

CS103 Introduction

David L. Dill
Department of Computer Science
Stanford University

1 / 33

Outline

- Course administration
- Propositional logic
 - Syntax
 - Truth tables
 - Tautology and logical equivalence

2 / 33

Course staff

Lecturer: David Dill

Course Assistants:

- Rakesh Achanta (rakesha@stanford.edu)
- Zavain Dar (zdar@stanford.edu)
- Miroslav Kukla (mkukla@stanford.edu)
- Hrysoula Papadakis (hpapadak@stanford.edu)
- Le Yu (billyue@stanford.edu)
- Qiao Zhao (bridger.zhao@gmail.com)

3 / 33

Purpose of the class

Our goal is to get you up-to-speed on the basic theory of computer science as quickly as you can.

- So you have some of the intellectual tools to “do” computer science.
- So you can understand the “secret language” of computer science.
- So you are ready for higher-level courses.
- “Education” – material in this course includes some of the greatest intellectual achievements of the last century.
- For fun.

CS103 is pretty intense, because it combines 2-3 courses from the curriculum of a few years ago that were already pretty intense (and, obviously, we leave a few things out).

We did it this way to reduce the number of requirements and give CS majors freedom to choose your direction later on in the major.

4 / 33

Topics

The topics we'll cover were chosen because they are basic foundations of computer science.

But this is not a survey course. These topics are all tightly related.

- Basic propositional and first-order logic.
- Basic sets, functions, relations, graphs, etc.
- Proof techniques
- Regular languages and finite automata
- Context-free languages and push-down automata.
- Turing machines and undecidability.
- NP-completeness

5 / 33

Skills

- Defining concepts precisely (using logic).
- Doing proofs.
- Solving problems (mathematically and computationally).
- Proving that problems are impossible or “hard”

6 / 33

Course organization

Detailed course information is on the web page. Please read it carefully.

- Web page is (or soon will be): <http://cs103.stanford.edu>
Check it frequently for announcements, scheduling changes, etc. Homeworks and handouts will often appear there with no paper copies of homeworks.
- Textbooks: Chapter 1 of “Discrete Mathematics and its Applications” by Kenneth R. Rosen and “Introduction to the Theory of Computation” by Michael Sipser (the whole thing). These should be in the bookstore.
- Grading: 60% Homeworks (9 of them), 15% midterm (1), 25% final (1). The lowest homework grade will be dropped. Otherwise, no lates without a dire excuse.

7 / 33

Course Organization, cont.

- Problem sessions: We'll have weekly problems sessions to work problems similar to the homeworks. Problem sessions are optional but highly recommended. Details to be announced.
- ~~Wednesday~~ Friday is homework day. The midterm will be on ~~Monday, February 13,~~ Thursday, February 16, in the evening.
- CS106A is a prerequisite because concepts from programming will make some of this class easier to understand (we hope). There is no programming required in this class.

8 / 33

Propositional Logic

This lecture and the next will be devoted to *propositional logic*.

Propositional logic is simple but extremely important in Computer Science

1. It is the basis for day-to-day reasoning (in programming, LSATs, etc.)
2. It is the theory behind digital circuits.
3. Many problems can be translated into propositional logic.
4. It is an important part of more complex logic (such as *first-order logic*, also called *predicate logic*, which we'll discuss shortly.)

In computer science, "Boolean" is often used interchangeably with "propositional."

The most famous unsolved problem in theoretical computer science is "How hard is it to decide whether a propositional logic formula is satisfiable?" (this is the most basic *NP-complete* problem).

9 / 33

Syntax

Syntax is the "way that linguistic elements are put together." It is what is defined by grammar.

It is a very important concept in linguistics, logic and computer science.

Rules of syntax are important in computer science because it requires precise definitions. This is important in theoretical computer science (especially in this class) but even more important in practical software.

We now encounter a syntactic problem: How to define the legal propositional logic formulas?

10 / 33

Syntax

We could write a context-free grammar for propositional logic formulas, but we haven't learned about context free grammars (we'll get to it in a few weeks). So I'll just explain the rules:

- We assume there are propositional variables, which will often look like P , Q , x_i , etc. Instead of defining them precisely, I'll just say you will know them when you see them (if you're not sure, please ask).
Any propositional variable by itself is a propositional formula.
- If α is a propositional formula, $(\neg\alpha)$ is also a propositional formula.
- If α and β are propositional formulas, the following are also propositional formulas: $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta)$, and $(\alpha \leftrightarrow \beta)$.

\neg , \wedge , \vee , \rightarrow , \leftrightarrow are called *propositional connectives*.

For example $(P \vee (Q \wedge (\neg(R \rightarrow P))))$ is a propositional formula.

