CS 245
Lecture Notes
Winter 2024

Collin Roberts

January 18, 2024

Contents

1 Lecture 01 - Introduction to CS 245

Outline

1. Introduction to CS 245
2. Introduction to Propositional Logic
   (a) Applications of Logic in Computer Science
   (b) Why We Need Formal Languages
3. Propositions
4. Connectives
5. Translations Between English and Propositional Logic
6. Logical Arguments

1.1 Q & A

1. Will you lecture from slides, or on the blackboard / document camera?
   A: I intend to lecture using prepared slides. Whenever I need to go off-script, I will write on the document camera, and add those notes to the prepared slides afterward. My Lecture Notes (on which my prepared slides will be based) are posted on LEARN.
2. How will tutorials work?
   A: Instructors will prepare material for the IAs to present during the hour. These materials will be posted on LEARN, at the end of the day on Friday.
3. Will all presentation / evaluation be "writable by hand"?
   A: Yes. CS 245 is very much like a MATH course. We will not code; instead we will work with logical objects that will help us to **think about coding**.
4. Do most students pass CS 245?
   A: Yes! The instructors are not seeking to "weed out" anyone. However we must evaluate based on our course syllabus.
5. When will Crowdmark Assignments and Marked Quizzes be due?
   A: 11:59 PM.
6. Will any Marked Quizzes be in-person?
   A: No. All Marked Quizzes will be delivered on LEARN.
7. What is Crowdmark?
   A: Crowdmark is a website, on which:
   (a) instructors create / distribute assignments,
   (b) students submit their answers to assignment questions (.pdf format is preferred; graphic formats are also allowed),
   (c) TAs grade the student answers, and
   (d) instructors release marked assignments back to the students.
8. Is the historical material in Logic01 examinable?
   A: No. The history is for interest. The **translation** material from Logic01 will appear on assignments and exams.
9. Can we use the smartphone version of the iClicker software?

A: Yes!

1.2 Introduction to CS 245

1. **Q:** How will the course run?
   A: Refer to the course website for high-level answers:
   `https://student.cs.uwaterloo.ca/~cs245/F23/`
2. **Q:** How will iClickers work?
   A: Bring your iClicker to class. Use your iClicker to answer occasional questions which will be presented during the lectures. There are no marks attached to iClicker use - it is 100% optional.
3. **Q:** What homework do I have before the next lecture?
   A: Read the Logic01 and Logic02 slide decks, and come to the next lecture prepared to ask any questions that you have about them.

Setting Course Expectations:

1. You should not expect that CS 245 will (directly) improve your coding skills.
2. You should instead expect that CS 245 will make you a more effective thinker about coding.
3. This will ultimately improve your coding, once you have assimilated CS 245 into your thinking.

1.3 Introduction to Propositional Logic

1.3.1 Applications of Logic in Computer Science

Topics in CS For Which Logic Is Relevant

1. SAT-solvers (Propositional Logic)
2. Database Analysis (First-Order Logic)
3. Properties of the Natural Numbers - Peano Axioms
4. Program Verification
5. Decidability and Undecidability
6. Definability and Undefinability
7. Provability and Unprovability
8. Artificial Intelligence
9. and many more ...

1.3.2 Why We Need Formal Languages

Motivation: We are often tempted to rely on our own understanding and intuition exclusively. However sometimes an apparently reasonable **decision problem** (Definition 1.3.1) has subtleties.

Definition 1.3.1. *A **decision problem** is a problem which calls for an answer of either yes* (1) *or no* (0)*, given some input.*

Definition 1.3.2. *A **paradox** is a declarative statement that*

1. *cannot be true, and*
2. *cannot be false.*

Examples:

1. "This sentence is false."
2. **The Barber Paradox** (Temporarily assume a universe of only men.)

   There is a barber who is said to shave each man, if and only if that man does not shave himself. **Q:** Does the barber shave himself?
   - Then if the barber shaves himself, then it is because he does not shave himself, and in turn this is because he does shave himself.
   - Also, if the barber does not shaves himself, then it is because he shaves himself.
   - "Does the barber shave himself?" is unanswerable.
3. **Barey's Paradox**

   Consider the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

   Remarks:

   (a) Unlike what you were likely told in MATH 135, in CS 245 we take 0 to be the first Natural number.

   Definition 1.3.3. *We say that a Natural Number,* n*, has a **compact definition** if there is an English sentence of at most 200 charcters that uniquely defines the number* n.

   Examples:

   (a) "n is 3."
   (b) "n is the difference of 10 and 7."
   (c) "n is one million."
   (d) "n is one million to the power of one million."
   (e) "n is the number of cells in my body."
   (f) "n is the number of grains of sand on a California beach."
   (g) The sentence "n is even." identifies no Natural number.
   (h) The sentence "Fruit flies like a banana." identifies no Natural number.

   Let B be the set of all Natural numbers that have a compact definition.

   Q: Is B a finite set? (CQ 2)

   A: Yes: $|B| \leq 40^{200} < \infty$ (where we get 40 from 26 letters, 10 digits and a few punctuation marks). There are only finitely many compact English descriptions.

   Since B is finite and $\mathbb{N}$ is infinite, we may consider the first Natural number, x, which does not have a compact definition.

   Q: Is $x \in B$?

   A:
   - If $x \in B$, then we have a contradiction, since by construction x is the first Natural Number such that $x \notin B$.

- If $x \notin B$, then by construction there exists a way to define x by an English sentence of length ≤ 200 (the preceding description of x constitutes a compact definition). So $x \in B$.

Then this shows the paradox.

Morals:

1. The presence of such paradoxes, arising from descriptions in some natural language (English in these cases), tells us that we will need formal languages to study logic carefully.
2. We will see that paradoxes cannot occur in formal logic: every statement (formula) in formal logic is either true (1) or false (0) (and not both) in some context.

1.4 Propositions

Definition 1.4.1. *A **proposition** is a declarative sentence that is either true or false, in some context.*

Examples:

1. If I feed my fish, and I change my fish's tank filter, then my fish will be healthy. (compound, not simple)

Definition 1.4.2. *An **atomic (simple) proposition** is a proposition that cannot be broken down into smaller propositions. A proposition that is not atomic (simple) is called **compound**.*

In particular, the presence of any **connective** indicates that the proposition is compound.

Examples:

1. My fish will be healthy. (atomic a.k.a. simple)

Remarks:

1. We assign **proposition symbols** like p, q and r to represent simple propositions when we translate from English into Propositional Logic.

1.5 Connectives

You likely met these connectives and their truth tables in MATH 135.

1. unary
   (a) $\neg$ (negation)
2. binary
   (a) $\wedge$ (conjunction)
   (b) $\vee$ (disjunction)
   (c) $\rightarrow$ (implication)
   Truth Table:

p	q	$(p \rightarrow q)$
1	1	1
1	0	0
0	1	1
0	0	1

(d) $\leftrightarrow$ (equivalence)
Truth Table:

p	q	$(p \leftrightarrow q)$
1	1	1
1	0	0
0	1	0
0	0	1

1.6 Translations Between English and Propositional Logic

First Example, As We Have Limited Time Remaining: the compound proposition from above.

1. If I feed my fish, and I change my fish's tank filter, then my fish will be healthy.
   $((p \land q) \rightarrow r)$, where
   - p: I feed my fish.
   - q: I change my fish's tank filter.
   - r: My fish will be healthy.

Remarks:

1. Choose your atomic propositions (which will correspond to proposition symbols) to be positive statements (i.e. without any embedded negations). E.g. translate "I did not change my fish's tank filter" as $(\neg q)$, where q means "I changed my fish's tank filter".
2. Negated statements are compound, not atomic.
3. Assemble atomic propositions, using the appropriate connectives to express the provided English sentence.
4. Implications are particularly interesting here.
5. You will get some more practice at this on next week's tutorial, and on the first marked quiz and the first assignment.

Translate the following examples from English into Formulas of Propositional Logic:

1. She is clever and hard working.
   $(p \land q)$, where
   - p: She is clever.
   - q: She is hard working.
2. He is clever but not hard working.
   $(p \land (\neg q))$, where
   - p: He is clever.

    - q: He is hard working.
3. If it rains, then he will be at home; otherwise he will go to the market or he will go to school.
   $((p \to q) \land ((\neg p) \to (r \lor s)))$, where
    - p: It rains.
    - q: He will be at home.
    - r: He will go to the market.
    - s: He will go to school.

   Note that we need $\land$ and not $\lor$ as the last binary connective in this formula!
4. The sum of two integers is even if and only if both integers are even or both integers are odd.
   $$(p \leftrightarrow ((q_1 \land q_2) \lor (r_1 \land r_2))),$$
   where
    - p: The sum of the two integers is even.
    - q_1: The first integer is even.
    - q_2: The second integer is even.
    - r_1: The first integer is odd.
    - r_2: The second integer is odd.

Remarks:

1. We could replace r_1 by $(\neg q_1)$, and r_2 by $(\neg q_2)$, but this would take us further away from the original English statement.

Translate the following examples from Formulas of Propositional Logic into English: Use the atoms:

- p: Today is Sunday.
- q: I do homework.
- r: I watch TV.

1. $(q \leftrightarrow (\neg p))$
    - “I do homework if and only if today is not Sunday.”, or
    - “I do homework every day except Sunday.” (less direct, but a bit more natural).
2. $(q \lor r)$
    - “I do homework or I watch TV.”
    - “I do homework unless I watch TV.” (N.B. “Unless” is messy in English - sometimes it indicates an inclusive or the way we have translated it here, and other times it indicates an exclusive or)
3. $(p \to r)$
    - “On Sundays I watch TV.”
    - “If today is Sunday then I watch TV.”, or
    - “I watch TV if today is Sunday.”, or
    - “Today is Sunday only if I watch TV.”

1.7 Logical Arguments

Given the premises:

1. If I study before my mid-term, and I get 8 hours' sleep before my mid-term, then I will pass my mid-term.
2. I studied before my mid-term.
3. I did not pass my mid-term.

We may conclude:

1. Therefore I must not have gotten 8 hours' sleep before my mid-term.

Q: Is this argument "convincing" to you?

A: I find this argument "convincing". I will justify this assertion, soon.

Shorthand Used In Slides: Separate the premises and the conclusion using a horizontal line.

2 Lecture 02 - Structural Induction

Outline

1. Inductive Definitions of Sets
2. Structural Induction
3. Introduction to the Syntax of Propositional Logic
4. Unique Readability of Propositional Formulas
5. Parse Trees
6. Precedence Rules for Propositional Connectives
   (a) Generation Sequences for Propositional Formulas
7. Structural Induction - More Examples

2.1 Inductive Definitions of Sets

1. In CS 245, **structural induction** will be a theme of the course.
2. In this lecture we give the setup which will permit us to carry out structural induction correctly in every situation.
3. The first example of an inductively defined set, for which we can employ structural induction, is the set of **propositional formulas**, Form($\mathcal{L}^p$).

Remark: You may find this approach is too abstract for your taste, especially so early in the course. While I understand this reaction, I assure you that if we invest the time to understand the general setup, then we will reap the rewards throughout the rest of the course.

What techniques do we have for declaring sets?

1. The empty set, $\varnothing$, is a set.
2. via some property of interest, e.g.
   (a) Even Natural Numbers $\{n \in \mathbb{N} \mid n \text{ is even}\}$, or $\{n \in \mathbb{N} \mid 2|n\}$.
3. The **power set of a set** S, denoted P(S) (i.e. the set of all subsets of S) is a set.
4. Explicitly list all elements of the set. For example, $\{3, 6, 7\}$. Drawback: we cannot do this for infinite sets.
5. Inductively, for example, the set of all my blood relatives. Core set = {me} Operations = { daughter of, son of, brother of, sister of, mother of, father of }

Formally: **Inductive definition of a set**: 3 ingredients:

1. a **Universe** of all elements denoted by X (e.g. $X = \mathbb{R}$),
2. a **Core set** denoted by A (for "atoms"), with $A \subseteq X$, (e.g. $A = \{0\}$) and
3. a set of **operations (functions)** $X \to X$, denoted by F.
   The elements of F are **functions**, f, each having some **arity**, $k \geq 1$. I.e. k is the number or arguments that f takes.
   E.g. $F = \{s(x) = x + 1\}$, i.e. s is the **successor function**.
   Different functions have different arities - there is not a single arity k which applies to all of the fs.

Definition Given any subset $Y \subseteq X$, and any set F of operations (functions $f: X^k \to X$ for any $k \geq 1$), Y is **closed under** F if, for every $f \in F$, (say f is a k-ary function) and every $y_1, \dots, y_k \in Y, f(y_1, \dots, y_k) \in Y$.

Examples:

1. Letting $Y = \varnothing$, the above definition is vaccuously satisfied.
2. Let Y be the set of even Natural numbers. Let F be the set of addition, and multiplication. Then we know Y is closed under F.
   (a) Y is not closed if we include subtraction in F. (Time permitting, give an example to demonstrate this.)

Definition Y is a **minimal set with respect to a property** R if

1. Y satisfies R, and
2. for every set Z that satisfies R, $Y \subseteq Z$.

Now we have the formal definition.

Definition $I(X, A, F)$ = The minimal subset of X that

1. contains A, and
2. is closed under the operations in F.

Motivating Example The set of Natural numbers:

$$\mathbb{N} = I\left(\mathbb{R}, \{0\}, \left\{\underbrace{s(x) = x + 1}_{\text{successor function}}\right\}\right).$$

Exercise: Prove it. One containment is easy. The other containment can be proved using POMI, the way you would have in MATH 135.

Example: $I(X, A, F)$ is the set of polynomials in variable z over a field K (say the field of real numbers), with:

$$\begin{aligned}
X &= \textbf{the set of all strings that can be written using} \\
&\quad \mathbb{R} \cup \{+, \cdot, z\} \\
&\quad \text{e.g. } + z \cdot 5 + +18 \cdot 0 \cdot 5 \text{ is garbage, while} \\
&\quad z \cdot z + 1 \text{ is a polynomial in z over } \mathbb{R} \\
A &= \{z\} \cup \mathbb{R} \\
F &= \{+, \cdot\}
\end{aligned}$$

2.2 Structural Induction

Motivation: Here we explain our strategy to use **structural induction** to prove a desriable property R holds for every element of an inductively defined set, $I(X, A, F)$.

1. Prove that $R(a)$ holds for every a in the core set A (the **base case**).
2. Prove that, for every k-ary $f \in F$ (for any $k \geq 1$), and any $y_1, \dots, y_k \in X$ such that $R(y_1), \dots, R(y_k)$ all hold, we also have that $R(f(y_1, \dots, y_k))$ holds (the **inductive case**).

Remarks:

1. Our first example using this technique will be to prove the **unique readability of propositional formulas in** $\text{Form}(\mathcal{L}^p)$.
2. Recalling that

$$\mathbb{N} = I\left(\mathbb{R}, \{0\}, \left\{ \underbrace{s(x) = x + 1}_{\text{successor function}} \right\}\right).$$

   we see that structural induction, as above, reduces to the familiar POMI (strong perhaps, but there is no real difference between strong and regular induction anyway).

2.3 Introduction to the Syntax of Propositional Logic

Using our setup above, we will define the set $\text{Form}(\mathcal{L}^p)$ inductively. We need these ingredients:

1. Let P be a set of **proposition symbols**, e.g. $P = \{p, q, r\}$. These will be our **atoms**.
2. Let C be the set of **propositional connectives**, namely $C = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.
3. Let X be the set of all strings that can be written using $P \cup C \cup \{(,)\}$.
4. Let F be the set containing the following functions defined on X:
   (a) $\text{neg}(x) = (\neg x)$ (unary)
   (b) $\text{and}(x, y) = (x \wedge y)$ (binary)
   (c) $\text{or}(x, y) = (x \vee y)$ (binary)
   (d) $\text{impl}(x, y) = (x \rightarrow y)$ (binary)
   (e) $\text{equiv}(x, y) = (x \leftrightarrow y)$ (binary)

Definition 2.3.1. *Using the notation above, the set* $\text{Form}(\mathcal{L}^p)$ *of* ***propositional formulas over*** F *is defined inductively, as*

$$I(X, P, F)$$

See also Definitions 2.2.1, 2.2.2 and 2.2.3 in the text.

Examples:

1. Each of the following is a in $\text{Form}(\mathcal{L}^p)$ over $P = \{p, q, r\}$.
   (a) p
   (b) $(\neg q)$
   (c) $(p \wedge q)$
   (d) $((p \wedge q) \wedge p)$
   (e) $(p \rightarrow (q \vee r))$
2. This string is not in $\text{Form}(\mathcal{L}^p)$.

$$)p \rightarrow \wedge \leftrightarrow rq()$$

How would you prove this fact? We will answer this soon.

Questions from the Class

1. Is it syntactically incorrect to add extra parentheses?
   A: Yes! We will be very strict to start; we will relax this later.
2. What is a proper initial segment of an expression?
   A: Denote our expression by s. A **proper initial segment** of s is a non-empty expression, X, such that S = XY, for some non-empty expression, Y.
   E.g. if s = "carrot", then X = "car" is a proper initial segment of s (with Y = "rot"). The set of proper initial segments of s is { "c", "ca", "car", "carr", "carro" }.

- CQs, as appropriate

Problems:

1. **Problem:** Prove by structural induction that every propositional formula in Form($\mathcal{L}^p$) contains at least one proposition symbol.
   Solution: Exercise.
2. **Problem:** Prove that every propositional formula in Form($\mathcal{L}^p$), A, has the same number of '(' and ')' symbols.
   Solution: Let A be any propositional formula in Form($\mathcal{L}^p$). The proof is by structural induction on A.
   Let R(A) be the property
   "A has equally many '(' and ')' symbols."
   Base Case (A is p for some proposition symbol, p): Then A has zero '(' and zero ')' symbols. Therefore R(A) holds in the base case.
   Induction Case: Define the notation
   - $\ell(A)$ denotes the number of '(' symbols in A.
   - r(A) denotes the number of ')' symbols in A.

   We have these subcases.
   - A is $(\neg B)$:
     - The inductive hypothesis is R(B), i.e. that $\ell(B) = r(B)$.
     - Then we have

$$\begin{aligned}\ell\big((\neg B)\big) &= 1 + \ell(B) \text{ (inspection)}\\ &= 1 + r(B) \text{ (induction hypothesis: R(B))}\\ &= r\big((\neg B)\big) \text{ (inspection)}\end{aligned}$$

   - A is $(B \star C)$, for some formulas B and C and some binary connective $\star$:
     - "Without loss of generality" clearly applies to all the **binary** connectives.
     - The inductive hypothesis is R(B) and R(C), i.e. that $\ell(B) = r(B)$ and $\ell(C) = r(C)$.

- Then we have

$$\begin{aligned}\ell\big((B \star C)\big) &= 1 + \ell(B) + \ell(C) \text{ (inspection)}\\ &= 1 + r(B) + r(C) \text{ (induction hypothesis: R(B) and R(C))}\\ &= r\big((B \star C)\big) \text{ (inspection)}\end{aligned}$$

Remarks:

1. **Q:** How could we use this result to demonstrate that our non-examples of formulas from Lecture 02 were correct?

2.4 Unique Readability of Propositional formulas

Theorem (Unique Readability of Propositional Formulas) 2.4.1. *Every propositional formula in* Form($\mathcal{L}^p$)*,* A*, is exactly one of an atom,* $(\neg B)$*,* $(B \wedge C)$*,* $(B \vee C)$*, or* $(B \to C)$*; and in each case* A *is of that form in exactly one way.*

Property R(A): A formula A has property R(A) iff it satisfies all three of the following.

1: The first symbol of A is either '(' or a variable.
2: A has an equal number of '(' and ')', and each proper initial segment of A has more '(' than ')'.
3: A has a unique construction as a formula.

Remarks:

1. (A **proper initial segment** of A is a non-empty expression X such that A is XY for some non-empty expression Y.)
2. We prove property R(A) for all formulas A, by Structural Induction on A.
3. We only need Property 3 in the end. Including Properties 1 and 2 gives our inductive hypothesis more strength when needed.

Base (A is p, for some proposition symbol, p):

- 1: trivial.
- 2:
  - first part: trivial.
  - second part: vaccuous (since A has no proper initial segments in this case).
- 3: trivial.

The induction step has two sub-cases.

1. A is $(\neg B)$, for some propositional formula B in Form($\mathcal{L}^p$):
   The inductive hypothesis is that the formula B has property R.
   - **1:** By construction, $(\neg B)$ has Property 1, since it begins with '('.
   - **2:** Since B has an equal number of left and right parentheses, therefore so does $(\neg B)$.

For the second part of Property 2, we check these subcases for every possible proper initial segment, x, of A.

