CS 591: Formal Methods in Security and Privacy

Introduction, Class Structure, Logistics, and Objectives

Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu

Security and Privacy

infosecurity STRATEGY | INSIGHT | TECHNOLOGY

Latest

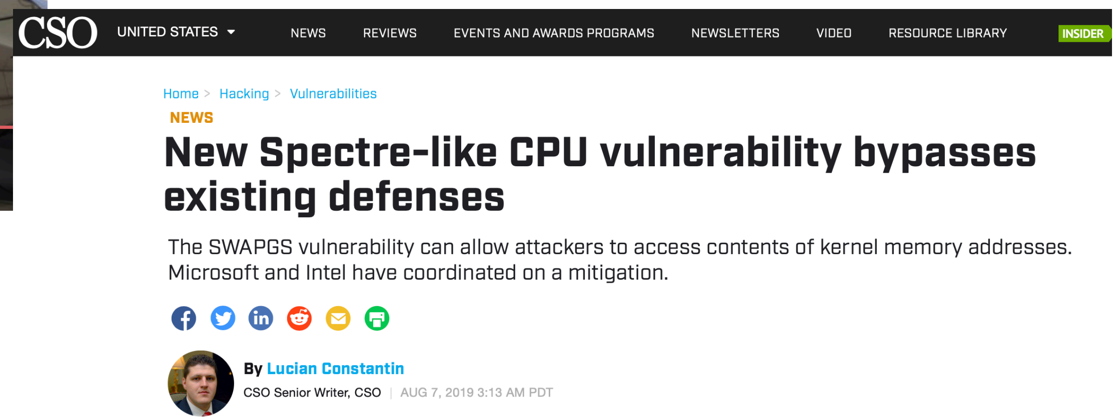
Security by Sector: Healthcare Orgs Continue to Suffer Security Headaches

News Topics Features Webinars White Papers Podcasts Events & Conferences Directory

