Overview of E0 205
Mathematical Logic and Theorem Proving

Deepak D'Souza and Kamal Lodaya

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

06 Jan 2025

Mathematical Logic and Theorem Proving

- "Mathematical" Logic (pioneered by Boole, Frege, Russell, Hilbert, Gödel, ...)
  - Goals related to foundations of Mathematics (formalizing set theory and mathematical reasoning techniques)
  - Applications in Math and Theoretical CS (e.g. Büchi's logical characterisations of regular languages)

  as opposed to Philosophical Logic.
- SMT (SAT+Decision Procedures for certain theories) vs Theorem Proving
  - Fully automated vs Interactive.

Why study Logic in Computer Science?

Computability

- Notions of computability were proposed to answer questions in logic
  - Formalizing mathematics (coming up with a complete proof system, deciding truth of logical statements) led to Hilbert proposing the "Entscheidungsproblem" (decision problem for logical validity).
  - Church and Turing separately proposed Lambda Calculus and Turing machines as notions of computability, and showed the Entscheidungsproblem was undecidable.
- Natural computational problems
  - SAT complete for NP, Horn-SAT complete for P
  - FO with fixpoints.

Why study Logic in Computer Science?

Verification and Synthesis

- Specification languages
  - Temporal Logic
  - Floyd-Hoare Logic
- Checking whether a program/system satisfies a specification
  - Program satisfies a pre-post specification if generated Verification Conditions (VCs) are logically valid.
  - Model-Checking procedures for Temporal Logics.
  - Constrained Horn Clauses
- Symbolic Analysis
  - Symbolic Model-Checking
  - Predicate abstraction
  - Controller Synthesis

Others (Proofs as Types, Algorithmic meta theorems, etc)

Course Contents

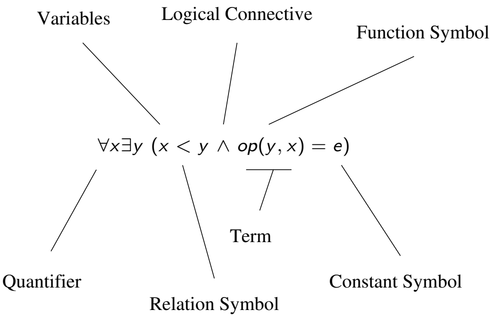
- Mathematical Logic
  - Propositional and First-Order Logic
  - Definability
  - Normal Forms
  - Sound and complete proof systems (Sequent Calculus)
  - Compactness and Lowenheim-Skolem Theorem
- Decision Procedures
  - DPLL procedure for Propositional Logic (SAT)
  - Equality and Uninterpreted Functions (EUF)
  - Real and Integer Linear Arithmetic
  - Array logic
  - Nelson-Oppen combination

Example FO Logic Formula

$$\forall x \exists y (x < y \wedge op(y, x) = e)$$

Example FO Logic Formula

$$\forall x \exists y (x < y \land op(y, x) = e)$$



FO Signature

A First-Order signature is a tuple

$$S = (R, F, C)$$

where

- R is a countable set of relation symbols
- F is a countable set of function symbols
- C is a countable set of constant symbols

Each relational/functional symbol comes with an associated "arity".

Example FO signatures

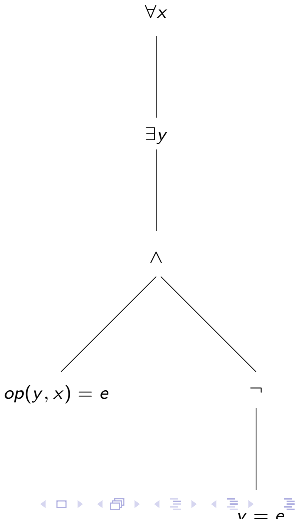
- $S_{gr} = (\{\}, \{op^{(2)}\}, \{e\})$ (Groups)
- $S_{ogr} = (\{<^{(2)}\}, \{op^{(2)}\}, \{e\})$ (Ordered Groups)
- $S_{ar} = (\{\}, \{+^{(2)}, \cdot^{(2)}\}, \{0, 1\})$ (Arithmetic)
- $S_{eq} = (\{r^{(2)}\}, \{\}, \{\})$ (Equivalence Relations)

Semantics: Example

Find the truth of the S_{gr}-formula

$$\forall x \exists y ((op(y,x) = e) \wedge \neg (y = e))$$

in the structure $(\mathbb{Z}, +, 0)$.



Theories

An S-theory is a set of S-sentences T which is satisfiable and closed under logical consequence.