11 / 33

Parentheses and precedence

Every kind of formula has parentheses around it to eliminate all ambiguity about grouping.

But we learn conventions about dropping parentheses in arithmetic in grade school:

- $-1 + 2$ is $((-1) + 2)$
- $1 + 2 \times 3$ is $(1 + (2 \times 3))$
- *etc.*

\times is "stickier" than $+$, and "unary minus" is stickier than binary operators (the approved term for "stickier" is "has higher precedence").

We treat \wedge like \times and \vee like $+$ and \neg like unary $-$. \wedge will usually be stickier than \rightarrow and \leftrightarrow , but I'll try not to depend on that.

So $(P \vee (Q \wedge (\neg(R \rightarrow P))))$ could be $P \vee Q \wedge \neg(R \rightarrow P)$

12 / 33

Abstract syntax

Precedence is very important when you write programs to read and write notation like this, but only matters now so that we can understand the formulas without writing a zillion parentheses.

For most of our purposes, we don't care about the notation so much as the nested structure of the formulas.

That nested structure can be captured in an *abstract syntax tree*. Here is the tree for $P \vee Q \wedge \neg(R \rightarrow P)$:

13 / 33

Abstract syntax, cont.

This has everything we really care about for understanding and doing things to the formula. The syntax is just to make the formulas easy to write on the screen/paper.

Computer programs that process programming languages usually convert everything to abstract syntax trees as quickly as can, and only keep the input syntax around for error messages.

14 / 33

Abstract syntax definitions

So, in the future, I'll just define the structure of the formula, and we'll use the parentheses when necessary to read and write it.

- If α is a propositional formula, $(\neg\alpha)$ is also a propositional formula.
- If α is a propositional formula, $\neg\alpha$ is also a propositional formula.
- If α and β are propositional formulas, the following are also propositional formulas: $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, and $\alpha \leftrightarrow \beta$.

This resembles a recursive function, with the propositional variables as the "base case."

The five propositional connectives are recursive, because they are defined in terms of the propositional formulas α and β .

15 / 33

Semantics

Syntax just tells us how the notation fits together, but it doesn't tell us what it means.

"Semantics" is the word in computer science for the precise definition of the meaning of a notation (and other things).

All attempts to define "meaning" or "semantics" end up being circular. I only understand and know how to explain it by example.

For now, we'll define the meaning of propositional formulas using *truth tables*.

16 / 33

Truth tables

A truth table is a way of exhaustively cataloging all the possible scenarios for truth functions.

Columns are sentences (atomic sentences to the left). Rows give all possible *truth assignments* to the atomic sentences.

P	$\neg P$
T	F
F	T

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

17 / 33

Complex truth tables

Truth tables can be written for more complex sentences. Associate a value with each connective: $\neg R$, $Q \wedge \neg R$, $P \vee (Q \wedge \neg R)$.

P	Q	R	$P \vee (Q \wedge \neg R)$
T	T	T	F
T	T	F	T
T	F	T	F
T	F	F	T
F	T	T	F
F	T	F	T
F	F	T	F
F	F	F	F

18 / 33

Tautology

A *tautology* is a sentence that is valid based only on truth-functional reasoning.

I.e., the entire column for the sentence in the truth table is "T."

P	$P \vee \neg P$
T	T
F	T

$P \vee \neg P$ is called the "law of the excluded middle."

A sentence and its negation cannot both be false.

19 / 33

Logical equivalence

Two sentences are *logically equivalent* if they always have the same truth value, based only on the rules of logic.

I will write $\alpha \equiv \beta$ when α and β are logically equivalent.

For practical purposes, logical equivalences is equality on logical formulas.

P	Q	$\neg(P \vee Q)$	$\neg P \wedge \neg Q$
T	T	F	F
T	F	F	F
F	T	F	F
F	F	T	T

$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$ is one of *DeMorgan's laws*

$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ is the other *DeMorgan's law*.

20 / 33

Tautological consequence

Proofs usually start with a set of assumptions, called *premises* and follow a set of logical rules to reach a conclusion.

In propositional logic, all premises and conclusions are propositional formulas.

A sentence is a *logical consequence* of premises if it is true whenever the premises are, based only on the rules of logic.

For propositional formulas, can also be read from the truth table.

Example: $P \vee Q$ is a tautological consequence of P .