INFOSECURITY MAGAZINE HOME » NEWS » SINGAPORE AIRLINES SOFTWARE BUG RESULTS IN BREACH

7 JAN 2019 NEWS

Singapore Airlines Software Bug Results in Breach



CSO UNITED STATES NEWS REVIEWS EVENTS AND AWARDS PROGRAMS NEWSLETTERS VIDEO RESOURCE LIBRARY INSIDER

Home > Hacking > Vulnerabilities

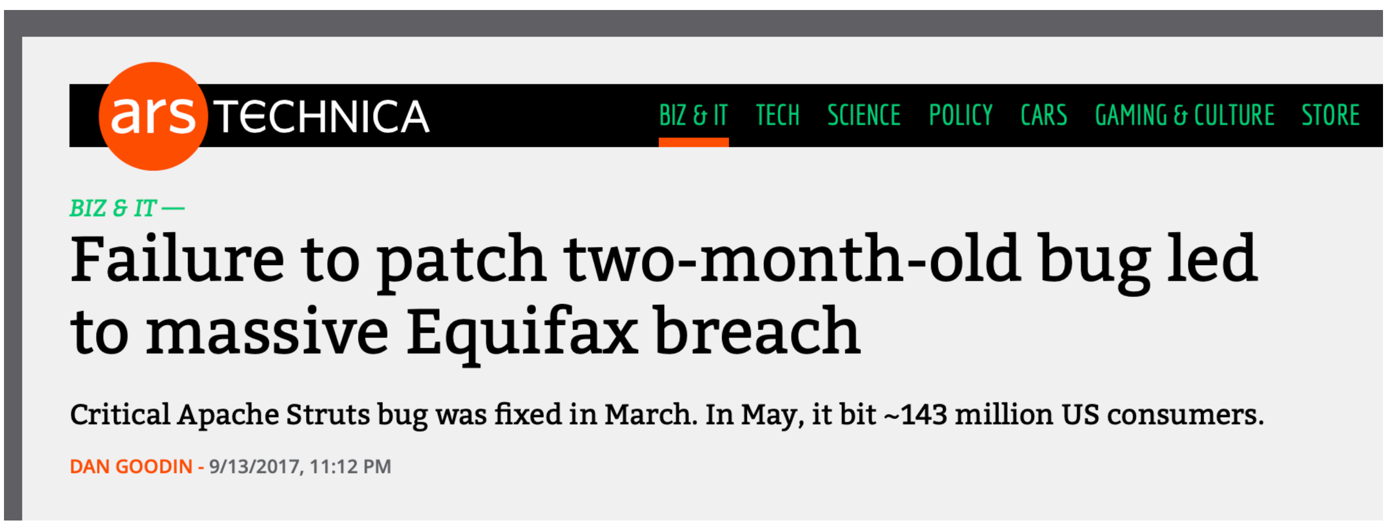
NEWS

New Spectre-like CPU vulnerability bypasses existing defenses

The SWAPGS vulnerability can allow attackers to access contents of kernel memory addresses. Microsoft and Intel have coordinated on a mitigation.

By Lucian Constantin

CSO Senior Writer, CSO | AUG 7, 2019 3:13 AM PDT



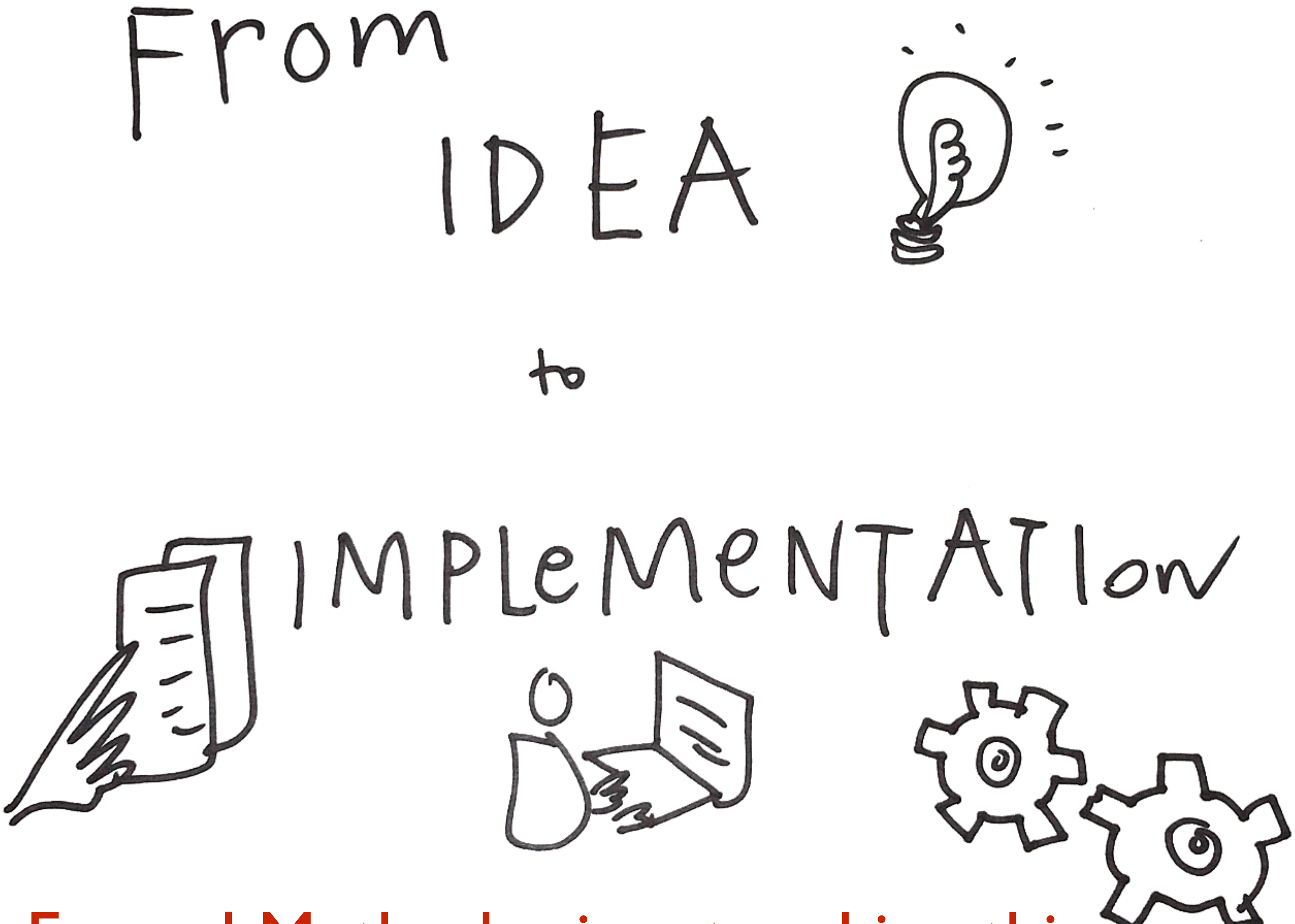
ars TECHNICA BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STORE

