Declarative Infrastructure Configuration Synthesis and Debugging

ConfigAssure system

Sanjai Narain, Gary Levin and Vikram Kaul, Telcordia Technologies, Inc.

Sharad Malik, Princeton University

Presented by Adam Bergstein

Oct 10, 2011

Overview

- Background
- Goals
- Implementation
- Example
- Missing clarity
- Analysis of solution
- Questions

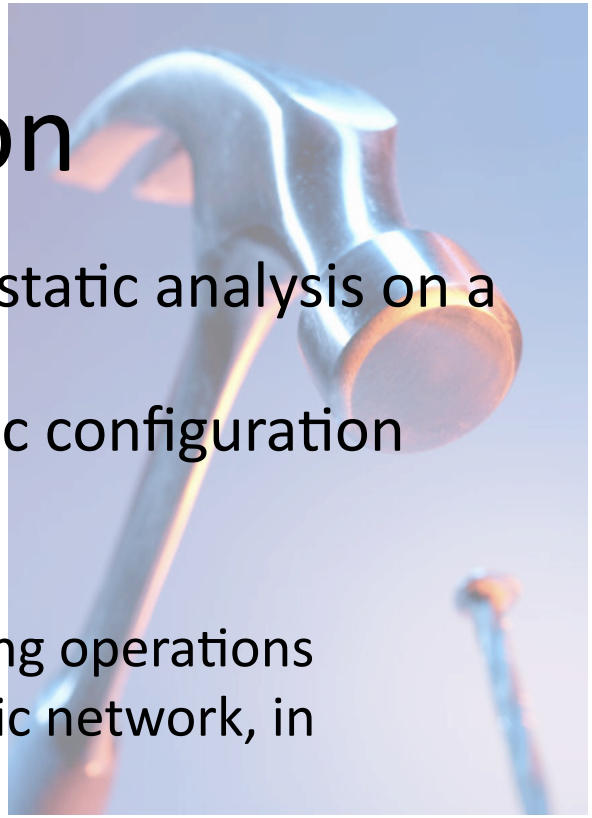
Background

- Difficult to verify configuration of large-scale networking implementations
- Well researched constraints and best practices of network implementations
- Common modeling techniques using SAT-Solvers
- Common languages to express logic, like Prolog

Goals

- Formally proving a network configuration over all known values
- Leverage known networking best practices and previous research
- Looking for an “end-to-end” solution that takes requirements and specifies appropriate configuration
- Identify problematic configuration for unsolvable solutions

Implementation



- Developed ConfigAssure as a way to do static analysis on a network
- Define requirements and prove a specific configuration meets the requirements
- Inputs:
  - General **requirements** to define networking operations
  - **Configuration database** to model a specific network, in variables and terms
  - Domain of allowable networking values (IP address ranges)
- Partial evaluator converts into a quantifier-free form of Boolean logic statement (QFF)
- QFFs sent to a solver (Kodkod/Zchaff SAT Solver)
- Solver returns possible solutions or identifies configurations that are problematic

