CS228 : Logic for Computer Science

Lecture 0: Introduction and logistics

Instructor: S. Akshay

IIT Bombay, India

27-07-2026

Bugs in programs can be costly!

Bugs in programs can be costly!

June 4, 1996 - Ariane 5 Flight 501.
"Working code for the Ariane 4 rocket is reused in the Ariane 5, but the Ariane 5's faster engines trigger a bug in an arithmetic routine inside the rocket's flight computer. The error is in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines cause the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4, triggering an overflow condition that results in the flight computer crashing."

Source: Simson Garfinkel, Wired Magazine, 2005: `https://www.wired.com/2005/11/historys-worst-software-bugs/`

Bugs in programs can be costly!

June 4, 1996 - Ariane 5 Flight 501.

1993-Intel Pentium floating point divide.
"A silicon error causes Intel's highly promoted Pentium chip to make mistakes when dividing floating-point numbers that occur within a specific range. For example, dividing 4195835.0/3145727.0 yields 1.33374 instead of 1.33382, an error of 0.006 percent... The bug ultimately costs Intel $475 million."

Source: Simson Garfinkel, Wired Magazine, 2005: `https://www.wired.com/2005/11/historys-worst-software-bugs/`

Bugs in programs can be costly!

June 4, 1996 - Ariane 5 Flight 501.

1993-Intel Pentium floating point divide.

2005 - Toyota Prius Bug.
"Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly. But unlike the large-scale auto recalls of years past, the root of the Prius issue wasn't a hardware problem - it was a programming error in the smart car's embedded code. The Prius had a software bug."

Source: Simson Garfinkel, Wired Magazine, 2005: `https://www.wired.com/2005/11/historys-worst-software-bugs/`

Bugs in programs can be costly!

June 4, 1996 - Ariane 5 Flight 501.

1993-Intel Pentium floating point divide.

2005 - Toyota Prius Bug.

How do you catch bugs?

Bugs in programs can be costly!

June 4, 1996 - Ariane 5 Flight 501.

1993-Intel Pentium floating point divide.

2005 - Toyota Prius Bug.

How do you catch bugs?

- Software testing not enough always...
- Need mathematically formal and rigorous guarantees.
- How do we reason about software/programs?

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason
   - about human thought

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason
   - about human thought – isn't that philosophy??

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason
   - about human thought – isn't that philosophy??
   - about computing objects: systems or machines or programs

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason
   - about human thought – isn't that philosophy??
   - about computing objects: systems or machines or programs
2. How to automate this reasoning

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason
   - about human thought – isn't that philosophy??
   - about computing objects: systems or machines or programs
2. How to automate this reasoning

OMG!: Machines reasoning about machines? Is this AI?!

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason
   - about human thought – isn't that philosophy??
   - about computing objects: systems or machines or programs
2. How to automate this reasoning

OMG!: Machines reasoning about machines? Is this AI?!

- Not quite: but as we will see, we can use these techniques to reason about AI too!

CS228: Logic for Computer Science

This course in a nutshell

1. How to reason
   - about human thought – isn't that philosophy??
   - about computing objects: systems or machines or programs
2. How to automate this reasoning

What we need are

1. languages to formalize reasoning: symbolic logic
2. mathematical models of the computing objects: abstractions, transition systems
3. algorithms and tools to automate this whole process!

What this course is not about and what it is about

The idea of this course is not to...

- ... cover a specific topic in depth
- ... exhaustively cover all logics, automata, models

What this course is not about and what it is about

The idea of this course is not to...

- ... cover a specific topic in depth
- ... exhaustively cover all logics, automata, models

The idea of this course is to...

- ... prepare students to have a background in formal logic.
- ... see where logic occurs in CS and how it can be used for CS.
- ... look at some logics, techniques in detail.
- ...

Who uses Logic in CS?

How many queries to a logic tool (SAT/SMT solver to be precise) does Amazon Web Services invoke per day?

- Option A: 1 thousand
- Option B: 100 thousand
- Option C: 1 Million
- Option D: 1 Billion

Who uses Logic in CS?

How many queries to a logic tool (SAT/SMT solver to be precise) does Amazon Web Services invoke per day?

- Option A: 1 thousand
- Option B: 100 thousand
- Option C: 1 Million
- Option D: 1 Billion

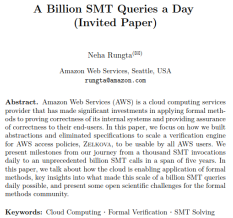
What do they use it for?