(a) x is “(”: Then x has 1 “(” symbol and 0 “)” symbols.
(b) x is “(¬”: Then x has 1 “(” symbol and 0 “)” symbols.
(c) x is “(¬”z, for some proper initial segment, z, of B: Since z has more “(” than “)” symbols, therefore so does x.
(d) x is “(¬”B: Since B equally many “(” and “)” symbols, therefore x has more “(” than “)” symbols.

In every case, x has more “(” than “)” symbols. Hence (¬B) has Property 2.

- **3:** Because B has Property 3, therefore by construction so does (¬B).

This shows that A has Property R.

2. A is (B ★ C), for some propositional formulas B, C in Form($\mathcal{L}^p$) and some binary connective ★:
The inductive hypothesis is that each formula B and C has property R.
   - **1:** Clearly, A has property 1.
   - **2**: Since B and C have equal numbers of left and right parentheses, therefore so does (B ★ C).
     For the second part of Property 2, we check these subcases for every possible proper initial segment, x, of A.
     (a) x is “(”: Then x has 1 “(‘ ’symbol and 0 “)” symbols.
     (b) x is “(”z, for some proper initial segment, z, of B: Since z has more “(‘ than “)” symbols, therefore so does x.
     (c) x is “(”B: Since B equally many “(‘ and “)” symbols, therefore x has more “(‘ than “)” symbols.
     (d) x is “(”B★: Since B equally many “(‘ and “)” symbols, therefore x has more “(‘ than “)” symbols.
     (e) x is “(”B ★ z, for some proper initial segment, z, of C: Since B equally many “(‘ and “)” symbols, and z has more “(‘than “)” symbols, therefore x has more “(‘ than “)” symbols.
     (f) x is “(”B ★ C: Since B and C have equally many “(‘ and “)” symbols, therefore x has more “(‘ than “)” symbols.
     In every case, x has more “(‘ than “)” symbols. Hence (¬B) has Property 2.
   - **3:** We must show
     If A is (B′ ★′ C′) for **formulas** B′ and C′, then B = B′, ★ = ★′ and C = C′.
     If ‖B′‖ = ‖B‖, then B′ = B (both start at the second symbol of A). Thus also ★′ = ★ and C′ = C, as required. So we are finished if we can prove that ‖B′‖ = ‖B‖.
     – Towards a contradiction, assume that either B′ is a proper initial segment of B or B is a proper initial segment of B′.
     – The inductive hypothesis applies to B and B′. In particular, each has property 2.
     – Therefore B and B′ have a balanced number of “(” and “)” characters, by property 2.

  - But if B is a proper initial segment of B′, then B has more "(" than ")" characters, also by property 2. This is a contradiction.
  - We reach a similar contradiction if we assume that B′ is a proper initial segment of B. Thus neither B nor B′ can be a proper initial segment of the other.

  Therefore A has a unique derivation; it has Property 3, as required.

By the principle of structural induction, every Propostional formula has Properties 1, 2 and 3.

This shows that Unique Readability (Property 3) holds for every Propositional Formula.

This is what we set out to prove.

Explanation of the Connection Between B **and** B′**:**

- In the past, some students have been confused about why it holds that either $B = B'$, B is a proper initial segment of B′ or vice versa.
- The key fact to remember here is that both B and B′ arose from a choice of how to decompose the given formula A. In detail,

$$(B \star C) = A = (B' \star' C').$$

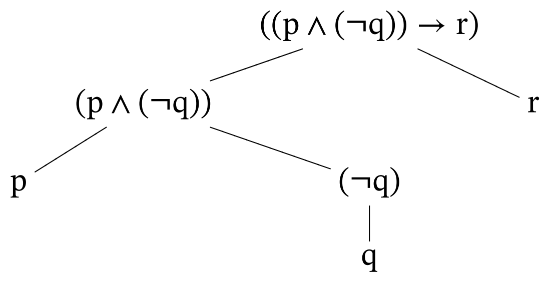
- Because we actually mean **equality** of formulas (i.e. symbol-by-symbol equality of the expressions constituting the formulas) here, we now see that the above fact about B and B′ must hold.

Why We Care About Unique Readability:

1. For the rules of propositional logic semantics to be well-defined, it is crucial that every propositional formula can be parsed in only one way.

2.5 Parse Trees

- A **parse tree** for a formula represents the formation sequence as a tree with its root at the top, and each internal node corresponding with an application of one of the formation rules.
- For example, this is a parse tree for the formula A which is $((p \land (\neg q)) \to r)$:



- This parse tree has height 3.
- See also p24 of the text.

- Another typical question would be to provide a parse tree, and to ask for the formula that the tree represents.

Remarks:

1. If we follow our construction rules to the letter, then $(p \wedge q \wedge r)$ is **not** in Form($\mathcal{L}^p$).
2. To put this formula into Form($\mathcal{L}^p$), we would have to write
   (a) $((p \wedge q) \wedge r)$ or
   (b) $(p \wedge (q \wedge r))$.
3. These are non equal as (syntactic) formulas.
4. These formulas are **tautologically equivalent**, i.e. they behave the same way in every semantic context (equivalently, they have the same truth tables).

2.6 Precedence Rules for Propositional Connectives

Order of Precedence for Propositional Connectives: As on p 33 of the text, we may omit some parentheses once we agree on an order of precedence for the connectives. The order is

1. $\neg$
2. $\wedge$
3. $\vee$
4. $\rightarrow$
5. $\leftrightarrow$

Examples: On each row of the following table, we give a formula with some (or all) parentheses omitted, followed by the formula in Form($\mathcal{L}^p$) that results from adding parentheses according to the above precedence rules.

some (or all) parentheses omitted	in Form($\mathcal{L}^p$)
$p \vee q \wedge r$	$(p \vee (q \wedge r))$
$\neg p \vee q$	$((\neg p) \vee q)$
$p \rightarrow q \wedge r$	$(p \rightarrow (q \wedge r))$
$p \rightarrow q \leftrightarrow r$	$((p \rightarrow q) \leftrightarrow r)$
$p \wedge q \rightarrow \neg r$	$(p \wedge q) \rightarrow (\neg r))$

Remarks:

1. We will say "propositional formula in Form($\mathcal{L}^p$)" to refer to a formula which is syntactially correct according to the earlier definition.
2. We will say "propositional formula", to refer to a formula which may be in Form($\mathcal{L}^p$), or may have some parentheses omitted, where the correct formula in Form($\mathcal{L}^p$) could be recovered according to the precedence rules.

2.6.1 Generation Sequences for Propositional Formulas

Examples:

1. Give a generation sequence for each of the following propositional formulas in Form($\mathcal{L}^p$) over $P = \{p, q, r\}$.
   (a) p
      Solution:
      i. p is in the core set.
   (b) q
      Solution:
      i. q is in the core set.
   (c) $(p \wedge q)$
      Solution:
      i. p is in the core set.
      ii. q is in the core set.
      iii. Applying and to lines 1(c)i and 1(c)ii yields $(p \wedge q)$.
   (d) $(p \to (q \vee r))$
      Solution:
      i. p is in the core set.
      ii. q is in the core set.
      iii. r is in the core set.
      iv. Applying or to lines 1(d)ii and 1(d)iii yields $(q \vee r)$.
      v. Applying impl to lines 1(d)i and 1(d)iv yields $(p \to (q \vee r))$.

2.7 Structural Induction - More Examples

Examples:

1. **Setup:**
   - Let A be the set $\{(0, 1, 0)\}$.
   - Suppose that we can operate on A by flipping any two elements from 0 to 1 or from 1 to 0.

   Problem: Is it possible that any sequence of such flips applied to A yields the triple $(0, 0, 0)$?
   Solution: Let $X = \{$ all triples of binary digits $\}$. Then consider $I(X, A, F)$, where $A = \{(0, 1, 0)\}$, $F = \{$ flip 1 and 2, flip 1 and 3, flip 2 and 3 $\}$. The problem is then equivalent to asking
      Is $(0, 0, 0) \in I(X, A, F)$?
   I claim that the answer is "No". I need to prove my answer, by structural induction on $I(X, A, F)$.
   For any binary triple (x, y, z), define $R(x, y, z)$ to be
      (x, y, z) has an even number of 0 digits.
   We will prove by structural induction that every triple in $I(X, A, F)$ has property R.
   - Base: $R(0, 1, 0)$ is clear.
   - Induction: Let (x, y, z) be any binary triple having property R. Then we check that each operation in F preserves property R, via the following table:

input triple	flip 1 and 2	flip 1 and 3	flip 2 and 3
(1, 1, 1)	(0, 0, 1)	(0, 1, 0)	(1, 0, 0)
(0, 0, 1)	(1, 1, 1)	(1, 0, 0)	(0, 1, 0)
(0, 1, 0)	(1, 0, 0)	(1, 1, 1)	(0, 0, 1)
(1, 0, 0)	(0, 1, 0)	(0, 0, 1)	(1, 1, 1)

All the output triples have property R. This completes the inductive step, and the proof.

Now since $(0, 0, 0)$ does not have property R, therefore $(0, 0, 0) \notin I(X, A, F)$.

Remarks:

(a) This example provides a strategy for proving that an element of the universe is **not** a member of an inductively defined set:

  i. Prove that all elements of the set have some property R.

  ii. Prove that the element of interest does **not** have property R.

E.g. since all propositional formulas in Form($\mathcal{L}^p$) have equaly many "(" and ")" symbols, therefore $)p \to \land \leftrightarrow rq()$ is not in Form($\mathcal{L}^p$).

3 Lecture 03 - Semantics of Propositional Logic

Outline

1. Semantics of Propositional Logic
   (a) Truth Valuations
   (b) Evaluating Any Propositional Formula
   (c) Properties of Propositional Formulas

3.1 Semantics of Propositional Logic

Remarks:

1. Until now, we have been focussing (mostly) on **syntax** alone, i.e. on the question, "of all possible expressions, which ones constitute propositional formulas in Form($\mathcal{L}^{\mathrm{p}}$)?".
2. **Semantics** is concerned with the question, "given some propositional formula $A \in$ Form($\mathcal{L}^{\mathrm{p}}$), is A 1 or 0 in some semnatic context?".
3. A **semantic context** in Propositional Logic means a choice of **truth valuation**, i.e. a choice, for each available proposition symbol, between 0 and 1. (See Definition 3.1.1.)
4. A **truth valuation** corresponds with **a single row of a truth table**.
5. See Lu pp19-21 for the truth tables of each connective.

3.1.1 Truth Valuations

Definition 3.1.1. *A* ***truth valuation*** *t is a function from proposition symbols to* $\{0,1\}$*. Notation:* p^t *denotes the value of* p *under t.*

In other words, a truth valuation, t, is a function

$$t\colon \mathcal{P} \to \{0,1\},$$

where $\mathcal{P}$ denotes the set of available proposition symbols.

Examples:

1. Let $\mathcal{P} = \{\mathrm{p},\mathrm{q},\mathrm{r}\}$. Define the function

$$\begin{array}{cccc} t & : & \{\mathrm{p},\mathrm{q}\} & \to & \{0,1\} \\ & & \mathrm{p} & \mapsto & 1 \\ & & \mathrm{q} & \mapsto & 0 \end{array}$$

   Q: Is t a truth valuation on $\mathcal{P}$? Why or why not?
   A: No. t is not defined for $\mathrm{r} \in \mathcal{P}$, hence t is not a function $t\colon \mathcal{P} \to \{0,1\}$.
2. Now let $\mathcal{P} = \{\mathrm{p},\mathrm{q}\}$. Define the function t as in the previous question. **Q:** Is t a truth valuation on $\mathcal{P}$? Why or why not?
   A: Yes. t is a function $t\colon \mathcal{P} \to \{0,1\}$.

3. Write a combined truth table for $(p \to q)$ and $((\neg p) \vee q)$.
Solution:

p	q	$(p \to q)$	$((\neg p) \vee q)$
1	1	1	1
1	0	0	0
0	1	1	1
0	0	1	1

Remarks:
(a) These formulas are **not equal**.
(b) However these forumulas are **tautologically equivalent** (Definition 3.1.3), i.e. they have the same truth table, i.e. they evaluate the same way under every possible truth valuation.
(c) Notation: $(p \to q) \models\mid ((\neg p) \vee q)$.

3.1.2 Evaluating Any Propositional Formula

Fix a truth valuation t. Then every formula C has a value under t, denoted C^t, defined inductively as follows.

1. p^t is given by the definition of t, for every proposition symbol, p.
2. $(\neg A)^t = \begin{cases} 1 & \text{if } A^t = 0 \\ 0 & \text{if } A^t = 1 \end{cases}$
3. $(A \wedge B)^t = \begin{cases} 1 & \text{if } A^t = B^t = 1 \\ 0 & \text{otherwise} \end{cases}$
4. $(A \vee B)^t = \begin{cases} 1 & \text{if } A^t = 1 \text{ or } B^t = 1 \text{ (or both)} \\ 0 & \text{otherwise} \end{cases}$
5. $(A \to B)^t = \begin{cases} 1 & \text{if } A^t = 0 \text{ or } B^t = 1 \text{ (or both)} \\ 0 & \text{otherwise} \end{cases}$
6. $(A \leftrightarrow B)^t = \begin{cases} 1 & \text{if } A^t = B^t \\ 0 & \text{otherwise} \end{cases}$

This handles every propositional forumla $C \in \text{Form}(\mathcal{L}^p)$ unambiguously, because of Unique Readability (Theorem 2.4.1).

Definition 3.1.2. *Let* C *be a propositional formula in* $\text{Form}(\mathcal{L}^p)$ *and let t be a truth valuation. We say that*

1. C *is* ***satisfied*** *under t if* $C^t = 1$, *and*
2. C *is* ***not satisfied*** *under t if* $C^t = 0$.

Remarks:

1. Definitinon 3.1.2 is well-defined, because of Theorem 2.4.1.

Examples:

1. Construct the truth table for each formula given.
   (a) $(\neg(p \wedge q))$
   Solution:

p	q	$(p \wedge q)$	$(\neg(p \wedge q))$
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	1

   (b) $((\neg p) \vee (\neg q))$
   Solution:

p	q	$(\neg p)$	$(\neg q)$	$((\neg p) \vee (\neg q))$
1	1	0	0	0
1	0	0	1	1
0	1	1	0	1
0	0	1	1	1

   Remarks:
   i. This particular **tautological equivalence** $(\neg(p \wedge q)) \models\!\dashv ((\neg p) \vee (\neg q))$ is an example of a **DeMorgan Law**.

   (c) $((p \wedge q) \rightarrow r)$
   Solution:

p	q	r	$(p \wedge q)$	$((p \wedge q) \rightarrow r)$
1	1	1	1	1
1	1	0	1	0
1	0	1	0	1
1	0	0	0	1
0	1	1	0	1
0	1	0	0	1
0	0	1	0	1
0	0	0	0	1

Definition 3.1.3. *Two propositional formulas* $A, B \in \mathrm{Form}(\mathcal{L}^p)$ *are called* ***(tautologically) equivalent*** *(denoted* $A \models\!\dashv B$*) if* $A^t = B^t$ *for every truth valuation, t. (Equivalently, if* A *and* B *have the same truth table.)*

3.1.3 Properties of Propositional Formulas

Problems:

1. Let $\mathcal{P}$ be a set of proposition symbols. Let t be a truth valuation defined on $\mathcal{P}$. Let C be a propositional formula in $\mathrm{Form}(\mathcal{L}^p)$. Prove by structural induction on C that $C^t \in \{0, 1\}$.
   Solution: For any propositional formula in $\mathrm{Form}(\mathcal{L}^p)$, C, define R(C) to be the statement

$$C^t \in \{0, 1\}.$$

   <u>Base</u> ($C = p$, for some proposition symbol $p \in \mathcal{P}$): Then $C^t = p^t \in \{0, 1\}$ (i.e. R(C) holds).

Induction
If $C = (\neg A)$, for some propositional formula A in Form($\mathcal{L}^p$), then the induction hypothesis, R(A), says that $A^t \in \{0, 1\}$. Therefore

$$C^t = (\neg A)^t = \begin{cases} 0 & \text{if } A^t = 1 \\ 1 & \text{if } A^t = 0 \end{cases}$$

This shows that $C^t \in \{0, 1\}$ (i.e. R(C) holds), completing this case.
If $C = (A \wedge B)$, for some propositional formulas, A, B in Form($\mathcal{L}^p$), then the induction hypotheses, R(A) and R(B), say that $A^t \in \{0, 1\}$ and $B^t \in \{0, 1\}$. Therefore

$$C^t = (A \wedge B)^t = \begin{cases} 1 & \text{if } A^t = B^t = 1 \\ 0 & \text{if } A^t = 0 \text{ or } B^t = 0 \text{ or both} \end{cases}$$

This shows that $C^t \in \{0, 1\}$ (i.e. R(C) holds), completing this case.
The remaining cases for the binary connectives $\vee$, $\rightarrow$ and $\leftrightarrow$ are similar to the case for $\wedge$, and thus are omitted.
Thus, by the Principle of Structural Induction, R(C) holds for every $C \in$ Form($\mathcal{L}^p$).

Definition 3.1.4. *A propositional formula* C *in* Form($\mathcal{L}^p$) *is a* ***tautology*** *(aka* ***valid formula****) if* $C^t = 1$, *for every truth valuation t.*

Examples:

1. $(p \vee (\neg p))$

Definition 3.1.5. *A propositional formula* C *in* Form($\mathcal{L}^p$) *is* ***satisfiable*** *if* $C^t = 1$, *for some truth valuation t.*

Definition 3.1.6. *A propositional formula* C *in* Form($\mathcal{L}^p$) *is a* **contradicton** *(aka* **not satisfiable***) if* $C^t = 0$, *for every truth valuation t.*

Examples:

1. $(p \wedge (\neg p))$

Remarks:

1. Being satisfiable is the negation of being a contradiction.
2. Being a tautology is **not** the negation of being a contradiction.
3. Every tautology is satisfiable.
4. However not every satisfiable formula is a tautology.
5. Strategies for determining the properties of a propositional formula:
   (a) truth table
   (b) valuation tree (e.g. use a valuation tree to show that $((p \rightarrow q) \rightarrow r))$ is satisfiable, and not a tautology.

Problems:

1. For each of the Propositional formulas given below, determine with proof whether the formula is a contradiction (i.e. not satisfiable), satisfiable and not a tautology, or a tautology (i.e. a valid formula). Use truth tables and/or valuation trees to justify each answer.
   (a) C = $(p \wedge (\neg p))$
   Solution: The given formula has the following truth table.

p	$(\neg p)$	$(p \wedge (\neg p))$
1	0	0
0	1	0

   C is a contradiction (not satisfiable) since there is no truth valuation in which the formula is true.
   (b) C = $(p \vee (\neg p))$
   Solution: The given formula has the following truth table.

p	$(\neg p)$	$(p \vee (\neg p))$
1	0	1
0	1	1

   C is a tautology, since the formula is true in all truth valuations.
   (c) C = $((p \to q) \to r)$
   Solution 1: The given formula has the following truth table.

p	q	r	$(p \to q)$	$((p \to q) \to r)$
1	1	1	1	1
1	1	0	1	0
1	0	1	0	1
1	0	0	0	1
0	1	1	1	1
0	1	0	1	0
0	0	1	1	1
0	0	0	1	0

   C is satisfiable since there is at least one truth valuation that makes C 1 (e.g. row 2).
   C is not a tautology, since at least one truth valuation makes C 0 (e.g. row 1).
   Solution 2: As the number of proposition symbols grows, it can become tedious to write down the entire truth table. We can instead solve the problem using a **valuation** tree.
   - By →-properties, a truth valuation t will make $C^t = 1$ if $r^t = 1$. (N.B. This is "if", not "if and only if", as we can see from the truth table in Solution 1.) This shows that C is satisfiable.
   - By →-properties, a truth valuation t will make $C^t = 0$ if and only if $(p \to q)^t = 1$ and $r^t = 0$. Again by →-properties, a truth valuation t will make $(p \to q)^t = 1$ if $q^t = 1$. (N.B. Again this is "if", not "if and only if".) This shows that C is not a tautology.

