Formal Methods In Networking

CS 598D, Spring 2010
Princeton University

Lead Instructor: Sanjai Narain, Telcordia Research
narain@research.telcordia.com, 908 337 3636

In Collaboration with
Ehab Al-Shaer, UNC Charlotte
Gary Levin, Telcordia Research
Boon Thau Loo, U. Penn
Sharad Malik, Princeton
Simon Ou, Kansas State
Andreas Voellmy, Yale
Pamela Zave, AT&T Research

Outline

• Course goals and plan
• Why study formal methods?
• Formal methods to be covered
• Their applications to networking problems
  – Theory of configuration
  – Protocol verification
  – Routing protocol design
• Projects
• Reading list
• Schedule
• Notes on logic
Course Goal And Plan

• Obtain working knowledge of formal methods that can solve real problems; stimulate new research ideas

• Instructors will
  – Discuss networking problems: theory of configuration, routing protocol design, protocol verification
  – Discuss formal methods for solving these
  – Identify open problems

• Students will
  – Select one method
  – Read 1-2 papers about it
  – Use it to solve problems, possibly around a testbed
  – Present findings to class
  – Speculate on approaches to open problems

• Teams are encouraged. Need synthesis of programming language and networking expertise

• Lectures Mondays, Fridays 9:30-10:50am, Room 302
Why Study Formal Methods?

• Formal method system = Specification language + Inference engine

• We specify “what” is required, i.e., relationships

• Inference engine figures out “how” to compute it

• Precise requirement specification, even if incomplete, is useful

• There is empirical evidence of their usefulness
Formal Methods To Be Covered

- **Boolean logic:** \( \text{ipsec\_to\_a} \supset \text{uniform\_mtu} \lor \text{permitted\_icmp\_a} \)
  - SAT solvers solve millions of constraints in millions of Boolean variables in seconds
  - BDDs an alternative to SAT but number of variables handled is much less

- **EUF:** \( \text{ipsec\_to=} \text{ip\_address}(r_0, e_0) \supset \text{uniform\_mtu}=\text{true} \lor \text{permitted\_icmp=} \text{ip\_address}(r_0, e_0) \)
  - Don’t have to name each variable
  - SMT solver faster than SAT for this language

- **Prolog:** \( \text{permitted\_icmp}(\text{ip\_address}(R, E)) \subset \text{ipsec\_to}(\text{ip\_address}(R, E)) \)
  - Quantification over individual variables
  - Only one condition in conclusion: “procedural” interpretation; write efficient specification
  - Programming language + DB
  - SLD resolution. 10s of millions of facts efficiently queried
  - Datalog: Prolog without complex terms

- **First-order logic:** \( \text{ipsec\_to}(X) \supset \text{uniform\_mtu} \lor \text{permitted\_icmp}(X) \)
  - Quantification over individual variables
  - No restriction on number of conditions on left or right side of implication
  - Alloy: First-order logic with finite domains. Compile into Boolean; use SAT

- **HOL:** Quantification over individual, function and predicate variables, e.g., induction principle

- **Promela:** Quantification over state variables. Used to specify dynamic behavior
Problem 1. Theory of Configuration

Narain, Al-Shaer, Ou
hostname DemoRouter-5
!
router ospf 50
no redistribute connected subnets
redistribute static subnets
network 10.10.6.0 0.0.0.255 area 9
network 104.104.104.0 0.0.0.255 area 9
network 105.105.105.0 0.0.0.255 area 9
!
router ospf 20
no redistribute connected subnets
redistribute static subnets
network 192.168.6.0 0.0.0.255 area 0
!
crypto isakmp policy 1
hash sha
authentication pre-share
!
interface Ethernet1
ip address 192.168.6.1 255.255.255.255

Specification of Fault-Tolerant VPN

Implementation (configuration)
Consequences of Configuration Errors

• Setting it [security] up is so complicated that it’s hardly ever done right. While we await a catastrophe, simpler setup is the most important step toward better security.

• ...human factors, is the biggest contributor—responsible for 50 to 80 percent of network device outages.

• We don’t need hackers to break the systems because they’re falling apart by themselves.

• Things break. Complex systems break in complex ways.
  – Steve Bellovin, Columbia University. Above article
Bridging Gap Between Requirement and Configuration

Why are these hard?

