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

Course page: http://www.cs.princeton.edu/courses/archive/spring10/cos598D/FormalMethodsNetworkingOutline.html

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: `ipsec_to_a` $\supset$ `uniform_mtu` $\vee$ `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: `ipsec_to=ip_address(`r_0`,` e_0`)`$\supset$`uniform_mtu=true` $\vee$ `permitted_icmp=ip_address(`r_0`,` e_0`)`
  - Don't have to name each variable
  - SMT solver faster than SAT for this language

- Prolog: `permitted_icmp(ip_address(R, E))` $\subset$ `ipsec_to(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: `ipsec_to(X)` $\supset$ `uniform_mtu` $\vee$ `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

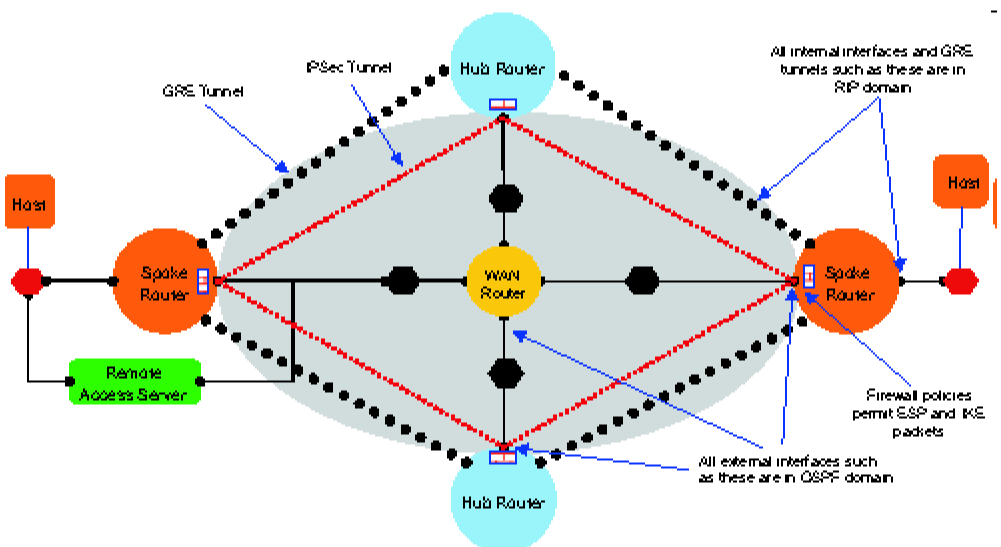
- HOL: Quantification over individual, function and predicate variables, e.g., induction principle

- Promela: Quantification over state variables. Used to specify dynamic behavior

Problem 1. Theory of Configuration

Narain, Al-Shaer, Ou

The Gap Between Requirement and Configuration (Glue)




```
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 192.168.6.1 255.255.255.0
```

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.
  - Butler Lampson, MIT. Computer Security in the Real World. *IEEE Computer*, June 2004

- …human factors, is the biggest contributor—responsible for 50 to 80 percent of network device outages.
  - What’s Behind Network Downtime? Proactive Steps to Reduce Human Error and Improve Availability of Networks, 2008. http://www.juniper.net/solutions/literature/white_papers/200249.pdf

- We don’t need hackers to break the systems because they’re falling apart by themselves.
  - Peter G. Neumann, SRI. “Who Needs Hackers”, NY Times, September 7, 2007. http://www.nytimes.com/2007/09/12/technology/techspecial/12threat.html

- Things break. Complex systems break in complex ways.
  - Steve Bellovin, Columbia University. Above article