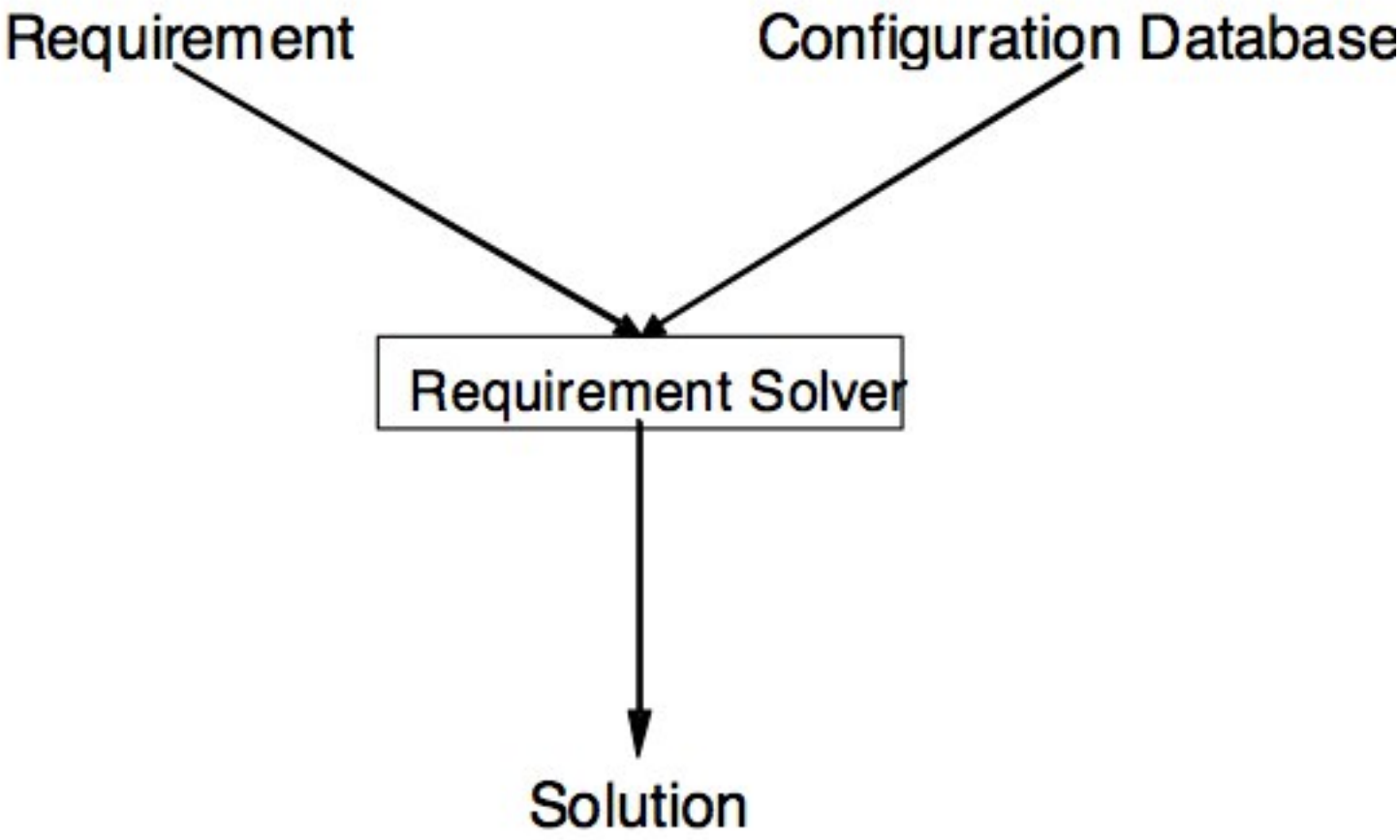
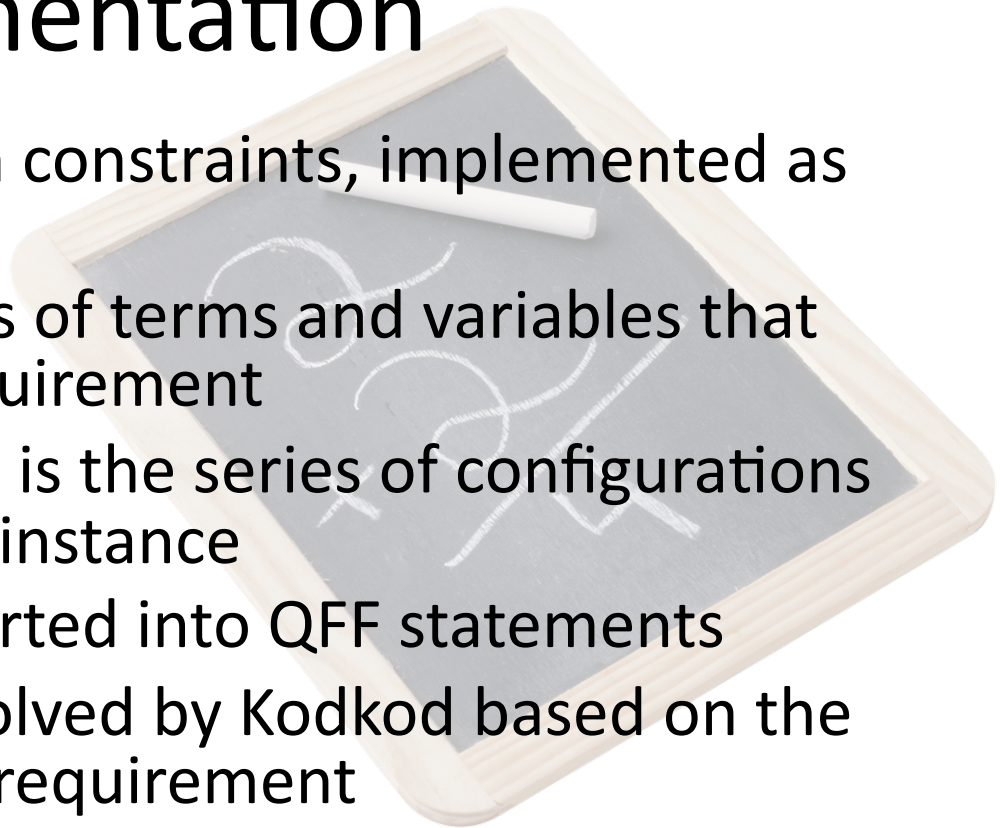
