

Verification For Security: Introduction

Klaus v. Gleissenthall



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
- We'll have two assignments that teach you how to do verification hands-on.
- Then, you get to apply what you learned in a project.

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),
compilers (compcert), processors
(ARM/Intel), distributed systems (AWS),
networks (AWS), enclaves, etc.

SYNOPSYS®



TOYOTA

AIRBUS



Microsoft

galois |



Astrée

Turing Awards in PL / Verification

Dijkstra



Floyd



Hoare



Milner



Pnueli



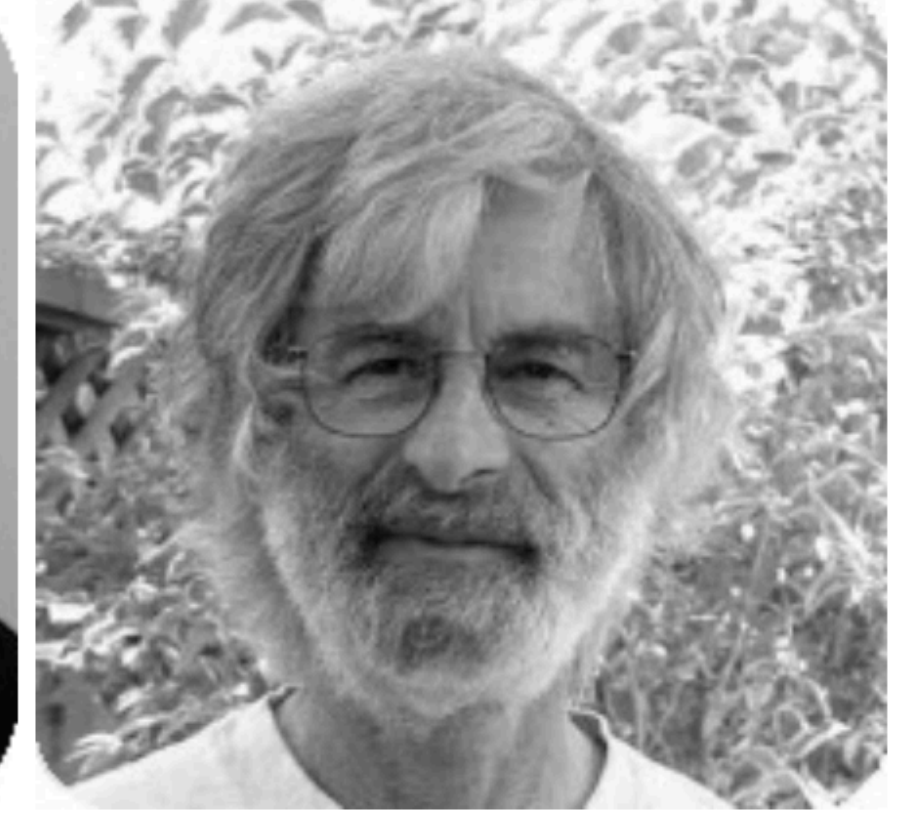
Clarke



Emerson



Sifakis



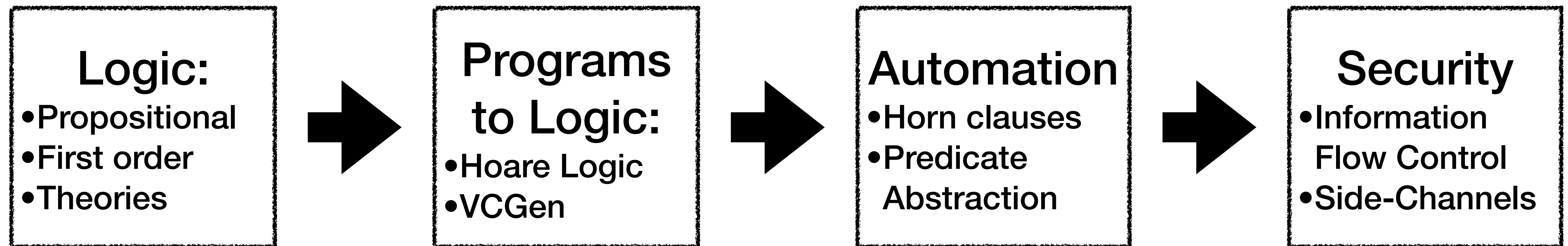
Lamport

Plan

Two parts:

- Algorithms for showing that programs are correct
- Later: Showing that programs are secure
- Not an easy course, but you will learn a lot

Plan



Evaluation

- 2 Practical Assignments (25%)
- Final Project (25%)
- Written Exam (50%)
- Must pass assignment and project with grade ≥ 4
- Must pass the exam with grade ≥ 5.5

Practical Assignments

- Implement an SMT solver & a program verifier in Haskell!
- Verify programs in a language called Nano
- 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, 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: Pick your own Project

Practical Assignments

- You'll need to know Haskell (quite well!)
- **Start reading the tutorial now!**
- <http://learnyouahaskell.com/chapters>
- Intro assignment due on Sep 10!
- Look at our monad tutorial here
- **Download GHC and start coding**

Project

- This is where you get to show us what you learned.
- The project needs to be written in Haskell
- Groups of 2, as before.
- First, 1 page proposal.
- Then, you'll implement the project and write up a report of up to three pages.
- List of project ideas on the website

Exam

- Check understanding of the material
- Shows individual contribution, as the rest of the course is pair projects.
- We'll release online Canvas quizzes that you can use to prepare for the exam.

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

Office Hours

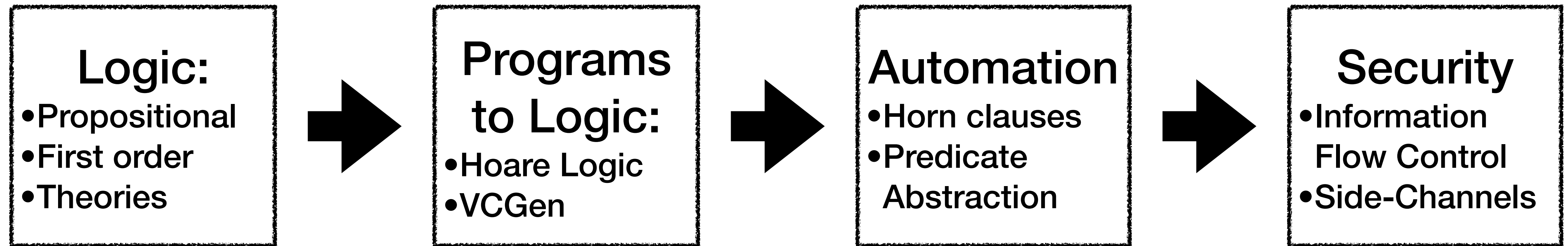
- We have a practical session on Friday
- This will be used as office hours for asking questions about lecture, programming assignments and the project

Getting the most out of this course

- Take what I say as a starting point and read more!
- I will add links to further reading & other similar courses
- Google: "topic site:edu slides" to find more material
- Think about how & why things work
- Would they still work if we changed x, y ?
- Ask questions!
- Use the canvas message board liberally (but don't share code)!

Questions? ... let's start!

Let's start!



Logic

- Why are we talking about logic?
- Logic is the *Calculus of Computation*!
- May seem abstract now (*what's up with all these symbols?*) ...
- ... 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), \perp (false)

propositional variables p, q, r, p_1, p_2, \dots

Literal: an atom a or its negation $\neg a$

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

$\neg F$ “not” negation

$F_1 \wedge F_2$ “and” conjunction

$F_1 \vee F_2$ “or” disjunction

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 disjunction
----------------	------	-------------------

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), \perp (false)

p, q, r, p_1, p_2, \dots

Literal: an atom a or its negation $\neg a$

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

$$I : \{p \rightarrow \top, q \rightarrow \perp, r \rightarrow \perp, \dots\}$$

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

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

We write $I \not\models F$, if F evaluates to \perp 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

$$I \models \top \quad I \not\models \perp$$

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

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

Inductive Case

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

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

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

What about \rightarrow , \leftrightarrow ? Already defined.