Bridging Gap Between Requirement and Configuration

End-To-End Requirements

↑

Requirement specification

Configuration synthesis

Diagnosis

Repair

Reconfiguration planning

Verification

Distributed configuration

Configuration file analysis

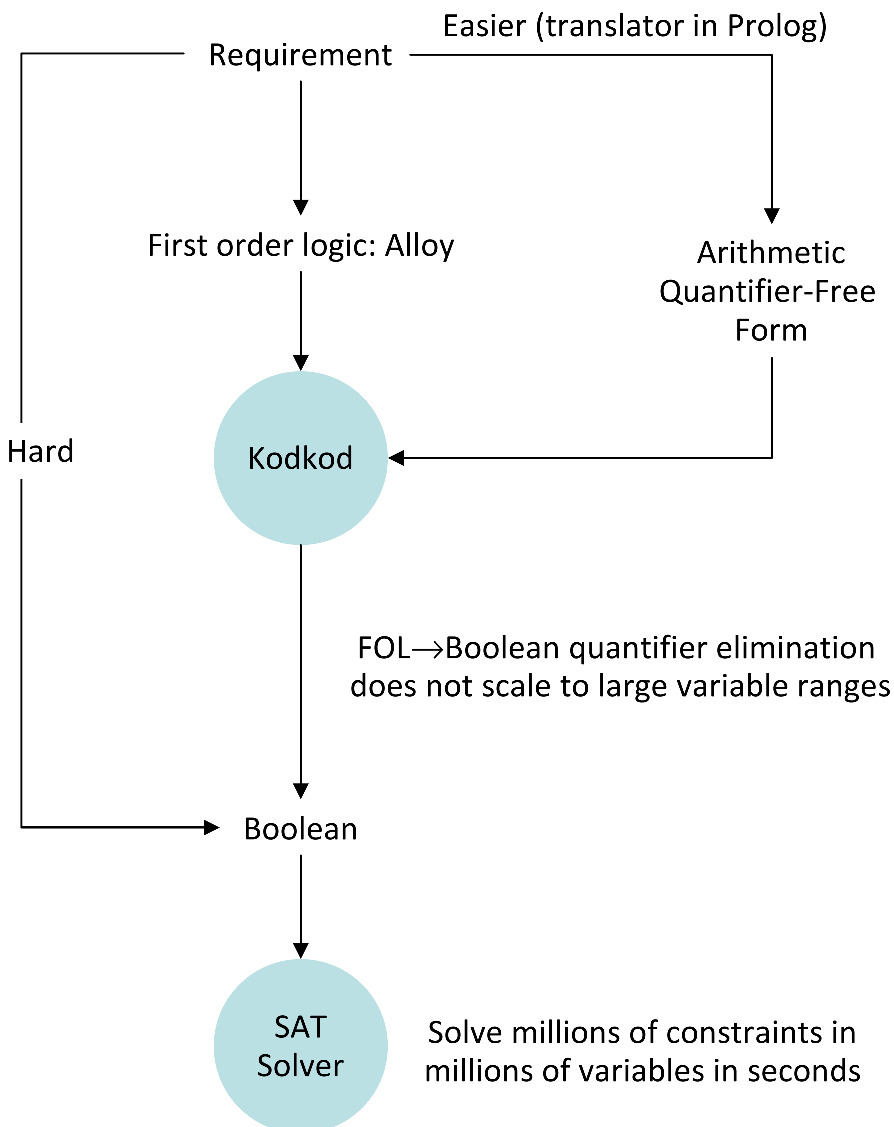
↓

Configurations (machine language)

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



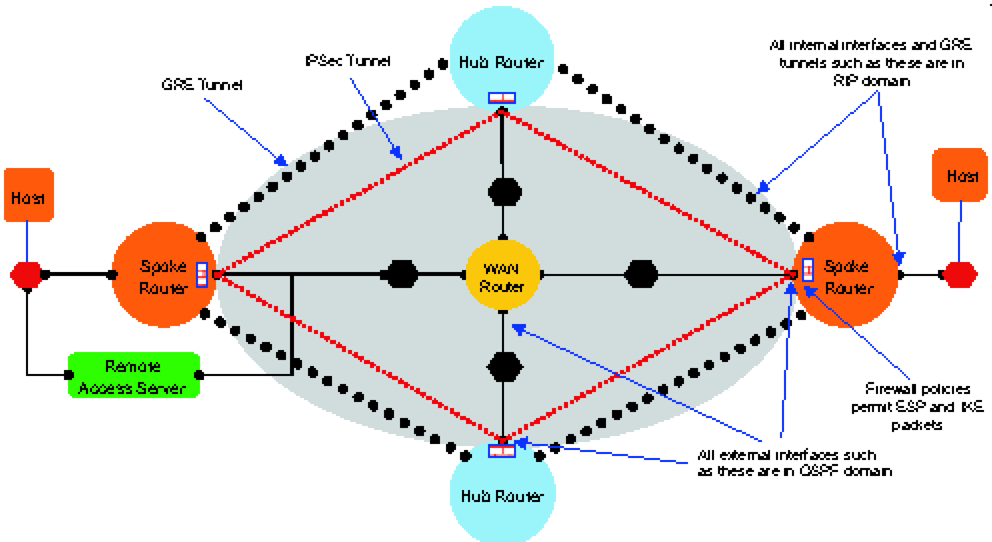
- Specification: Security, connectivity, performance, reliability requirements specified as constraints
- Synthesis: Solve constraints
- Diagnosis: Analyze UNSAT-CORE
- Repair: If x=c appears in UNSAT-CORE, it is a root-cause. Remove it and re-solve
- Reconfiguration planning: Transform safety invariant 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