Who uses Logic in CS?

How many queries to a logic tool (SAT/SMT solver to be precise) does Amazon Web Services invoke per day?

- Option A: 1 thousand
- Option B: 100 thousand
- Option C: 1 Million
- Option D: 1 Billion

What do they use it for?



A Billion SMT Queries a Day
(Invited Paper)

Neha Rungta(✉)

Amazon Web Services, Seattle, USA
rungta@amazon.com

Abstract. Amazon Web Services (AWS) is a cloud computing services provider that has made significant investments in applying formal methods to proving correctness of its internal systems and providing assurance of correctness to their end-users. In this paper, we focus on how we built abstractions and eliminated specifications to scale a verification engine for AWS access policies, Zelkova, to be usable by all AWS users. We present milestones from our journey from a thousand SMT invocations daily to an unprecedented billion SMT calls in a span of five years. In this paper, we talk about how the cloud is enabling application of formal methods, key insights into what made this scale of a billion SMT queries daily possible, and present some open scientific challenges for the formal methods community.

Keywords: Cloud Computing · Formal Verification · SMT Solving

Who uses Logic in CS?

Widely used in industry

- Hardware and software verification
- Cyber-physical systems: Avionics, Automobiles, space!
- AI and databases
- Hot area: explainability in AI/ML!

Including Amazon, Intel, IBM, Microsoft Research, Google, Samsung, TCS, NASA, Mathworks ...

Who uses Logic in CS?

Widely used in industry

- Hardware and software verification
- Cyber-physical systems: Avionics, Automobiles, space!
- AI and databases
- Hot area: explainability in AI/ML!

Including Amazon, Intel, IBM, Microsoft Research, Google, Samsung, TCS, NASA, Mathworks ...

Widely used in Academia too

- research groups in all major universities around the world
- An interplay of mathematics and computer science, logic and coding, theory and practice.

Who uses Logic in CS?

Widely used in industry

- Hardware and software verification
- Cyber-physical systems: Avionics, Automobiles, space!
- AI and databases
- Hot area: explainability in AI/ML!

Including Amazon, Intel, IBM, Microsoft Research, Google, Samsung, TCS, NASA, Mathworks ...

Widely used in Academia too

- research groups in all major universities around the world
- An interplay of mathematics and computer science, logic and coding, theory and practice.

For a more colorful video, see `https://www.youtube.com/watch?v=AM_gwEKjnGY`

Course structure

Topics to be covered in this course:

- Module 1: Symbolic Logics and formal reasoning

Course structure

Topics to be covered in this course:

- Module 1: Symbolic Logics and formal reasoning
- Module 2: The problem of satisfiability and its applications

Course structure

Topics to be covered in this course:

- Module 1: Symbolic Logics and formal reasoning
- Module 2: The problem of satisfiability and its applications
- Module 3: Hoare Logic and Program verification

Course structure

Topics to be covered in this course:

- Module 1: Symbolic Logics and formal reasoning
- Module 2: The problem of satisfiability and its applications
- Module 3: Hoare Logic and Program verification
- Module 4: Temporal Logics and Model checking

There are advanced courses in each of these topics! CS433, CS615, CS 6104, CS713, CS735, CS738, CS739

Course structure

Topics to be covered in this course:

- Module 1: Symbolic Logics and formal reasoning
- Module 2: The problem of satisfiability and its applications
- Module 3: Hoare Logic and Program verification
- Module 4: Temporal Logics and Model checking

There are advanced courses in each of these topics! CS433, CS615, CS 6104, CS713, CS735, CS738, CS739 But this course binds them all.... :-)


*

*By Peter J. Yost - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=98351026

Course logistics

Mode of instruction

- Lectures: 3 hrs per week

Course logistics

Mode of instruction

- Lectures: 3 hrs per week
- Weekly problem solving & self study: 3+ hrs per week, compulsory!
- Problemsheets will be given every week.

Course logistics

Mode of instruction

- Lectures: 3 hrs per week
- Weekly problem solving & self study: 3+ hrs per week, compulsory!
- Problemsheets will be given every week.
- (Optional) problem solving sessions by TAs: $\sim$ 1 hr per week

Course logistics

Mode of instruction

- Lectures: 3 hrs per week
- Weekly problem solving & self study: 3+ hrs per week, compulsory!
- Problemsheets will be given every week.
- (Optional) problem solving sessions by TAs: $\sim$ 1 hr per week
- Piazza for doubts and more.