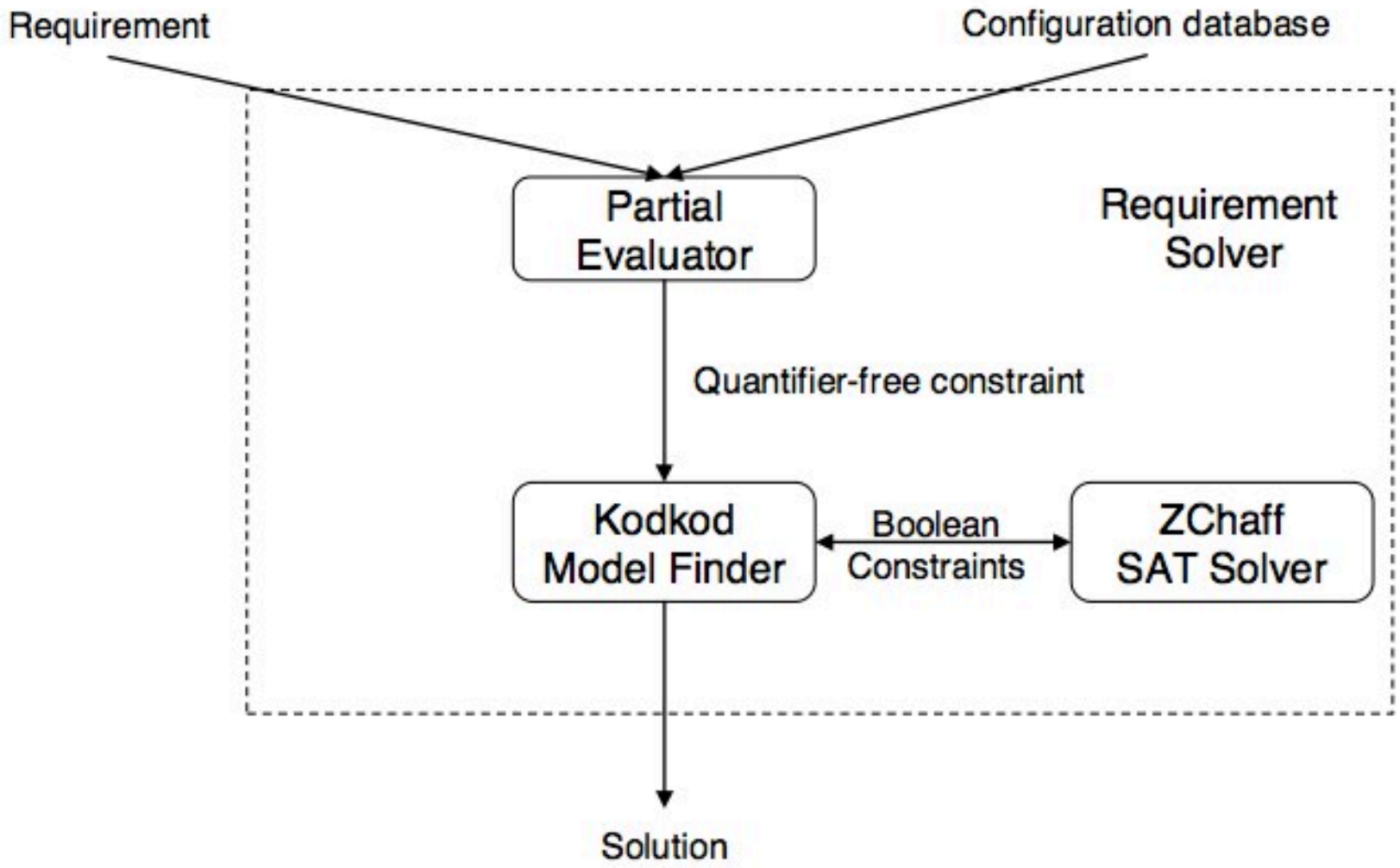