- How to intuitively specify connectivity, security, performance and reliability requirements?
- Synthesis, reconfiguration planning and verification require searching very large spaces
- Security and functionality interact
- Components can correctly work in isolation but not together
- Removing one error can cause another
- Distributed configuration is not well-understood
- Hard to formalize configuration language grammar documented in hundreds of pages of English
Progress Towards Theory of Configuration: ConfigAssure

- Specification: Security, connectivity, performance, reliability requirements specified as constraints
- Synthesis: Solve constraints
- Diagnosis: Analyze UNSAT-CORE
- Repair: If x=c appears in UNSAT-CORE, it is a root-cause. Remove it and re-solve
- Reconfiguration planning: Transform safety invariant into a constraint on times at which variables change from initial to final value. Solve.
- Verification: Represent firewall rule-set as a constraint on generic packet header and check equivalence
- Configuration file analysis: Represent commands as a Prolog database and query
- Future: Evaluating EUF and SMT
Progress Towards Theory of Configuration: MulVAL and ConfigChecker

- **MulVAL**
  - Specifies conditions for adversary success
  - Optimal identification of configurations to change to prevent attacks
  - Specification language: Datalog
  - Uses properties of Datalog proofs and MinCost SAT solvers

- **ConfigChecker**
  - Firewall verification with BDD-based model-checking
  - Symbolic reachability analysis: Answer questions e.g.: “Does firewall policy strengthening change the set of packets flowing from A to B?”
Possible Testbeds To Be Built For Theory of Configuration

Fault-Tolerant VPN
Narain, LISA-2005

Built at Telcordia by Tiger Qie of Princeton
LISA-2003
Theory of Configuration 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

- **BDDs**
  - Evaluate ConfigChecker on testbed configurations
  - Compare ConfigChecker security-policy verification with ConfigAssure’s

- **Datalog+MinCost SAT**
  - Implement MulVAL’s minimum-cost vulnerability mitigation algorithm
  - Evaluate against testbed

- **Software systems**
  - SWI-Prolog
  - XSB Prolog
  - SAT: Zchaff, Minisat
  - SMT: Yices, CVC3, OpenSMT
  - ConfigChecker

- **Open problems**
  - Creating a specification language usable by administrators
  - Scalability of all algorithms
  - Convergence of repair algorithm
  - Distributed configuration
Problem 2. Protocol Verification

Zave, Voellmy
Protocol Verification

- Verification of distributed systems is hard
- Approach: Check that a system satisfies a behavior invariant
  - Lightweight verification of network protocols: The case of Chord
  - Proof of an interdomain policy: A load-balancing multi-homed network
- Alloy verification project
  - Reproduce results of above paper
  - Others, TBD
- Promela/SPIN verification project: TBD
- Isabelle verification projects:
  - Isabelle/HOL tutorial: [http://isabelle.in.tum.de/dist/Isabelle/doc/tutorial.pdf](http://isabelle.in.tum.de/dist/Isabelle/doc/tutorial.pdf) Read chapter 1-3,5-7. Chapter 10 demonstrates an application of Isabelle/HOL to proving the correctness of a security protocol.
  - Also, read about Isar (the proof language for Isabelle/HOL) in this short tutorial: [http://isabelle.in.tum.de/dist/Isabelle/doc/isar-overview.pdf](http://isabelle.in.tum.de/dist/Isabelle/doc/isar-overview.pdf)
Problem 3. Routing Protocol Design

Loo
Routing Protocol Design

• Declarative routing: Express routing protocols using a database query language (Datalog)
• Implemented to date:
  - Textbook routing protocols (3-8 lines, UCB/Wisconsin)
  - Chord DHT overlay routing (47 lines, UCB/IRB)
  - Narada mesh (16 lines, UCB/Intel)
  - Distributed Gnutella/Web crawlers (Dataflow, UCB)
  - Lamport/Chandy snapshots (20 lines, Intel/Rice/MPI)
  - Paxos distributed consensus (44 lines, Harvard)
• Project
  - Implement routing protocol on declarative networking system called Rapidnet
• Open problems
  - Comparing Datalog vs other programming paradigms (Prolog, functional languages and constraint-logic programming) for designing/implementing networks
  - Integration with verification tools (e.g. Alloy, PVS)
  - Integration with existing router platforms such as XORP and IOS
  - Synthesizing network protocols and configuration from high level declarative constraints and rules
  - In addition, read http://netdb.cis.upenn.edu/research.pdf for ongoing research efforts and discuss with Prof. Loo for project ideas.
Reading List

