Wrapping up the first half:

First-order logic for security analysis, First-order logic in Coq, Constructive logic, & Inductive proofs on paper/Coq

Aquinas Hobor and Martin Henz

Network Security Analysis via Predicate Logic

Process for applying theory to practice

1. Learn about problem

2. Create a formal model of the problem

3. State the goal

4. Use some kind of tool (theorem prover, SAT solver, etc.) to solve

Process for applying theory to practice

1. Learn about problem
2. Create a formal model of the problem
3. State the goal
4. Use some kind of tool (theorem prover, SAT solver, etc.) to solve

Problem

- We have a network of many computers (100s-1,000s-10,000s)

- Each computer only allows certain kinds of connections (example: the accounting computer only allows the CEO's computer to access it; anyone in the world can access the http services of the web server)

- Each computer is running different kinds of software
  - Mail software
  - Sales software
  - Office software
  - Web hosting software
  - etc.

- Often different computers are running different versions, different patches, etc.

Problem

We wish to guarantee some security policy, such as:

– Only the CEO can access at the accounting data

How can we try to do this?

Fact: most security breaches are exploits of known vulnerabilities. Defending against truly new vulnerabilities is really hard, so let's concentrate on the common case.

Process for applying theory to practice

1. Learn about problem

2. Create a formal model of the problem

3. State the goal

4. Use some kind of tool (theorem prover, SAT solver, etc.) to solve

Why do you take CS courses?

In this class, we are teaching you a set of tools

- Propositional Logic
- SAT Solving
- Natural Deduction
- Theorem Proving
- Predicate Logic
- Modal Logic
- Temporal Logic
- Model Checkers
- Hoare Logic

Why do you take CS courses?

In this class, we are teaching you a set of tools

- Propositional Logic ✓
- SAT Solving ✓
- Natural Deduction ✓
- Theorem Proving ✓
- Predicate Logic ✓
- Modal Logic
- Temporal Logic
- Model Checkers
- Hoare Logic

Learning the tools is not easy…

Learning the tools is not easy…

… but figuring out which tools can help in which situations is <u>hard</u> (knowing the tools well is a prerequisite, which is why you take courses…)

Usually you have to study a problem for some time before you get a good idea.

Model

- We will model the network with a series of implications (essentially how an attacker would break our policy)

- We have two basic classes of rules:
  - Network topology
  - Attack vulnerability

- Example rules (network topology):
  - forall (p : computer), AccessHTTP(p, WebServerComputer)
  - …
  - RunningApache1.0(WebServerComputer)
  - …

More rules

- Attack vulnerability rule:
  - …
  - KnownAttack42: forall (p1 : computer) (p2 : computer),
    RunningApache1.0(p2) -> AccessHTTP(p1,p2) -> TakeOver(p1,p2)
  - …

Uh oh…

It appears that anyone can take over the webserver!

More rules

- …
- TakeOver(CEOComputer, AccountingComputer)
- …

The CEO likes direct access to the accounting computer so that he can see the latest sales results.

More rules

- …
- AccessReportTool(WebServerComputer, CEOComputer)
- …

The CEO likes to get regular reports and statistics from his webserver, so he uses AccessReportTool, which is this really great piece of software, to do this.

More rules

- …
- KnownAttack212: forall p1 p2,

  AccessReportTool(p1,p2) -> TakeOver(p1,p2)
- …

Unfortunately, he downloaded it from a hacker website...

How to hack the accounting computer (and why an evildoer would want to)

1. Access the webserver:
   – forall (p : computer), AccessHTTP(p, WebServerComputer)

2. Since the webserver is running an old version of Apache, take it over:
   – RunningApache1.0(WebServerComputer)
   – KnownAttack42: forall (p1 : computer) (p2 : computer),
     RunningApache1.0(p2) -> AccessHTTP(p1,p2) -> TakeOver(p1,p2)

3. Since the CEO is nice enough to have installed AccessReportTool and let it access his machine, use it to take it over:
   – AccessReportTool(WebServerComputer, CEOComputer)
   – KnownAttack212: forall p1 p2,
     AccessReportTool(p1,p2) -> TakeOver(p1,p2)

4. Since the CEO likes direct access to the accounting computer, you can now take over the accounting computer
   – TakeOver(CEOComputer, AccountingComputer)

