CS257: Introduction to Automated Reasoning

Propositional Logic Basics

Stanford
University

CENTVR

Learning objectives

Through active engagement and completion of course activities, you will be able to:

- Understand the syntax, semantics, and properties of different logical languages for encoding different decision problems
- Understand and prove properties about popular automated reasoning procedures (e.g., CDCL, DPLL(T), Simplex, Nelson-Oppen)
- Understand the roles of automated reasoning in real-world applications such model checking, symbolic execution, and synthesis
- Get hands-on experience in using off-the-shelf SAT/SMT solvers
- Get exposure to formal methods literature and engage in formal methods research

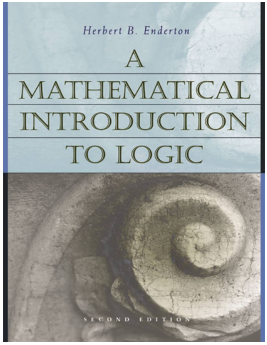
Assessments

- In-class participation
  - We use PollEveryWhere for in-class “quizzes” and discussions
- 3 homework assignments (due dates in the syllabus)
  - Programming components and writing components
  - Hand in on Canvas

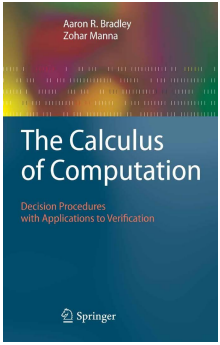
Assessments (cont.)

- Take-home midterm exam
  - 48 hour period to take exam starting at 7 AM on Thursday Nov. 2
  - Completed on your own
  - In-person review session 10:30 AM–12 PM Tuesday Oct. 31
  - We will book a back-up room for your use
- Final project
  - Conducted in pairs
  - **Option 1:** Implement a decision procedure and an optimization of it
  - **Option 2:** Investigate a new AR-related research problem
  - Deliverables: project proposal, final report, and code/proof artifact

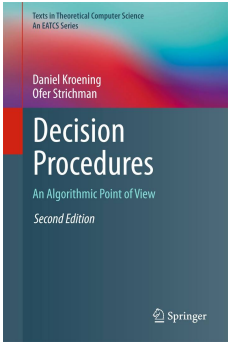
Textbooks



MI



CC



DP

- All freely available through Stanford (see syllabus)
- Please complete the assigned readings (see course website) before the lecture!

Other Course Details

- Course website: `http://web.stanford.edu/class/cs257/`
- Canvas: `https://canvas.stanford.edu/courses/176963`
- Ed Discussion: `edstem.org/us/courses/44522`
- All slides/assignments linked on the course website
- All assignment materials accessible via Canvas
- All announcements via Ed Discussion

People

Faculty Instructor

Caroline Trippel
OH: Wed 12-1 PM

Student Instructor

Andrew (Haoze) Wu
OH: Tues 9-10 AM

Course Assistant

Hanna Lachnitt
OH: Mon 2-3 PM,
Thu 3-4 PM

Guest Lecturers

- Clark Barrett, Proof systems, Theory of Arrays, Decision procedures for strings and sequences
  - Professor of Computer Science at Stanford
  - Co-founded the field of satisfiability modulo theories (SMT)
- Nina Narodytska, Formal Explanation of AI
  - Senior Researcher at VMware
  - Leading expert on MaxSAT and Formal XAI
- Mathias Preiner, Theory of Bitvectors
  - Research Scientist at Stanford in the Centaur Lab
  - One of the main developers of the SMT solvers Boolector and Bitwuzla
- Christopher Hahn, AI for AR
  - Former Visiting Assistant Professor of Computer Science at Stanford, Incoming Research Scientist at Google X
- John Matthews, Hardware Security Verification
  - Senior Principal Engineer at Intel

Propositional Logic

- Syntax (MI Ch. 1.0-1.1)
- Semantics, Satisfiability, and Validity (MI Ch. 1.2)
- Proof by deduction (CC Ch. 1.1-1.5)

* Some of the slides today are adapted from those of Clark Barrett's and Emina Torlak's.

Propositional Logic: Warm-up

We are about to construct a language into which we can translate English sentences.

Unlike natural languages (such as English or Chinese), it will be a formal language, with precise formation rules.

English	Formal language
Traces of potassium were observed.	K
Traces of potassium were not observed.	$\neg K$
The sample contained chlorine.	C
If traces of potassium were observed, then the sample did not contain chlorine.	?
Neither did the sample contain chlorine, nor were traces of potassium observed	?