BIZ & IT —

Failure to patch two-month-old bug led to massive Equifax breach

Critical Apache Struts bug was fixed in March. In May, it bit ~143 million US consumers.

DAN GOODIN - 9/13/2017, 11:12 PM



Formal Methods aim at making this process mathematically rigorous.

Goal of formal methods: building applications that are correct.

Why correctness matters?

What does “correct” mean?

A program is correct if it respects the specification:

- What is computed (functional aspects)
- How it is computed (non-functional aspects).

Why correctness matters?

An example:
DARPA HACMS (High Assurance Cyber Military Systems)

Infosec Institute

Is correctness easy to guarantee?

Is this code correct?

```
Function Add(x: int, y: int) : int
{
  r = 0;
  n = y;
  while n != 0
  {
    r = r + 1;
    n = n - 1;
  }
  return r
}
```

Is this code correct?

```
Function Add(x: int, y: int) : int
{
  r = 0;
  n = y;
  while n != 0
  {
    r = r + 1;
    n = n - 1;
  }
  return r
}
```

Something seems wrong.

Is this code correct?

```
Function Add(x: int, y: int) : int
{
  r = 0;
  n = y;
  while n != 0
  {
    r = r + 1;
    n = n - 1;
  }
  return r
}
```

Something seems wrong.

Is It the name or the program?

Adding the specification

```
Function Add(x: int, y: int) : int
{
  r = 0;
  n = y;
  while n != 0
  {
    r = r + 1;
    n = n - 1;
  }
  return r
}
Postcondition: r == x + y
```

Adding the specification

```
Precondition: x ≥ 0 and y ≥ 0
Function Add(x: int, y: int) : int
{
  r = 0;
  n = y;
  while n != 0
  {
    r = r + 1;
    n = n - 1;
  }
  return r
}
Postcondition: r == x + y
```

Does the program comply with the specification?

```
Precondition: x ≥ 0 and y ≥ 0
Function Add(x: int, y: int) : int
{
  r = 0;
  n = y;
  while n != 0
  {
    r = r + 1;
    n = n - 1;
  }
  return r
}
Postcondition: r == x + y
```

Does the program comply with the specification?

```
Precondition: x ≥ 0 and y ≥ 0
Function Add(x: int, y: int) : int
{
  r = 0;
  n = y;
  while n != 0
  {
    r = r + 1;
    n = n - 1;
  }
  return r
}
Postcondition: r == x + y
```

Fail to meet the specification

In the rest of the class

- We will focus on methods to guarantee correctness of programs with respect to security and privacy properties.
- We will address in particular three properties of security and privacy.
- We will focus on program-based methods, where the security and privacy guarantees depend on the program design.
- This is in contrast to (whole) system security and privacy, which require to consider many other aspects.

Why formal methods?