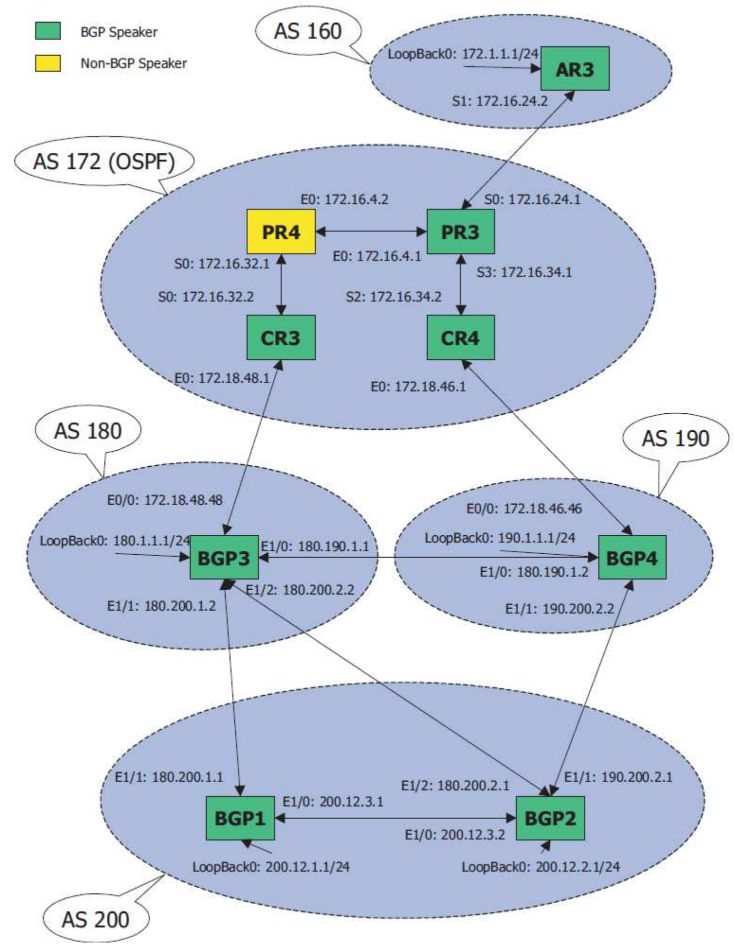
- MulVAL
  - Specifies conditions for adversary success
  - Optimal identification of configurations to change to prevent attacks
  - Specification language: Datalog
  - Uses properties of Datalog proofs and MinCost SAT solvers

- 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 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

