SMT Solvers For Configuration And Some Configuration Projects

Lecture 5

CS 598D, Spring 2010
Princeton University
Sanjai Narain
narain@research.telcordia.com
908 337 3636
References For Lectures 2-5

• Use of Prolog for configuration file analysis, specification, validation, verification and configuration

• ConfigAssure: Solutions to configuration problems and use of Kodkod/SAT

• Testbed based on description in
The Story So Far

• Discussed how Prolog can be used:
  – To analyze ad hoc configuration language files
  – To evaluate whether requirements are true of configurations
  – As a metalevel language to convert to other forms such as Graphviz dot files

• Motivated the need for constraint solvers for firewall verification
  – Used Prolog as a metalevel language to generate constraints and exploit the power of modern constraint solvers

• Discussed the use of constraint solvers for configuration problems:
  – Synthesis
  – Diagnosis
  – Repair
  – Repair at minimum cost
  – Reconfiguration planning (later)
Today

- Show how to build in partial evaluation into the definition of eval to improve solution efficiency
- Discuss the problem of naming large numbers of configuration variables
- Motivate the use of SMT solvers for EUF to solve it
- Discuss configuration-related projects
Partial Evaluation In eval Predicate

- `eval(Req, QFF)` transforms a requirement into an equivalent QFF

- Before submitting the QFF to Kodkod, evaluate parts of it to drastically reduce size of Kodkod/SAT problem

- Even better: don’t even generate true or false parts of a QFF

- Example:
  - If there are N distinct addresses, and new host has to be added
  - Then, there are N*(N+1)/2 constraints to specify that all addresses are distinct
  - However, only N constraints are needed to ensure that host address is distinct from existing ones

- This logic can be built into eval
Variable Naming Problem: QFFs Reference Configuration Variables

?- eval(and(good, not(bad)), C)
C= and[
    gre_a_local=ra_addr
    gre_a_remote=rb_addr
    dest=rb_addr
    not [
        or[
            and[
                gre_a_local=ra_addr
                gre_a_remote=rx_addr
            ]
            dest=rx_addr
        ]
    ]
]

Configuration database with variables
- gre(ra, tunnel_0, gre_a_local, gre_a_remote).
- ipAddress(ra, eth_0, ra_addr, 0).
- ipAddress(rb, eth_0, rb_addr, 0).
- ipAddress(rx, eth_0, rx_addr, 0).
- static_route(ra, dest, mask, 400).
But, How To Name Large Numbers of Variables?

It is hard to give a distinct name to each variable, and remember it when constructing the QFF

Solution: use function applications. Construct large number of variables by combining a small number of function symbols:

- \text{gre}_a\_local
- \text{gre}_a\_remote
- \text{ra}\_addr
- \text{rb}\_addr
- \text{rx}\_addr
- \text{local}_gre(ra, tunnel\_0)
- \text{remote}_gre(ra, tunnel\_0)
- \text{ip}\_address(ra, eth\_0)
- \text{ip}\_address(rb, eth\_0)
- \text{ip}\_address(rx, eth\_0)
- next_hop(ra, \text{ip}\_address(rx, eth\_0), 32)

Now rewrite eval rules:

\begin{align*}
\text{eval}(\text{gre}_\text{tunnel}(RX, RY), \text{and} (\text{remote}_gre(RX, tunnel\_0) = \text{ip}\_address(RY, eth\_0), \\
\text{local}_gre(RX, tunnel\_0) = \text{ip}\_address(RX, eth\_0))) \).
\text{eval}(\text{route}\_\text{available}(X, Y), \text{not}(\text{next}_\text{hop}(X, \text{ip}\_\text{address}(Y, eth\_0), 32) = 0)).
\end{align*}
The Equality With Uninterpreted Functions Language

- Fortunately, SMT solvers for the Equality with Uninterpreted Function symbols can take constraints with variables as function applications, and efficiently reason with these.

- The EUF language is as follows from "Exploiting Positive Equality in a Logic of Equality with Uninterpreted Functions" by R. Bryant, S. German, M. Velev

  \[
  \begin{align*}
  \text{term} & := \text{ITE}(\text{formula}, \text{term}, \text{term}) \\
  & \mid \text{function-symbol}(\text{term}, \ldots, \text{term}) \\
  \text{formula} & := \text{true} \mid \text{false} \mid (\text{term} = \text{term}) \\
  & \mid (\text{formula} \land \text{formula}) \mid (\text{formula} \lor \text{formula}) \mid \neg \text{formula} \\
  & \mid \text{predicate-symbol}(\text{term}, \ldots, \text{term})
  \end{align*}
  \]

- ITE is the if-then-else operator.