Figure 1. Requirement Solver

Implementation



- Requirements are known constraints, implemented as Prolog programs
- A configuration is a series of terms and variables that implement a defined requirement
- A configuration database is the series of configurations that define one network instance
- Configurations are converted into QFF statements
- All QFF statements are solved by Kodkod based on the Prolog equivalent of the requirement
- Kodkod returns a solution or an unsolvable QFF
  - A solution is a set of variables and accepting values in configuration
  - An unsolvable QFF identifies a specific configuration that is not solvable, which assists with mediation


Requirement
Configuration database
Partial Evaluator
Requirement Solver
Quantifier-free constraint
Kodkod Model Finder
Boolean Constraints
ZChaff SAT Solver
Solution

Implementation

- If Kodkod can identify problematic configurations, how do you resolve them?
  - Remove the specific configuration
  - Identify how the configuration needs altered (which changes the implementation)
- ConfigAssure also supports a “relaxable” set of values for variables
  - Each variable can have a set of possible values
  - If Kodkod cannot solve a configuration with specific values of variables, it will substitute other values from each variable’s relaxable set

Example

- Requirements example (Prolog)
  - All Physical IP Addresses Distinct

```
eval(all_physical_addresses_distinct, C):-,
    findall(X, H^I^M^ipAddress(H, I, X, M), S),
    eval(no_duplicates(S), C).

eval(no_duplicates([]), true).
eval(no_duplicates([U|V]), and(D, E)):-
    eval(no_duplicates(V), D),
    eval(non_member(U, V), E).

eval(non_member(U, []), true).
eval(non_member(U, [A|B]), and(C, D)):-
    check([not(U=A)], C),
    eval(non_member(U, B),D).
```