5. Transfer money to secret bank account

6. Flee country

Process for applying theory to practice

1. Learn about problem

2. Create a formal model of the problem

3. State the goal

4. Use some kind of tool (theorem prover, SAT solver, etc.) to solve

Goal

What you want to show is that:

forall p, p <> CEOComputer ->

~TakeOver(p, AccountingComputer)

This is one way to formally state the policy; as the policy gets more complicated it gets harder to state it…

Process for applying theory to practice

1. Learn about problem

2. Create a formal model of the problem

3. State the goal

4. Use some kind of tool (theorem prover, SAT solver, etc.) to solve

5. Building a business…

- Network Topology
  - Which connections different computers accept
  - This must be determined by some kind of network analysis tool, maybe that you run each night
- Known Attacks
  - Distributed by some security firm (think antivirus software)

(unfortunately, other people have already patented this idea…)

- First-order logic in Coq
- Constructive logic
- An example inductive proof

First-order logic in Coq

- (See script)

- ~~First-order logic in Coq~~
- Constructive logic
- An example inductive proof

Constructive Logic

- Definition & History
- Impact on Computing
- Mixing Constructive and Classical Logic

Definition

- Constructive logic (a.k.a. Intuitionistic Logic) is obtained from standard logic by removing certain rules such as:
  - Law of Excluded Middle
  - Double Negation Elimination
  - Axiom of Choice

Definition

- The focus is on producing witnesses and/or justification as opposed to only establishing truth

- From these points, you can see that Constructive logic is *weaker* than Classical logic in the sense of what you can prove

- But *stronger* in what a proof means/gives you

History

- At the beginning of the 1900s there was a major effort (e.g., Frege, Hilbert) to put all of mathematics on a common base of axioms
- Unfortunately, these failed; first in a not-obviously-fatal way (e.g., Russell); and then later in a more profound way (Gödel)

History

- The result was a considerable debate: how could mathematics be done in a sound way?
- Consider two statements:
  - There is a prime number larger than 100
  - 101 is a prime number larger than 100
- The first one says something exists – but does not help you find a witness
- The second one not only tells you that something exists but *what that object is*.

An example proof

- Goal: show that there exists two irrational numbers, a & b, such that a^b is rational.

- Proof: let x be the square root of 2 (that is, $2^{(1/2)}$). We know that x is irrational. Now consider the number $y = x^x$. By law of excluded middle, we know that either
  - y is rational: in this case a = b = x
  - y is irrational: in this case observe that $y^x = 2$. Thus a = y and b = x.

So that's pretty cool…

- But what are a and b?
- Well, *probably* x^x is irrational, so *probably* a = x^x and b = x.
- But can you prove it? Proving concrete numbers are irrational is usually pretty hard.
- You'd have to prove it without LEM, too…

What a constructive proof would mean

- If we had a *constructive* proof then we would be able to examine the proof and calculate exactly what a and b are.

- One of the philosophical positions that was advocated (e.g., by Brouwer) during the debate was that all of mathematics should be constructive: if you prove something exists you need to be able to find a witness.

History

- Arend Heyting (and others) discovered that the inability to find a witness was related to the use of a small number of proof rules:
  - Law of Excluded Middle (as you just saw!)
  - …

- Constructivists believed that all of mathematics should be rebuilt without using these rules

History

- Constructivists lost the debate, and for many years modern mathematics has freely continued to use LEM; constructive logic largely was ignored for the last 100 years.

- However, the rise of computing as a field has changed that.

Constructive Logic and Computing

- Computing is deeply concerned with finding witnesses to problems (e.g., it is not enough to know that a list can be sorted: we want to produce the sorted list in question!)

- Constructive proofs, remember, can be “mined” to produce witnesses.

Constructive Logic and Computing

- Thus, if someone gives you a constructive proof of the existence of some object, an *algorithm* exists that can find that object.

- Even better, the steps of the algorithm can be determined by examining the proof!

- This means that often constructive logic is very helpful for reasoning about computation.

Mixing Constructive and Classical Logic

- So if constructive logic has a use in reasoning about computation, then why don’t we just teach you pure constructive logic and forget about LEM?
  - Often we don’t care about computation even though we are in computing: we just care about whether something is true. For example, if we can prove that a computer system is secure using LEM, that is enough!
  - In the cases when we don’t care, constructive proofs are often much harder to develop (is “y” irrational?)
  - The semantics of constructive logic are considerably more complicated (e.g., no truth tables!).