Discuss with your neighbors, and submit your answers at

`https://pollev.com/andreww095`

Propositional Logic: Motivations

We are about to construct a language into which we can translate English sentences.

Unlike natural languages (such as English or Chinese), it will be a formal language, with precise formation rules.

English	Formal language
Traces of potassium were observed.	K
Traces of potassium were not observed.	$\neg K$
The sample contained chlorine.	C
If traces of potassium were observed, then the sample did not contain chlorine.	$K \to \neg C$
Neither did the sample contain chlorine, nor were traces of potassium observed	$\neg(C \wedge K)$ or $(\neg C) \wedge (\neg K)$

Suppose, the chemist emerges from her laboratory and announces that she observed traces of potassium but that the sample contained no chlorine...

Defining Propositional Logic

Formal language/logic allow us to escape from the ambiguities of natural languages.

But as the cost, formal logics will have a sharply limited degree of expressiveness.

A **formal logic** is defined by its syntax and semantics.

The **syntax** of a logical language consists of a set of symbols and rules for combining them to form "sentences" (in this case, formulae) of the language.

The **semantics** of a logic provides its meaning. In propositional logic, meaning is given by the truth values *true* and *false*, where $\mathit{true} \neq \mathit{false}$.

Syntax: symbols and expressions

An **alphabet** is a set of symbols. The alphabet of propositional logic consists of

- **atoms**:
  - **Truth symbols (constants)**: $\top$ ("true"), $\bot$ ("false")
  - **Propositional variables**: $p, q, r, \ldots$

  We use $\mathcal{B}$ to denote the set of atoms.
- **logical symbols:** logical connectives (i.e., "$\neg$", "$\wedge$", "$\vee$", "$\rightarrow$", "$\leftrightarrow$"), parentheses (i.e., "(", ")")

An **expression** is a finite sequence of symbols:

- $(p \wedge q)$
- $((\neg p) \rightarrow r)$
- $)) \leftrightarrow)s$ (Is this an expressions?)

Not all expressions make sense. Part of the job of the syntax is to restrict the kinds of expressions that will be allowed.

Syntax: Formula-building operations

We use a formal inductive definition to define the set $\mathcal{W}$ of well-formed formulas (or simply formulas or wffs) in propositional logic. These are the set of expressions that are allowed in propositional logic.

Formula-building operations $\mathcal{F}$:

- $\mathcal{E}_{\neg}(\alpha) = (\neg\alpha)$ (negation)
- $\mathcal{E}_{\wedge}(\alpha, \beta) = (\alpha \wedge \beta)$ (conjunction)
- $\mathcal{E}_{\vee}(\alpha, \beta) = (\alpha \vee \beta)$ (disjunction)
- $\mathcal{E}_{\rightarrow}(\alpha, \beta) = (\alpha \rightarrow \beta)$ (implication)
- $\mathcal{E}_{\leftrightarrow}(\alpha, \beta) = (\alpha \leftrightarrow \beta)$ (iff)

applying operations in $\mathcal{F}$ some finite number of times

The set of well-formed formulas is the set of all expressions generated by $\mathcal{F}$ from $\mathcal{B}$. I.e., every atom $A \in \mathcal{B}$ is a wff. If α and β are wffs, so are the expressions generated from them by $\mathcal{F}$.

Structural Induction

Induction Principle

If C is the set generated from B by F and S is a set which includes B and is closed under F, then we say S is inductive with respect to C.

Note that this also shows that $C \subseteq S$.

We often use the induction principle to show that a set C defined in this way has a particular property. The argument looks like this: (i) Define S to be the subset of U with some property P; (ii) Show that S is inductive with respect to C.

This proves that $C \subseteq S$ and thus all elements of C have property P.

We often use structural induction to prove properties about formulas themselves.

Structural Induction: Example

Given our inductive definition of well-formed formulas, we can use the induction principle to prove things about the set $\mathcal{W}$ of well-formed formulas.

Example

Prove that any **wff** has the same number of left parentheses and right parentheses.

Proof

Let $l(\alpha)$ be the number of left parentheses and $r(\alpha)$ the number of right parentheses in an expression α. Let S be the set of all expressions α such that $l(\alpha) = r(\alpha)$. We wish to show that $\mathcal{W} \subseteq S$. This follows from the induction principle if we can show that S is inductive.

Base Case:

We must show that $\mathcal{B} \subseteq S$. Recall that $\mathcal{B}$ is the set of expressions consisting of a single propositional symbol. It is clear that for such expressions, $l(\alpha) = r(\alpha) = 0$.

