example-guided synthesis: An overview

An NDLog program consists of a set of symbolic rules:

- \[ \text{Fwd}(\text{swi}, \text{dstIP}, \text{srcIP}, \text{prt}) :- \text{PacketIn}(\text{swi}, \text{srcIP}, \text{dstIP}, \text{prt}), \text{InBound}(\text{swi}, \text{prt}). \]
- \[ \text{Fwd}(\text{swi}, \text{srcIP}, \text{dstIP}, \text{prt}) :- \text{PacketIn}(\text{swi}, \text{srcIP}, \text{dstIP}, \text{prt2}), \text{InBound}(\text{swi}, \text{prt2}), \text{Outbound}(\text{swi}, \text{prt}). \]

### Input-output

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>PacketIn</td>
<td>Fwd</td>
</tr>
<tr>
<td>10.0.0.1 -&gt; 172.217.11.46</td>
<td>10.0.0.1, port 2</td>
</tr>
<tr>
<td>PacketIn</td>
<td>Fwd</td>
</tr>
<tr>
<td>10.0.0.1 -&gt; 172.217.11.46</td>
<td>172.217.11.46, port 1</td>
</tr>
</tbody>
</table>

### Symbolic Rules

- \[ \text{Fwd}(\text{swi}, \text{dstIP}, \text{srcIP}, \text{prt}) :- \text{PacketIn}(\text{swi}, \text{srcIP}, \text{dstIP}, \text{prt}), \text{InBound}(\text{swi}, \text{prt}). \]
- \[ \text{Fwd}(\text{swi}, \text{srcIP}, \text{dstIP}, \text{prt}) :- \text{PacketIn}(\text{swi}, \text{srcIP}, \text{dstIP}, \text{prt2}), \text{InBound}(\text{swi}, \text{prt2}), \text{Outbound}(\text{swi}, \text{prt}). \]
Synthesis algorithm

1. Divide-and-conquer principle: one rule at a time, combine them into the final program
   - because NDLog evaluates each rule independently

2. Prune search space
   - Only search within the syntax-correct rule space
**Synthesis algorithm**

Find the set of rules cover all examples

Fwd(Switch, Dst, Src, Port) :-
PacketIn(Switch, Src, Dst, Port),
InBound(Switch, Port).

<table>
<thead>
<tr>
<th>Inbound</th>
<th>Outbound</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch 1</td>
<td>switch 1</td>
</tr>
<tr>
<td>port 1</td>
<td>port 2</td>
</tr>
</tbody>
</table>

**background knowledge**

**Input-output examples**

<table>
<thead>
<tr>
<th>PacketIn</th>
<th>Fwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch 1, 10.0.0.1 -&gt; 172.217.11.46, port 1</td>
<td>switch 1, 172.217.11.46, 10.0.0.1, port 1</td>
</tr>
<tr>
<td>switch 1, 10.0.0.1 -&gt; 172.217.11.46, port 1</td>
<td>switch 1, 10.0.0.1, 172.217.11.46, port 2</td>
</tr>
</tbody>
</table>
Synthesize individual rule

Inbound
switch 1 | port 1
Outbound
switch 1 | port 2

background knowledge

Input-output examples

<table>
<thead>
<tr>
<th>PacketIn</th>
<th>Fwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch 1, 10.0.0.1 -&gt; 172.217.11.46, port 1</td>
<td>switch 1, 172.217.11.46, 10.0.0.1, port 1</td>
</tr>
<tr>
<td>switch 1, 10.0.0.1 -&gt; 172.217.11.46, port 1</td>
<td>switch 1, 10.0.0.1, 172.217.11.46, port 2</td>
</tr>
</tbody>
</table>

relation name variable names

? (? ,?) :- ?(?,?,?), ...(Order of relations within the rule body does not matter)

Skeleton of an NDLog rule

4 possible Relation Names:
PacketIn, Fwd, Inbound, Outbound

Fwd(? ,?) :- PacketIn(? ,?,?,?), Inbound(? ,?), Outbound(? ,?).
Synthesize individual rule

<table>
<thead>
<tr>
<th>Inbound</th>
<th>Outbound</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch 1, port 1</td>
<td>switch 1, port 2</td>
</tr>
</tbody>
</table>

**background knowledge**

<table>
<thead>
<tr>
<th>Input-output examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PacketIn</td>
<td>Fwd</td>
</tr>
<tr>
<td>switch 1, 10.0.0.1 -&gt; 172.217.11.46, port 1</td>
<td>switch 1, 172.217.11.46, 10.0.0.1, port 1</td>
</tr>
<tr>
<td>switch 1, 10.0.0.1 -&gt; 172.217.11.46, port 1</td>
<td>switch 1, 10.0.0.1, 172.217.11.46, port 2</td>
</tr>
</tbody>
</table>

Fwd(? ,?) :- PacketIn(? ,? ,? ,?), Inbound(? ,?), Outbound(? ,?).

Enumerate on all possible variable instantiation, until we find a rule that covers some examples.
Preliminary results

Synthesis programs:

• Reachability
  - Query if any pair of nodes can reach each other in the network

• MAC learning switch

• Stateful firewall

• App-based forwarding
  - Look up forward destination by application
Preliminary results

These reductions come from two insights:
1. factor program into individual rules
2. type information

<table>
<thead>
<tr>
<th>Program (# possible programs)</th>
<th># rules tried</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>reachability (10^5)</td>
<td>226</td>
<td>0.4</td>
</tr>
<tr>
<td>MAC learning (10^6)</td>
<td>11</td>
<td>0.02</td>
</tr>
<tr>
<td>stateful firewall (10^11)</td>
<td>13497</td>
<td>72</td>
</tr>
<tr>
<td>APP-based forwarding (10^14)</td>
<td>28829</td>
<td>149</td>
</tr>
</tbody>
</table>

- The major bottleneck of synthesis efficiency comes from the enumerative nature.
- Examples were carefully hand-crafted, in order to synthesize correct programs.
Ongoing work

• Speed up synthesis
  - model it as reinforcement problem, use heuristic to direct searching

• Automatic example generation
  - collect from network program execution traces

• Richer DSL support
Conclusion

- Propose new approach: synthesize declarative controller program using input-output examples
- Synthesis algorithm: leverage both syntactic restrictions and semantic features of declarative programs
- Proof-of-concept prototype: synthesize declarative programs with fewer than 4 relations, within 2 minutes.