Mixing Constructive and Classical Logic

- However, since it is sometimes useful we want you to be aware of it

- We also want you to be aware that when there is an application of constructive logic, it is useful to *also* use classical logic in the parts of the proof where witnesses are not needed.

Mixing Constructive and Classical Logic

- In this case, one needs to add extra logical connectives when they differ between constructive proofs and classical proofs.
  - Use \/ for classical disjunction, and + for constructive
    - P \/ Q means, either P or Q is true
    - P + Q means, either P or Q is true and here is an algorithm for deciding
  - Use “exists” for classical existence, and “existsC” for constructive
    - exists x, P means there is an x that makes P true
    - existsC x, P means there is an x that makes P true, and if you want I can tell you exactly which x it is
  - etc.

Mixing Constructive and Classical Logic

- Some operators you don't need to do this:
  - P /\ Q has the same meaning in both constructive and classical logic
  - So does "forall", "->", etc.

- Also, obviously, all of the constructive operators imply the classical ones:
  - P + Q -> P \/ Q
  - existsC x, P -> exists x, P

Mixing Constructive and Classical Logic

- Of course, the other way around does not hold: P \/ Q ("y irrational" \/ "y rational") does not imply P + Q.

- By using these special connectives then it is possible to mix constructive and classical logic and be sure that the things one needs to be computable remain that way.

- Coq has this kind of functionality built-in, but it is beyond the scope of this module to use it.

- ~~First-order logic in Coq~~
- ~~Constructive logic~~
- An example inductive proof

Inductive definitions

- Consider the following definitions:
  - Nat = Z | S(Nat)
  - Add(a,b) =

b	when a = Z
S(Add(a',b))	when a = S(a')

- For example, Add(S(S(Z)), S(Z)) = S(Add(S(Z),S(Z))) = S(S(Add(Z,S(Z)))) = S(S(S(Z)))

Goal

- We would like to show that
  - forall a b, Add(a,b) = Add(b,a)

- How can we do this?

- Structural induction.

Lemma 1

- Rather than attack from problem as a whole, we will break it into pieces (called lemmas):
  - Lemma 1: forall a b, Add (a, S(b)) = S(Add(a,b))

- Proof: we will do **structural induction** on the structure of "a", using induction hypothesis

  P = "Add (a,S(b)) = S(Add (a,b))".

- **Very Important**
  - Say what you are doing induction on (structure of a)
  - Give your induction hypothesis **explicitly** at the beginning of the proof (P)
  - In your induction hypothesis, occurrences of the object over which you are doing induction (a) are **free variables**.
  - We are **not** using the induction hypothesis "forall a b, Add (a, S(b)) = S(Add (a, b))" – **this is what we are trying to prove**.

Break into cases

- We now get two cases. Recall that Nat = Z | S(Nat):

  - Case 1: a = Z. Then we need to prove P with a = Z:
    – Add (Z,S(b)) = S(Add(Z, b))

  - Case 2: a = S(a’). Then we **may assume** P on a’:
    – Add (a’, S(b)) = S(Add(a’, b))
  - and **must prove** P on a :
    – Add (a, S(b)) = S(Add (a, b))

- One we prove both cases, we have used structural induction to prove “forall a, P” – that is,

– forall a b, Add(a, S(b)) = S(Add(a, b))

– Note: now a is **not free** in this formula since it is bound by the forall.

Lemma 1, Case 1

- Case 1: a = Z.
  - We want to prove [a => Z] P
  - That is, Add(Z,S(b)) = S(Add(Z,b))

1. Add(Z,S(b)) = S(b)	By def. of Add
2. b = Add (Z,b)	By def. of Add
3. Add(Z, S(b)) = S(Add(Z,b))	Substitute (2) into (1)

  - So we are done with case 1.