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

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

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

*From Logic: Form and Function, J.A. Robinson, Elsevier, 1979

Origins Of Modern Logic

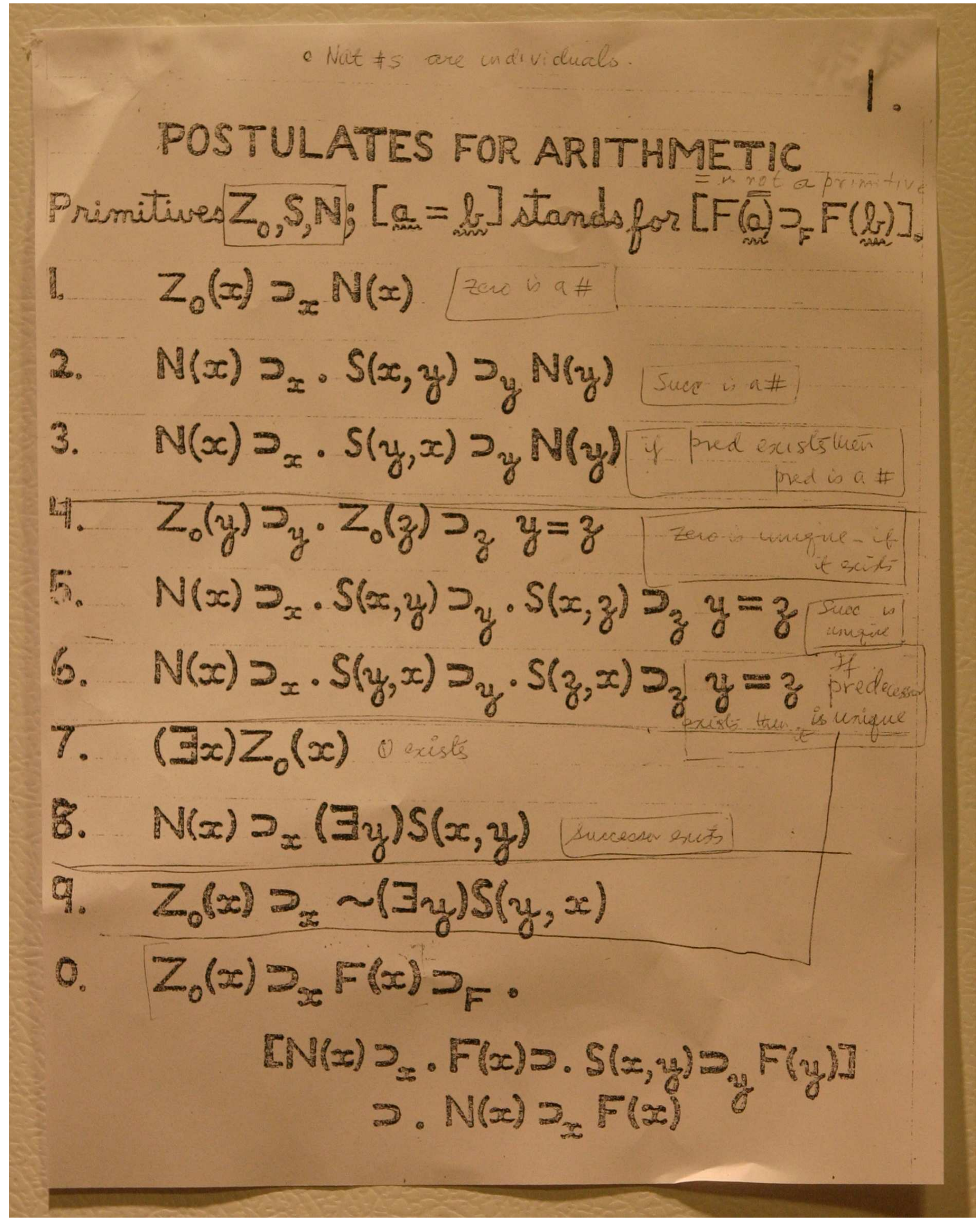
- 1854: George Boole invents Boolean algebra

- 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



Nat #s are individuals.

1.

POSTULATES FOR ARITHMETIC

Primitives $\boxed{Z_0, S, N}$; $[a = b]$ stands for $[F(a) \supset_F F(b)]$. (= is not a primitive)

1. $Z_0(x) \supset_x N(x)$ — Zero is a #

2. $N(x) \supset_x . S(x,y) \supset_y N(y)$ — Succ is a #

3. $N(x) \supset_x . S(y,x) \supset_y N(y)$ — if pred exists then pred is a #

4. $Z_0(y) \supset_y . Z_0(z) \supset_z y = z$ — Zero is unique – if it exists

5. $N(x) \supset_x . S(x,y) \supset_y . S(x,z) \supset_z y = z$ — Succ is unique

6. $N(x) \supset_x . S(y,x) \supset_y . S(z,x) \supset_z y = z$ — If predecessor exists then it is unique

7. $(\exists x) Z_0(x)$ — 0 exists

8. $N(x) \supset_x (\exists y) S(x,y)$ — Successor exists

9. $Z_0(x) \supset_x \sim(\exists y) S(y,x)$

0. $Z_0(x) \supset_x F(x) \supset_F .$

$$[N(x) \supset_x . F(x) \supset . S(x,y) \supset_y F(y)] \supset . N(x) \supset_x F(x)$$

1901. Russell’s Paradox

$$\exists \overset{\text{set}}{\underset{\text{barber}}{S}}.\forall \overset{\text{set}}{\underset{\text{person}}{T}}.\neg\, \overset{\text{belongs}}{\underset{\text{shaves}}{\alpha}}(T, T) \Leftrightarrow \alpha(T, S)$$

Russell’s paradox (top labels)

Barber’s “paradox” (bottom labels)

- 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 α = human, β=mortal. But not other way around
  - $(\forall x.\ \alpha(x) \supset \beta\ (x) \wedge \alpha(p)) \supset \beta(p)$ is valid (true no matter what α, β, 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