Course logistics

Mode of instruction

- Lectures: 3 hrs per week
- Weekly problem solving & self study: 3+ hrs per week, compulsory!
- Problemsheets will be given every week.
- (Optional) problem solving sessions by TAs: $\sim$ 1 hr per week
- Piazza for doubts and more.
- Office hours, if needed, by appointment.

Course logistics

Mode of instruction

- Lectures: 3 hrs per week
- Weekly problem solving & self study: 3+ hrs per week, compulsory!
- Problemsheets will be given every week.
- (Optional) problem solving sessions by TAs: $\sim$ 1 hr per week
- Piazza for doubts and more.
- Office hours, if needed, by appointment.

Evaluation Disclaimer: Tentative!

Course logistics

Mode of instruction

- Lectures: 3 hrs per week
- Weekly problem solving & self study: 3+ hrs per week, compulsory!
- Problemsheets will be given every week.
- (Optional) problem solving sessions by TAs: $\sim$ 1 hr per week
- Piazza for doubts and more.
- Office hours, if needed, by appointment.

Evaluation Disclaimer: Tentative!

- Quizzes (2 or 3) : 30%
- Midsem: 25%
- Final exam: 40%
- Other: Pop quizzes, Class participation : 5%

Module 1: Logic, a language and calculus for reasoning

1. Men are mortal
2. Socrates is a man

Socrates is mortal

Module 1: Logic, a language and calculus for reasoning

1. Men are mortal
2. Socrates is a man

Socrates is mortal

Intuitive Rule/Pattern:

1. α are β
2. γ is an α

γ is β

Module 1: Logic, a language and calculus for reasoning

1. Men are mortal
2. Socrates is a man

Socrates is mortal

Intuitive Rule/Pattern:

1. α are β
2. γ is an α

γ is β

1. A barber shaves all those who don't shave themselves
2. The barber needs a shave

Who shaves the barber?

Module 1: Logic, a language and calculus for reasoning

1. Men are mortal
2. Socrates is a man

Socrates is mortal

Intuitive Rule/Pattern:

1. α are β
2. γ is an α

γ is β

1. A barber shaves all those who don't shave themselves
2. The barber needs a shave

Who shaves the barber?

A symbolic logic zoo for reasoning

- Propositional logic
- Predicate logic
- Temporal logic
- Modal logic

Module 2: Satisfiability

Boolean Propositional Satisfiability

- Is a sentence written in the propositional logic satisfiable?
- That is, is there an assignment that evaluates it to true?

Module 2: Satisfiability

Boolean Propositional Satisfiability

- Is a sentence written in the propositional logic satisfiable?
- That is, is there an assignment that evaluates it to true?
- A central problem in Algorithms, complexity - NP-complete!

Module 2: Satisfiability

Boolean Propositional Satisfiability

- Is a sentence written in the propositional logic satisfiable?
- That is, is there an assignment that evaluates it to true?
- A central problem in Algorithms, complexity - NP-complete!
- Applications paramount – A whole book of problems that can be reduced to it.

Module 2: Satisfiability

Boolean Propositional Satisfiability

- Is a sentence written in the propositional logic satisfiable?
- That is, is there an assignment that evaluates it to true?
- A central problem in Algorithms, complexity - NP-complete!
- Applications paramount – A whole book of problems that can be reduced to it.
- Our formal reasoning, via logic, will lead to building efficient algorithms that solve many instances of this problem easily!

Module 2: Satisfiability

Boolean Propositional Satisfiability

- Is a sentence written in the propositional logic satisfiable?
- That is, is there an assignment that evaluates it to true?
- A central problem in Algorithms, complexity - NP-complete!
- Applications paramount – A whole book of problems that can be reduced to it.
- Our formal reasoning, via logic, will lead to building efficient algorithms that solve many instances of this problem easily!

Theory behind powerful solvers

- SAT solvers
- SMT solvers

Module 3: Application to reasoning about programs

```
int foo(int n) {
    int k, j;
    k = 0; j = 1;
    while (k != n) {
        k = k + 1;
        j = 2*j;
    }
    return(j);
```

```
list * bar(list * i) {
    list *j, *k;
    j = NULL;
    while (i != NULL) {
        k = i→next;
        i→next = j;
        j = i;
        i = k;
    }
    return(i)
```

- If "foo" is called with $n > 0$, does it always return 2^n?
- If "bar" is called with i pointing to an acyclic list, does i always point to an acyclic list when "bar" returns?