• Available on course site
## Schedule

<table>
<thead>
<tr>
<th>Week of</th>
<th>Instructor</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/01/10</td>
<td>Narain</td>
<td>Introduction and logic programming theory</td>
</tr>
<tr>
<td>02/08/10</td>
<td>Narain</td>
<td>Introduction to Prolog, and application of Alloy to configuration theory</td>
</tr>
<tr>
<td>02/15/10</td>
<td>Narain</td>
<td>Application of SAT and SMT solvers to configuration theory</td>
</tr>
<tr>
<td>02/22/10</td>
<td>Loo</td>
<td>Datalog and its application to routing protocol design</td>
</tr>
<tr>
<td>03/01/10</td>
<td>Malik</td>
<td>SAT and SMT solvers</td>
</tr>
<tr>
<td>03/08/10</td>
<td>Ou</td>
<td>Datalog+MinCost SAT solvers for network vulnerability analysis and mitigation</td>
</tr>
<tr>
<td>3/15/10</td>
<td></td>
<td>NO CLASS</td>
</tr>
<tr>
<td>03/22/10</td>
<td>Zave</td>
<td>Promela and application to protocol verification</td>
</tr>
<tr>
<td>03/29/10</td>
<td>Zave</td>
<td>Alloy and application to protocol verification</td>
</tr>
<tr>
<td>04/05/10</td>
<td>Al-Shaer</td>
<td>Binary decision diagrams and their application to security policy verification</td>
</tr>
<tr>
<td>04/12/10</td>
<td>Voellmy/Narain</td>
<td>Isabelle and BGP verification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Review of papers</td>
</tr>
<tr>
<td>04/19/10</td>
<td>Narain</td>
<td>Review of papers</td>
</tr>
<tr>
<td>04/26/10</td>
<td>Narain</td>
<td>Student paper presentations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Student paper review reports due 4/30</td>
</tr>
<tr>
<td>05/03/10</td>
<td>Narain</td>
<td>Student paper presentations</td>
</tr>
<tr>
<td>05/10/10</td>
<td>Narain</td>
<td>Student software project presentations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Software project reports due 5/11</td>
</tr>
</tbody>
</table>
Notes on Logic
What is Logic?

• Study of what follows from what

• Study of what is a correct inference by examining only form not content

• If “all epihorins are febrids” and “all febrids are turpy” then “all epihorins are turpy”
  – We don’t need to know all the words

• Correct inference
  – I have seen a picture of Obama
  – Obama is the president of US
  – So, I have seen a picture of the president of US

• Incorrect inference
  – I have seen a picture of someone
  – Someone is the president of US
  – So, I have seen a picture of the president of US

Origins Of Modern Logic

• 1854: George Boole invents Boolean algebra

• 1879: Gottlob Frege invents Begriffsschrift or Concept Language
  – Today, it is called the Predicate Calculus
  – Extends Boolean algebra with Boolean-valued functions, individual and function variables and quantifiers over these
  – Motivated by trying to derive arithmetic from logic, i.e., prove Peano postulates from axioms of logic
  – This was called the Logicism program

• Peano postulates
  – 0 is a natural number
  – 0 is not the successor of any natural number
  – Every natural number has a successor
  – No two natural numbers have the same successor
  – Principle of induction: If F holds for 0, and for any n if F holds for n then it holds for the successor of n, then F holds for all natural numbers
Peano Postulates in Predicate Calculus

By Alonzo Church
UCLA Philosophy Department Course
~1986
1901. Russell’s Paradox

\[ \exists S. \forall T. \neg \alpha(T, T) \iff \alpha(T, S) \]

- Is the Barber’s “paradox” an instance of Russell’s?

- No. The barber does not exist. But saying that the set does not exist contradicts an assumption of set theory that for every condition, there must exist a set of objects for which the condition is true.

- Russell proposed type theory to avoid the paradox – but strict adherence to it means arguments such as Cantor’s diagonal argument cannot be carried out. So, he introduced the Axiom of Reducibility.

- How can a set belong to itself? Consider the set S of all sets in which every set has more than 5 members. S has more than 5 members, so it must belong to itself.
Logic Structure

• Logic has syntax, semantics, axioms and rules of inference

• Syntax: Defines well-formed formulas, wffs

• Semantics: About meanings of wffs
  – \( \forall x. \alpha(x) \supset \beta(x) \) is true under the interpretation \( \alpha = \text{human} \), \( \beta = \text{mortal} \). But not other way around
  – \( (\forall x. \alpha(x) \supset \beta(x) \land \alpha(p)) \supset \beta(p) \) is valid (true no matter what \( \alpha, \beta, p \) mean)

• Model checking: Evaluate if a wff is true in a given interpretation

• Model finding: Find an interpretation in which a wff is true. A.k.a. constraint solving

• Axioms: Valid wffs

• Rules of inference: Derive wffs from others
  – Modus ponens: From A and \( A \supset B \), infer B.

• Proof: Sequence of wffs starting at axioms, obtained by applications of rules of inference

• Properties of rules of inference:
  – Soundness: Starting with axioms, every derived wff is valid
  – Completeness: Every valid wff is derivable from axioms
  – Consistency: Cannot derive both \( A \) and \( \neg A \)