4 Lecture 04 - Tautological Consequence

Outline

1. Tautological Consequence
   (a) Satisfaction of a Set of formulas
   (b) Definition of Tautological Consequence
   (c) Subtleties About Tautological Consequence
   (d) Argument Validity

4.1 Tautological Consequence

4.1.1 Satisfaction of a Set of formulas

Definition 4.1.1. *We say that a truth valuation, t,* ***satisfies a set*** Σ*,* ***of propositional formulas in*** $\mathrm{Form}(\mathcal{L}^p)$*, (Notation:* $\Sigma^t = 1$*) if, for every* $C \in \Sigma$*, we have* $C^t = 1$*. If there exists* $C \in \Sigma$ *such that* $C^t = 0$*, then we say that t* ***does not satisfy*** Σ *(Notation:* $\Sigma^t = 0$*).*

Remarks:

1. Analogously to a formula, we say that Σ is **satisfiable** if there exists a truth valuation *t* such that $\Sigma^t = 1$. Otherwise, we say that Σ is **not satisfiable**.
2. When we write $\Sigma^t = 0$, we are **not** asserting that every $C \in \Sigma$ has $C^t = 0$; we just assert that there is **at least one** such C.

Problems:

1. Verify that $\Sigma = \{((p \to q) \vee r), ((p \vee q) \vee s)\}$ is satisfiable.
   Solution 1: Let *t* be any truth valuation such that $q^t = 1$. p^t, r^t, s^t can be 0 or 1 - it will not matter in this example. Then
   - $(p \to q)^t = 1$ (by $\to$-rule), so that $((p \to q) \vee r)^t = 1$ (by $\vee$-rule), and
   - $(p \vee q)^t = 1$ (by $\vee$-rule), so that $((p \vee q) \vee s)^t = 1$ (by $\vee$-rule).

   This shows that *t* satisfies Σ. N.B. This defines a family of truth valuations that work. There are $2^3 = 8$ such truth valuations.
   Solution 2: Construct a joint truth table for $((p \to q) \vee r)$ and $((p \vee q) \vee s)$ and verify that at least one row has 1 in both of its final columns.
2. Is the set $\Sigma = \{(p \to (p \wedge q)), ((\neg p) \vee (\neg q)), ((\neg p) \to p)\}$ satisfiable? Prove your answer.
   Solution 1: I claim that the set is not satisfiable. For a contradiction, suppose that there exists a truth valuation, *t*, such that $\Sigma^t = 1$. Then
   - $(p \to (p \wedge q))^t = 1$, so we must have one of
     - $(p \wedge q)^t = 1$, which requires $p^t = q^t = 1$. However this implies that $((\neg p) \vee (\neg q))^t = 0$, which contradicts $\Sigma^t = 1$, completing this case.
     - $p^t = 0$, which implies $((\neg p) \to p)^t = 0$, which contradicts $\Sigma^t = 1$, completing this case.
   - All possibilities lead to contradiction. This completes the proof.

Solution 2: Construct a joint truth table for $(\mathrm{p} \to (\mathrm{p} \wedge \mathrm{q}))$, $((\neg \mathrm{p}) \vee (\neg \mathrm{q}))$ and $((\neg \mathrm{p}) \to \mathrm{p})$ and verify that no row has 1 in all three of its final columns.

4.1.2 Definition of Tautological Consequence

Definition 4.1.2. *We say that a set* Σ *of propositional formulas in* $\mathrm{Form}(\mathcal{L}^{\mathrm{p}})$ ***tautologically implies*** *a propositional formula* C *in* $\mathrm{Form}(\mathcal{L}^{\mathrm{p}})$ *(Notation:* $\Sigma \models \mathrm{C}$*), if, whenever* $\Sigma^t = 1$ *for some truth valuation t, we also have* $\mathrm{C}^t = 1$*. In this situtation, we also say that* C *is a* ***tautological consequence*** *of* Σ*.*

Remarks:

1. The notion of **tautological consequence** will be a key ingredient in defining the **soundness** and **completeness** of our proof system(s), later on.
2. It is crucial to understand that Definition 4.1.2 asserts **nothing** in the situation where $\Sigma^t = 0$.
3. To prove that $\Sigma \models \mathrm{C}$, apply Defintion 4.1.2.
4. To prove that $\Sigma \not\models \mathrm{C}$, for some Σ and C, we must exhibit a choice of a truth valuation, t, such that $\Sigma^t = 1$ and $\mathrm{C}^t = 0$.
5. If Σ is not satisfiable, then $\Sigma \models \mathrm{C}$, for **any** C.
6. Let $\Sigma = \emptyset$. Let t be any truth valuation. Then $\emptyset^t = 1$. This is counter-intuitive. But we must accept it, given Definition 4.1.1. There is no $\mathrm{C} \in \emptyset$ such that $\mathrm{C}^t = 0$.

Problems:

1. Q: Is it true that $\{(\mathrm{p} \wedge \mathrm{q})\} \models \mathrm{p}$?
   A: Yes. Any truth valuation, t such that $\{(\mathrm{p} \wedge \mathrm{q})\}^t = 1$ has $(\mathrm{p} \wedge \mathrm{q})^t = 1$, and by properties of $\wedge$, this requires $\mathrm{p}^t = 1 = \mathrm{q}^t$. Hence t satisfies p.
2. Q: Is it true that $\{(\mathrm{p} \wedge \mathrm{q})\} \models \mathrm{r}$?
   A: No. Consider the truth valuation
   $$\begin{array}{rccc} t & : & \{\mathrm{p},\mathrm{q},\mathrm{r}\} & \to & \{0,1\} \\ & & \mathrm{p}^t & = & 1 \\ & & \mathrm{q}^t & = & 1 \\ & & \mathrm{r}^t & = & 0 \end{array}$$
   Then we have that $\{(\mathrm{p} \wedge \mathrm{q})\}^t = 1$, and $\mathrm{r}^t = 0$.
3. Q: Is it true that $\{(\mathrm{p} \vee \mathrm{q})\} \models \mathrm{p}$?
   A: No. Consider the truth valuation
   $$\begin{array}{rcccc} t & : & \{\mathrm{p},\mathrm{q}\} & \to & \{0,1\} \\ & & \mathrm{p}^t & = & 0 \\ & & \mathrm{q}^t & = & 1 \end{array}$$
   Then we have that $\{(\mathrm{p} \vee \mathrm{q})\}^t = 1$, and $\mathrm{p}^t = 0$.

4. Prove or disprove the tautological consequence $\{(p \to r), (q \to (\neg r))\} \models (p \to (\neg q))$.
Solution: The truth table below shows the valuations of all of the formulas involved. The lines marked with $\lhd$ are the ones for which all the assumptions $\{(p \to r), (q \to (\neg r))\}$ are all true. In all such cases, the conclusion $(p \to (\neg q))$ is also true.

p	q	r	$(p \to r)$	$(q \to (\neg r))$	$(p \to (\neg q))$	
0	0	0	1	1	1	$\lhd$
0	0	1	1	1	1	$\lhd$
0	1	0	1	1	1	$\lhd$
0	1	1	1	0	1	
1	0	0	0	1	1	
1	0	1	1	1	1	$\lhd$
1	1	0	0	1	0	
1	1	1	1	0	0	

Thus the tautological consequence holds.

4.1.3 Subtleties About Tautological Consequence

1. Any statement like "For all $x \in \varnothing$, $x < 6$" is true. The empty set, $\varnothing$, provides no counterexample $x \in \varnothing$, such that $x < 6$ fails to hold.
2. By the same token, "For all $C \in \varnothing$, $C^t = 1$" is true, for any truth valuation t.
   (a) It is important to understand that the negation of the statement in the previous bullet is "There exists $C \in \varnothing$, such that $C^t = 0$" (which is clearly not true), and **not** "For all $C \in \varnothing$, $C^t = 0$" (which is true, as above).
3. As pointed out earlier, **the empty set is satisfied under any truth valuation,** *t*.
4. This shows by Definition 4.1.2 that if $\varnothing \models C$, then C is a tautology.
   (a) Some authors will write "fresh air" instead of $\varnothing$. I will always write $\varnothing$ explicitly.
5. For the reverse implication, note that if C is a tautology, $\Sigma \models C$, for **any** Σ.
6. As pointed out earlier, in the case where Σ is not satisfiable, we see that $\Sigma \models C$, for **any** C.
7. An equivalent alternative to Definition 3.1.3, using Definition 4.1.2 is that

$$A \models\!\!\!\dashv B \text{ if and only if } \big(\{A\} \models B \text{ and } \{B\} \models A\big).$$

8. Writing $\Sigma \models \varnothing$ makes no sense. $\varnothing$ is not a Propsitional formula.
9. In class we developed this summary table about the subtleties of tautological consequence. Make sure that you understand the definition of tautological consequence, and you do not try to rely on this table alone!

Σ	C	$\Sigma \models C$?
not satisfiable	contradiction	yes
not satisfiable	satisfiable, not a tautology	yes
not satisfiable	tautology	yes
satisfiable	contradiction	no
satisfiable	satisfiable, not a tautology	maybe
satisfiable	tautology	yes

4.1.4 Argument Validity

As explained in the slides, the argument "Σ proves A" is **valid** if and only if $\Sigma \models A$.

Example:

Given the premises:

1. If I study before my mid-term, and I get 8 hours' sleep before my mid-term, then I will pass my mid-term.
2. I studied before my mid-term.
3. I did not pass my mid-term.

We may conclude:

1. Therefore I must not have gotten 8 hours' sleep before my mid-term.

First, we translate from English into propositional logic; second we prove the resulting tautological consequence.

Define these atomic propositions:

1. s: I studied before my mid-term.
2. h: I got 8 hours' sleep before my mid-term.
3. p: I passed my mid-term.

We then encode the above argument as follows. We will explain all the notation below, soon.

$$\{((s \land h) \to p), s, (\neg p)\} \underbrace{\models}_{\text{"tautologically implies"}} (\neg h).$$

We will see soon that the premises on the left of $\models$ are sufficient to tautologically imply $(\neg h)$, the formula on the right of $\models$. This is what I meant by "convincing", above.

Solution 1:

- Let $\Sigma = \{((s \land h) \to p), s, (\neg p)\}$.
- Towards a contradiction, suppose that there exists a truth valuation, t, such that $\Sigma^t = 1$ and $(\neg h)^t = 0$, i.e. $h^t = 1$.

- Then we have $s^t = 1$, and $(\neg p)^t = 1$, so that $p^t = 0$.
- By $\wedge$-properties, it follows that $(s \wedge h)^t = 1$.
- By $\rightarrow$-properties, it follows that $p^t = 1$.
- This contradiction completes the proof.

Solution 2: Construct a truth table displaying all of the formulas involved. This is left as an exercise.

5 Lecture 05 - Normal Forms

Outline

1. Replacement and Duality Theorems
2. Propositional Calculus
3. Normal Forms
4. Disjunctive Normal Form
   (a) From Truth Tables
5. Conjunctive Normal Form
   (a) From Truth Tables
6. How to Obtain Normal Forms

5.1 Replacement and Duality Theorems

These Theorems are stated in the Logic03 slide deck. Both proofs are by structural induction, and are ommitted.

Theorem (Replacability) 5.1.1. *Let* A *be a formula containing a subformula* B. *Assume* $\mathrm{B} \models\!\dashv \mathrm{C}$, *and let* A' *be the formula obtained by simultaneously replacing in* A *some (but not necessarily all) occurrences of* B *by the formula* C. *Then* $\mathrm{A}' \models\!\dashv \mathrm{A}$.

Theorem (Duality) 5.1.2. *Suppose* A *is a formula composed only of atoms and the connectives* $\neg, \vee, \wedge$, *by the formation rules for these three connectives. Suppose* $\Delta(\mathrm{A})$ *results from simulaneously replacing in* A *all occurrences of* $\wedge$ *with* $\vee$, *all occurrences of* $\vee$ *with* $\wedge$, *and each atom with its negation. Then* $\neg\mathrm{A} \models\!\dashv \Delta(\mathrm{A})$.

5.2 Propositional Calculus

To transform formulas into either of the normal forms that we care about (e.g. so that we can perform Resolution on them), we will need to know how to eliminate implications and equivalences, leaving only $\neg, \wedge, \vee$.

To remove the connective $\rightarrow$ one uses the logical equivalence

$$\mathrm{A} \rightarrow \mathrm{B} \models\!\dashv \neg\mathrm{A} \vee \mathrm{B}$$

There are two ways to remove the connective $\leftrightarrow$:

$$\mathrm{A} \leftrightarrow \mathrm{B} \models\!\dashv (\mathrm{A} \wedge \mathrm{B}) \vee (\neg\mathrm{A} \wedge \neg\mathrm{B})$$

and

$$\mathrm{A} \leftrightarrow \mathrm{B} \models\!\dashv (\mathrm{A} \rightarrow \mathrm{B}) \wedge (\mathrm{B} \rightarrow \mathrm{A}) \models\!\dashv (\neg\mathrm{A} \vee \mathrm{B}) \wedge (\neg\mathrm{B} \vee \mathrm{A})$$

Example. Remove $\rightarrow$ and $\leftrightarrow$ from the following formula:

$$((p \rightarrow (q \wedge r)) \vee ((r \leftrightarrow s) \wedge (q \vee s))).$$

Solution.

$$((\neg p \vee (q \wedge r)) \vee (((\neg r \vee s) \wedge (\neg s \vee r)) \wedge (q \vee s))).$$

Law	Name
$A \vee \neg A \vDash\!\dashv 1$	Excluded middle law
$A \wedge \neg A \vDash\!\dashv 0$	Contradiction law
$A \vee 0 \vDash\!\dashv A$, $A \wedge 1 \vDash\!\dashv A$	Identity laws
$A \vee 1 \vDash\!\dashv 1$, $A \wedge 0 \vDash\!\dashv 0$	Domination laws
$A \vee A \vDash\!\dashv A$, $A \wedge A \vDash\!\dashv A$	Idempotent laws
$\neg(\neg A) \vDash\!\dashv A$	Double-negation law
$A \vee B \vDash\!\dashv B \vee A$, $A \wedge B \vDash\!\dashv B \wedge A$	Commutativity laws
$(A \vee B) \vee C \vDash\!\dashv A \vee (B \vee C)$ $(A \wedge B) \wedge C \vDash\!\dashv A \wedge (B \wedge C)$	Associativity laws
$A \vee (B \wedge C) \vDash\!\dashv (A \vee B) \wedge (A \vee C)$ $A \wedge (B \vee C) \vDash\!\dashv (A \wedge B) \vee (A \wedge C)$	Distributivity laws
$\neg(A \wedge B) \vDash\!\dashv \neg A \vee \neg B$ $\neg(A \vee B) \vDash\!\dashv \neg A \wedge \neg B$	De Morgan's laws

From these laws one can derive further laws, for example, the absorption laws

$$A \vee (A \wedge B) \vDash\!\dashv A$$

$$A \wedge (A \vee B) \vDash\!\dashv A.$$

Hint: To prove the first equivalence, use the identity law ($A \vDash\!\dashv A \wedge 1$), distributivity law to factor out A, domination law, and identity laws again.

Another important law (and its dual):

$$(A \wedge B) \vee (\neg A \wedge B) \vDash\!\dashv B$$

$$(A \vee B) \wedge (\neg A \vee B) \vDash\!\dashv B.$$

Hint: Use distributity laws to factor out B.

5.3 Normal Forms

There are two types of normal forms in propositional calculus:

1. the **Disjunctive Normal Form (DNF)**, and
2. the **Conjunctive Normal Form (CNF)**.

Definition.

1. A **literal** is either a proposition symbol, or the negation of a proposition symbol.
2. A disjunction with literals as disjuncts (e.g. $(p \vee (\neg q) \vee r)$) is called a **disjunctive clause**.
3. A conjunction with literals as conjuncts (e.g. $((\neg p) \wedge q \wedge (\neg r))$) is called a **conjunctive clause**.
4. Any clause, disjunctive or conjunctive, is called a **clause**.

5.4 Disjunctive Normal Form

Definition 5.4.1. *A disjunction of conjunctive clauses is said to be in* ***Disjunctive Normal Form (DNF).***

A formula in **Disjunctive Normal Form (DNF)** is of the form $(A_{11} \wedge \cdots \wedge A_{1n_1}) \vee \cdots \vee (A_{k1} \wedge \cdots \wedge A_{kn_k})$ where A_{ij} are literals.

The formulas $(A_{j1} \wedge \cdots \wedge A_{jn_j})$ are the **conjunctive clauses** of the formula in DNF.

Examples:

1. Each of $(p \wedge q) \vee (p \wedge \neg q) \vee p$, $p \vee (q \wedge r)$, and $\neg p \vee q$ is in disjunctive normal form.
2. The formula $\neg(p \wedge q) \vee r$ is **not** in disjunctive normal form.

5.4.1 From Truth Tables

Example: What is the DNF of the formula f, given by the following truth table?

p	q	r	f
1	1	1	1*
1	1	0	0
1	0	1	1*
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	1*
0	0	0	0

Answer:

$$f \models\!\!\!\!\!\dashv (p \wedge q \wedge r) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge \neg q \wedge r).$$

5.5 Conjunctive Normal Form

Definition 5.5.1. *A conjunction of disjunctive clauses is said to be in* ***Conjunctive Normal Form (CNF).***

A formula in **Conjunctive Normal Form (CNF)** is of the form $(A_{11} \vee \cdots \vee A_{1n_1}) \wedge \cdots \wedge (A_{k1} \vee \cdots \vee A_{kn_k})$, where A_{ij} are literals.

The formulas $(A_{j1} \vee \cdots \vee A_{jn_j})$ are the **disjunctive clauses** of the formula in CNF.

Examples:

1. Each of $p \wedge (q \vee r) \wedge (\neg q \vee r)$ and $p \wedge q$ is in conjunctive normal form.
2. The formula $p \wedge (r \vee (p \wedge q))$ is **not** in conjunctive normal form.

5.5.1 From Truth Tables

- **Duality** can be used to obtain conjunctive normal forms from truth tables.
- Recall: if A is a formula containing only the connectives $\neg$, $\vee$ and $\wedge$, then its **dual**, $\Delta(A)$, is formed by replacing all $\vee$ by $\wedge$, all $\wedge$ by $\vee$, and all atoms by their negations.
- **Example:** Find the dual of the formula $A = (p \wedge q) \vee \neg r$.
  Answer: $\Delta(A) = (\neg p \vee \neg q) \wedge \neg\neg r \mathrel{\models\!\!\!\dashv} (\neg p \vee \neg q) \wedge r$.
- Recall that, by the Duality Theorem (5.1.2), $\Delta(A) \mathrel{\models\!\!\!\dashv} \neg A$.
- Also note that, if a formula A is in DNF, then its dual can easily be transformed into an equivalent formula in CNF, using double-negation if necessary.
- This idea can be used to find the **conjunctive normal form** from the truth table of a formula f.
  - Determine the **disjunctive normal form** for $\neg f$.
  - If the resulting DNF formula is A, then $A \mathrel{\models\!\!\!\dashv} \boldsymbol{\neg f}$.
  - Compute $\Delta(A) \mathrel{\models\!\!\!\dashv} \neg A$, by the Duality Theorem.
  - $\Delta(A) \mathrel{\models\!\!\!\dashv} \neg A \mathrel{\models\!\!\!\dashv} \neg(\boldsymbol{\neg f}) \mathrel{\models\!\!\!\dashv} \mathbf{f}$.
  - $\Delta(A)$ can easily be converted into an equivalent formula in **CNF**.

Example: Find the CNF of the formula f_1 given by the truth table:

p	q	r	f_1
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	1

Solution:

p	q	r	f_1	$\neg f_1$
1	1	1	1	0
1	1	0	1	0
1	0	1	0	1
1	0	0	0	1
0	1	1	1	0
0	1	0	1	0
0	0	1	0	1
0	0	0	1	0

The DNF of $\neg f_1$, based on the truth table for $\neg f_1$, is the formula:

$$A = (p \land \neg q \land r) \lor (p \land \neg q \land \neg r) \lor (\neg p \land \neg q \land r) \Dashv\vDash \neg f_1.$$

The CNF for f_1 is equivalent to the dual of formula A, namely

$$\begin{aligned} & \Delta(A) \\ \Dashv\vDash\ & \neg A \\ \Dashv\vDash\ & (\neg p \lor q \lor \neg r) \land (\neg p \lor q \lor r) \land (p \lor q \lor \neg r) \\ \Dashv\vDash\ & \neg(\neg f_1) \\ \Dashv\vDash\ & f_1. \end{aligned}$$

5.6 How to Obtain Normal Forms

Use the following tautological equivalences:

(1) $A \to B \Dashv\vDash \neg A \lor B$.

(2) $A \leftrightarrow B \Dashv\vDash (\neg A \lor B) \land (A \lor \neg B)$.

(3) $A \leftrightarrow B \Dashv\vDash (A \land B) \lor (\neg A \land \neg B)$.

(4) $\neg\neg A \Dashv\vDash A$.

(5) $\neg(A_1 \land \cdots \land A_n) \Dashv\vDash \neg A_1 \lor \cdots \lor \neg A_n$.

(6) $\neg(A_1 \lor \cdots \lor A_n) \Dashv\vDash \neg A_1 \land \cdots \land \neg A_n$.