Module 3: Application to reasoning about programs

```
int foo(int n) {
    int k, j;
    k = 0; j = 1;
    while (k != n) {
        k = k + 1;
        j = 2*j;
    }
    return(j);
}
```

```
list * bar(list * i) {
    list *j, *k;
    j = NULL;
    while (i != NULL) {
        k = i→next;
        i→next = j;
        j = i;
        i = k;
    }
    return(i)
}
```

- If "foo" is called with $n > 0$, does it always return 2^n?
- If "bar" is called with i pointing to an acyclic list, does i always point to an acyclic list when "bar" returns?

- In general, we can't answer this (why?), but then what?

Module 3: Application to reasoning about programs

```
int foo(int n) {
    int k, j;
    k = 0; j = 1;
    while (k != n) {
        k = k + 1;
        j = 2*j;
    }
    return(j);
}
```

```
list * bar(list * i) {
    list *j, *k;
    j = NULL;
    while (i != NULL) {
        k = i→next;
        i→next = j;
        j = i;
        i = k;
    }
    return(i)
}
```

- If "foo" is called with $n > 0$, does it always return 2^n?
- If "bar" is called with i pointing to an acyclic list, does i always point to an acyclic list when "bar" returns?

- In general, we can't answer this (why?), but then what?
- The "art and science" of proving programs (in)correct

Module 3: Application to reasoning about programs

```
int foo(int n) {
    int k, j;
    k = 0; j = 1;
    while (k != n) {
        k = k + 1;
        j = 2*j;
    }
    return(j);
```

```
list * bar(list * i) {
    list *j, *k;
    j = NULL;
    while (i != NULL) {
        k = i→next;
        i→next = j;
        j = i;
        i = k;
    }
    return(i)
```

- If "foo" is called with $n > 0$, does it always return 2^n?
- If "bar" is called with i pointing to an acyclic list, does i always point to an acyclic list when "bar" returns?

- In general, we can't answer this (why?), but then what?
- The "art and science" of proving programs (in)correct – using logic and reasoning!

Module 4: Application to Model checking

System

satisfy?

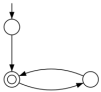
specification

System Model

model-checking

$\models$?

Spec Model



ϕ

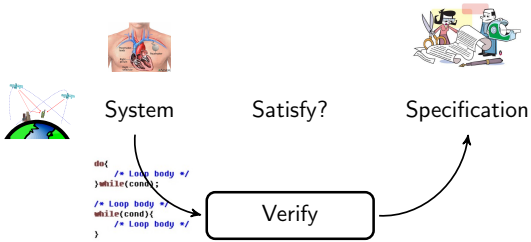
A Powerful Technique: Formal Verification

System Satisfy? Specification

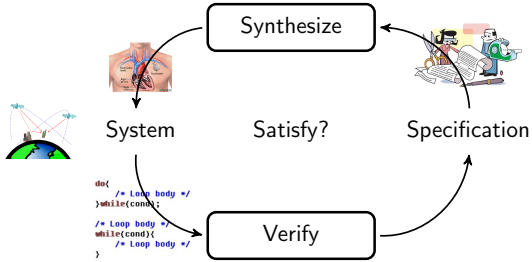
```
do{
    /* Loop body */
}while(cond);

/* Loop body */
while(cond){
    /* Loop body */
}
```

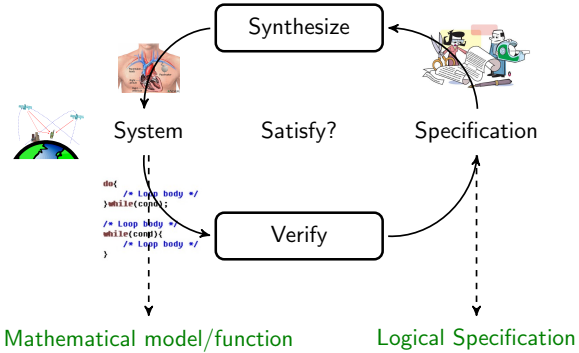
A Powerful Technique: Formal Verification