- Formal proofs can be checked by a machine, and designed with the support of a machine.
- Formal methods provide strong guarantees of software correctness.
- Formal systems usually allow us to create digital certificates that can be used to verify the trust of the software component.
- In applications that require strong security and privacy guarantees, we cannot make mistakes.

What is the recipe we will follow?

- We will look at specific notions of security and privacy
- We will formalize these notions, through some formal model useful to determine if a program satisfy this notion or not.
- We will then look at a formal logic which allow us to mechanize this reasoning.
- We will use the formal logic on several examples.

What will we look at?

- Basic methods to reason formally about whether programs meet their specification: Hoare Logic
- Adapt this method to reason about security in terms of information flow: Relational Hoare Logic
- Adapt this method to reason about security in terms of formal cryptography: Probabilistic Relational Hoare Logic
- Adapt this method to reason about data privacy in terms of differential privacy: Approximate Probabilistic Relational Hoare Logic

Marco Gaboardi

- Ph.D. 2007 from University of Torino (Italy) and INPL (France)
- Research in:
  Programming Languages Theory
  Data Privacy
- First year at BU
- Three and a half year at the University at Buffalo
- Two and a half year at the University of Dundee, Scotland, UK.
- Office Hours: 2-4pm Tuesday

Alley Stoughton

- Ph.D. 1987 from University of Edinburgh (Scotland)
- Research in:

  Programming Languages Theory

  Formal Methods for Security and Cryptography
- Research Professor at BU
- Previously: Sussex (England), Kansas State, MIT Lincoln Laboratory, IMDEA (Spain)
- Office Hours: 2–4pm Thursday or by appointment

Syllabus for the course

Location: MCS B29
Time: Tu-Tr 11:00 - 12:15
Office Hours:
Marco Gaboardi (MCS 116) Tues 2-4pm or by appointment;
Alley Stoughton (MCS 122) Thurs 2-4pm or by appointment

Course load:

- participation in class and on Piazza,
- completing the assignments,
- working on a project and presenting the results.

Course Webpages: Piazza

- Course will use Piazza and the class webpage
- Piazza: piazza.com/bu/spring2020/cs591g1/home
  - Q/A
  - Homework and solutions
- Class webpage: http://cs-people.bu.edu/gaboardi/teaching/S20-CS591.html
  - Class slides
  - Materials

Notes

An introduction to relational program verification

Suggested Citation: Gilles Barthe (2020), "An introduction to relational program verification", : Vol. xx, No. xx, pp 1–1. DOI: 10.1561/XXXXXXXXX.

Gilles Barthe

now
the essence of knowledge
Boston — Delft

http://software.imdea.org/~gbarthe/__introrelver.pdf

Schedule

http://cs-people.bu.edu/gaboardi/teaching/S20-CS591.html

Grade Break Down

- 40% Assignments (tentatively 4-6)
- 50% Project
- 10% Participation in class and on piazza

Proving Assignments

- 1 assignment to get you familiar with Hoare Logic
  (Out in Week 2 and due in Week 3 - tentatively)
- 1 assignment on Non-Interference and Relational Hoare Logic
  (Due in Week 5 - tentatively).
- 1 assignment on Differential Privacy and Approximate Probabilistic Relational Hoare Logic
  (Due in Week 7 - tentatively).
- 1 assignment on Formal Cryptography and Probabilistic Relational Hoare Logic
  (Due tentatively at the end of November)

Proving Assignments

- Discussions with other students about the assignment are permitted and encouraged.
- However, the solutions to the programming assignments must be your own. No pair or group submission is allowed.

Academic integrity policy

The Department of Computer Science takes the academic integrity of all students seriously. In order to uphold the integrity of our programs and the university, we rely on students to behave appropriately and take responsibility for their mistakes. Please review the following pages to better understand the expectations of the department, college, and university, as well as the process of any academic misconduct matters.