The theory of a set of S-formulas T, written "$Th(T)$", is the set of S-sentences that are logical consequences of T. That is:

$$Th(T) = \{\varphi \in L_0^S \mid T \models \varphi\}.$$

Theory of Groups $Th(\Phi_{gr})$

Let Φ_{gr} be the set of formulas (group axioms):

$$\forall x \forall y \forall z\ (op(op(x,y),z) = op(x, op(y,z))) \quad (1)$$

$$\forall x\ (op(x,e) = x) \quad (2)$$

$$\forall x \exists y\ (op(x,y) = e) \quad (3)$$

Then $Th(\Phi_{gr})$

- Contains $\forall x \exists y (op(y,x) = e)$, but

Example of Group Theory

Group Axioms Φ_{Gr}

$$\forall x \forall y \forall z\ ((x \circ y) \circ z = x \circ (y \circ z)) \tag{4}$$
$$\forall x\ (x \circ e = x) \tag{5}$$
$$\forall x \exists y\ (x \circ y = e) \tag{6}$$

Structures for Φ_{Gr}: $(\mathbb{Z}, +, 0)$ and $(\mathbb{R}, +, 0)$; but not $(\mathbb{R}, \cdot, 1)$.

Theorem: Every element of a group has a left-inverse: $\forall x \exists y (y \circ x = e)$.

Question: is there a complete proof system for Group theory? That is, whenever we have $\Phi_{Gr} \models \varphi$, then we also have $\Phi_{Gr} \vdash \varphi$.

Gödel's Completeness Theorem

Let $\Phi \vdash \varphi$ denote a derivation of φ from Φ using the Sequent Calculus proof system.

Theorem (Completeness)

For any set of first-order logic sentences Φ:

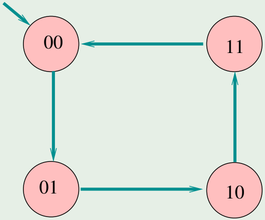
$$\Phi \models \varphi \text{ iff } \Phi \vdash \varphi.$$

Some consequences of the theorem and its proof:

- There is a complete proof system for Group Theory (Sequent Calculus + Φ_{Gr} as axioms).
- (Lowenheim-Skolem) If a set of FO formulas Φ is satisfiable then it is satisfiable in a countable model.
- (Compactness) If a set of formulas Φ is unsatisfiable, then there is a finite subset of Φ which is unsatisfiable.

Boolean SAT solving

Does the system satisfy the temporal logic formula $G(b \implies X(\neg b))$?



In bounded model-checking we could ask for a path of length 2 that violates the specification: Is

$$\neg a_0 \wedge \neg b_0 \wedge T(a_0, b_0, a_1, b_1) \wedge T(a_1, b_1, a_2, b_2) \wedge b_1 \wedge b_2,$$

where $T(a, b, a', b') = (\neg a \wedge a' \wedge b \iff b') \vee (a \wedge \neg a' \wedge b \iff \neg b')$, satisfiable?

Linear Arithmetic

Bounded model-checking for programs:

```
int x = 19;
int y = 15;
while (x >= 10) {
  int z = -1;
  x = x + z;
}
assert(y >= x);
```

Does there exist zero-iteration execution violating the assertion: Is

$$x_1 = 19 \land y_1 = 15 \land x_1 < 10 \land y_1 < x_1$$

satisfiable?

Linear Arithmetic

Floyd-Hoare style verification of programs:

```
int x = 19;
int y = 15;
// inv: x <= 20
while (x >= 10) {
  int z = -1;
  x = x + z;
}
assert(y >= x);
```

Are the constraints: $\forall x, y, z, x'$:

$$x = 19 \implies x \leq 20$$

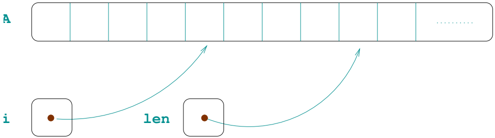
$$x \leq 20 \wedge x \geq 10 \wedge z = -1 \wedge x' = x + z \implies x' \leq 20$$

$$x \leq 20 \wedge y = 15 \wedge \neg(x \geq 10) \implies y \geq x'$$

valid?

Array Logic

```
ainit(int A[], int len) {
// Pre: 0 <= len
  int i = 0;
  while (i < len) {
    A[i] = 0;
    i = i + 1;
  }
}
// Post:
forall k: ((0 <= k < len)
              => A[k] = 0)
```



Loop invariant:

$$(0 \leq i \leq len) \wedge \forall k((0 \leq k < i) \implies A[k] = 0)$$

Verification condition:

$$[(0 \leq i \leq len) \wedge \forall k((0 \leq k < i) \implies A[k] = 0) \wedge \neg(i < len)] \implies$$

$$\forall k : ((0 \leq k < len) \implies A[k] = 0).$$

Program Transformation

Example: Are these programs equivalent?

```
S1: z := (x1 + y1) * (x2 + y2);
```

```
T1: u1 := (x1 + y1);
T2: u2 := (x2 + y2);
T3: z := u1 * u2;
```

We want to check whether (forall $x_1, x_2, y_1, y_2, z_1, z_2, u_1, u_2$)

$$
\begin{aligned}
&(z_1 = (x_1 + y_1) * (x_2 + y_2) \wedge \\
&\; u_1 = x_1 + y_1 \;\wedge u_2 = x_2 + y_2 \;\wedge z_2 = u_1 * u_2) \\
&\rightarrow z_1 = z_2.
\end{aligned}
$$

Since reasoning about 32 bit ints and addition and multiplication is difficult, We could instead check whether the EUF formula:

$$
\begin{aligned}
&(z_1 = G(F(x_1, y_1), F(x_2, y_2)) \wedge \\
&\; u_1 = F(x_1, y_1) \;\wedge u_2 = F(x_2 + y_2) \;\wedge z_2 = G(u_1, u_2)) \\
&\rightarrow z_1 = z_2.
\end{aligned}
$$

is valid.

