Verification For Security: Introduction

Klaus v. Gleissenthall

VRIJE UNIVERSITEIT AMSTERDAM

What is Algorithmic Verification?

Algorithms, Techniques and Tools to ensure that:

- Program’s don’t have bugs.
- Later: are secure in some sense.

But what does that *mean*? Stay tuned :)

Course Website

- Course website at: Verification-for-Security.github.io
- Contains lecture slides, calendar, additional details

Goals

- **Deep dive** into:
  - The state of the art of program verification
  - Programming Languages (PL) techniques for security
  - Learn our way of thinking about these problems
  - Prepare for research in the area
  - Learn how to use this in practice
  - Make you a better programmer
  - Have fun!

This course

- Starts from zero, but moves fast
- Want to explain everything, end-to-end
- We start with lectures that give you the necessary background
- Very hands-on, wrt programming.
- We'll have three assignments that teach you how to apply your knowledge in practice.

Why should you care?

- Many success stories

  AWS, Airbus, Microsoft, NASA,

  Galois, Facebook, Google

- Verified crypto in google Chrome (Fiatcrypto)

- Verified Operating System (sel4, hi-star, verve), compilers (compcert), processors (ARM/Intel), distributed systems (AWS), networks (AWS), enclaves, etc.

Turing Awards in PL / Verification

Dijkstra

Floyd

Hoare

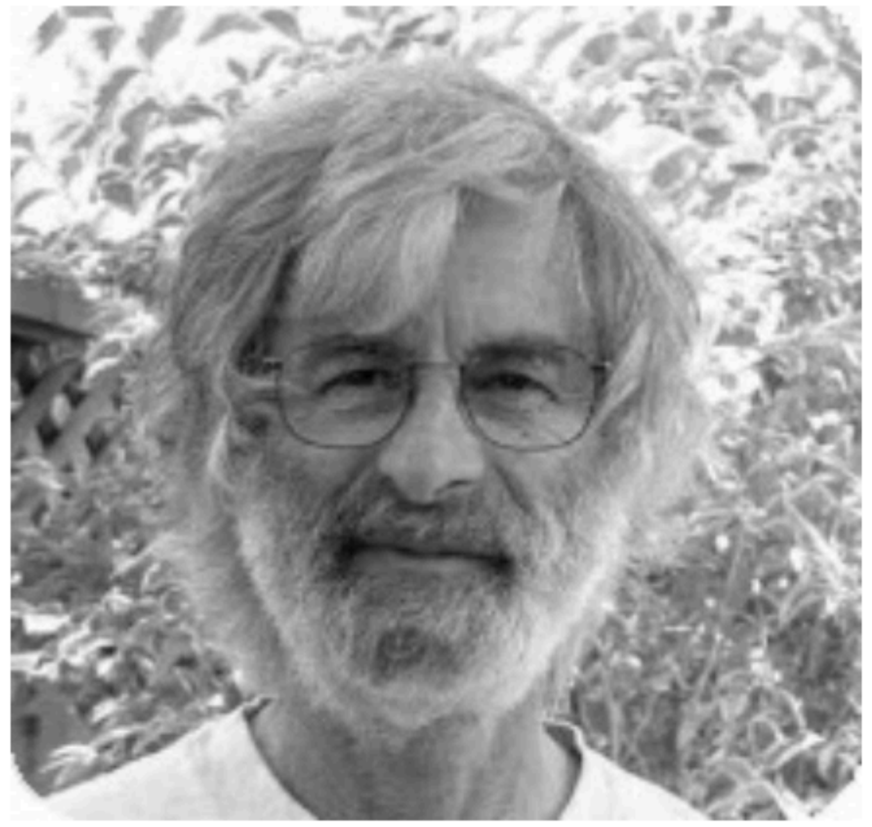
Milner

Pnueli

Clarke

Emerson

Sifakis



Lamport

Plan

Two parts:

- Algorithms for showing that programs are correct
  - Doesn’t divide by zero
  - Doesn’t access arrays out of bounds
  - Mergesort output is sorted
- Later: Showing that programs are secure
  - Doesn’t leak secrets to an attacker