- https://www.bu.edu/academics/policies/academic-conduct-code/
- https://www.bu.edu/cs/undergraduate/undergraduate-life/academic-integrity/

EasyCrypt

- Proof assistant for mechanizing proofs in the program logics we'll study in this course
- You'll be using EasyCrypt in the course's assignments, and perhaps in the course project
- To date, EasyCrypt has mostly been used to mechanize proofs from theoretical cryptography
- But we'll also consider other applications, e.g., to noninterference and differential privacy

EasyCrypt

- EasyCrypt proofs are about simple assignment-oriented programming language with while loops and random assignments
- EasyCrypt has logics supporting:
  - Reasoning about single programs
  - Reasoning about pairs of programs—relational logic
  - Reasoning in a typed higher-order logic—for general mathematics, and connecting results from other logics

EasyCrypt

- EasyCrypt proofs:
  - structured as sequences of lemmas
  - lemmas are proved using tactics, which reduce goals to zero or more subgoals
  - developed interactively using a special mode of Emacs text editor, with EasyCrypt running as subprocess
  - maybe rechecked either interactively, or in batch mode

EasyCrypt Lab Sessions

- We would like to schedule a weekly EasyCrypt lab session for showing more details about EasyCrypt than there is time for in lectures
- That way, lectures can focus on theory and applications, with lab sessions focusing on realization in EasyCrypt
- According the “course matrix”, these are possible times:
  - Mondays from 11:10am-12noon
  - Tuesdays from 6:30-7:20pm
  - Wednesdays from 11:10am-12noon
  - Thursdays from 6:30-7:20pm
  - Fridays from 3:30-4:20pm
- Discussion?

Final Projects

Projects can take different forms depending on the interest of each student but all the projects must have a research component.

Some examples:

- using EasyCrypt or one of its extensions to prove security and privacy of a new complex algorithm,
- design or implementation of a new programming language, system, or tool for security and privacy,
- development and implementation of heuristics and optimizations to speed up the verification tasks for security and privacy,
- investigation of new applications of relational logics.

Final Projects

We will provide some specific ideas for possible projects but other ideas may be accepted if well motivated and discussed with us.

You may work on your project alone or with others. Groups can be composed by at most two students. Each group is invited to meet with us regularly (3-4 times during the term) to check on the advancements and directions of the project.

The deadline for choosing a project is February 25.

Questions?

Formal Logic

- We will need to reason extensively about formal specifications/requirements.
- A convenient way to express these requirements is by means of logical formulas.

```
X=Y+1 and Z=X+S

not X=Y  or  Y < X

For all n, X=n  implies  Y = n+1
```

Classical Logic Formulas: Basic Predicates

We can assume that we have some basic predicates that give us some basic formulas

```
X=Y

X<Y

True

False
```

We can think about this as some primitive operations whose validity we are able to establish in an atomic way.

Classical Logic Formulas: Connectives

If we have a formula `P` we can create the formula `not P`

If we have a formula `P` and a formula `Q` we can create the formula

`P and Q`

If we have a formula `P` and a formula `Q` we can create the formula

`P or Q`

If we have a formula `P` and a formula `Q` we can create the formula

`P implies Q`

Classical Logic Formulas: Quantifiers

If we have a formula `P(x)` which depends on the variable `x` we can create the universally quantified formula `for all x, P(x)`

If we have a formula `P(x)` which depends on the variable `x` we can create the existentially quantified formula `exists x, P(x)`

Classical Logic: Proving formulas

When we are working with logical formulas we are in general interested in proving them true.

We could define a formal system for building proofs. This can be achieved for example by a formal system managing proof rules of the form:

$$\frac{\text{assumption}_1 \text{ is true} \qquad \text{assumption}_k \text{ is true}}{\text{conclusion is true}}$$

For this lecture we will just rely an intuitive explanation of how we can prove a formula, starting from a set of assumptions, and how this can be translated in rules.

Classical Logic - Proving formulas: Conjunction

To prove that a conjunction formula `P and Q` is true, we need to show that both `P` and `Q` are true.