- Good SMT solvers are Yices, CVC3 and OpenSMT. They also contain bitshift operators that can be used for network addressing.
New Constraint And Its Solution With SMT Solver For EUF

?- eval(and(good, not(bad)), C)

C =
    and[
      remote_gre(ra, tunnel0)=ip_address(rb, eth0)
      local_gre(ra, tunnel0)=ip_address(ra, eth0)
      not [
        next_hop(ra, ip_address(rb, eth0), 32)=0
      ]
    ]

  not [
    or[
      and[
        remote_gre(ra, tunnel0)=ip_address(rx, eth0)
        local_gre(ra, tunnel0)=ip_address(ra, eth0)
      ]
      not [
        next_hop(ra, ip_address(rx, eth0), 32)=0
      ]
    ]
  ]

Solver produces

ip_address(ra, eth0)=34
local_gre(ra, tunnel0)=34
ip_address(rb, eth0)=33
remote_gre(ra, tunnel0)=33
next_hop(ra, 33, 32)=35
ip_address(rx, eth0)=36
next_hop(ra, 36, 32)=0
VPN Implemented With Current Practice

Administrator Creates 12 files like this

```plaintext
interfaces {
    restore-original-config-on-shutdown: true
    interface gre1 {
        description: "Tunnel to XR1"
        disable: false
        default-system-config
    }
    interface gre2 {
        description: "Tunnel to XR3"
        disable: false
        default-system-config
    }
    interface eth2 {
        description: "Local Hosts"
        disable: false
        default-system-config
    }
}
```
New VPN Implementation Practice

Administrator creates specification like this

% Host-side router interfaces
subnet([xr1-eth2])
subnet([xr2-eth2]).
subnet([xr3-eth2]).
subnet([xr4-eth3]).

% GRE tunnels
subnet([xr1-gre1, xr4-gre2]).
subnet([xr4-gre1, xr3-gre2]).
subnet([xr3-gre1, xr2-gre2]).
subnet([xr2-gre1, xr1-gre2]).

% OSPF domain
ospf([xr1-gre1, xr4-gre2, xr4-gre1, xr3-gre2, xr3-gre1, xr2-gre2, xr2-gre1, xr1-gre2, xr1-eth2, xr2-eth2, xr3-eth2, xr4-eth2]).

% Static routing
next_hop(host1, 0.0.0.0, 32)=ip_address(xr1-eth2).
next_hop(host2, 0.0.0.0, 32)=ip_address(xr3-eth2).

Synthesis system will generate configurations and then all the files
A Web-Based Configuration Service

- Specification
  + Configuration Files

Configuration Server

- Repaired configuration files
  + Safe reconfiguration plan
  + Root-cause analysis and visualization
Towards A Requirement/Constraint Library

• Create a set of useful constraints

• Allow a user to compose these with logical operators to define complex constraints

• Classes of constraints
  – Integrity of logical structures associated with protocols
  – Connectivity
  – Security
  – Reliability
  – Performance
  – Best practices
Requirement Library For Fault-Tolerant VPN

Configuration variables are of the form
- ip_address(H, I)
- mask(H, I)
- local_gre(H, I)
- remote_gre(H, I)
- next_hop(H, Dest, Mask)
- ospf_area(H, I)
- ospf_hello_interval(H, I)
- ospf_dead_interval(H, I)

Primitive constraints are of the form
configuration variable=value

Complex constraints

\[ \text{gre}_\text{tunnel}(G_1, T_1, G_2, T_2) \] \rightarrow
\[ \text{remote}_\text{gre}(G_1, T_1) = \text{local}_\text{gre}(G_2, T_2) \] and
\[ \text{local}_\text{gre}(G_1, T_1) = \text{remote}_\text{gre}(G_2, T_2) \]

local_gre(G, T) is an address on G

\[ \text{subnet}([H_1-l_1,..,H_k-l_k]) \] \rightarrow
\[ \forall i. \text{ip}_\text{address}(H_i, T_i) \text{ bitwiseand mask}(H_i, T_i) \text{ is same} \]

\[ \text{ospf}_\text{subnet}([H1-l1,..,Hk-lk]) \] \rightarrow
\[ \forall i. \text{ospf}_\text{area}(H_i, I_i) \text{ is same, and} \]
\[ \forall i. \text{ospf}_\text{hello}_\text{interval}(H_i, I_i) \text{ is same, and} \]
\[ \forall i. \text{ospf}_\text{dead}_\text{interval}(H_i, I_i) \text{ is same} \]

All IP addresses are distinct

All IP addresses are in a given range
Configuration-Related Projects

- Prolog: Implement
  - Configuration file analyzer
  - Configuration file builder
  - Configuration visualizer
  - Configuration validator
  Evaluate against testbed

- SMT solver: Implement ConfigAssure’s
  - Synthesis algorithm
  - Minimum-cost repair algorithm
  - Reconfiguration planning algorithm
  Evaluate against testbed

Jointly Build This For Fault-Tolerant VPN
Next Class: The Use of Alloy For Configuration

- The challenges that arose and the resolution that led to ConfigAssure

- Will also be preparation for Pamela Zave’s lectures on Alloy for verification