Plan

Logic:
- Propositional
- First order
- Theories

→

Programs to Logic:
- Hoare Logic
- VCGen

→

Automation
- Horn clauses
- Predicate Abstraction

→

Security
- Information Flow Control
- Side-Channels

Flipped Classroom

- We will use the flipped classroom setting
- You're <u>expected to watch videos</u> before lecture slot
- We'll use the slot for Q&A

Evaluation

- 3 Practical Assignments (50%)
- Written Exam (50%)
- Must pass assignment and project with grade >=4
- Must pass the exam with grade >=5.5

Practical Assignments

- You'll build your own SAT and SMT solver
- You'll build your own program verifier, similar to Dafny!

  https://dafny.org/

- Groups of 2
- We give you skeleton code, need to fill in main functions
- Pass all the tests, full points

Practical Assignments

- Late submissions: one malus point per day
- One late-day, per group no questions asked
- Exceptions for hardship
  - Need to be approved by study advisor

Practical Assignments

- Assignment 1: SAT & SMT Solver
- Assignment 2: Implement a Program Verifier
- Assignment 3: IFC Type System

Practical Assignments

- Prerequisite: Functional Programming

  (for example, Equational Programmig at VU)
- You'll need to know Haskell (quite well!)
- If you don't already know it, this course will be very tough!
- http://learnyouahaskell.com/chapters
- Intro assignment due on Sep 8!
- We wrote a tutorial about monads, take a look here
- **Download GHC and start coding**

Exam

- Check understanding of the material
- Shows individual contribution, as the rest of the course is pair projects.
- We'll release a mock exam and will discuss the solution in class.

Plagiarism

- Please don’t do it! It’s bad for everyone
- Don’t copy & don’t share solutions
- We will check automatically & manually
- There are no existing solutions, the assignments are new

Practical Session

- We have a practical session on Friday
- This will be primarily used to help you with the programming assignments

Getting the most out of this course

- Take what I say as a starting point and read more!
- Look at links for further reading & other similar courses
- Think about how & why things work
- Would they still work if we changed x or y?
- When you don't know an answer to a question look at the definition!
- Take a piece of paper and write your own notes
- Answer quizzes in lectures on by yourself!
- Take the mock exam

Getting the most out of this course

- Use the course resources!
- We want you to succeed!
- Come to the Q&A and ask questions, we're here to help!
- In our experience: students who actively participate tend to do a lot better.
- Practicals are here to help you with the assignments!
- If you don't know how to get started, come and ask!

Changes from Last year

- No more project
- No Horn solving project
- Lecture videos are online
- Better instructions for assignments

- Let us know what works and what doesn't!
- Your feedback matters and will help improve this course!

Let’s start!

Logic:
- Propositional
- First order
- Theories

Programs to Logic:
- Hoare Logic
- VCGen

Automation
- Horn clauses
- Predicate Abstraction

Security
- Information Flow Control
- Side-Channels

Logic

- Why are we talking about logic?
- Logic is the *Calculus of Computation*!
- May seem abstract now (*why are we doing this?*) …
- … but much/all of program analysis can be boiled down to logic.
- Think of logic as a language for **asking questions** about programs!

Decision Procedures

- Efficient Algorithms to **answer questions** about programs
- Easily enough to teach one or many courses
- We will only scratch the surface to give a feel
- Good resource: Calculus of Computation

Propositional Logic

- You probably already know this, but good to recap
- A logic is a **language**, described by:
  - Syntax of its formulas (variables, connectives, …)
  - Semantics of formulas (when is a formula *satisfied*, or *valid*)

Propositional Logic: Syntax

Atom:	truth symbols	$\top$ (true), $\bot$ (false)
	propositional variables	$p, q, r, p_1, p_2, \ldots$
Literal:	an atom α or its negation $\neg \alpha$	

Formula: A literal, or application of a **logical connective** to formula $F, F_1, F_2, \ldots$

$\neg F$	"not"	negation
$F_1 \wedge F_2$	"and"	conjunction
$F_1 \vee F_2$	"or"	discjunction

Propositional Logic: Syntax

Formula: A literal, or application of a **logical connective** to formula F, F_1, F_2, …

$\neg F$	"not"	negation
$F_1 \wedge F_2$	"and"	conjunction
$F_1 \vee F_2$	"or"	F_1 discjunction

We can define additional connectives in terms of these basic ones:

$F_1 \rightarrow F_2$	"implies"	$\neg F_1 \vee F_2$
$F_1 \leftrightarrow F_2$	"if and only if"	$(F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$

Propositional Logic: Syntax

Atom:

$\top$ (true), $\bot$ (false)

$p, q, r, p_1, p_2, \ldots$

Literal: an atom α or its negation $\neg \alpha$

Formula: literal or

$\neg F \quad F_1 \wedge F_2 \quad F_1 \vee F_2$

Quiz:

a) is p an atom?

b) Is p a literal?

c) is p a formula?

d) what about $\neg p$?

e) what about $\neg\neg p$?