This corresponds to the following rule:

$$\frac{\texttt{P true} \qquad\qquad \texttt{Q true}}{\texttt{P and Q true}}$$

Classical Logic - Proving formulas: Conjunction

To prove that a disjunction formula `P` or `Q` is true, it is sufficient to show that one between `P` and `Q` is true.

This corresponds to the following two rules:

$$\frac{\texttt{P true}}{\texttt{P or Q true}} \qquad \frac{\texttt{Q true}}{\texttt{P or Q true}}$$

Classical Logic - Proving formulas: Negation

To prove that a negation formula `not P` is true, we can show that under the assumption that `P` is true, we can conclude `False`.

This corresponds to the following rule:

$$\dfrac{\begin{array}{c}\texttt{P true} \\ \vdots \\ \texttt{False true}\end{array}}{\texttt{not P true}}$$

Classical Logic - Proving formulas: Negation

To prove that a negation formula `not P` is true, we can show that under the assumption that `P` is true, we can conclude `False`.

This corresponds to the following rule:

$$\dfrac{\begin{array}{c}\texttt{P true}\\ \vdots \\ \texttt{False true}\end{array}}{\texttt{not P true}}$$

$\boxed{\text{Not very precise}}$

Notation

$$\frac{\begin{array}{c}\mathtt{P\ true}\\ \vdots \\ \mathtt{False\ true}\end{array}}{\mathtt{not\ P\ true}}$$

This way of writing can be confusing, sometime is better to just write:

$$\frac{\begin{array}{c}\mathtt{P}\\ \vdots \\ \mathtt{False}\end{array}}{\mathtt{not\ P}}$$

Classical Logic - Proving formulas: Implication

To prove that an implication formula `P implies Q` is true, we can show that under the assumption that `P` is true, we can conclude `Q`.

This corresponds to the following rule:

$$\frac{\begin{array}{c}\texttt{P} \\ \vdots \\ \texttt{Q}\end{array}}{\texttt{P implies Q}}$$

Classical Logic - Proving formulas: Implication

To prove that an implication formula `P implies Q` is true, we can show that under the assumption that `P` is true, we can conclude `Q`.

This corresponds to the following rule:

$$\dfrac{\begin{array}{c}\texttt{P}\\ \vdots\\ \texttt{Q}\end{array}}{\texttt{P implies Q}}$$

Not very precise

Classical Logic - some examples

Classical Logic - some examples

```
P implies (P or Q)
```

Classical Logic - some examples

$$\frac{}{\texttt{P implies (P or Q)}}$$

Classical Logic - some examples

$$\frac{\texttt{P or Q}}{\texttt{P implies (P or Q)}}$$

Classical Logic - some examples

$$\frac{\overline{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

`P implies (Q implies (P and Q))`

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\quad}{\texttt{P implies (Q implies (P and Q))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\texttt{Q implies (P and Q)}}{\texttt{P implies (Q implies (P and Q))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{}{\texttt{Q implies (P and Q)}}}{\texttt{P implies (Q implies (P and Q))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\texttt{P and Q}}{\texttt{Q implies (P and Q)}}}{\texttt{P implies (Q implies (P and Q))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\dfrac{}{\texttt{P and Q}}}{\texttt{Q implies (P and Q)}}}{\texttt{P implies (Q implies (P and Q))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\dfrac{\texttt{P}}{\texttt{P and Q}}}{\texttt{Q implies (P and Q)}}}{\texttt{P implies (Q implies (P and Q))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\dfrac{\texttt{P} \qquad\qquad \texttt{Q}}{\texttt{P and Q}}}{\texttt{Q implies (P and Q)}}}{\texttt{P implies (Q implies (P and Q))}}$$

Classical Logic - Proving formulas: Universal Quantification

To prove that a universally quantified formula `for all x,P(x)` is true, we can show that under the assumption that `x` is an arbitrary element of the universe, we can conclude `P(x)`.

This corresponds to the following rule:

$$\frac{\texttt{P(x)}}{\texttt{for all x,P(x)}}$$

Classical Logic - Proving formulas: Universal Quantification

To prove that a universally quantified formula `for all x,P(x)` is true, we can show that under the assumption that `x` is an arbitrary element of the universe, we can conclude `P(x)`.