Structural Induction: Example

Inductive Case:

We must show that S is closed under each formula-building operator in $\mathcal{F}$.

- $\mathcal{E}_\neg$
  Suppose $\alpha \in S$. We know that $\mathcal{E}_\neg(\alpha) = (\neg\alpha)$. It follows that $l(\mathcal{E}_\neg(\alpha)) = 1 + l(\alpha)$ and $r(\mathcal{E}_\neg(\alpha)) = 1 + r(\alpha)$.
  But because $\alpha \in S$, we know that $l(\alpha) = r(\alpha)$, so it follows that $l(\mathcal{E}_\neg(\alpha)) = r(\mathcal{E}_\neg(\alpha))$, and thus $\mathcal{E}_\neg(\alpha) \in S$.
- $\mathcal{E}_\wedge$
  Suppose $\alpha, \beta \in S$. We know that $\mathcal{E}_\wedge(\alpha, \beta) = (\alpha \wedge \beta)$. Thus $l(\mathcal{E}_\wedge(\alpha, \beta)) = 1 + l(\alpha) + l(\beta)$ and $r(\mathcal{E}_\wedge(\alpha, \beta)) = 1 + r(\alpha) + r(\beta)$.
  As before, it follows from the inductive hypothesis that $\mathcal{E}_\wedge(\alpha, \beta) \in S$
- The arguments for $\mathcal{E}_\vee$, $\mathcal{E}_\rightarrow$, and $\mathcal{E}_\leftrightarrow$ are exactly analogous to the one for $\mathcal{E}_\wedge$.

Since S includes $\mathcal{B}$ and is closed under the operations in $\mathcal{F}$, it is inductive. It follows by the induction principle that $\mathcal{W} \subseteq S$.

Notational conventions for formulas

- A countably infinite set of propositional variables exist, we typically use $p, q, r, p_1, p_2, p_3, \ldots$ to denote them.
- Can omit outermost parentheses: $p \wedge q$ instead of $(p \wedge q)$
- Can further omit parentheses by defining **order of operations (precedence)**:
  - Negation binds the strongest with small as possible scope: $\neg p \wedge q$ means $((\neg p) \wedge q)$
  - $\wedge$ bind stronger than $\vee$: $p_1 \wedge p_2 \vee p_3$ means $(p_1 \wedge p_2) \vee p_3$
  - $\vee$ bind stronger than $\rightarrow$, $\leftrightarrow$: $p_1 \wedge p_2 \rightarrow \neg p_3 \vee p_4$ means $((p_1 \wedge p_2) \rightarrow ((\neg p_3) \vee p_4))$
  - When one symbol is used repeatedly grouping is to the right: $p_1 \wedge p_2 \wedge p_3$ is $(p_1 \wedge (p_2 \wedge p_3))$

Propositional Logic: Semantics

The meaning of a wff α is a truth value T or F.

Intuitively, given a mapping v from propositional symbols in α to $\{0, 1\}$, where 0 represents F and 1 represents T, we can determine the meaning of α.

This mapping v is called an **variable assignment** (or **interpretation**) of α.

From v, we can define an extension $\overline{v}$ as follows:

- $\overline{v}(\bot) = 0$ and $\overline{v}(\top) = 1$
- For propositional symbol p_i, $\overline{v}(p_i) = v(p_i)$
- $\overline{v}(\mathcal{E}_\neg(\alpha)) = 1 - \overline{v}(\alpha)$
- $\overline{v}(\mathcal{E}_\wedge(\alpha, \beta)) = \min(\overline{v}(\alpha), \overline{v}(\beta))$
- $\overline{v}(\mathcal{E}_\vee(\alpha, \beta)) = \max(\overline{v}(\alpha), \overline{v}(\beta))$
- $\overline{v}(\mathcal{E}_\rightarrow(\alpha, \beta)) = \max(1 - \overline{v}(\alpha), \overline{v}(\beta))$
- $\overline{v}(\mathcal{E}_\leftrightarrow(\alpha, \beta)) = 1 - |\overline{v}(\alpha) - \overline{v}(\beta)|$

These statements are equivalent:

- $\overline{v}(\alpha) = 1$
- $v \vDash \alpha$
- v is a **model** of α
- v is a **satisfying assignment** of α
- v **satisfies** α

Satisfiability, logical implication, and validity

A wff α is **satisfiable** iff there exists an interpretation v such that $\overline{v}(\alpha) = 1$.

α is **unsatisfiable** iff it is not satisfiable, that is, $\overline{v}(\alpha) = 0$ for all v.