Propositional Logic: Semantics

- A logic is a **language**, described by:
  - Syntax of its formulas (predicates, connectives, …)
  - Semantics of formulas (when is a formula *satisfied*, or *valid*)

Propositional Logic: Semantics

An interpretation **I** maps propositional variables to truth values

$$I : \{p \rightarrow \top,\ q \rightarrow \bot,\ r \rightarrow \bot, \cdots\}$$

We can **evaluate** a formula F **under I** by substituting each p for **I**(p)

We write $I \vDash F$, if F evaluates to $\top$ under I, we call I a **satisfying interpretation or model**

We write $I \nvDash F$, if F evaluates to $\bot$ under I, I is a **falsifying interpretation or counter-model**

Propositional Logic: Semantics

We can **evaluate** a formula F **under I** by substituting each p for **I**(p)

We can make this more precise through an **inductive definition**

Base case

$I \models \top \qquad I \not\models \bot$

$I \models p, \quad \text{if} \quad I(p) = \top$

$I \not\models p, \quad \text{if} \quad I(p) = \bot$

Inductive Case

$I \models \neg F, \quad \text{if} \quad I \not\models F$

$I \models F_1 \wedge F_2, \quad \text{if} \quad I \models F_1 \text{ and } I \models F_2$

$I \models F_1 \vee F_2, \quad \text{if} \quad I \models F_1 \text{ or } I \models F_2$

What about $\rightarrow$, $\leftrightarrow$? Already defined.

Propositional Logic: Semantics

Example

Base case

$I \models \top \qquad I \not\models \bot$

$I \models p, \quad \text{if} \quad I(p) = \top$

$I \not\models p, \quad \text{if} \quad I(p) = \bot$

Inductive Case

$I \models \neg F, \quad \text{if} \quad I \not\models F$

$I \models F_1 \wedge F_2, \quad \text{if} \quad I \models F_1 \text{ and } I \models F_2$

$I \models F_1 \vee F_2, \quad \text{if} \quad I \models F_1 \text{ or } I \models F_2$

Quiz: Consider formula $F \triangleq (\neg p \vee q)$ and interpretation $I \triangleq \{p \rightarrow \top, q \rightarrow \bot\}$

Which of the following is true?

a) $I \models F$ b) $I \not\models F$

What about formula $(p \wedge q) \rightarrow (\neg p \vee q)$?

Propositional Logic: Semantics

Satisfiability and Validity

F is <u>satisfiable</u>, iff *there exists* an interpretation I such that $I \models F$

F is <u>valid</u>, iff *for all* interpretations I, $I \models F$

F is <u>contingent</u>, iff it is satisfiable but not valid

Duality between satisfiability and validity:

F is <u>valid</u>, iff $\neg F$ is <u>unsatisfiable</u>

Thus: If we can decide whether a formula is satisfiable, we can also check if it is valid

Propositional Logic: Semantics

Example

Quiz:

Are these formulas sat, unsat, or valid?

1. $(p \land q) \rightarrow \neg p$
2. $(p \land q) \rightarrow (p \lor \neg q)$
3. $(p \rightarrow (q \rightarrow r)) \land \neg((p \land q) \rightarrow r)$

Deciding Satisfiability

- Logic: **asking questions** about programs
- But how to answer?
- We want to decide whether a formula is satisfiable

Quiz:
- Why only satisfiable?