(7) $A \land (B_1 \lor \cdots \lor B_n) \Dashv\vDash (A \land B_1) \lor \cdots \lor (A \land B_n)$.

$(B_1 \lor \cdots \lor B_n) \land A \Dashv\vDash (B_1 \land A) \lor \cdots \lor (B_n \land A)$.

(8) $A \lor (B_1 \land \cdots \land B_n) \Dashv\vDash (A \lor B_1) \land \cdots \land (A \lor B_n)$.

$(B_1 \land \cdots \land B_n) \lor A \Dashv\vDash (B_1 \lor A) \land \cdots \land (B_n \lor A)$.

By the Theorem of Replaceability of Tautologically Equivalent Formulas, we can use the equivalences on the previous slide to convert any formula into a tautologically equivalent formula in either normal form:

1. By (1)–(3) we eliminate $\rightarrow$ and $\leftrightarrow$.
2. By (4)–(6) we transform a formula into an equivalent one, where every $\neg$ symbol has only an atom as its scope.
3. By (7) we eliminate $\vee$ from the scope of $\wedge$.
4. By (8) we eliminate $\wedge$ from the scope of $\vee$.

Note: For a tautology A, the required DNF may simply be $p \vee \neg p$ (this is also in CNF, with one clause), where p is any atom in A.

Similarly, for a contradiction B, the required CNF may be $p \wedge \neg p$ (which is also in DNF, with one clause), where p is any atom in B.

Example. Convert the following formula into a conjunctive normal form.

$$\neg((p \vee \neg q) \wedge \neg r).$$

A tautologically equivalent formula in conjunctive normal form can be found by the following sequence of derivations:

$\neg((p \vee \neg q) \wedge \neg r)$	$\models\!\mid \neg(p \vee \neg q) \vee \neg\neg r$	De Morgan
	$\models\!\mid \neg(p \vee \neg q) \vee r$	Double-negation
	$\models\!\mid (\neg p \wedge \neg\neg q) \vee r$	De Morgan
	$\models\!\mid (\neg p \wedge q) \vee r$	Double-negation
	$\models\!\mid (\neg p \vee r) \wedge (q \vee r)$	Distributivity

Algorithm for Conjunctive Normal Form

1. Eliminate equivalence and implication. (Use $A \rightarrow B \models\!\mid \neg A \vee B$ and $A \leftrightarrow B \models\!\mid (\neg A \vee B) \wedge (A \vee \neg B)$.)
2. Move negations inward, so that they will only apply to atoms, in literals. (Use double-negation and DeMorgan.)
3. Eliminate $\vee$s from the scopes of $\wedge$s, and eliminate $\wedge$s from the scopes of $\vee$s. (Use distributivity.)
4. **Recursive procedure** CNF(A):
   3.1 If A is a literal then return A.
   3.2 If A is $B \wedge C$ then return $\mathrm{CNF}(B) \wedge \mathrm{CNF}(C)$.
   3.3 If A is $B \vee C$ then
   - call CNF(B) and CNF(C)
   - suppose $\mathrm{CNF}(B) = B_1 \wedge B_2 \wedge \cdots \wedge B_n$
   - suppose $\mathrm{CNF}(C) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$
   - return $\wedge_{i=1\ldots n,\, j=1\ldots m}(B_i \vee C_j)$

   Note: The last step is similar to using distributivity to expand $(x_1 + x_2 + \ldots + x_n) \cdot (y_1 + y_2 + \ldots + y_m)$.

6 Lecture 06 - Adequate Sets of Connectives, Boolean Algebra, Logic Gates

Outline:

1. Adequate Sets of Connectives
2. Boolean Algebras
3. Logic Gates

6.1 Adequate Sets of Connectives

Definition 6.1.1. *The* ***arity*** *of a function,* f, *is the number,* $n \geq 1$, *of inputs that the function takes.*

Remarks:

1. 1-ary functions (e.g. $\neg$) are called **unary**.
2. 2-ary functions (e.g. $\wedge$) are called **binary**.
3. A propositional connective with arity n is a function

$$f: \{0, 1\}^n \to \{0, 1\}.$$

Q: If we permitted $n = 0$, then what would an 0-ary (nullary) function look like? **A:** Constant!

Definition 6.1.2. *A set,* S, *of propositional connectives is called* ***adequate for propositional logic*** *if* ***every*** *propositional connective (of any arity) can be implemented using the connectives from* S.

Theorem 6.1.3. *The set* $S_0 = \{\wedge, \vee, \neg\}$ *is an adequate set of connectives for propositional logic.*

Proof. Fix any $n \geq 1$. Fix any function

$$f: \{0, 1\}^n \to \{0, 1\}.$$

Write the arguments to f as $p_1, \dots, p_n$. It suffices to define f in terms of $S_0 = \{\wedge, \vee, \neg\}$.

- Construct the truth table for the function f.
- Use the theorem about the existence of Disjunctive Normal Forms to obtain a formula A_{S_0}, in DNF, with $f(p_1, \dots, p_n) \models\!\dashv A_{S_0}$.
- By construction, A_{S_0} has the same truth table as f, and A_{S_0} is constructed in terms of S_0.

■

Remarks:

1. Here is a (simple) example to demonstrate the construction in the proof of Theorem 6.1.3. Suppose that we start with the truth table of a binary connective f:

p_1	p_2	$f(p_1, p_2)$
0	0	0
0	1	1
1	0	0
1	1	1

- For row 2, we construct $((\neg p_1) \wedge p_2)$.
- For row 4, we construct $(p_1 \wedge p_2)$.
- Then we assemble the formula $(((\neg p_1) \wedge p_2) \vee (p_1 \wedge p_2))$, which by construction has the same truth table as f, and is implemented using only $\{\wedge, \vee, \neg\}$.

Theorem 6.1.4. *Each of the sets* $\{\wedge, \neg\}$, $\{\vee, \neg\}$ *and* $\{\rightarrow, \neg\}$ *is an adequate set of connectives for propositional logic.*

Proof. By Theorem 6.1.3, it suffices to show that

1. We can define $\vee$ in terms of $\{\wedge, \neg\}$,
   (a) $(A_1 \vee A_2) = (\neg((\neg A_1) \wedge (\neg A_2)))$✓
2. We can define $\wedge$ in terms of $\{\vee, \neg\}$
   (a) $(A_1 \wedge A_2) = (\neg((\neg A_1) \vee (\neg A_2)))$✓
   and
3. We can define $\wedge$ and $\vee$ in terms of $\{\rightarrow, \neg\}$.
   (a) $(A_1 \wedge A_2) = (\neg(A_1 \rightarrow (\neg A_2)))$✓
   (b) $(A_1 \vee A_2) = ((\neg(A_1) \rightarrow A_2))$✓

■

Remarks:

1. Because of Theorem 6.1.4, to prove another set of connectives is adequate for propositional logic, it would now suffice to show that the given set can implement any of $\{\wedge, \neg\}$, $\{\vee, \neg\}$ and $\{\rightarrow, \neg\}$
2. To prove that a given set of propositional connectives is **not** adequate for propositional logic, it would suffice to exhibit one propositional connective (of any arity), which **cannot** be implemented using the connectives in the given set.

Adequate sets containing a single connective exist. Refer the reader to Logic 05, Slides 11-13.

6.2 Boolean Algebras

Definition 6.2.1. *A* ***Boolean Algebra*** *is a set* B, *together with two binary operations* $+$ *and* $\cdot$, *and a unary operation* $\bar{\ }$. *The set* B *contains elements* ***0*** *and* ***1****, is closed under the application of* $+, \cdot$ *and* $\bar{\ }$, *and the following properties hold for all* $x, y, z \in B$:

1. ***Identity laws:*** $x + 0 = x$ *and* $x \cdot 1 = x$.
2. ***Complement laws:*** $x + \bar{x} = 1$, $x \cdot \bar{x} = 0$.
3. ***Associativity laws:*** $(x + y) + z = x + (y + z)$, $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.

4. ***Commutativity laws:*** $x + y = y + x$, $x \cdot y = y \cdot x$.
5. ***Distributivity laws:*** $x + (y \cdot z) = (x + y) \cdot (x + z)$ *and* $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

Boolean Algebra Examples:

1. The set of **formulas** in Form($\mathcal{L}^p$), with $\vee$ as $+$, $\wedge$ as $\cdot$, $\neg$ as $-$, **0** and **1**, and where $=$ is $\models\!\mid$, is a Boolean algebra.
2. The set of **subsets** of a universal set U, with the **union** operator $\cup$ as $+$, the **intersection** operator $\cap$ as $\cdot$, the **set complementation** operator ^{c} as $-$, the **empty set** $\emptyset$ as **0**, and the **universal set** U as **1**, is a Boolean algebra.

Logical equivalences	**Set properties**
$\neg(\neg p) \models\!\mid p$	$(A^c)^c = A$
$p \vee p \models\!\mid p, \quad p \wedge p \models\!\mid p$	$A \cup A = A, \quad A \cap A = A$
$p \vee \mathbf{0} \models\!\mid p, \quad p \wedge \mathbf{1} \models\!\mid p$	$A \cup \emptyset = A, \quad A \cap \mathbf{U} = A$
$p \wedge \mathbf{0} \models\!\mid \mathbf{0}, \quad p \vee \mathbf{1} \models\!\mid \mathbf{1}$ $p \vee \neg p \models\!\mid 1, \quad p \wedge \neg p \models\!\mid 0$	$A \cap \emptyset = \emptyset, \quad A \cup \mathbf{U} = \mathbf{U}$ $A \cup A^c = \mathbf{U}, \quad A \cap A^c = \emptyset$
$\neg(p \wedge q) \models\!\mid (\neg p \vee \neg q)$	$(A \cap B)^c = (A^c \cup B^c)$
$\neg(p \vee q) \models\!\mid (\neg p \wedge \neg q)$	$(A \cup B)^c = A^c \cap B^c$

Boolean algebra properties

1. Note that, using the laws given in the definition of a Boolean algebra, it is possible to prove many other laws that hold for every Boolean algebra (e.g. DeMorgan's Laws).
2. Thus, to establish results about propositional logic, or about sets, we need only prove results about Boolean algebras.

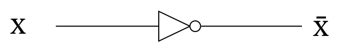
6.3 Logic Gates

Boolean algebra and computers

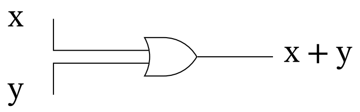
1. Boolean algebra is used to model the behaviour of electronic circuits, that work on a **binary** basis (i.e. everything is represented over the set $\{0, 1\}$.
2. A **Boolean variable** represents one bit (1/true or 0/false, which are also called **Boolean constants**).
3. An n-variable **Boolean function** is a function $f\colon \{0, 1\}^n \to \{0, 1\}$
4. An electronic computer is made up of a number of **circuits**, each of which implements a Boolean function.
5. The smallest building blocks of circuits are called **logic gates**. We will start with $\wedge, \vee$ and $\neg$.

Basic logic gates:

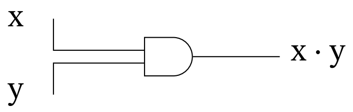
1. An **inverter**, or a **NOT** gate, is a logic gate that implements negation ($\neg$). It accepts the value of a Boolean variable as input, and produces the negation of its value as its output.



2. **The OR gate**
   (a) The inputs to this gate are the values of two Boolean variables. The output is the Boolean sum + (denoting ∨) of their values.

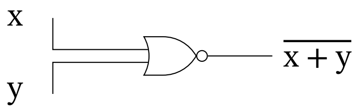


3. **The AND gate**
   (a) The inputs to this gate are the values of two Boolean variables. The output is the Boolean product (denoting ∧) of their values.

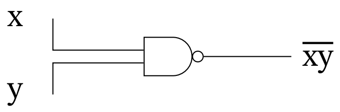


4. We invert the output of ∨ or ∧ by putting the negation circle on its output.

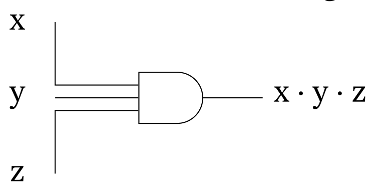
NOR



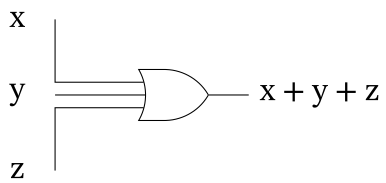
NAND



5. **Circuit notations and conventions**
   (a) In circuit design, we use the following notations:
      i. $x + y$ denotes $x \vee y$,
      ii. $x \cdot y$ and xy both denote $x \wedge y$
      iii. $\bar{x}$ denotes $\neg x$
      iv. $=$ denotes tautological equivalence ⟚.
   (b) We sometimes permit multiple inputs to AND gates (top) and OR gates (bottom), as illustrated below (unambiguous because ∧ and ∨ are associative).
      i.



      ii.

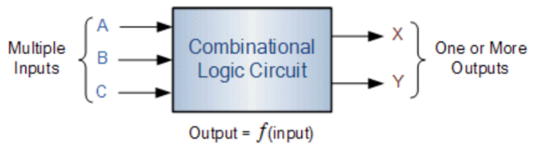


Combinational circuits

- **Combinational logic circuits** (sometimes called **combinatorial circuits**) are **memo-**

ryless digital logic circuits whose output is a function of the present value of the inputs only.

- A **combinational circuit** is implemented as a combination of NOT gates, OR gates, and AND gates. In general such a circuit has n inputs and m outputs in $\{0, 1\}$.

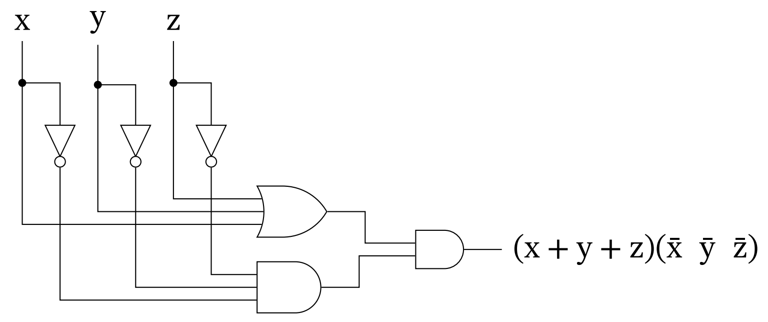


- In contrast, sequential logic circuits - not described in this course – are basically combinational circuits with the additional properties of storage (to remember past inputs) and feedback.

Example: Design a circuit that produces the following output:

$$(x + y + z)(\bar{x}\ \ \bar{y}\ \ \bar{z}).$$

Solution:



Note that propositional calculus can be used to simplify this formula to 0. Formula simplification often leads to smaller/cheaper circuits.

7 Lecture 07 - Circut Design and Minimization, Code Analysis and Simplification

Outline

1. Circuit Design and Minimization
2. Code Analysis and Simplification

7.1 Circuit Design and Minimization

Example (Logic05, Slide 48): Design a circuit for a 3-switch light fixture
Sometimes light fixtures are controlled by more than one switch. Circuits need to be designed so that flipping any one of the switches for the fixture turns the light on when it is off, and turns the light off when it is on. Design a circuit that accomplishes this task, when there are three switches.

Solution: The inputs are three Boolean variables, x, y, z, one for each switch.

Let $x = 1$ if the first switch is closed, and $x = 0$ if it is open, and similarly for y and z.

The output function is $F(x, y, z)$ defined as $F(x, y, z) = 1$ if the light is on, and $F(x, y, z) = 0$ if the light is off.

Truth table for 3-switch light fixture We can choose to specify that the light be on when all three switches are closed, so that $F(1, 1, 1) = 1$.

This determines all the other values of F:

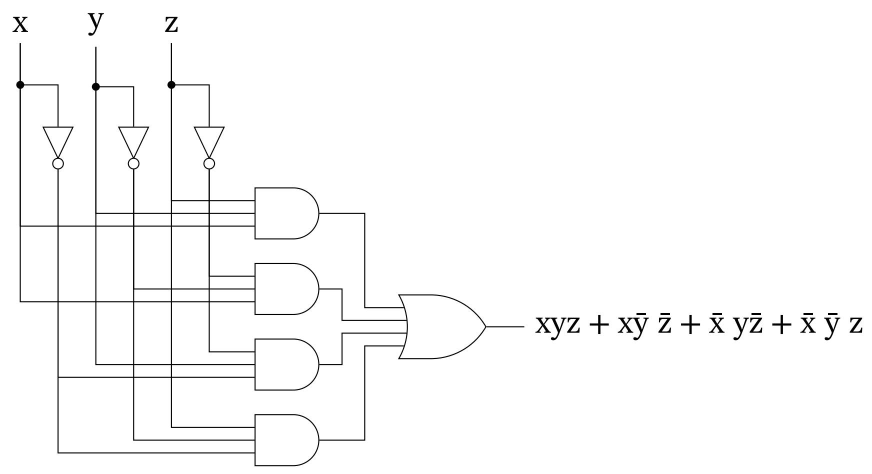
x	y	z	F(x, y, z)
1	1	1	**1**
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

The formula in DNF corresponding to this truth table is

$$F(x, y, z) \models\!\!\!\dashv\; xyz + x\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z$$

Circuit for 3-switch light fixture Below is a circuit implementing the function

$$F(x, y, z) \models\!\!\!\dashv\; xyz + x\bar{y}\,\bar{z} + \bar{x}\,y\bar{z} + \bar{x}\,\bar{y}\,z$$



Example (Logic05, Slide 60)
Minimize the circuit implementing the formula with the following truth table.

x	y	z	A
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	0

Strategy:

1. Find a formula in DNF that has the same truth table as A.
2. Use the **laws of Boolean Algebra** to simplify the formula as much as possible.
3. Draw the circuit for the simplified formula.

Remarks:

1. Some simplification by inspection is possible in this example. I will show a technique that will always work, even if such “hacks” are not available.

We have the following formula according to the truth table:

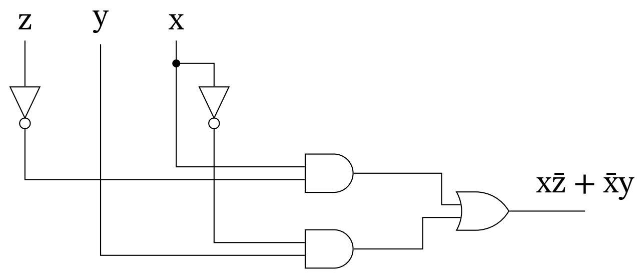
$$A \mathrel{\models\!\!\!\dashv} xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}y\bar{z}$$

To build this circuit without simplifying at all, we would need 8 gates in total: 3 ¬, 4 ∧ (3 inputs each), 1 ∨ (4 inputs)).

We can simplify the formula, to minimize the required circuit, as follows:

$$\begin{aligned}
A &\boxminus x y \bar{z} + x \bar{y} \bar{z} + \bar{x} y z + \bar{x} y \bar{z} \\
&\boxminus x \bar{z} y + x \bar{z} \bar{y} + \bar{x} y z + \bar{x} y \bar{z} \text{ (commutativity of } \cdot) \\
&\boxminus (x \bar{z} y + x \bar{z} \bar{y}) + (\bar{x} y z + \bar{x} y \bar{z}) \text{ (associativity of } +) \\
&\boxminus x \bar{z}(y + \bar{y}) + \bar{x} y(z + \bar{z}) \text{ (distributivity)} \\
&\boxminus x \bar{z}(1) + \bar{x} y(1) \text{ (excluded middle)} \\
&\boxminus x \bar{z} + \bar{x} y \text{ (identity)}
\end{aligned}$$

The minimized circuit needs only 5 gates: 2 $\neg$, 2 $\wedge$, 1 $\vee$.



7.2 Code Analysis and Simplification

Example: Slides 61-63 Consider the code fragment:

```
if ($q_1$ or not $q_2$) then
      if (not ($q_2$ and $q_3$)) then
        $P_1$
      else
           if ($q_2$ and not $q_3$) then
             $P_2$
           else
             $P_3$
else
  $P_4$
```

where q_1, q_2, q_3 are true/false conditions (proposition symbols), and P_1, P_2, P_3, P_4 are sub-fragments of code.

1. Analyze the code with a truth table:

q_1	q_2	q_3	$q_1 \vee \neg q_2$	$\neg(q_2 \wedge q_3)$	$q_2 \wedge \neg q_3$	Action
1	1	1	1	0	0	P_3
1	1	0	1	1	1	P_1
1	0	1	1	1	0	P_1
1	0	0	1	1	0	P_1
0	1	1	0	0	0	P_4
0	1	0	0	1	1	P_4
0	0	1	1	1	0	P_1
0	0	0	1	1	0	P_1

By analyzing the above truth table, we can observe under what combination of conditions q_1, q_2, q_3 are various fragments of code executed. For example, P_3 is executed iff $(q_1)^t = (q_2)^t = (q_3)^t = 1$.