$U = \{\alpha_1, ...\}$ is **satisfiable** iff there exists v such that $\overline{v}(\alpha_i) = 1$ for all $\alpha_i \in U$.

U **logically implies** wff β (written $U \models \beta$) iff every satisfying assignment v to U also satisfies β. We say β is a **logical consequence** of U.

Special cases:

- If $\varnothing \models \alpha$, then we say α is a **tautology** or α is **valid** and write $\models \alpha$.
- If U is **unsatisfiable**, then $U \models \alpha$ for every wff α.
- α_1, α_2 are **logically equivalent**, written $\alpha_1 \models\models \alpha_2$, iff $\{\alpha_1\} \models \alpha_2$ and $\{\alpha_2\} \models \alpha_1$.

Satisfiability, logical implication, and validity

Satisfiability and validty are dual concepts: a wff α is valid iff $\neg\alpha$ is unsatisfiable.

If we have a procedure that can check satisfiability of a propositional formula α, then we can also check the validity of α and vice versa.

Relation between $\models$ and $\rightarrow$

- $\alpha_1 \models\!\models \alpha_2$ and $\alpha_1 \models \alpha_2$ are not formulas.
- **Logical equivalence:** $\alpha_1 \models\!\models \alpha_2$ iff $\alpha_1 \leftrightarrow \alpha_2$ is valid.
- **Logical implication:** $\alpha_1 \models \alpha_2$ iff $\alpha_1 \rightarrow \alpha_2$ is valid.

A couple of notes on notations

- While we use v to denote an interpretation, it is also common to use $\mathcal{I}$ to denote it (e.g., CC Sec. 1.2).
- While we use $\alpha \models\models \alpha'$ to denote logical equivalence between two wffs, it is also common to use $\alpha \equiv \alpha'$.
- We have already seen three ways to use the turnstile symbol "$\models$":
  - An interpretation v satisfies a wff α: $v \models \alpha$
  - A wff α' logically implies a wff α: $\alpha' \models \alpha$
  - A set of wffs U logically implies a wff α: $U \models \alpha$

  This "abuse" of the turnstile symbol is common in the literature and generally it donotes some kind of "entailment". There will be no confusion as long as we clarify the meaning of the left-hand-side.

Defining One Operator in Terms of Another

We say a binary opeartor $\circ$ is **defined from** a set of operators $\{\circ_1, ..., \circ_n\}$ if for all α and β, $\alpha \circ \beta \vDash\dashv \gamma$, where γ is constructed by applying operators in $\{\circ_1, ..., \circ_n\}$ to α and β a finite number of times.

Boolean opeartors $(\vee, \wedge, \rightarrow, \leftrightarrow)$ can be defined from $\neg$ and one of $\vee, \wedge, \rightarrow, \leftrightarrow$.

Example: defining $\vee, \wedge, \leftrightarrow$ using $\neg$ and $\rightarrow$.

- $\alpha \wedge \beta \vDash\dashv \neg(\alpha \rightarrow \neg\beta)$
- $\alpha \vee \beta \vDash\dashv \neg\alpha \rightarrow \beta$
- $\alpha \leftrightarrow \beta \vDash\dashv \neg((\alpha \rightarrow \beta) \rightarrow \neg(\beta \rightarrow \alpha))$

Why do we care about this?

- Ease for structural induction.
- Many algorithms are defined over normal forms using a specified subset of the boolean operators.

Decision Procedure in Propositional Logic

Given a set of wffs U, and a wff α, a **decision procedure** for U is a **terminating** procedure[1] that takes α and returns

- yes if $\alpha \in U$,
- no if $\alpha \notin U$.

This class:

- We consider decision procedure for validity/satisfiability (U is the set of valid/satisfiable formulas).

[1] A procedure does not necessarily terminate, whereas an algorithm terminates.

Basic Decision Procedures for Validity/Satisfiability

Two fundamental strategies for deciding validity/satisfiability:

- **Search-based procedure**: enumerate all possible interpretations of the given wff.
- **Deduction-based procedure**: use a mechanisms of reasoning based on axioms and inference rules to deduce validity.

SAT solvers (covered later) interleave search and deduction.

The Truth-table Method

In propositional logic, enumerating solutions can be done using **truth tables**.

Example: is $\alpha := (p \wedge q) \rightarrow (p \vee \neg q)$ a valid formula?

p	q	$p \wedge q$	$\neg q$	$p \vee \neg q$	α
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

Drawbacks?

- Need to evaluate a formula for each of 2^n possible interpretations. Memory efficient, but runtime inefficient.
- Works when the number of solutions is finite.