Nelson-Oppen Combination

Example: Is this sentence satisfiable?

$$f(f(x) - f(y)) \neq f(z) \wedge x \leq y \wedge y + z \leq x \wedge z \geq 0$$

Nelson-Oppen Combination

Example: Is this sentence satisfiable?

$$f(f(x) - f(y)) \neq f(z) \wedge x \leq y \wedge y + z \leq x \wedge z \geq 0$$

No, because the arithmetic constraints imply that $x = y$ and $z = 0$; and the functional constraints must then imply that $f(f(x) - f(y)) = f(0) = f(z)$.

Nelson-Oppen Combination

Shows how we can combine decision procedures for two theories into a decision procedure for their union.

"Equality Sharing" Procedure:

Is this sentence satisfiable?

$$f(f(x) - f(y)) \neq f(z) \wedge x \leq y \wedge y + z \leq x \wedge z \geq 0$$

Arithmetic Constraints

$$\begin{array}{rcl} x & \leq & y \\ y + z & \leq & x \\ z & \geq & 0 \\ g_2 - g_3 & = & g_1 \end{array}$$

Function Constraints

$$\begin{array}{rcl} f(g_1) & \neq & f(z) \\ f(x) & = & g_2 \\ f(y) & = & g_3 \end{array}$$

Constrained Horn Clauses

```
int x = 19;
while (*) {
  int z = f();
  x = x + z;
}
int y = g();
assert(y >= x);
```

Find unary relations f, g and *inv* such that:

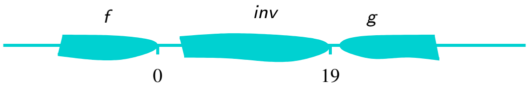
$$x = 19 \implies inv(x)$$
$$inv(x) \wedge \mathrm{f}(z) \wedge x' = x + z \implies inv(x')$$
$$inv(x) \wedge \mathrm{g}(y) \implies y \geq x$$

Constrained Horn Clauses

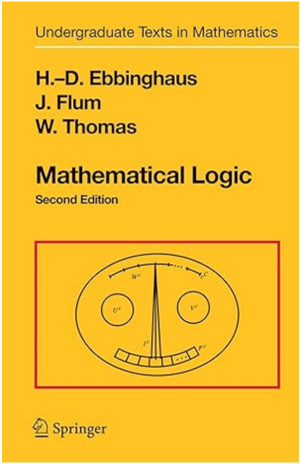
```
int x = 19;
while (*) {
  int z = f();
  x = x + z;
}
int y = g();
assert(y >= x);
```

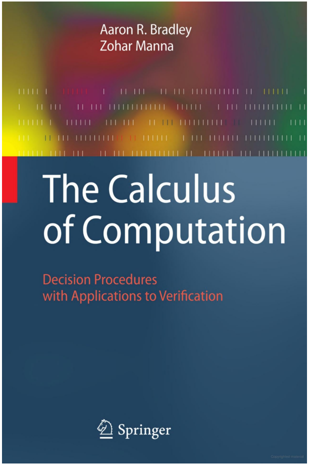
Find unary relations f, g and *inv* such that:

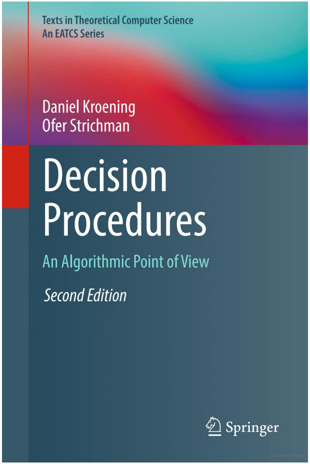
$$x = 19 \implies inv(x)$$
$$inv(x) \wedge \mathrm{f}(z) \wedge x' = x + z \implies inv(x')$$
$$inv(x) \wedge \mathrm{g}(y) \implies y \geq x$$



Course Textbooks







Course Details

- Weightage: 40% assignments + seminar, 20% midsem exam, 40% final exam.
- Assignments to be done on your own.
- Dishonesty Policy: Any instance of copying in an assignment will fetch you a 0 in that assignment + one grade reduction + report to DCC.
- Seminar (in pairs) can be chosen from list on course webpage or your own topic.
- Course webpage:
  `www.csa.iisc.ac.in/~deepakd/logic-2025`
- Teaching assistants for the course: Alan Jojo and Abhishek Uppar
- Those interested in crediting/auditing please send me an email so that I can add you to the course Teams / mailing list.