2. Prove that P_2 is a dead code (without the truth table).
Definition: Dead code is code that is never executed.
The condition for P_2 to be executed is to satisfy

$$\begin{aligned} & (q_1 \vee \neg q_2) \wedge \neg\neg(q_2 \wedge q_3) \wedge (q_2 \wedge \neg q_3) \\ \models\!\!\!\dashv\ & (q_1 \vee \neg q_2) \wedge q_2 \wedge q_3 \wedge q_2 \wedge \neg q_3 \\ \models\!\!\!\dashv\ & 0 \end{aligned}$$

Since this condition can never be true (it is a contradiction), this means that P_2 can never be executed, i.e., it is dead code.

3. Is P_3 a dead code? No, according to the truth table, row 1.
Without a truth table, we can reason that the condition for P_3 to be executed is to satisfy

$$(q_1 \vee \neg q_2) \wedge (q_2 \wedge q_3) \wedge \neg(q_2 \wedge \neg q_3).$$

This formula has a satisfying truth valuation, namely $(q_1)^t = (q_2)^t = (q_3)^t = 1$, so P_3 is not dead code.

4. **Simplified Code (Slide 63 of Logic05):** The following simplified code is equivalent to the original:

```
if ($q_1$ and $q_2$ and $q_3$) then
  $P_3$
else
    if (not $q_1$ and $q_2$) then
      $P_4$
    else
      $P_1$
```

   (a) One can use the **laws of propositional calculus** to verify that this simplified code is equivalent to the original code.
   (b) **Hint:** Show that the formula with proposition symbols q_1, q_2, q_3 that leads to the execution of P_1 in the original code is logically equivalent to the formula that leads to the execution of P_1 in the simplified code. Show that the same holds for P_2, P_3 and P_4.

(c) We have the following truth table for the simplified code:

q_1	q_2	q_3	$q_1 \wedge q_2 \wedge q_3$	$\neg q_1 \wedge q_2$	Action
1	1	1	1	0	P_3
1	1	0	0	0	P_1
1	0	1	0	0	P_1
1	0	0	0	0	P_1
0	1	1	0	1	P_4
0	1	0	0	1	P_4
0	0	1	0	0	P_1
0	0	0	0	0	P_1

We note that this table results in the same fragments of code being executed under the same conditions as the truth table for the original code (the last columns of the truth tables for the original and simplified code are identical).

(d) If we do not want to use truth tables to compare the original code with the simplified code, we can use the laws of propositional calculus to compare the conditions for each fragment of code to be executed.

i. In the original code, P_1 is executed when this formula is satisfied:

$$\begin{aligned}
& (q_1 \vee \neg q_2) \wedge \neg(q_2 \wedge q_3) \\
\models\!\!\!\!\models\ & (q_1 \vee \neg q_2) \wedge (\neg q_2 \vee \neg q_3) \\
\models\!\!\!\!\models\ & (q_1 \wedge \neg q_2) \vee (q_1 \wedge \neg q_3) \vee (\neg q_2 \wedge \neg q_2) \vee (\neg q_2 \wedge \neg q_3) \\
\models\!\!\!\!\models\ & (q_1 \wedge \neg q_2) \vee (q_1 \wedge \neg q_3) \vee (\neg q_2 \vee (\neg q_2 \wedge \neg q_3)) \\
\models\!\!\!\!\models\ & (q_1 \wedge \neg q_2) \vee (q_1 \wedge \neg q_3) \vee \neg q_2 \\
\models\!\!\!\!\models\ & \neg q_2 \vee (q_1 \wedge \neg q_2) \vee (q_1 \wedge \neg q_3) \\
\models\!\!\!\!\models\ & \neg q_2 \vee (q_1 \wedge \neg q_3) \text{ (use absorption law)}
\end{aligned}$$

ii. In the original code, P_2 is executed when this formula is satisfied:

$$\begin{aligned}
& (q_1 \vee \neg q_2) \wedge ((q_2 \wedge q_3) \wedge (q_2 \wedge \neg q_3)) \\
\models\!\!\!\!\models\ & (q_1 \vee \neg q_2) \wedge (q_2 \wedge (q_3 \wedge \neg q_3)) \\
\models\!\!\!\!\models\ & (q_1 \vee \neg q_2) \wedge (q_2 \wedge 0) \\
\models\!\!\!\!\models\ & 0
\end{aligned}$$

In the simplified code, P_2 does not exist.

iii. In the original code, P_3 is executed when this formula is satisfied:

$$\begin{aligned}
& (q_1 \vee \neg q_2) \wedge (q_2 \wedge q_3) \wedge \neg(q_2 \wedge \neg q_3) \\
\mathrel{\text{⊨⊣}} \; & (q_1 \vee \neg q_2) \wedge q_2 \wedge (q_3 \wedge (\neg q_2 \vee q_3)) \\
\mathrel{\text{⊨⊣}} \; & (q_1 \vee \neg q_2) \wedge q_2 \wedge q_3 \\
\mathrel{\text{⊨⊣}} \; & (q_1 \wedge q_2 \wedge q_3) \vee (\neg q_2 \wedge q_2 \wedge q_3) \\
\mathrel{\text{⊨⊣}} \; & (q_1 \wedge q_2 \wedge q_3) \vee 0 \\
\mathrel{\text{⊨⊣}} \; & q_1 \wedge q_2 \wedge q_3
\end{aligned}$$

In the simplified code, P_3 is executed when this formula is satisfied:

$$q_1 \wedge q_2 \wedge q_3.$$

iv. In the original code, P_4 is executed when this formula is satisfied:

$$\neg(q_1 \vee \neg q_2) \mathrel{\text{⊨⊣}} \neg q_1 \wedge q_2.$$

In the simplified code, P_4 is executed when this formula is satisfied:

$$\begin{aligned}
& \neg(q_1 \wedge q_2 \wedge q_3) \wedge (\neg q_1 \wedge q_2) \\
\mathrel{\text{⊨⊣}} \; & (\neg q_1 \vee \neg q_2 \vee \neg q_3) \wedge \neg q_1 \wedge q_2 \\
\mathrel{\text{⊨⊣}} \; & \neg q_1 \wedge q_2 \text{ (use absorption law)}
\end{aligned}$$

8 Lecture 08 - Propositional Formal Deduction

Outline

1. Introduction to Proof Systems
2. Formal Deduction
   (a) Proof Rules
3. Formal Deduction Examples

Questions from the Class at the Start

1. **Q:** Do the Formal Deduction proof rules correspond to how we write proofs in English (Math 135 style, say)?
   A: Sometimes.
   - The negation rules look very much like versions of proof-by-contradiction.
   - $\lor-$ can be thought of as proof-by-cases.

   Whether you find these proof rules intuitive or not, they have been chosen for good syntactic reasons (provability side), and good semantic reasons (tautological consequence side).

8.1 Introduction to Proof Systems

Goal: Develop a technique for rigourously checking whether a proof is correct or not (i.e. checkable by computer, no human intuition required).

E.g., consider the following argument.

Given the premises:

1. If I study before my mid-term, and I get 8 hours' sleep before my mid-term, then I will pass my mid-term.
2. I studied before my mid-term.
3. I did not pass my mid-term.

We may conclude:

1. Therefore I must not have gotten 8 hours' sleep before my mid-term.

Q: Is this argument "convincing" to you?

A: I find this argument "convincing". I will justify this assertion, soon.

First, we translate from English into propositional logic; second we write a Formal Deduction proof of the argument.

Define these atomic propositions:

1. s: I studied before my mid-term.
2. h: I got 8 hours' sleep before my mid-term.

3. p: I passed my mid-term.

We then encode the above argument as follows. We will explain all the notation below, soon.

$$\{((s \wedge h) \to p), s, (\neg p)\} \underbrace{\vdash}_{\text{"proves"}} (\neg h).$$

We will see soon that the premises on the left of $\vdash$ are sufficient to prove $(\neg h)$, the formula on the right of $\vdash$. This is what I meant by “convincing”, above.

We read the shorthand $\Sigma \vdash A$ as “the set Σ proves the formula A”.

Q: What is the difference between $\models$ and $\vdash$?

A: $\models$ is semantic; $\vdash$ is syntactic. The Formal Deduction proof rules are 100% syntactic, 0% semantic. The connections between $\models$ and $\vdash$ arise as the **soundness** and **soundness** of Formal Deduction.

Here is a formal proof of the above result, in the proof system of Formal Deduction.

(1)	$((s \wedge h) \to p), s, (\neg p), h$	$\vdash$	s	(by $(\in)$)
(2)	$((s \wedge h) \to p), s, (\neg p), h$	$\vdash$	h	(by $(\in)$)
(3)	$((s \wedge h) \to p), s, (\neg p), h$	$\vdash$	$(s \wedge h)$	(by $(\wedge+)$, (1), (2))
(4)	$((s \wedge h) \to p), s, (\neg p), h$	$\vdash$	$((s \wedge h) \to p)$	(by $(\in)$)
(5)	$((s \wedge h) \to p), s, (\neg p), h$	$\vdash$	p	(by $(\to -)$ (3), (4))
(6)	$((s \wedge h) \to p), s, (\neg p), h$	$\vdash$	$(\neg p)$	(by $(\in)$)
(7)	$((s \wedge h) \to p), s, (\neg p)$	$\vdash$	$(\neg h)$	(by $(\neg+)$, (5), (6))

Exercises:

1. Prove
$$\{((s \wedge h) \to p), s, h\} \vdash p.$$

Remarks:

1. The Formal Deduction proof system we'll study first closely mimics the way that we write proofs already (say in MATH 135 style).
2. **Notation:**
   (a) $A, B, C, \ldots$ are formulas, and
   (b) Σ is a set of formulas.

Definition 8.1.1. *A **proof** in Formal Deduction is a sequence of lines of the form $\Sigma \vdash A$ (“Σ proves A”), for some set Σ and some formula A.*

3. Each line of the proof has columns, in order left-to-right:
   (a) a line number,
   (b) a set of premise formulas,
   (c) $\vdash$,
   (d) a conclusion formula, and

   (e) a justification, either "by supposition", or according to some **proof rule** of Formal Deduction, based on any of the lines already in evidence.
4. A **proof rule** takes one or more lines as input and returns a new line as output, according to predetermined rules.
5. The choice of proof rules determines the proof system S (at this point S is Formal Deduction).
   (a) Other proof systems exist.
   (b) If two proof systems are both sound and complete, then whatever is provable in one is necessarily provable in the other.
6. If we use our proof rules correctly, then each line asserts a correct fact about provability, from any given assumptions (i.e. suppositions) with which we started.
7. We are finished writing down our proof when the desired assertion $\Sigma \vdash A$ appears on its last line.
8. A proof is 100 % syntactic, and 0 % semantic.
9. The connection between $\models$ and $\vdash_S$ will come in the form of the **soundness** and **completeness** of the proof system S.
10. **Notation:** $\Sigma \vdash_S C$ reads as "There is a proof, in proof system S, with premises Σ and conclusion C". Read $\vdash_S$ as "proves in S".
11. **Notation:** $\Sigma \vdash A$ reads as "There is a proof in Formal Deduction, with last line $\Sigma \vdash A$."

8.2 Formal Deduction

8.2.1 Proof Rules

The Formal Deduction proof system has

1. one axiom ($\in$) (which can be proved using Ref and + - see pp47-48 in the text),
2. one Reflexive rule (Ref),
3. one Addition of Premises Rule (+) and
4. an introduction and an elimination rule for each Propositional connective (5 connectives, therefore 10 rules).

Formal Deduction Proof Rules

1. Axiom ($\in$): If $A \in \Sigma$, then $\Sigma \vdash A$.
2. (Ref): $A \vdash A$.
3. (+):
$$\begin{array}{lccl} \text{If} & \Sigma_1 & \vdash & A, \\ \text{then} & \Sigma_1, \Sigma_2 & \vdash & A. \end{array}$$
4. ($\neg-$):
$$\begin{array}{lccl} \text{If} & \Sigma, (\neg A) & \vdash & B, \\ & \Sigma, (\neg A) & \vdash & (\neg B), \\ \text{then} & \Sigma & \vdash & A. \end{array}$$
5. ($\neg+$):

If $\Sigma, A \vdash B$,
$\Sigma, A \vdash (\neg B)$,
then $\Sigma \vdash (\neg A)$.

This rule is also called **Reductio Ad Absurdum (RAA)**. This rule need not be a basic rule, because it can be proved from the basic rules - see below. We will prove it, then include it as a basic rule for convenience. See also Theorem 2.6.5 [2] in the text.

Remarks:

(a) Think of both $\neg$ rules as variations on proof by contradiction.

(b) Despite the fact that you may find some of these rules counter-intuitive, you have to get used to using them as stated.

6. $(\rightarrow -)$:
   If $\Sigma \vdash (A \rightarrow B)$,
   $\Sigma \vdash A$,
   then $\Sigma \vdash B$.
7. $(\rightarrow +)$:
   If $\Sigma, A \vdash B$,
   then $\Sigma \vdash (A \rightarrow B)$.
8. $(\wedge -)$:
   If $\Sigma \vdash (A \wedge B)$,
   then $\Sigma \vdash A$,
   $\Sigma \vdash B$.
9. $(\wedge +)$:
   If $\Sigma \vdash A$,
   $\Sigma \vdash B$,
   then $\Sigma \vdash (A \wedge B)$.
10. $(\vee -)$:
    If $\Sigma, A \vdash C$,
    $\Sigma, B \vdash C$,
    then $\Sigma, (A \vee B) \vdash C$.
11. $(\vee +)$:
    If $\Sigma \vdash A$,
    then $\Sigma \vdash (A \vee B)$,
    $\Sigma \vdash (B \vee A)$.
12. $(\leftrightarrow -)$:
    If $\Sigma \vdash (A \leftrightarrow B)$,
    $\Sigma \vdash A$,
    then $\Sigma \vdash B$.
    If $\Sigma \vdash (A \leftrightarrow B)$,
    $\Sigma \vdash B$,
    then $\Sigma \vdash A$.
13. $(\leftrightarrow +)$:
    If $\Sigma, A \vdash B$,
    $\Sigma, B \vdash A$,
    then $\Sigma \vdash (A \leftrightarrow B)$.

Remarks:

1. As in the text, we will write $\Sigma_1 \vdash \Sigma_2$ to mean that, for every $C \in \Sigma_2, \Sigma_1 \vdash C$.
2. In my experience, the rules for $\wedge, \rightarrow, \leftrightarrow$ are more intuitive, while the rules for $\vee, \neg$ are less intuitive. I will include as many examples using the less intuituive rules during our time together.
3. It may seem as if these proofs come "out of thin air". I have worked out my examples ahead of time. It will take you some practice to become comfortable with these proof rules.
4. The premise set on the last line is fixed (no choice). The premise sets on earlier lines are freely chosen by us.
5. Working **backwards** can reveal what additional premise formulas are required on earlier lines.

8.3 Proof Examples

1. Proof of $(p \wedge q), (r \wedge s) \vdash (p \wedge s)$:

(1)	$(p \wedge q), (r \wedge s)$	$\vdash$	$(p \wedge q)$	(by $(\in)$)
(2)	$(p \wedge q), (r \wedge s)$	$\vdash$	p	(by $(\wedge-)$, (1))
(3)	$(p \wedge q), (r \wedge s)$	$\vdash$	$(r \wedge s)$	(by $(\in)$)
(4)	$(p \wedge q), (r \wedge s)$	$\vdash$	s	(by $(\wedge-)$, (3))
(5)	$(p \wedge q), (r \wedge s)$	$\vdash$	$(p \wedge s)$	(by $(\wedge+)$, (2), (4))

2. Proof of $A, (\neg A) \vdash B$ (Theorem 2.6.5 [4] in the text):

(1)	$A, (\neg A), (\neg B)$	$\vdash$	A	(by $(\in)$)
(2)	$A, (\neg A), (\neg B)$	$\vdash$	$(\neg A)$	(by $(\in)$)
(3)	$A, (\neg A)$	$\vdash$	B	(by $(\neg-)$, (1), (2))

3. Proof of $((\neg A) \vee B) \vdash (A \rightarrow B)$ (one direction of Theorem 2.6.9 [5] in the text):

(1)	$(\neg A), A$	$\vdash$	B	(by 2.6.5 [4])
(2)	$(\neg A)$	$\vdash$	$(A \rightarrow B)$	(by $(\rightarrow +)$, (1))
(3)	B, A	$\vdash$	B	(by $(\in)$)
(4)	B	$\vdash$	$(A \rightarrow B)$	(by $(\rightarrow +)$, (3))
(5)	$((\neg A) \vee B)$	$\vdash$	$(A \rightarrow B)$	(by $(\vee-)$, (2), (4))

Remarks:

(a) The Σ in the $\vee-$ rule on line 5 is $\emptyset$.

4. Proof of $(A \rightarrow B) \vdash ((\neg A) \vee B)$ (other direction of Theorem 2.6.9 [5] in the text):

(1)	$(A \to B), (\neg((\neg A) \lor B)), (\neg A)$	$\vdash$	$(\neg A)$	(by $(\in)$)
(2)	$(A \to B), (\neg((\neg A) \lor B)), (\neg A)$	$\vdash$	$((\neg A) \lor B)$	(by $(\lor+)$, (1))
(3)	$(A \to B), (\neg((\neg A) \lor B)), (\neg A)$	$\vdash$	$(\neg((\neg A) \lor B))$	(by $(\in)$)
(4)	$(A \to B), (\neg((\neg A) \lor B))$	$\vdash$	A	(by $(\neg-)$, (2), (3))
(5)	$(A \to B), (\neg((\neg A) \lor B)), A$	$\vdash$	A	(by $(\in)$)
(6)	$(A \to B), (\neg((\neg A) \lor B)), A$	$\vdash$	$(A \to B)$	(by $(\in)$)
(7)	$(A \to B), (\neg((\neg A) \lor B)), A$	$\vdash$	B	(by $(\to -)$, (5), (6))
(8)	$(A \to B), (\neg((\neg A) \lor B)), A$	$\vdash$	$((\neg A) \lor B)$	(by $(\lor+)$, (7))
(9)	$(A \to B), (\neg((\neg A) \lor B)), A$	$\vdash$	$(\neg((\neg A) \lor B))$	(by $(\in)$)
(10)	$(A \to B), (\neg((\neg A) \lor B))$	$\vdash$	$(\neg A)$	(by $(\neg+)$, (8), (9))
(11)	$(A \to B)$	$\vdash$	$((\neg A) \lor B)$	(by $(\neg-)$, (4), (10))

5. Theorem 2.6.2 in the text If $\Sigma \vdash A$, then there exists a finite subset $\Sigma_0 \subseteq \Sigma$ such that $\Sigma_0 \vdash A$.
   Proof. Exercise. By structural induction on $\Sigma \vdash A$. The details are in the text. ■
6. Transitivity - Theorem 2.6.3 [2] in the text If $\Sigma \vdash \Sigma'$ and $\Sigma' \vdash A$, then $\Sigma \vdash A$.
   Proof. See the text. It is an exercise to write up the parts with hand-waving properly, using induction. ■
7. Proof of the DeMorgan Law $(\neg(A \land B)) \vdash ((\neg A) \lor (\neg B))$ (one direction of Theorem 2.6.9 [7] in the tex

(1)	$(\neg(A \land B))$	$\vdash$	$(A \to (\neg B))$	(by Theorem 2.6.8 [5])
(2)	$(A \to (\neg B))$	$\vdash$	$((\neg A) \lor (\neg B))$	(by Theorem 2.6.9 [5])
(3)	$(\neg(A \land B))$	$\vdash$	$((\neg A) \lor (\neg B))$	(by Transitivity, (1), (2))

8. Proof of the DeMorgan Law $((\neg A) \lor (\neg B)) \vdash (\neg(A \land B))$ (other direction of Theorem 2.6.9 [7] in the t

(1)	$(\neg A), (A \land B)$	$\vdash$	$(A \land B)$	(by $(\in)$)
(2)	$(\neg A), (A \land B)$	$\vdash$	A	(by $(\land-)$, (1))
(3)	$(\neg A), (A \land B)$	$\vdash$	$(\neg A)$	(by $(\in)$)
(4)	$(\neg A)$	$\vdash$	$(\neg(A \land B))$	(by $(\neg+)$, (2), (3))
(5)	$(\neg B), (A \land B)$	$\vdash$	$(A \land B)$	(by $(\in)$)
(6)	$(\neg B), (A \land B)$	$\vdash$	B	(by $(\land-)$, (1))
(7)	$(\neg B), (A \land B)$	$\vdash$	$(\neg B)$	(by $(\in)$)
(8)	$(\neg B)$	$\vdash$	$(\neg(A \land B))$	(by $(\neg+)$, (2), (3))
(9)	$((\neg A) \lor (\neg B))$	$\vdash$	$(\neg(A \land B))$	(by $(\lor-)$, (4), (8))