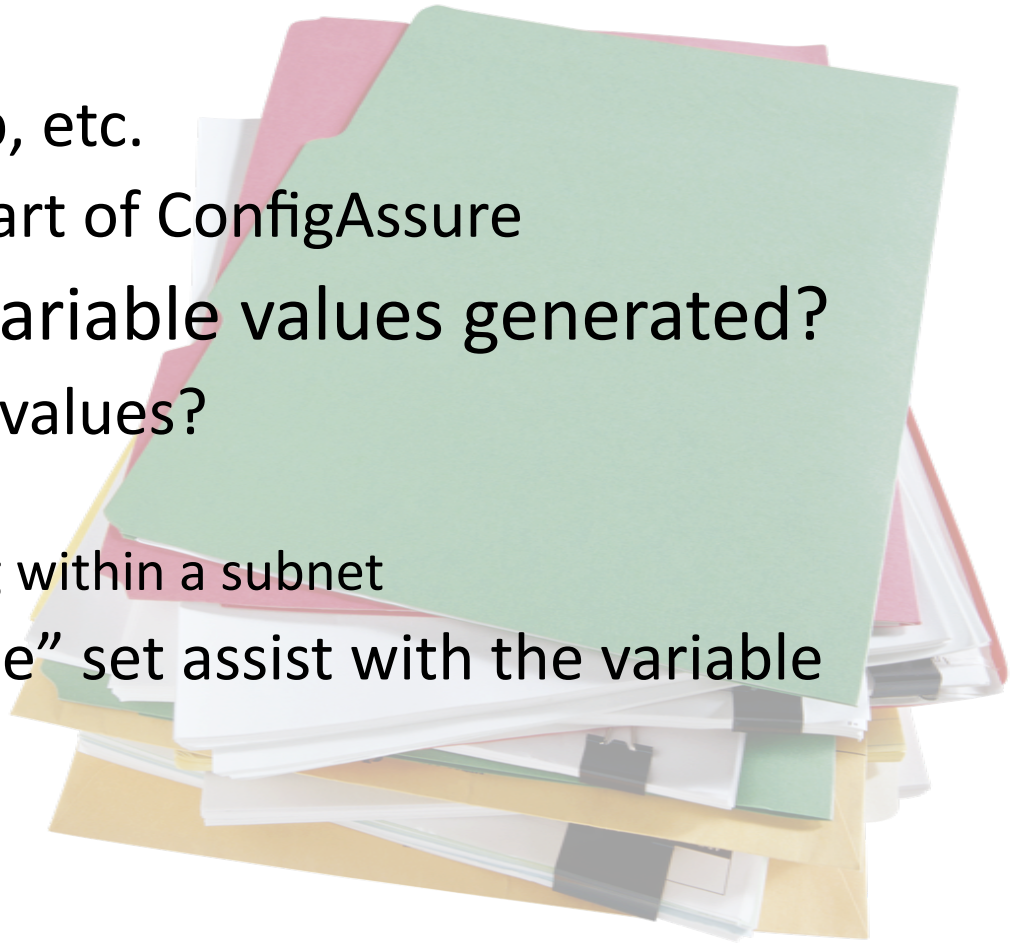
Example

- Converted configuration into QFF statements to be evaluated

```
eval(hsrp_subnet([]), true).
eval(hsrp_subnet([H-I]), true).
eval(hsrp_subnet([H1-I1, H2-I2|Rest]), and(C, CRest)):-
    hsrp(H1, I1, G1, V1),
    hsrp(H2, I2, G2, V2),
    ipAddress(H1, I1, A1, M1),
    check([contained(A1, M1, V1, 32)], C1),
    check([contained(A1, M1, V2, 32)], C2),
    check([G1=G2, V1=V2], C3),
    andEach([C1, C2, C3], C),
    eval(hsrp_subnet([H2-I2|Rest]), CRest).
```