- Let's look at two simple methods first:
  - Enumerating interpretations (aka truth tables)
  - Semantic Arguments (deductive proofs)

Deciding Satisfiability: Truth Table

- Let's look at the following formula $F \triangleq (p \land q) \rightarrow (p \lor \neg q)$
- We can enumerate its interpretations via a truth table

Deciding Satisfiability: Truth Table

- Let’s look at another example $F \triangleq (p \lor q) \rightarrow (p \land q)$

For n propositional variables, there are 2^n interpretations: impractical

Deciding Satisfiability: Semantic Argument

Try to prove validity of formula F through a proof by contradiction

- Assume F is not valid, i.e., there exists I s.t., $I \not\models F$
- Apply proof rules to derive $\perp$ along every branch (what's that?)
- If we succeed, F is valid

Deciding Satisfiability: Semantic Argument

Proof Rules

<u>neg</u>

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

<u>conj</u>

$$\frac{I \models F_1 \land F_2}{I \models F_1 \quad I \models F_2} \qquad \frac{I \not\models F_1 \land F_2}{I \not\models F_1 \ \underline{\text{or}}\ I \not\models F_2}$$

<u>disj</u>

$$\frac{I \models F_1 \lor F_2}{I \models F_1 \ \underline{\text{or}}\ I \models F_2} \qquad \frac{I \not\models F_1 \lor F_2}{I \not\models F_1 \quad I \not\models F_2}$$

<u>imp</u>

$$\frac{I \models F_1 \rightarrow F_2}{I \models \neg F_1 \ \underline{\text{or}}\ I \models F_2} \qquad \frac{I \not\models F_1 \rightarrow F_2}{I \models F_1 \quad I \not\models F_2}$$

<u>contr</u>

$$\frac{I \models F \quad I \not\models F}{I \models \bot}$$

$$\underline{\text{neg}} \quad \frac{I \vDash \neg F}{I \nvDash F} \qquad \frac{I \nvDash \neg F}{I \vDash F}$$

$$\underline{\text{conj}} \quad \frac{I \vDash F_1 \land F_2}{I \vDash F_1 \quad I \vDash F_2} \qquad \frac{I \nvDash F_1 \land F_2}{I \nvDash F_1 \ \underline{\text{or}}\ I \nvDash F_2}$$

$$\underline{\text{disj}} \quad \frac{I \vDash F_1 \lor F_2}{I \vDash F_1 \ \underline{\text{or}}\ I \vDash F_2} \qquad \frac{I \nvDash F_1 \lor F_2}{I \nvDash F_1 \quad I \nvDash F_2}$$

$$\underline{\text{imp}} \quad \frac{I \vDash F_1 \rightarrow F_2}{I \vDash \neg F_1 \ \underline{\text{or}}\ I \vDash F_2} \qquad \frac{I \nvDash F_1 \rightarrow F_2}{I \vDash F_1 \quad I \nvDash F_2}$$

$$\underline{\text{contr}} \quad \frac{I \vDash F \quad I \nvDash F}{I \vDash \bot}$$

Let's prove that $F \triangleq (p \land q) \rightarrow (p \lor \neg q)$ is valid

<u>neg</u>

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

<u>conj</u>

$$\frac{I \models F_1 \wedge F_2}{I \models F_1 \quad I \models F_2} \qquad \frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \ \underline{\text{or}}\ I \not\models F_2}$$

<u>disj</u>

$$\frac{I \models F_1 \vee F_2}{I \models F_1 \ \underline{\text{or}}\ I \models F_2} \qquad \frac{I \not\models F_1 \vee F_2}{I \not\models F_1 \quad I \not\models F_2}$$

<u>imp</u>

$$\frac{I \models F_1 \rightarrow F_2}{I \models \neg F_1 \ \underline{\text{or}}\ I \models F_2} \qquad \frac{I \not\models F_1 \rightarrow F_2}{I \models F_1 \quad I \not\models F_2}$$

<u>contr</u>

$$\frac{I \models F \quad I \not\models F}{I \models \bot}$$

Let's do $F \triangleq ((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$

Semantic arguments also offers no easy, algorithmic way to prove validity!

Logic:

- Propositional
- First order
- Theories

→

Programs to Logic:

- Hoare Logic
- VCGen

→

Automation

- Horn clauses
- Predicate Abstraction

→

Security

- Information Flow Control
- Side-Channels

Where are we?

- Propositional Logic Syntax & Semantics
- Naive ways of checking satisfiability
- Next lecture: checking sat efficiently
- Bigger picture:
  - Logic is the language of computation
  - Propositional logic is too restricted
  - Next, first order logic (too expressive) & undecidable
  - Goldilocks solution SMT: propositional logic + theories