9. Proof of $(\neg(\neg A)) \vdash A$ (Theorem 2.6.5 [1] in the text):

(1)	$(\neg(\neg A)), (\neg A)$	$\vdash$	$(\neg A)$	(by $(\in)$)
(2)	$(\neg(\neg A)), (\neg A)$	$\vdash$	$(\neg(\neg A))$	(by $(\in)$)
(3)	$(\neg(\neg A))$	$\vdash$	A	(by $(\neg-)$, (1), (2))

   Note the typo in the statement of this result in the text - the A is missing on the LHS of the $\vdash$ symbol!
10. Proof of RAA (Theorem 2.6.5 [2] in the text):

(1)	Σ, A	$\vdash$	B	(by supposition)
(2)	$\Sigma, (\neg(\neg A))$	$\vdash$	Σ	(by $(\in)$)
(3)	$(\neg(\neg A))$	$\vdash$	A	(by Theorem 2.6.5 [1])
(4)	$\Sigma, (\neg(\neg A))$	$\vdash$	A	(by $(+)$, (3))
(5)	$\Sigma, (\neg(\neg A))$	$\vdash$	B	(by (Tr) (2), (4), (1))
(6)	Σ, A	$\vdash$	$(\neg B)$	(by supposition)
(7)	$\Sigma, (\neg(\neg A))$	$\vdash$	Σ	(by $(\in)$)
(8)	$(\neg(\neg A))$	$\vdash$	A	(by Theorem 2.6.5 [1])
(9)	$\Sigma, (\neg(\neg A))$	$\vdash$	A	(by $(+)$, (8))
(10)	$\Sigma, (\neg(\neg A))$	$\vdash$	$(\neg B)$	(by (Tr) (7), (9), (6))
(11)	Σ	$\vdash$	$(\neg A)$	(by $(\neg -)$ (5), (10))

11. Proof of $\{\neg B \to \neg A\} \vdash A \to B$ (Theorem 2.6.6 [4] in the text with A and B swapped):

(1)	$\neg B \to \neg A, A, \neg B$	$\vdash$	A	(by $(\in)$)
(2)	$\neg B \to \neg A, A, \neg B$	$\vdash$	$\neg B \to \neg A$	(by $(\in)$)
(3)	$\neg B \to \neg A, A, \neg B$	$\vdash$	$\neg B$	(by $(\in)$)
(4)	$\neg B \to \neg A, A, \neg B$	$\vdash$	$\neg A$	(by $(\to -)$, (2), (3))
(5)	$\neg B \to \neg A, A$	$\vdash$	B	(by $(\neg -)$, (1), (4))
(6)	$\neg B \to \neg A$	$\vdash$	$A \to B$	(by $(\to +)$, (5))

9 Lecture 09 - Soundness and Completeness of Propositional Formal Deduction

Outline

1. The Soundness of Propositional Formal Deduction
   (a) How to Prove a Tautological Consequence in General
   (b) Proof of the Soundness Theorem
2. The Completeness of Propositional Formal Deduction
   (a) Useful Ingredients
   (b) Proof of the Completeness Theorem (Optional)
3. Soundness and Completeness Revisited

9.1 The Soundness of Propositional Formal Deduction

9.1.1 How to Prove a Tautological Consequence in General

To prove $\Sigma \vDash C$,

1. directly: let t be any truth valuation such that $\Sigma^t = 1$, then prove that $C^t = 1$, OR
2. by contradiction: towards a contradiction, suppose that t is a truth valuation such that $\Sigma^t = 1$ and $C^t = 0$, then seek a contradiction.

9.1.2 Proof of the Soundness Theorem

Theorem (Soundness of Propositional Formal Deduction) 9.1.1. *If* $\Sigma \vdash C$*, then* $\Sigma \vDash C$.

Proof.
- By structural induction on $\Sigma \vdash C$, where the set of proofs is defined as $I(X, A, F)$, with
  - X is all sequences of numbered lines of the form: $\Sigma \vdash C$ (justification).
  - A is { single lines of the form: $C \vdash C$ (by (Ref)) }.
    (N.B. Think about how proofs "by Supposition" work. They sit outside of this framework, as they should.)
  - F is the 11 **basic** proof rules of formal deduction.
- Base: $C \vdash C$ by (Ref). It is clear that $C \vDash C$, so that the base case holds.
- Induction: As we are not in the base case, we have these cases for the **last** proof rule used in the proof $\Sigma \vdash C$.
  2. (+):
     - The induction hypothesis is $\Sigma_1 \vDash A$.
     - We want to prove that $\Sigma_1 \cup \Sigma_2 \vDash A$.
     - Let t be any truth valuation such that $(\Sigma_1 \cup \Sigma_2)^t = 1$.
     - Then in particular $\Sigma_1^t = 1$.
     - Then we have $A^t = 1$.

3. <u>$(\neg-)$</u>:
   - The induction hypothesis is $\Sigma \cup \{\neg A\} \models B$ and $\Sigma \cup \{\neg A\} \models \neg B$.
   - We want to prove that $\Sigma \models A$.
   - Towards a contradiction, suppose that there exists a truth valuation t such that $\Sigma^t = 1$ and $A^t = 0$.
   - Then by $\neg$-rule, $(\neg A)^t = 1$.
   - Then by the induction hypothesis and the definition of tautological consequence, we have $B^t = 1$ and $(\neg B)^t = 1$.
   - By $\neg$-rule, we have $B^t = 1$ and $B^t = 0$.
   - This contradiction completes the proof of this case.
4. <u>$(\to -)$</u>:
   - The induction hypothesis is $\Sigma \models A \to B$ and $\Sigma \models A$.
   - We want to prove that $\Sigma \models B$.
   - Let t be any truth valuation such that $\Sigma^t = 1$.
   - Then by the definition of tautological consequence, we have $(A \to B)^t = A^t = 1$.
   - Thus by the $\to$-rule, $B^t = 1$.
5. <u>$(\to +)$</u>:
   - The induction hypothesis is $\Sigma \cup \{A\} \models B$.
   - We want to prove that $\Sigma \models A \to B$.
   - Let t be any truth valuation such that $\Sigma^t = 1$. We have these two cases for A^t.
     * If $A^t = 0$, then by the $\to$-rule, we have $(A \to B)^t = 1$.
     * If $A^t = 1$, then $(\Sigma \cup \{A\})^t = 1$, so that, by the definition of tautological consequence, we have $B^t = 1$. Then by the $\to$-rule, we have $(A \to B)^t = 1$.

   In either case we have $(A \to B)^t = 1$.
6. <u>$(\wedge-)$</u>:
   - The induction hypothesis is $\Sigma \models (A \wedge B)$.
   - We want to prove that $\Sigma \models A$ and $\Sigma \models B$.
   - Let t be any truth valuation such that $\Sigma^t = 1$.
   - Then by the definition of tautological consequence, we have $(A \wedge B)^t = 1$.
   - Then by the $\wedge$-rule, we have $A^t = B^t = 1$.
7. <u>$(\wedge+)$</u>:
   - The induction hypothesis is $\Sigma \models A$ and $\Sigma \models B$.
   - We want to prove that $\Sigma \models (A \wedge B)$.
   - Let t be any truth valuation such that $\Sigma^t = 1$.
   - Then by the definition of tautological consequence, we have $A^t = B^t = 1$.
   - Then by the $\wedge$-rule, we have $(A \wedge B)^t = 1$.
8. <u>$(\vee-)$</u>:
   - The end of the proof then looks like:

(k)	Σ, A	$\vdash$	C	(by (?), (?))
(ℓ)	Σ, B	$\vdash$	C	(by (?), (?))
(?)	$\Sigma, (A \vee B)$	$\vdash$	C	(by ($\vee-$), (k), (ℓ))

- The induction hypotheses are
  - $\Sigma \cup \{A\} \models C$, and
  - $\Sigma \cup \{B\} \models C$.
- We want to prove that $\Sigma \cup \{A \vee B\} \models C$.
- Let t be any truth valuation such that $(\Sigma \cup (A \vee B))^t = 1$.
- Then since $(A \vee B)^t = 1$, it follows by $\vee$-rule that $A^t = 1$, $B^t = 1$, or both.
  - If $A^t = 1$, then $(\Sigma \cup \{A\})^t = 1$, so that, by the definition of tautological consequence, we have $C^t = 1$.
  - If $B^t = 1$, then $(\Sigma \cup \{B\})^t = 1$, so that, by the definition of tautological consequence, we have $C^t = 1$.

  In either case, we have $C^t = 1$.

9. $\underline{(\vee+)}$:
   - The induction hypothesis is $\Sigma \models A$.
   - We want to prove that $\Sigma \models A \vee B$ and $\Sigma \models B \vee A$.
   - Let t be a truth valuation such that $\Sigma^t = 1$.
   - Then by the induction hypothesis, we have that $A^t = 1$.
   - By $\vee$-rule, $(A \vee B)^t = (B \vee A)^t = 1$.
10. $\underline{(\leftrightarrow -)}$:
    (a) - The induction hypothesis is $\Sigma \models (A \leftrightarrow B)$ and $\Sigma \models A$.
        - We want to prove that $\Sigma \models B$.
        - Let t be a truth valuation such that $\Sigma^t = 1$.
        - Then by the definition of tautological consequence, we have $(A \leftrightarrow B)^t = A^t = 1$.
        - Then by $\leftrightarrow$ properties, $B^t = 1$.

    (b) - The induction hypothesis is $\Sigma \models (A \leftrightarrow B)$ and $\Sigma \models B$.
        - We want to prove that $\Sigma \models A$.
        - Let t be a truth valuation such that $\Sigma^t = 1$.
        - Then by the definition of tautological consequence, we have $(A \leftrightarrow B)^t = B^t = 1$.
        - Then by $\leftrightarrow$ properties, $A^t = 1$.
11. $\underline{(\leftrightarrow +)}$:
    - The induction hypothesis is $\Sigma \cup \{A\} \models B$ and $\Sigma \cup \{B\} \models A$.
    - We want to prove that $\Sigma \models A \leftrightarrow B$.
    - Let t be a truth valuation such that $\Sigma^t = 1$. We have these subcases:
      (a) $A^t = B^t = 0$. Then $(A \leftrightarrow B)^t = 1$.
      (b) $A^t = 0$ and $B^t = 1$. Then $(\Sigma \cup \{B\})^t = 1$, so that by the definition of tautological consequence, $A^t = 1$. This is a contradiction, so this case cannot occur.
      (c) $A^t = 1$ and $B^t = 0$. Then $(\Sigma \cup \{A\})^t = 1$, so that by the definition of tautological consequence, $B^t = 1$. This is a contradiction, so this case cannot occur.
      (d) $A^t = B^t = 1$. Then $(A \leftrightarrow B)^t = 1$.

In all cases that are possible, $(A \leftrightarrow B)^t = 1$.

- All induction cases are now handled. By the principle of structural induction, we are finished.

■

Soundness - Remarks on the Proof

- This proof works even if Σ is infinite (even uncountable).
- But because any individual proof includes finitely many steps, it can have only finitely many premises.
- So every proof can be written using a finite subset $\Sigma_0 \subset \Sigma$, even if Σ is infinite.

Applications of Soundness and Completeness

1. **Problem:** Prove that $\{(A \to B)\} \nvdash (B \to A)$.
   Solution: The contrapositive of soundness is: If $\Sigma \not\models C$, then $\Sigma \nvdash C$. For a counterexample, let $A = p, B = q$. Then the truth valuation $p^t = 0, q^t = 1$ witnesses that $\{(A \to B)\} \not\models (B \to A)$. So we are done.
2. **Problem:** Let Σ, C satisfy $\Sigma \models C$. Does it follow that $\Sigma \not\models (\neg C)$?
   Solution: No. For example, let $\Sigma = \{C, (\neg C)\}$. More generally, let Σ be any inconsistent set.
3. Similarly, note that, as shown below, an inconsistent set proves any formula.

9.2 The Completeness of Propositional Formal Deduction

9.2.1 Useful Ingredients

Definitions

Definition 9.2.1. *Let Σ be a set of propositional formulas in* $\text{Form}(\mathcal{L}^p)$. *We call Σ **consistent** if there exists a formula B such that $\Sigma \nvdash B$.*

Definition 9.2.2. *Let Σ be a set of propositional formulas in* $\text{Form}(\mathcal{L}^p)$. *We call Σ **consistent** if, for every Propositional formula A, if $\Sigma \vdash A$ then $\Sigma \nvdash (\neg A)$.*

Theorem 9.2.3. *Definitions 9.2.1 and 9.2.2 are equivalent.*

Proof.
- For the forward direction, suppose that there exists a formula, B, such that $\Sigma \nvdash B$.
  - Let A be any formula such that $\Sigma \vdash A$.
  - Towards a contradiction, suppose that $\Sigma \vdash (\neg A)$.
  - Then we have

    (1) $\Sigma \vdash A, (\neg A)$ (by supposition)
    (2) $A, (\neg A) \vdash B$ (by Theorem 2.6.5 [4])
    (3) $\Sigma \vdash B$ (by (Tr), (1), (2))

    This contradiction completes this direction.

- For the backward direction, suppose that there for every formula, A, if $\Sigma \vdash A$ then $\Sigma \nvdash (\neg A)$.
  - Towards a contradiction, suppose that $\Sigma \vdash B$, for every formula B.
  - Let A be any formula. By our above assumption, $\Sigma \vdash A$ and $\Sigma \vdash (\neg A)$.
  - This contradiction with the assumption completes the proof in this direction.

■

Definition 9.2.4. *Let Σ be a set of propsitional formulas in* $\mathrm{Form}(\mathcal{L}^p)$*. We call Σ **inconsistent** if Σ is not consistent.*

Tautological Consequence and Provability

- We call Σ **unsatisfiable** if **no** truth valuation t makes $\Sigma^t = 1$.
- Recall: If Σ is unsatisfiable then $\Sigma \vDash A$, for any A.
- Similarly, if Σ is inconsistent then $\Sigma \vdash A$, for any A. (Exercise: Prove it!)

Lemmas

Lemma 9.2.5. *Let Σ be a set of propositional formulas in* $\mathrm{Form}(\mathcal{L}^p)$*. Let* A *be a Propositional formula. Then $\Sigma \vDash A$ if and only if $\Sigma \cup \{(\neg A)\}$ is unsatisfiable.*

Proof. For the forward direction, assume that $\Sigma \vDash A$. Let t be any truth valuation. We have these cases for Σ^t.

- If $\Sigma^t = 1$, then because $\Sigma \vDash A$, it follows that $A^t = 1$. Hence $(\neg A)^t = 0$. Therefore $\Sigma \cup \{(\neg A)\}^t = 0$.
- If $\Sigma^t = 0$, then $\Sigma \cup \{(\neg A)\}^t = 0$.

In either case, $\Sigma \cup \{(\neg A)\}^t = 0$. Since t was arbitrary, this shows that $\Sigma \cup \{(\neg A)\}$ is unsatisfiable.

For the backward direction, assume that $\Sigma \cup \{(\neg A)\}$ is unsatisfiable. Let t be a truth valuation such that $\Sigma^t = 1$. We have these cases for A^t.

- If $A^t = 0$, then $(\neg A)^t = 1$. But then $\Sigma \cup \{(\neg A)\}^t = 1$. This contradicts the fact that $\Sigma \cup \{(\neg A)\}$ is unsatisfiable, and so this case cannot occur.
- The only remaining possibility, namely that $A^t = 1$, must occur.

This proves that $\Sigma \vDash A$. ■

Lemma 9.2.6. $\Sigma \vdash A$ *if and only if* $\Sigma \cup \{(\neg A)\}$ *is inconsistent.*

Proof. For the forward direction, assume that $\Sigma \vdash A$. Then we have

(1)	Σ	$\vdash$	A	(by supposition)
(2)	$\Sigma, (\neg A)$	$\vdash$	A	(by (+), (1))
(3)	$\Sigma, (\neg A)$	$\vdash$	$(\neg A)$	(by ($\in$))

This shows that $\Sigma \cup \{(\neg A)\}$ violates Definition 9.2.1 of being consistent.

For the backward direction, assume that $\Sigma \cup \{(\neg A)\}$ is inconsistent. Let B be any Propositional formula. Then we have

(1)	$\Sigma, (\neg A)$	$\vdash$	B	(by negation of Definition 9.2.1)
(2)	$\Sigma, (\neg A)$	$\vdash$	$(\neg B)$	(by negation of Definition 9.2.1)
(3)	Σ	$\vdash$	A	(by $(\neg -)$, (1), (2))

■

9.2.2 Proof of the Completeness Theorem (Optional)

Theorem (Completeness of Propositional Formal Deduction) 9.2.7. *If* $\Sigma \vDash A$, *then* $\Sigma \vdash A$.

Proof.
- It suffices to prove that if Σ is consistent, then Σ is satisfiable. Why?
  - This is because the contrapositive of this statement (replacing Σ with $\Sigma \cup \{(\neg A)\}$ throughout) is: if $\Sigma \cup \{(\neg A)\}$ is unsatisfiable, then $\Sigma \cup \{(\neg A)\}$ is inconsistent.
  - By Lemma 9.2.6, we can re-write this as: if $\Sigma \cup \{(\neg A)\}$ is unsatisfiable, then $\Sigma \vdash A$.
  - Rewriting this using Lemma 9.2.5, we get: if $\Sigma \vDash A$, then $\Sigma \vdash A$, the exact statement of Theorem 9.2.7.
- So to prove the Theorem, we will prove that
  $\boxed{\text{every consistent set is satisfiable.}}$
- Let Σ be an arbitrary consistent set.
- WLOG, assume that all the formulas in Σ are constructed using only $\{\neg, \wedge, \vee, \to\}$ (i.e. no $\leftrightarrow$ connectives, which can be replaced using $\to$ and $\wedge$ whenever needed). Once we finish the proof, it will be clear how to include the $\leftrightarrow$ connective if you want.
- Suppose that Σ is **countable** (i.e. Σ is finite or we can write $\Sigma = \{q_0, q_1, q_2, \ldots\}$), and assume that we can write a sequence of all the propositional formulas in Form($\mathcal{L}^p$):

$$A_0, A_1, \ldots, A_i, \ldots$$

- Now define

$$\begin{aligned} \Sigma_0 &= \Sigma \\ \Sigma_{i+1} &= \begin{cases} \Sigma_i \cup \{A_i\} & \text{if } \Sigma_i \cup \{A_i\} \text{ is consistent} \\ \Sigma_i & \text{otherwise} \end{cases}, i \geq 0 \end{aligned}$$

- Observe that each Σ_i is consistent by its construction. (Exercise: Prove it, by induction on $i \geq 0$.)
- **Remark:** The assumptions about countability are **too strong**. Σ may not be countable. We could fix this using **transfinite induction**, which is beyond the scope of this course.
- Let

$$M = \bigcup_{i=0}^{\infty} \Sigma_i.$$

- We use M to denote a "monster".

- Observe that M is consistent, by its construction. (Exercise: Prove it. If not, then some Σ_i is inconsistent, contradicting an earlier observation.)
- The idea behind the construction of M is that M should be the largest possible set that both
  - is consistent, and
  - contains Σ.
- Define a truth valuation t via $p^t = 1$ if and only if $p \in M$. This is a truth valuation, because every proposition symbol either lies in M or lies outside of M.
- I claim that $M^t = 1$, i.e. that M is satisfiable.
- This is enough since $M \supseteq \Sigma$ by construction.
- Let C be an arbitrary Propositional formula.
- Define R(C) to be the property that $C^t = 1$ if and only if $C \in M$.
- We will prove that R(C) holds for every propositional formula C in Form($\mathcal{L}^p$).
- The proof is by structural induction on C.
- We make some useful observations before giving the body of the proof.
- Observation #1: For any formula A, M contains A or $(\neg A)$ and not both (since M is consistent by construction).
- Observation #2: For any formula A, if $M \vdash A$, then $A \in M$.
  - By Observation #1, either $A \in M$, or $(\neg A) \in M$, and not both.
  - Towards a contradiction, suppose that $(\neg A) \in M$.
  - Then $M \vdash (\neg A)$.
  - This shows that M is inconsistent.
  - This contradiction completes the proof of this observation.
- Observation #3: For any formulas A and B, if $A \in M$ and $(A \to B) \in M$, then $B \in M$. Proof:
  - $A \in M$, so $M \vdash A$.
  - $(A \to B) \in M$, so $M \vdash (A \to B)$.
  - By $(\to -)$, $M \vdash B$.
  - By Observation #2, $B \in M$.
- Observation #4: If $B \in M$, then $(A \to B) \in M$, for any A. Proof:
  - Let A be arbitrary.
  - Let $B \in M$.
  - By $(\in)$, $M \vdash B$.
  - By $(+)$, $M \cup \{A\} \vdash B$.
  - By $\to +$, $M \vdash (A \to B)$.
  - By Observation #2, $(A \to B) \in M$.
- Observation #5: If $A \notin M$, then $(A \to B) \in M$, for any B. Proof:
  - Let B be arbitrary.
  - Let $A \notin M$.
  - By Observation #1, $(\neg A) \in M$.
  - By $(\in)$, $M \vdash (\neg A)$.
  - By $(+)$, $M \cup \{A\} \vdash (\neg A)$.