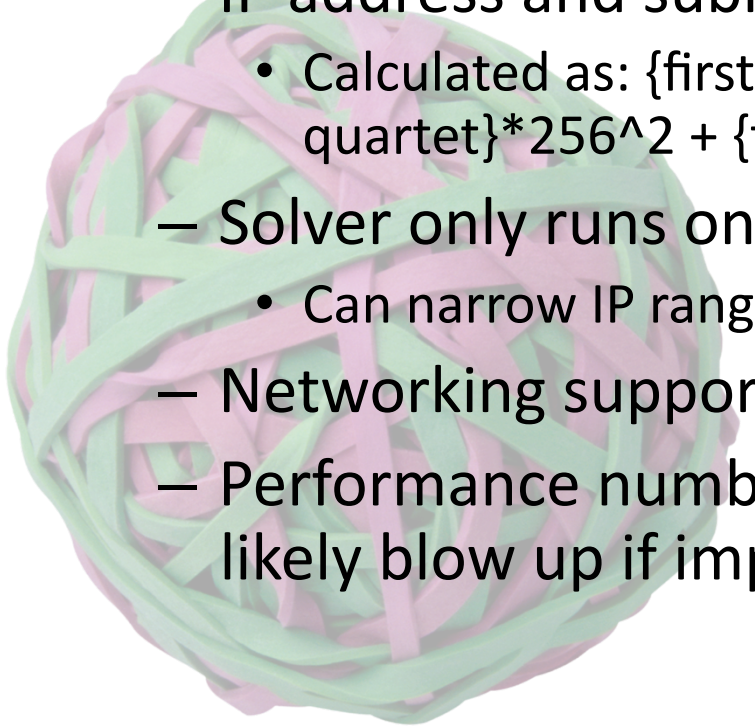
Missing Clarity

- Where is the definition for certain Prolog functions?
  - ipAddress, subnet, hsrp, etc.
  - Must be defined as a part of ConfigAssure
- How are the possible variable values generated?
  - Does it use all possible values?
    - IP-Addressing bounds
    - Bounds of IP-addressing within a subnet
  - How does the “relaxable” set assist with the variable values?

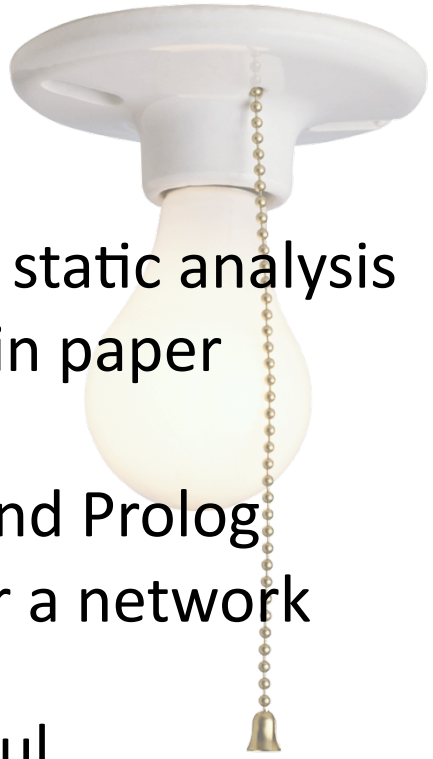


Analysis of solution

- Is this useful only for networking? Very likely
  - Specific Prolog functions just for networking and no mention of program language analysis
  - IP address and subnets lend itself well to this solution
    - Calculated as: {first quartet}*256^3 + {second quartet}*256^2 + {third quartet}*256 +{fourth quartet}
  - Solver only runs on fixed bounds of possible IPs
    - Can narrow IP range down based on subnet as well
  - Networking supports bitwise operations
  - Performance numbers looked positive, but would likely blow up if implementing the bounds of IPv6



Analysis of solution



- We have read a lot of papers on solvers and static analysis
- Very similar solution to MulVAL mentioned in paper
- What is innovative here?
  - ConfigAssure strongly relies on Kodkod and Prolog
  - Created a way to define requirements for a network and analyze a given configuration
  - "Relaxed" sets makes this tool more useful
    - Although, ConfigAssure does not define what should be in the set
    - Relies on the end user, which could limit the tool's effectiveness
    - "I will prove *this*. But if *this* is meaningless, it will do you no good"
  - Determined some QFFs could be solved more efficiently outside of Kodkod

Questions