- **Very Important**
  - Say which case you are in “a = Z”
  - Say what you want to prove “Add (Z,S(b)) = S(Add(Z,b))”

Lemma 1, Case 2

- Case 2: a = S(a’).
  - We may assume [a => a’] P
  - That is, Add(a’, S(b)) = S(Add(a’, b))
  - We want to prove [a => S(a’)] P
  - That is, Add(S(a’), S(b)) = S(Add(S(a’), b))

  1. Add(S(a’), S(b)) = S(**Add(a’, S(b))**) By def. of Add
  2. Add(S(a’), S(b)) = S(**S(Add(a’, b))**) Substitute IH into (2)
  3. Add(S(a’), S(b)) = S(Add(S(a’), b)) By def. of Add

  - So we are done with case 2.

- **Very Important**
  - Say precisely what the induction hypothesis is.

Lemma 1, conclusion

- Are we done?

- NO. We must finish the proof by saying something like,

- “Structural induction lets us conclude,
  - forall a b, Add(a, S(b)) = S(Add(a, b))”

Lemma 2

- We continue with another lemma (subproof):
  - Lemma 2: forall a, Add (a,Z) = Add (Z,a)

- Proof: we will do **structural induction** on the structure of “a”, using induction hypothesis P = “Add (a,Z) = Add (Z,a)”.
  - **Very Important**
    - Say what you are doing induction on (structure of a)
    - Give you induction hypothesis **explicitly** at the beginning of the proof (P)
    - In your induction hypothesis, occurrences of the object over which you are doing induction are **free variables**.

Break into cases

- We now get two cases. Recall that Nat = Z | S(Nat):
  - Case 1: a = Z. Then we need to prove P with a = Z: “Add (Z,Z) = Add(Z,Z)”.
  - Case 2: a = S(a’). Then we **may assume** P on a’: “Add (a’, Z) = Add(Z, a’)”, and **must prove** P on a : “Add (a,Z) = Add (Z,a)”
- One we prove both cases, we have used structural induction to prove “forall a, P” – that is, “forall a, Add(a,Z) = Add(Z,a)”

– Note: now a is **not free** in this formula since it is bound by the forall.

Lemma 2, Case 1

- Case 1: a = Z.
  - We want to prove [a => Z] P
  - That is, Add(Z,Z) = Add(Z,Z)
  - This is directly form reflexivity of equality
  - So we are done

- **Very Important**
  - Say which case you are in (a = Z)
  - Say what you want to prove (Add (Z,Z) = Add(Z,Z))

Lemma 2, Case 2

- Case 2: a = S(a’)
  - We may assume [a => a’] P
  - That is, Add(a’, Z) = Add (Z, a’)
  - We want to prove [a => S(a’)] P
  - That is, Add (S(a’), Z) = Add(Z, S(a’))

Lemma 2, Case 2

- Add(S(a’), Z) = S(Add(a’,Z)) by def. of Add
- S(Add(a’,Z)) = S(Add(Z,a’)) by [a => a’] P (IH)
- S(Add(Z,a’)) = Add (Z, S(a’)) by Lemma 1

- And we are done with case 2; now we can conclude:
  - Thus by structural induction we prove “forall a, Add(a,Z) = Add(Z,a)”.

Theorem, Proof

- Goal: forall a b, Add(a,b) = Add(b,a)
- Proof: we will use structural induction on “a” using the induction hypothesis P:
  - forall b, Add (a, b) = Add(b,a)
- Important: note that this is a **stronger** hypothesis then the weaker P’:
  - Add (a,b) = Add(b,a)
- In the second hypothesis, “b” is a **constant** (like 7); in the first “b” is a bound universal variable
  - P -> P’, but P’ does not imply P.

Theorem, Case 1

- Case 1: a = Z; we must prove
  - forall b, Add (Z, b) = Add (b,Z)

- But this is just Lemma 2, so we are done.

Theorem, Case 2

- Case 2: a = S(a’) and we can assume the induction hypothesis
  - forall b, Add (a’, b) = Add (b, a’)

- We want to prove
  - forall b, Add (S(a’), b) = Add (b, S(a’))

  - Add (S(a’), b) = S(Add(a’, b)) Definition of Add
  - S(Add(a’,b)) = S(Add(b, a’)) IH
  - S(Add(b, a’)) = Add(b, S(a’)) Lemma 1

Conclusion

- Structural Induction thus lets us conclude that
  - forall a b, Add (a, b) = Add (b, a)

- Now let’s do the same proof in Coq.
  - (See script)