  - By ($\in$), $M \cup \{A\} \vdash A$.
  - Then $M \cup \{A\}$ us inconsistent.
  - Therefore $M \cup \{A\} \vdash B$.
  - Hence by ($\rightarrow +$), $M \vdash (A \rightarrow B)$.
  - By Observation #2, $(A \rightarrow B) \in M$.
- <u>Base (C is p for some proposition symbol p):</u>
  - By the construction of *t*, we then have $C^t = p^t$ which equals 1 if and only if $C \in M$.
- <u>Induction</u> We have the following sub-cases depending on the construction of C.
  - C is $(\neg A)$, for some A:
    - The induction hypothesis is that $A^t = 1$ if and only if $A \in M$.
    - For the forward direction, assume that $C^t = 1$, i.e. $(\neg A)^t = 1$.
      - Then, by $\neg$-properties, we have that $A^t = 0$.
      - By the induction hypothesis, we have that $A \notin M$.
      - By Observation #1, we have $(\neg A) \in M$.
    - For the backward direction, assume that $C \in M$, i.e. $(\neg A) \in M$.
      - By Observation #1, we have that $A \notin M$.
      - By the induction hypothesis, we have that $A^t = 0$.
      - Then, by $\neg$-properties, we have that $(\neg A)^t = 1$.
  - C is $(A \wedge B)$, for some A, B:
    - The induction hypothesis is
      - $A^t = 1$ if and only if $A \in M$, and
      - $B^t = 1$ if and only if $B \in M$.
    - For the forward direction, assume that $C^t = 1$, i.e. $(A \wedge B)^t = 1$.
      - By $\wedge$-properties, we have $A^t = 1$ and $B^t = 1$.
      - By the induction hypothesis, we have $A \in M$ and $B \in M$.
      - By $\wedge+$, we have $M \vdash (A \wedge B)$.
      - By Observation #2, we have $(A \wedge B) \in M$.
    - For the backward direction, assume that $C \in M$, i.e. $(A \wedge B) \in M$.
      - By two applications of $\wedge-$, we have $M \vdash A$ and $M \vdash B$.
      - By two applications of Observation #2, we have $A \in M$ and $B \in M$.
      - By the induction hypothesis, we have $A^t = 1$ and $B^t = 1$.
      - By $\wedge$-properties, we have $(A \wedge B)^t = 1$.
  - C is $(A \vee B)$, for some A, B:
    - The induction hypothesis is
      - $A^t = 1$ if and only if $A \in M$, and
      - $B^t = 1$ if and only if $B \in M$.
    - For the forward direction, assume that $C^t = 1$, i.e. $(A \vee B)^t = 1$.
      - By $\vee$-properties, we have $A^t = 1$ or $B^t = 1$, or both.
      - If $A^t = 1$, then by the induction hypothesis, we have $A \in M$.
      - By $\vee+$, we have $M \vdash (A \vee B)$.
      - By Observation #2, we have $(A \vee B) \in M$.
      - If $B^t = 1$, then the proof that $(A \vee B) \in M$ is similar.

    * For the backward direction, assume that $C \in M$, i.e. $(A \vee B) \in M$.
        - We are finished if we can prove that $(A \vee B)^t = 1$.
        - Towards a contradiction, suppose that $A^t = 0$ and $B^t = 0$.
        - By the induction hypothesis, we have $A \notin M$ and $B \notin M$.
        - By Observation #1, we have $(\neg A) \in M$ and $(\neg B) \in M$.
        - By $\wedge+$, $M \vdash ((\neg A) \wedge (\neg B))$.
        - By Observation #2, $((\neg A) \wedge (\neg B)) \in M$.
        - By Theorem 2.6.9 [6], $M \vdash (\neg(A \vee B))$.
        - This contradicts the fact that M is consistent.
        - Therefore we have $(A \vee B)^t = 1$, as desired.
- C is $(A \to B)$, for some A, B:
    * The induction hypothesis is
        - $A^t = 1$ if and only if $A \in M$, and
        - $B^t = 1$ if and only if $B \in M$.
    * For the forward direction, assume that $C^t = 1$, i.e. $(A \to B)^t = 1$.
        - Thus $B^t = 1$ or $A^t = 0$.
        - If $B^t = 1$, then by induction $B \in M$. By Observation #4, $(A \to B) \in M$.
        - If $A^t = 0$, then by induction $A \notin M$. By Observation #5, $(A \to B) \in M$.
    * For the backward direction, assume that $C \in M$, i.e. $(A \to B) \in M$.
        - If $A \in M$, then by induction $A^t = 1$. By Observation #3, $B \in M$. By induction, $B^t = 1$. By the properties of $\to$, we have $(A \to B)^t = 1$.
        - If $A \notin M$, then by induction $A^t = 0$. By the properties of $\to$, we have $(A \to B)^t = 1$.

■

9.3 Soundness and Completeness Revisited

Here is a different but equivalent statement of the Soundness Theorem.

Theorem (Soundness of Propositional Formal Deduction) 9.3.1. *If Σ is satisfiable, then Σ is consistent.*

Theorem 9.3.2. *Theorems 9.1.1 and 9.3.1 are equivalent.*

Proof.
- For the forward direction, assume that Σ satisfiable implies Σ consistent.
    - The contrapositive, replacing Σ by $\Sigma \cup \{(\neg A)\}$ throughout, is that $\Sigma \cup \{(\neg A)\}$ is inconsistent implies that $\Sigma \cup \{(\neg A)\}$ is unsatisfiable.
    - Via Lemma 9.2.5, we can rewrite this as $\Sigma \cup \{(\neg A)\}$ is inconsistent implies that $\Sigma \models A$.
    - Via Lemma 9.2.6, we can rewrite this as $\Sigma \vdash A$ implies that $\Sigma \models A$.
- For the backward direction, assume that $\Sigma \vdash A$ implies $\Sigma \models A$.
    - Via Lemma 9.2.6, we can rewrite this as $\Sigma \cup \{(\neg A)\}$ is inconsistent implies $\Sigma \models A$.

- Via Lemma 9.2.5, we can rewrite this as $\Sigma \cup \{(\neg A)\}$ is inconsistent implies $\Sigma \cup \{(\neg A)\}$ is unsatisfiable.
- The contrapositive (replacing $\Sigma \cup \{(\neg A)\}$ by Σ throughout) is Σ is satisfiable implies Σ is consistent.

■

As we proved already (See Theorem 9.2.7), an equivalent alternative rephrasing of the Completeness Theorem is:

Theorem (Completeness of Propositional Formal Deduction) 9.3.3. *If Σ is consistent, then Σ is satisfiable.*

10 Lecture 10 - Resolution Proof System

Outline

1. Resolution Proof System
   (a) Proving argument validity with resolution
   (b) Resolution Procedure
   (c) Soundness of Resolution
2. Set-of-Support Strategy
3. Davis-Putnam Procedure (DPP)
   (a) Soundness and Completeness of DPP (Proof Optional)

10.1 Resolution Proof System

Instructor Request: Please read the entire Logic07 slide deck in conjunction with reading these Lecture Notes.

Motivational Question: We've already started learning about Formal Deduction. Why do we need yet another proof system?

Answer: While Formal Deduction appeals to our intuition (MATH 135-style), which is desirable, a drawback of Formal Deduction is that there is $\underline{\text{no algorithm for finding proofs}}$.

On the other hand, Resolution is less intuitive, but $\underline{\text{much more amenable to automation}}$, which we like.

Resolution is actually used in industry - you may see it during a future work term if you haven't already.

10.1.1 Proving argument validity with resolution

Starting Assumption: We need formulas A, B, both written as some $\underline{\text{disjunction of literals}}$. E.g.

$$\begin{aligned} A &= p \vee q \vee \neg r \\ B &= \neg p \vee r \end{aligned}$$

To apply the **resolution rule** to A and B, there must exist a proposition symbol, which occurs in A, with its negation occurring in B (or vice versa).

In our example, A has p, while B has ¬p; similarly, A has ¬r, while B has r.

Let's work with p here. Write
A as $p \vee C$, and
B as $\neg P \vee D$,
for subformulas C, D dictated by A, B.

Then the **resolvent** of A, B is C ∨ D (i.e. the disjunction of the "left-overs" of A, B, after we resolve p with ¬p). In our example,

$$\begin{aligned} C &= q \lor \neg r, \text{ and} \\ D &= r, \text{ so that the resolvent is} \\ C \lor D &= q \lor \neg r \lor r. \end{aligned}$$

Since all input formulas must be disjunctions of literals, therefore we must convert all inputs into CNF before we can start resolution. Once we have converted our formulas into CNF, we process their disjunctive clauses, using the resolution rule.

Goal: Determine whether the starting set of clauses is satisfiable, or not.

Why we care: Fact (ingredient of soundness proof for Formal Deduction):

$$\Sigma \vDash C \text{ if and only if } \Sigma \cup \{\neg C\} \text{ is unsatisfiable.}$$

So if Resolution can (via soundness) answer questions about satisfiability, then it can answer questions about tautological consequence, i.e. about argument validity.

Two Possible Outcomes From a Resolution Refutation Proof We start with the clauses arising from a premise set, Σ, plus the negated conclusion, ¬C.

1. Resolve everything possible: arrive at the **empty clause**, ⊥.
   (a) The empty clause, ⊥, is not satisfiable.
   (b) By the soundness of resolution, this implies that the starting set of clauses was not satisfiable.
   (c) Hence the original argument was valid.
2. Resolve everything possible: arrive at the **empty set**, ∅.
   (a) The empty set, ∅, is satisfiable.
   (b) By the soundness of resolution, this implies that the starting set of clauses was satisfiable.
   (c) Hence the original argument was not valid.

10.1.2 Resolution Procedure

- **Input:** Set of disjunctive clauses $\mathcal{S} = \{D_1, D_2, \ldots, D_m\}$.
- REPEAT, trying to get the empty clause, ⊥:
  - Choose two clauses, one with p and one with ¬p, for some proposition symbol, p.
  - Resolve and call the resolvent D.
  - If D = ⊥ (i.e. if we resolve formulas p, ¬p, for some proposition symbol, p) then **output empty clause**.
  - Else add D to $\mathcal{S}$.

10.1.3 Soundness of Resolution

Theorem (Soundness of Resolution) 10.1.1. *The resolvent is tautologically implied by its parent clauses, which makes resolution a sound rule of formal deduction.*

Proof.
- Let p be a proposition symbol, and let C and D be clauses.
- Then we have that $p \vee C, \neg p \vee D \ \vdash_r \ C \vee D$.
- We want to prove that $p \vee C, \neg p \vee D \ \models \ C \vee D$.
  - (i) If at least one of C or D is not empty, then we prove:
    Claim: $p \vee C, \neg p \vee D \models C \vee D$ for any clauses C, D, not both empty.
    - Consider a truth valuation t such that $(p \vee C)^t = (\neg p \vee D)^t = 1$.
    - We have these cases for p^t:
      * If $\underline{p^t = 0}$, then $C^t = 1$, because otherwise $(p \vee C)^t = 0$.
      * Similarly, if $\underline{p^t = 1}$, then $D^t = 1$, because otherwise $(\neg p \vee D)^t = 0$.
    - In either situation, $(C \vee D)^t = 1$, therefore $p \vee C, \neg p \vee D \models C \vee D$.
    - This proves the **Claim**.
  - (ii) If both C and D are empty then, by definition, the resolvent of p and ¬p is the empty clause ⊥ (denoting $p \wedge \neg p$).
    In this case $p, \neg p \ \models \ \perp$ because the premises are contradictory. (Recall that an unsatisfiable set tautologically implies **any** formula.)
- Since $C \vee D$ (or ⊥) is the resolvent of the parent clauses $p \vee C$ and $\neg p \vee D$ on p, In both cases, (i) and (ii), the required tautological consequence holds.
- This proves the soundness of resolution.

■

Remarks:

1. The rest of this lecture is about how to choose <u>what to resolve, when</u>.

10.2 Set-of-Support Strategy

Motivation: Without some systematic way of deciding what to resolve, it is possible that we "go around in circles".

Strategy: The **set of support** is the set of formulas obtained, in some number of steps, **from the negated conclusion**.

Rule: Every resolution step must use at least one formula from the set of support.

Example (a valid argument from earlier):

$$\{((s \wedge h) \to p), s, (\neg p)\} \models (\neg h).$$

1. Convert Premises and Negated Conclusion to CNF

$$\begin{aligned}(s \wedge h) \to p &\;\boxminus\; \neg(s \wedge h) \vee p \text{ (elim } \to)\\ &\;\boxminus\; (\neg s \vee \neg h) \vee p \text{ (DML)}\\ &\;\boxminus\; \neg s \vee \neg h \vee p \text{ (Assoc)}\\ s &\;\boxminus\; s\\ \neg p &\;\boxminus\; \neg p\\ \neg\neg h &\;\boxminus\; h \text{ (double } \neg)\end{aligned}$$

2. Resolve

1.	$\neg s \vee \neg h \vee p$	premise	
2.	s	premise	
3.	$\neg p$	premise	
4.	h	negated conclusion	$SoS = \{4\}$
5.	$\neg s \vee \vee p$	resolvent: 1,4	$SoS = \{4, 5\}$
6.	$\neg s$	resolvent: 3,5	$SoS = \{4, 5, 6\}$
7.	$\perp$	resolvent: 2,6	$SoS = \{4, 5, 6, 7\}$

Since we obtained the empty clause, the set is unsatisfiable, and hence the original argument was **valid**.

10.3 Davis-Putnam Procedure (DPP)

Motivation: The Set of Support Strategy is a strategy, not an algorithm. The David-Putnam Procedure will afford us an algorithm for Resolution.

Algorithm: See Logic07, Slide 31 for the algorithm, and the explanation for the new notation for clauses.

Previous Example Re-Done Using DPP: In our new notation, we obtain the following starting set, S, of clauses:

$$\mathcal{S} = \big\{\{\neg h, p, \neg s\}, \{s\}, \{\neg p\}, \{h\}\big\}$$

Apply the Davis-Putnam procedure to find out whether or not the set of clauses is satisfiable or not. Explain how you interpret the outcome of the Davis-Putnam procedure.

1. Show in detail all the intermediary steps.
2. In particular, for each elimination of a variable, show which are the sets S_i, S_i', T_i and U_i.
3. For each resolvent indicate which are the parent clauses.
4. Eliminate the variables in lexicographic order: h, p, s.

Solution.

$$\mathcal{S}_1 = \mathcal{S}$$

1. Eliminate h:

$$\begin{aligned} \mathcal{S}_1' &= \mathcal{S}_1 \\ \mathcal{T}_1 &= \big\{\{\neg h, p, \neg s\}, \{h\}\big\} \\ \mathcal{U}_1 &= \big\{\{p, \neg s\}\big\} \\ \mathcal{S}_2 &= (\mathcal{S}_1' \setminus \mathcal{T}_1) \cup \mathcal{U}_1 \\ &= \big\{\{s\}, \{\neg p\}, \{p, \neg s\}\big\}. \end{aligned}$$

   Note, h is now eliminated.

2. Eliminate p:

$$\begin{aligned} \mathcal{S}_2' &= \mathcal{S}_2 \\ \mathcal{T}_2 &= \big\{\{\neg p\}, \{p, \neg s\}\big\} \\ \mathcal{U}_2 &= \big\{\{\neg s\}\big\} \\ \mathcal{S}_3 &= (\mathcal{S}_2' \setminus \mathcal{T}_2) \cup \mathcal{U}_2 \\ &= \big\{\{s\}, \{\neg s\}\big\}. \end{aligned}$$

   Note, p is now eliminated.

3. Eliminate s:

$$\begin{aligned} \mathcal{S}_3' &= \mathcal{S}_3 \\ \mathcal{T}_3 &= \big\{\{s\}, \{\neg s\}\big\} \\ \mathcal{U}_3 &= \{\bot\} \\ \mathcal{S}_4 &= (\mathcal{S}_3' \setminus \mathcal{T}_3) \cup \mathcal{U}_3 \\ &= \{\bot\}. \end{aligned}$$

   Note, s is now eliminated.

As we obtained the empty clause (i.e a contradiction from resolving ¬s with s) the set of clauses S is not satisfiable, and the argument from which it originated is valid.

10.3.1 Soundness and Completeness of DPP (Proof Optional)

See the proof in the slides, if you are interested in the details.

11 Lecture 11 - Introduction to First-Order Logic

Outline

1. Introduction to First-Order Logic
2. Translations Between English and First-Order Logic

11.1 Introduction to First-Order Logic

Remarks:

1. The complexity, and hence the difficulty, of the course material increases at this point.
2. Propositional Logic is simple and nice, but as a consequence, it is a bit limiting.
3. E.g. consider this argument:
   - Premises
     - All humans are mortal.
     - I am a human.
   - Conclusion Therefore I am mortal.
4. This argument "makes sense".
5. However it cannot be adequately expressed, much less proved, in Propositional Logic.
6. One reason: The first premise needs a ∀ quantifier to correctly express it.
7. Adding quantifiers is one major enhancement that we make when we move from Propositional Logic to First-Order Logic. These are the **quantifiers** ∀ and ∃, which you met in Math 135. Quantifiers have the same meanings as in MATH 135 – we will formalize their use, soon.
8. For this lecture, we will keep things informal – we will formalize later.
9. First-Order logic extends Propositional logic. Everything we already know about Propositional logic still works in First-Order Logic.

Examples:

1. "For all integers adding 0 returns the same integer", or "For every integer x, x + 0 = x" could translate to First-Order logic as

$$\big(\forall x(x + 0 = x)\big)$$

   Question from the Class: Where did "x is an integer" go in this translation?
   Answer:
   (a) We can work in the universe of integers, $\mathbb{Z}$. In this universe, this translation tells the whole story.
   (b) If the universe also contains non-integers (e.g. $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$), then we need to enhance our translation to express that x is an integer. Define

$$I(x) = \begin{cases} 1 & \text{if x is an integer} \\ 0 & \text{otherwise} \end{cases}$$

Think of I(x) as the assertion “x is an integer”. With this notation, we can re-translate as

$$\big(\forall x(I(x) \rightarrow (x + 0 = x))\big).$$

Note, the following incorrect suggestion was made in the past:

$$\big(\forall x(I(x) \land (x + 0 = x))\big).$$

Make certain that you understand why we need $\rightarrow$ and not $\land$ to correctly express the English sentence.

2. The First-Order formula

$$\big(\forall x(x \cdot 1 = x)\big)$$

or, if the universe could contain non-integers,

$$\big(\forall x(I(x) \rightarrow (x \cdot 1 = x))\big)$$

could translate to English as “For every integer x, $x \cdot 1 = x$” or “for every integer, multiplying by 1 returns the same integer”.

For these examples, a natural semantic choice (where all symbols have their usual meanings) is

- Domain (a.k.a. “Universe”) $\mathbb{Z}$ (or $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$)
  - **Remark:** In First-Order Logic, individual, function and relation symbols have **no intrinsic meanings**. These symbols get their meanings in some semantic context; in different semantic contexts, the same symbol can get **different meanings**.
- Individual symbols $\{0, 1\}$
- Function symbols $\{+^{(2)}, \cdot^{(2)}\}$
- Relation symbols $\{I^{(1)}, =^{(2)}\}$ (we could easily add $<$, $\leq$, etc.)

Question from the Class: Why does “=” belong to the relation symbols and not to the function symbols?

Answer: Fundamentally,

1. A function takes a tuple of domain elements and returns a new domain element as its output.
2. A relation takes a tuple of domain elements and returns 0 or 1 as its output. A relation stands for a statement, which is either 0 or 1 in some semantic context. We use “=” in the sense of comparison, not in the sense of assignment. As a comparison, “=” is clearly a relation symbol and not a function symbol.