This corresponds to the following rule:

$$\frac{\texttt{P(x)}}{\texttt{for all x,P(x)}}$$

Not very precise

Classical Logic - Proving formulas: Existential Quantification

To prove that an existentially quantified formula `exists x,P(x)` is true, we can show that `P(t)` is true for an element t of the universe.

This corresponds to the following rule:

$$\frac{\texttt{P(t)}}{\texttt{exists x,P(x)}}$$

Classical Logic - Proving formulas: Existential Quantification

To prove that an existentially quantified formula `exists x,P(x)` is true, we can show that `P(t)` is true for an element t of the universe.

This corresponds to the following rule:

$$\frac{\texttt{P(t)}}{\texttt{exists x,P(x)}}$$

Not very precise

Classical Logic: other rules for proving formulas

We have few other principles that we can use to prove formulas

If we have contradictory assumptions, we can prove anything

This corresponds to the following rule:

$$\frac{\texttt{P and (not P)}}{\texttt{Q}}$$

Classical Logic: other rules for proving formulas

If we have the assumption `P implies Q` and the assumption `P`, then we can prove `Q`.

This corresponds to the following rule:

$$\frac{\texttt{P implies Q} \qquad \texttt{P}}{\texttt{Q}}$$

Classical Logic: other rules for proving formulas

If we have the assumption `P or Q` and we want to prove `R`, then we can prove that `R` follows from `P` and that `R` follows from `Q`.

This corresponds to the following rule:

$$\dfrac{P \texttt{ or } Q \qquad \begin{array}{c} P \\ \vdots \\ R \end{array} \qquad \begin{array}{c} Q \\ \vdots \\ R \end{array}}{R}$$

Classical Logic: other rules for proving formulas

In classical logic we have the Law of the Excluded Middle saying that a formula P is always either true or false.

This can be formulated as the following rule

$$\frac{}{\texttt{P or (not P)}}$$

Classical Logic: negation

The negation of composed formulas can be often rewritten

```
not(P and Q) ≅ (not P) or (not Q)
not(P or Q) ≅ (not P) and (not Q)
not(P implies Q) ≅ P and (not Q)
not(not P) ≅ P
```

```
not exists x, P(x) ≅ for all x, not P(x)
not for all x, P(x) ≅ exists x, not P(x)
```

Classical Logic - some examples

Classical Logic - some examples

```
P implies (P or Q)
```

Classical Logic - some examples

```
P implies (P or Q)
```

Classical Logic - some examples

$$\frac{\texttt{P or Q}}{\texttt{P implies (P or Q)}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

`not ((P or not P) implies (P and not P))`

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\texttt{False}}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{}{\texttt{False}}}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\texttt{P and not P}}{\texttt{False}}}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\dfrac{}{\texttt{P and not P}}}{\texttt{False}}}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\dfrac{\texttt{(P or not P) implies (P and not P)}}{\texttt{P and not P}}}{\texttt{False}}}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic - some examples

$$\cfrac{\cfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\cfrac{\cfrac{\cfrac{\texttt{(P or not P) implies (P and not P)} \qquad \texttt{P or not P}}{\texttt{P and not P}}}{\texttt{False}}}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic - some examples

$$\dfrac{\dfrac{\texttt{P}}{\texttt{P or Q}}}{\texttt{P implies (P or Q)}}$$

$$\dfrac{\dfrac{\dfrac{\texttt{(P or not P) implies (P and not P)} \qquad \dfrac{}{\texttt{P or not P}}}{\texttt{P and not P}}}{\texttt{False}}}{\texttt{not ((P or not P) implies (P and not P))}}$$

Classical Logic: proving complex fact

Classical logic can be combined with specific theories to prove more complex facts. As an example, if we combine classical logic with a theory of integers we can prove true formulas such as:

```
for all x,y,z, x+y=z implies x=z-y
```

Questions?