Propositional Logic: Semantics

Example

Base case

Quiz:

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

$I \models \top$ $I \not\models \perp$

$I \models p$, if $I(p) = \top$

$I \not\models p$, if $I(p) = \perp$

Which of the following is true?

a) $I \models F$

b) $I \not\models F$

Inductive Case

$I \models \neg F$, if $I \not\models F$

$I \models F_1 \wedge F_2$, if $I \models F_1$ and $I \models F_2$

$I \models F_1 \vee F_2$, if $I \models F_1$ or $I \models F_2$

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

Propositional Logic: Semantics

Satisfiability and Validity

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

F is valid, iff *for all* interpretations I, $I \models F$

F is contingent, iff it is satisfiable but not valid

Duality between satisfiability and validity:

F is valid, iff $\neg F$ is unsatisfiable

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. $(p \wedge q) \rightarrow \neg p$
2. $(p \wedge q) \rightarrow (p \vee \neg q)$
3. $(p \rightarrow (q \rightarrow r)) \wedge \neg((p \wedge q) \rightarrow r)$

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 $F \triangleq (p \wedge q) \rightarrow (p \vee \neg q)$
- We can enumerate its interpretations via a truth table

Deciding Satisfiability: Truth Table

- Let's look at another example $F \triangleq (p \vee q) \rightarrow (p \wedge q)$

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., $I \models F$
- 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>	$\frac{I \models \neg F}{I \not\models F}$	$\frac{I \not\models \neg F}{I \models F}$	<u>conj</u>	$\frac{I \models F_1 \wedge F_2}{I \models F_1 \quad I \models F_2}$	$\frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \text{ \underline{or} } I \not\models F_2}$
<u>disj</u>	$\frac{I \models F_1 \vee F_2}{I \models F_1 \text{ \underline{or} } I \models F_2}$	$\frac{I \not\models F_1 \vee F_2}{I \not\models F_1 \quad I \not\models F_2}$	<u>imp</u>	$\frac{I \models F_1 \rightarrow F_2}{I \models \neg F_1 \text{ \underline{or} } I \models F_2}$	$\frac{I \not\models F_1 \rightarrow F_2}{I \models F_1 \quad I \not\models F_2}$
<u>contr</u>	$\frac{I \models F \quad I \not\models F}{I \models \perp}$				

<u>neg</u>	$\frac{I \models \neg F}{I \not\models F}$	$\frac{I \not\models \neg F}{I \models F}$	<u>conj</u>	$\frac{I \models F_1 \wedge F_2}{I \models F_1 \quad I \models F_2}$	$\frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \text{ \underline{or} } I \not\models F_2}$
<u>disj</u>	$\frac{I \models F_1 \vee F_2}{I \models F_1 \text{ \underline{or} } I \models F_2}$	$\frac{I \not\models F_1 \vee F_2}{I \not\models F_1 \quad I \not\models F_2}$	<u>imp</u>	$\frac{I \models F_1 \rightarrow F_2}{I \models \neg F_1 \text{ \underline{or} } I \models F_2}$	$\frac{I \not\models F_1 \rightarrow F_2}{I \models F_1 \quad I \not\models F_2}$
<u>contr</u>	$\frac{I \models F \quad I \not\models F}{I \models \perp}$				

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

<u>neg</u>	$\frac{I \models \neg F}{I \not\models F}$	$\frac{I \not\models \neg F}{I \models F}$	<u>conj</u>	$\frac{I \models F_1 \wedge F_2}{I \models F_1 \quad I \models F_2}$	$\frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \text{ \underline{or} } I \not\models F_2}$
<u>disj</u>	$\frac{I \models F_1 \vee F_2}{I \models F_1 \text{ \underline{or} } I \models F_2}$	$\frac{I \not\models F_1 \vee F_2}{I \not\models F_1 \quad I \not\models F_2}$	<u>imp</u>	$\frac{I \models F_1 \rightarrow F_2}{I \models \neg F_1 \text{ \underline{or} } I \models F_2}$	$\frac{I \not\models F_1 \rightarrow F_2}{I \models F_1 \quad I \not\models F_2}$
<u>contr</u>	$\frac{I \models F \quad I \not\models F}{I \models \perp}$				

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!

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