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

21 / 33

Boolean Identities

It is often useful to think of Boolean connectives as algebraic operations, with accompanying laws:

Identity	Name
$\neg\neg P \equiv P$	double negation
$P \wedge \mathbf{T} \equiv P$ $P \vee \mathbf{F} \equiv P$	identity
$P \wedge \mathbf{F} \equiv \mathbf{F}$ $P \vee \mathbf{T} \equiv \mathbf{T}$	domination
$P \wedge \neg P \equiv \mathbf{F}$ $P \vee \neg P \equiv \mathbf{T}$	inverse laws
$P \wedge P \equiv P$ $P \vee P \equiv P$	idempotence
$P \wedge Q \equiv Q \wedge P$ $P \vee Q \equiv Q \vee P$	commutativity

22 / 33

More Identities Involving \wedge , \vee , and \neg

Identity	Name
$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$ $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$	associativity
$(P \wedge Q) \vee R \equiv (P \vee R) \wedge (Q \vee R)$ $(P \vee Q) \wedge R \equiv (P \wedge R) \vee (Q \wedge R)$	distributivity
$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$	de Morgan's laws

Note that properties come in pairs.

Duality: If you swap \wedge and \vee and \mathbf{T} and \mathbf{F} in an identity, you get another identity.

23 / 33

Substitution of Logical Equivalents

Logical equivalence is similar to equality.

If $S(P)$ is a sentence containing another sentence P , and $S(Q)$ is the same sentence except with Q replacing P , and $P \equiv Q$, then $S(P) \equiv S(Q)$.

This principle allows algebraic manipulation of logical formulas:

$$\begin{aligned}
 P \wedge (\neg P \vee Q) &\equiv (P \wedge \neg P) \vee (P \wedge Q) && \text{Distributivity of } \wedge \text{ over } \vee \\
 &\equiv \mathbf{F} \vee (P \wedge Q) && \text{Inverse} \\
 &\equiv P \wedge Q && \text{Identity}
 \end{aligned}$$

24 / 33

Implication

The implication $P \rightarrow Q$ formalizes statements like “If P holds, then Q must hold.”

\rightarrow is a Boolean operator, like \wedge and \vee .

The truth table is:

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

WARNING: The last line puzzles a lot of students. “How can $P \rightarrow Q$ be true if P is false?”

If this bothers you, do whatever it takes to become comfortable with it (write lots of examples, come argue with us) – or risk being lost forever.

25 / 33

Implication Does Not Increase Expressiveness

Implication doesn't enable you to say anything you couldn't say before.

$$P \rightarrow Q \equiv \neg P \vee Q$$

P	Q	$\neg P$	\vee	Q
T	T	F	T	
T	F	F	F	
F	T	T	T	
F	F	T	T	

... but sometimes they say it better.

Biggest advantage: Makes proofs more natural.

26 / 33

Translation from English

Many people find \rightarrow more confusing than the other Boolean connectives.

Basic translations:

“If P then Q ” “If P , Q ”
“ P implies Q ” “ P , so Q ”
“If P , therefore Q .” etc.

all mean $P \rightarrow Q$.

27 / 33

Implication vs. Causality

It is tempting to think of $P \rightarrow Q$ as “ P caused Q .” **Don't.**

“Fred is sneezing, so he must have a cold.” — sneezing is the effect of the cold, not the cause.

“Fred is coughing, so he must be sneezing.” — sneezing and coughing have a common cause.

However,

“Fred has a cold, so he must be sneezing.” might also be a valid statement.

28 / 33

Necessary and Sufficient Conditions

“P is sufficient for Q” means $P \rightarrow Q$.

“P is necessary for Q” means $Q \rightarrow P$.

“P if Q” means $Q \rightarrow P$.

“P only if Q” means $P \rightarrow Q$.

29 / 33

Variations on implications

Given an implication $p \rightarrow q$, there are several related implications:

implication	name
$q \rightarrow p$	converse
$\neg p \rightarrow \neg q$	inverse
$\neg q \rightarrow \neg p$	contrapositive

Important identities:

$p \rightarrow q \equiv \neg q \rightarrow \neg p$ (statement \equiv contrapositive)

$q \rightarrow p \equiv \neg p \rightarrow \neg q$ (converse \equiv inverse)

30 / 33

The Fallacy of Converse

It is a common error in reasoning to mistake an implication and its converse/inverse.

Statement: “If I have a fever, I am ill.”

Inverse: “If I do not have a fever, I am not ill.”

Contrapositive: “If I am not ill, I do not have a fever.”

Converse: “If I am ill, I have a fever.”

31 / 33

Biconditional

$P \leftrightarrow Q$ means $P \rightarrow Q$ and $Q \rightarrow P$.

It is true whenever P and Q have the same truth value.

Here is the truth table:

P	Q	$P \leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

32 / 33

Biconditionals in English

All of the following are standard ways to indicate biconditionals in English:

“P if and only if Q”

“P iff Q.”

“P exactly when Q”

“P just in case Q”

“P is necessary and sufficient for Q” etc.