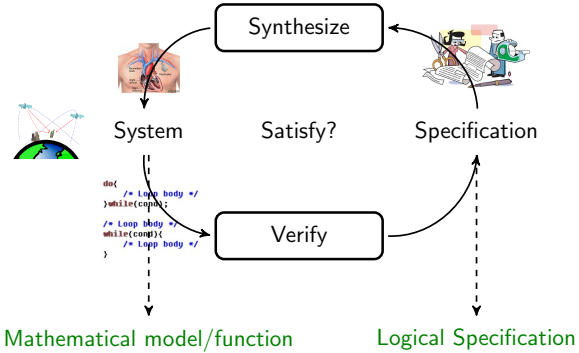
A Powerful Technique: Formal Verification



A Powerful Technique: Formal Verification



A Powerful Technique: Formal Verification



- Goal: Develop Efficient Algorithms to Verify and Synthesize
- Challenges: Expressivity of models, Scalability of Algorithms
- We stand on the Shoulders of Giants! Turing Awards in 1996 and 2007 were on this work.

Overlaps and links to other courses

- Unsurprisingly, there are overlaps with several courses

Undergrad courses

- Discrete structures (CS105) - reasoning ideas same, same but different!

Overlaps and links to other courses

- Unsurprisingly, there are overlaps with several courses

Undergrad courses

- Discrete structures (CS105) - reasoning ideas same, same but different!
- Automata theory (CS 310): last part will lead to automata theory

Overlaps and links to other courses

- Unsurprisingly, there are overlaps with several courses

Undergrad courses

- Discrete structures (CS105) - reasoning ideas same, same but different!
- Automata theory (CS 310): last part will lead to automata theory

Other linked courses

- CS 433: Automated reasoning: advanced course more practical
- CS 713: Specialized course on Automata & logic connections
- CS 738: Model checking, cyber-physical systems
- CS 6104: Quantitative Verification
- CS 781: Formal methods and machine learning
- CS 766: Analysis of concurrent programs

Links to other domains

Programs and Systems are everywhere and everyone wants to make sure they work!

Links to other domains

Programs and Systems are everywhere and everyone wants to make sure they work!

- Machine learning and AI
  - Techniques to verify and explain, check robustness of DNNs.
  - Formal reasoning and logic started AI.
  - Probabilistic logics
  - Prob systems: Markov chains, Markov decision processes, POMDPs
  - Logical expressiveness of GNNs, Transformers, LLMs...

Links to other domains

Programs and Systems are everywhere and everyone wants to make sure they work!

- Machine learning and AI
  - Techniques to verify and explain, check robustness of DNNs.
  - Formal reasoning and logic started AI.
  - Probabilistic logics
  - Prob systems: Markov chains, Markov decision processes, POMDPs
  - Logical expressiveness of GNNs, Transformers, LLMs...
- Programming languages
- Blockchains and other concurrent, distributed algorithms: verifying smart contracts.

Links to other domains

Programs and Systems are everywhere and everyone wants to make sure they work!

- Machine learning and AI
  - Techniques to verify and explain, check robustness of DNNs.
  - Formal reasoning and logic started AI.
  - Probabilistic logics
  - Prob systems: Markov chains, Markov decision processes, POMDPs
  - Logical expressiveness of GNNs, Transformers, LLMs...
- Programming languages
- Blockchains and other concurrent, distributed algorithms: verifying smart contracts.
- Data bases Specifying properties and using solvers!
- Cryptography security protocols and formalizing attacks!

Links to other domains

Programs and Systems are everywhere and everyone wants to make sure they work!

- Machine learning and AI
  - Techniques to verify and explain, check robustness of DNNs.
  - Formal reasoning and logic started AI.
  - Probabilistic logics
  - Prob systems: Markov chains, Markov decision processes, POMDPs
  - Logical expressiveness of GNNs, Transformers, LLMs...
- Programming languages
- Blockchains and other concurrent, distributed algorithms: verifying smart contracts.
- Data bases Specifying properties and using solvers!
- Cryptography security protocols and formalizing attacks!
- Cyber-physical, real-time systems, etc.

In conclusion

CS228: Logic for CS

- Formalizing using Logic
- Mathematical Reasoning about programs and systems
- Building efficient algorithms when possible
- A gateway to the fascinating area of formal methods in CS!

In conclusion

CS228: Logic for CS

- Formalizing using Logic
- Mathematical Reasoning about programs and systems
- Building efficient algorithms when possible
- A gateway to the fascinating area of formal methods in CS!

- Visit the course webpage for all details!
  `http://www.cse.iitb.ac.in/~akshayss/courses/cs228`