Clicker Questions:

1. CQ 7

11.2 Translations Between English and First-Order Logic

From English to First-Order Logic (Examples from Logic10, Slide 39)
Translate the following propositions into formulas of first-order logic. The **domain** (universe) is the set $\mathbb{C}$ of all **complex numbers**. Use the relations

1. N(u) : u is a natural number.
2. Q(u) : u is a rational number.
3. R(u) : u is a real number.
4. Even(u) : u is even.
5. Odd(u) : u is a odd.

1. All rational numbers are real numbers.
   Ans: $(\forall x(Q(x) \rightarrow R(x)))$ (NOT $(\forall x(Q(x) \land R(x))))$
2. Not all real numbers are rational numbers.
   Ans: $(\neg(\forall x(R(x) \rightarrow Q(x)))$
3. Some real numbers are not rational numbers.
   Ans: $(\exists x(R(x) \land (\neg Q(x))))$
   Remark: We will see soon that these formulas #2 and #3 are **logically equivalent**. For intuition, think about the DeMorgan Law that would transform the first into the second.
   Moral: There are frequently multiple correct translations.
4. Every natural number is either odd or even.
   Ans: $(\forall x(N(x) \rightarrow (\mathrm{Even}(x) \lor \mathrm{Odd}(x))))$
5. No natural number is both odd and even.
   Ans: $(\neg(\exists x(N(x) \land (\mathrm{Odd}(x) \land \mathrm{Even}(x)))))$
   Alternative: $(\forall x(N(x) \rightarrow (\neg(\mathrm{Odd}(x) \land \mathrm{Even}(x)))))$

From First-Order Logic Into English
Use the same domain and relations as above. Translate the following first-order formulas into English.

1. $(\forall x(N(x) \rightarrow (\mathrm{Even}(x) \rightarrow (\neg \mathrm{Odd}(x)))))$
   Ans: If a natural number is even, then it is not odd.
2. $(\neg(\forall x R(x)))$
   Ans: Not every number is real.
3. This example goes a bit beyond the Logic10 material – however it is a good example to think about, even now. Just for this part, add a function symbol sqrt(u), which returns the (positive) square root of u.
   $(\exists x(R(x) \land (\neg R(\mathrm{sqrt}(x)))))$
   Ans: Some real numbers have non-real square roots.

12 Lecture 12 - Syntax and Semantics of First-Order Logic

Outline

12.1 Definitions

Remarks:

1. The building blocks of "First-Order Logic" are relations (a.k.a. predicates) on some set, X.

Definition 12.1.1. *Let* X *be a non-empty set. Let* $k \geq 1$. *A* k*-ary* ***relation (a.k.a. predicate)*** *on* X *is any set of* k*-tuples of elements of* X. *(*k *is the* ***arity*** *of the relation.)*

Examples:

1. Let $X = \{1, 2, 3, 4, 5\}$. Then
   (a) $\{1, 2, 3\}$ is an example of a unary relation (predicate) ($k = 1$) on X. Equivalently,
   $$\{x \in X \mid x \leq 3\}.$$
   (b) $\{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5)\}$ is an example of a binary relation (predicate) ($k = 2$) on X. Equivalently,
   $$\{(x, y) \in X \times X \mid x = y\},$$
   in other words the relation (predicate) of **equality** on X.
   [Note, $X \times X$ means all ordered pairs, with co-ordinates from X.]
   (c) $\{(1, 2, 3), (2, 2, 5), (3, 1, 2)\}$ is an example of a 3-ary relation (predicate) on X.

2. Let $X = \mathbb{Z}$. Then
$$\{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x < y\}$$
is an example of a binary relation (predicate) ($k = 2$) on $\mathbb{Z}$.
3. Let $X = \mathbb{R}$. Then the parabola
$$\{(x, y) \in \mathbb{R} \times \mathbb{R} \mid y = x^2\}$$
is an example of a binary relation (predicate) ($k = 2$) on $\mathbb{R}$. This is an example of a relation (predicate) which is also a function (think of the vertical line test).
Moral: Every k-ary function gives rise to a $(k + 1)$-ary relation.
4. Let $X = \mathbb{R}$. Then the unit circle
$$\{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x^2 + y^2 = 1\}$$
is an example of a binary relation (predicate) ($k = 2$) on $\mathbb{R}$. This is an example of a relation (predicate) which fails to be a function (think of the vertical line test).
Moral: Not every $k + 1$-ary relation arises from a k-ary function.
5. $\mathcal{D} = \mathbb{R} \times \mathbb{Z}$ is a good domain for the **floor function**. Note that the floor function is **not** an example of a binary relation (predicate), because the sets from which the two co-ordinates are taken to make the pairs are **not** the same.

Remarks:

1. With binary functions (e.g. $+, \cdot$) and predicates (e.g. $\approx, >$), we are used to writing the function or relation (predicate) symbol between its arguments. For example, we would write $3 + 6$ and $3 < 6$ (i.e. "infix" notation).
2. For consistency with the standard notation for any possible arity, we could always write the function/relation first, and its arguments after, for example

Symbol Type	Arithmetic	DrRacket	First-Order logic
function	$x + y$	$(+\ x\ y)$	sum(x, y)
function	$x \cdot y$	$(*\ x\ y)$	product(x, y)
relation	$x \approx y$	$(\approx\ x\ y)$	Equals(x, y)
relation	$x > y$	$(>\ x\ y)$	Greater(x, y)

12.2 Ingredients of First-Order Logic

The following ingredients make up the **language** in which we write our formulas of First-Order logic.

1. Individual symbols. Usually $a, b, a_1, b_2, \dots, c_1, c_2 \dots$ Sometimes $0, 1, \pi$, or anything approprate. These are constants with names.
2. Variable symbols. Usually $x, y, z, \dots x_1, x_2, \dots, y_1, y_2 \dots$
We use x, y, z for **bound** variables and u, v, w for **free** variables (we will explain this distinction soon).

3. Function symbols. Usually f, g, h, … $f_1, f_2, \dots, g_1, g_2, \dots$ Sometimes sum(u, v), or anything descriptive (comes with an arity, k)
4. Relation symbols. F, G, …F_1, F_2, …, G_1, G_2, …Sometimes Equal(u, v), or anything descriptive (comes with an arity, k)
5. Propositional Connectives. $\neg$, $\land$, $\lor$, $\rightarrow$, and $\leftrightarrow$
6. Quantifiers. $\forall$ and $\exists$
7. Punctuation. '(', ')', and ','

See pp 74-76 in the text.

Remarks:

1. Semantically, we need to interpret 1–4.
2. 5–7 always retain the same meanings.

12.3 Syntax of First-Order Logic

Motivation: What are the syntactically correct formulas of First-Order logic?

12.3.1 Terms

Remarks:

1. A term is a placeholder for a domain element.
2. Semantically, a term will evaluate to a domain element.
3. Examples of **atomic terms**: u, v, 0, 1.
4. Examples of **non-atomic terms**: $(u + v), f(u), ((u + v) + v), ((u + v) + (u + v))$.
5. Semantically, a **domain** (a.k.a. universe) is a non-empty set of elements, about which our First-Order formulas will make assertions. Some common domains are $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$, any finite set, any non-empty set.
   Q: Why does the domain need to be non-empty?
   A: Many things that we want to study "collapse" if the domain can be empty, e.g. we should be able to prove $\{(\forall xF(x))\} \vdash (\exists xF(x))$. By soundness, $\{(\forall xF(x))\} \vDash (\exists xF(x))$. If the domain can be empty, then $(\forall xF(x))$ is (vaccuously) satisfied, but $(\exists xF(x))$ is not.

Inductive Definition of Terms: Let

1. X be the set of all strings that can be written using ingredients 1-7 above,
2. A be the set of individual symbols plus free variable symbols, and
3. F contains one function per function symbol in the language:
   - For a function symbol f, of arity k, F contains a function (call it f) of arity k on the set of terms already constructed,
   - I.e. for $f^{(k)}$, and for any terms $t_1, \dots, t_k$, $f(t_1, \dots, t_k)$ is a term.

Then the set of **terms** of First-Order logic is I(X, A, F). See Definition 3.2.1 in the text.

Examples:

1. Suppose $+^{(2)}$ is our only function symbol. Suppose a is our only individual symbol, and u is our only free variable symbol. Then some terms from this setup are

$$\{a, u, (a + a), (a + u), ((a + a) + (a + u), (a + (a + u)\}.$$

Difference Between Free Variables and Individual Symbols

1. Individual Symbols: refer to (special) named constants, that don't change (individual symbol = constant symbol).
2. Free Variable Symbols: refer to **any** domain element - they are free to change. (free variable = parameter)

This will become clearer when we discuss semantics.

12.3.2 Atomic formulas

Remarks:

1. Atomic formulas play the role formerly played by proposition symbols (i.e. the simplest objects that become 0 or 1 in some semantic context).

Examples of atomic formulas:

1. S(u),
2. F(u, v),
3. (u > 0),
4. F(f(u), 0).

Definition of Atomic formulas: The set of **atomic formulas** is defined as the union of

1. $F(t_1, \dots, t_k)$, for every relation symbol F of arity k, and terms $t_1, \dots, t_k$, and
2. $\approx (t_1, t_2)$, for terms t_1 and t_2 (shorthand: $t_1 \approx t_2$).

See Definition 3.2.2 in the text.

12.3.3 General formulas

Inductive Definition of formulas: Let

1. X be the set of all strings that can be written using ingredients 1-7 above,
2. A be the set of atomic formulas .
3. Let F contain one function per Propositional connective, plus one function per quantifier:
    - the Propositional connectives behave exactly as in the Propositional case, and
    - if A(u) is a formula, and if x does not occur in A(u), then ∀xA(x) and ∃xA(x) are formulas (in both cases A(x) is called the **scope** of the quantifier - See Definition 3.2.8 in the text).

Then the set of formulas is $I(X, A, F)$. See Definition 3.2.3 in the text.

Examples:

1. All formulas involved in our translation exercises from the last lecture are built according to this recipe.

Remarks:

1. $\underline{\text{Precedence rule for quantifiers:}}$ bind the first syntactically correct formula to the immediate right of the quantifier.
2. As in the text, we do not add parentheses when introducing quantifiers, unless required by the above precedence rule.
3. The variable x in a formula of the form $\forall x A(x)$ or $\exists x A(x)$ is called **bound**. Bound variables are only for book-keeping.
4. A variable u is **free** if it is not bound.
5. A formula is a placeholder for a statement.
6. Semantically, a formula will evaluate to 0 or 1.
7. A formula is either
   (a) **simple** if it is of the form $F(t_1, \dots, t_n)$ for some n-ary relation symbol P and some terms $t_1, \dots, t_n$ (e.g. $I(u)$ asserting "u is an integer"), or
   (b) constructed from simpler formulas using connectives and/or quantifiers (e.g. $\forall x(I(x) \to (x + 0 \approx x))$).
8. Examples of non-simple formulas:
   (a) $\big(\forall x F(f(x), 0)\big)$.
   (b) $(S(u) \wedge E(u, v))$.

Questions From The Class:

1. **Q:** What is the difference between a function and a relation?
   A: Jumping ahead to semantics,
   (a) a function consumes a tuple and returns a domain element, and
   (b) a relation consumes a tuple and returns 0 or 1, depending on whether the tuple belongs to the relation or not.

12.3.4 Parse Trees

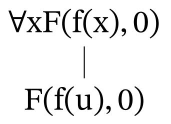
Remarks:

1. Parse trees in the First-Order Logic case are similar to those in the propositional logic case (bound variables become free as we move towards a leaf).
   (a) Every node is a formula, and
   (b) every leaf is atomic.
2. See p79 in the text.

Examples: Write the parse tree for each formula given.

1.

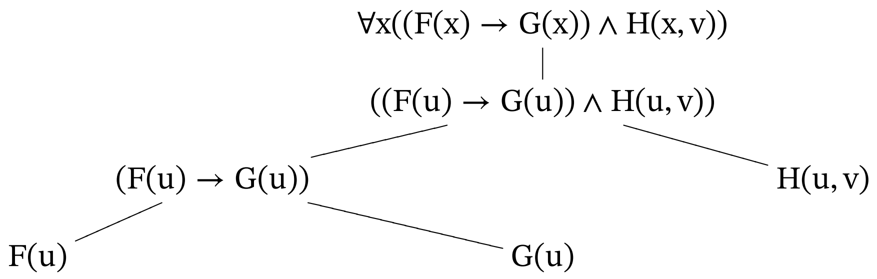
$$\forall x F(f(x), 0).$$



Remarks: Note that the variable name changes from a bound name to a free name, when we remove the quantifier.

2.

$$(\forall x((F(x) \to G(x)) \land H(x, v))).$$



See the example on p79 of the text.

12.3.5 Variables

Remarks:

1. Since all variables are (atomic) terms, therefore a variable is a placeholder for a domain element.
2. Recall:
   (a) x is **bound** in $\big(\forall x(x + 0 \approx x)\big)$, and
   (b) u is **free** in $(u + 0 \approx u)$, and
3. A formula with a **free variable** asserts something about the domain element for which the free variable stands in. For example, u is free in each of the following:
   (a) Even(u) could assert "u is even".
   (b) Greater(u, 5) could assert "$u > 5$".
   (c) (∃yGreater(u, y)) could assert "There exists y such that $u > y$".
4. Often, but not always, some choices of domain element for a **free variable** u satisfy the formula, while other choices do not.
5. Many of our examples of formulas in this course will be **closed formulas** or **sentences** (i.e. formulas with no free variables).

Examples:

1. Let A be $(u > 0)$.
   (a) A is an **atomic formula**. It is constructed using the relation symbol $>^{(2)}$, applied to the **atomic terms** u and 0.

    (b) The variable u in A is **free**.
    (c) A typical valuation, v, might choose
        i. domain: $\mathcal{D} = \mathbb{Z}$
        ii. individual symbols: 0 has its usual meaning in $\mathbb{Z}$
        iii. functions: $\varnothing$ (we can omit this line safely)
        iv. predicates: $>^{(2)}$ has its usual meaning in $\mathbb{Z}$
    (d) Note that this choice of v is **not enough** to interpret A, because A has a free variable, u. An **assignment** is needed to complete the interpretation of A.
2. Let B be $(\forall x(x > 0))$.
    (a) The x in B is bound.
    (b) Since B is a **sentence**, therefore the interpretation v suffices to interpret B.
    (c) Indeed B is 0 under v, since $-3 \in \mathbb{Z}$, and $(-3 > 0)$ is 0.
3. Let C be $(\exists x(x > 0))$.
    (a) The x in C is bound.
    (b) Since C is a **sentence**, therefore the interpretation v suffices to interpret C.
    (c) Indeed C is 1 under v, since $7 \in \mathbb{Z}$, and $(7 > 0)$ is 1.
4. If there are no free variables, then v suffices to interpret a formula.
5. In the First-Order formula $\forall x(ryF(x, y, w))$, the x and y variables are **bound** and the w variable is **free**. This formula is **not** a **sentence**, since it contains a free variable.
6. In the First-Order formula $(\forall x(\exists yF(x, y, a)))$ (where a is an individual symbol), the x and y variables are **bound**. This formula is a **sentence**, since it contains no free variables.

12.4 Semantics of First-Order Logic

Motivation: How do we assign 0 or 1 to each syntactically correct formula of First-Order logic?

12.4.1 Domains

Definition 12.4.1. *A **domain** $\mathcal{D}$ is a non-empty set.*

Examples: $\{1, 2, 3, 4, 5\}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$, { all humans on planet earth }, etc.

12.4.2 Interpretations

Definition 12.4.2. *An **interpretation** selects a domain $\mathcal{D}$, then maps*

- *each individual symbol to a domain element,*
- *each* k*-ary function symbol to a total function $\mathcal{D}^k \to \mathcal{D}$,*
- *each* k*-ary relation symbol to a* k*-ary relation on $\mathcal{D}$.*

The text does not define any notation for an interpretation. The text only defines notation for a valuation, later.

What It Means For A Function To Be Total: Consider a function f^v, of arity k, defined on a domain $\mathcal{D}$.

- This means that we have
$$f^v \colon \underbrace{\mathcal{D} \times \cdots \times \mathcal{D}}_{\text{k copies}} \to \mathcal{D},$$
where the inputs to the function come from $\mathcal{D}^k$, the k-fold product of copies of $\mathcal{D}$.
- To say that f^v is a **total** function of arity k defined on $\mathcal{D}$ means that
  - every tuple $(d_1, \dots, d_k) \in \mathcal{D}^k$ is a legal input to f^v, and
  - $f^v(d_1, \dots, d_k) \in \mathcal{D}$ for every tuple $(d_1, \dots, d_k) \in \mathcal{D}^k$.

12.4.3 Assignments

Definition 12.4.3. *An **assignment** uses a selected a domain $\mathcal{D}$ from an interpretation, then maps*

- *each (free) variable symbol to a domain element.*

The text does not define any notation for an assignment. The text only defines notation for a valuation, later.

12.4.4 Valuations

Remarks:

1. A valuation v, makes all needed choices of a semantic context such that, under v, we can interpret any
   (a) term, t, as t^v, making it a domain element, and
   (b) formula, A, as A^v, making it 0 or 1.

Definition 12.4.4. *A **valuation** v is an interpretation plus an assignment.*

Notation:

Given	Notation	Means
individual symbol a	a^v	domain element to which v maps a
free variable symbol u	u^v	domain element to which v maps u
k-ary function symbol f	f^v	total function $\mathcal{D}^k \to \mathcal{D}$ to which v maps f
k-ary relation symbol F	F^v	relation on $\mathcal{D}^k$ to which v maps F

12.4.5 Terms

Definition 12.4.5. *Fix a valuation v. For each term* t*, the domain element obtained from interpreting* t *under v, denoted t^v, is as follows.*

- *If* t *is an individual symbol* a*, then* t^v *is* a^v.
- *If* t *is a free variable* u*, then* t^v *is* u^v.
- *If* t *is* $f(t_1, \dots, t_k)$*, then* t^v *is* $f^v(t_1^v, \dots, t_k^v)$.

See p88 in the text.

Remarks:

1. Recall that a **term** is a placeholder for a domain element.
2. Hence when evaluated under some valuation, a term evaluates to a domain element:

Proposition 12.4.6. *Let* t *be any term of First-Order logic. Let* v *be any valuation, with domain* $\mathcal{D}$*. Then* $\mathrm{t}^v \in \mathcal{D}$.

Proof. The proof is by structural induction on t and is left as an exercise. ■

Lemma 12.4.7. *Let* t *be any term of First-Order Logic. Let* v_1, v_2 *be any valuations, over the same domain* $\mathcal{D}$*, and such that*

- $\mathrm{a}^{v_1} = \mathrm{a}^{v_2}$*, for every individual symbol* a *in* t*,*
- $\mathrm{u}^{v_1} = \mathrm{u}^{v_2}$*, for every free variable symbol* u *in* t*,*
- $\mathrm{f}^{v_1} = \mathrm{f}^{v_2}$*, for every function symbol* f *in* t*, and*
- $\mathrm{F}^{v_1} = \mathrm{F}^{v_2}$*, for every relation symbol* F *in* A.

Then

$$\mathrm{t}^{v_1} = \mathrm{t}^{v_2}.$$

Proof. The proof is by structural induction on t and is left as an exercise. ■

12.4.6 Formulas

A Notation Needed for Quantified formulas To evaluate quantified formulas , we need a way to "override" a valuation on a free variable.

Definition 12.4.8. *Let*

- v *be a valuation,*
- $\mathcal{D}$ *be its domain,*
- $\alpha \in \mathcal{D}$*, and*
- u *be a free variable.*

Then define a new valuation

$$v(\mathrm{u}/\alpha)$$

which is the same as v*, except that* $\mathrm{u}^{v(\mathrm{u}/\alpha)} = \alpha$*. In other words,* $v(\mathrm{u}/\alpha)$ *is* v*, with the evaluation of* u *"overridden" to* α*.*

See p88 in the text.

We can compose overrides as many times as needed, e.g. we can form

$$v(\mathrm{u}/\alpha)(\mathrm{w}/\beta)$$

Definition 12.4.9.
- *We write* $\mathrm{A}^v = 1$ *to indicate that a valuation* v ***satisfies*** *a formula* A.
- *We write* $\mathrm{A}^v = 0$ *to indicate that a valuation* v ***does not satisfy*** *a formula* A.

Form of A	Condition for $A^v = 1$
$F(t_1, \dots, t_k)$	$(t_1^v, \dots, t_k^v) \in F^v$
$(\neg B)$	$B^v = 0$
$(B \wedge C)$	$B^v = 1$ *and* $C^v = 1$
$(B \vee C)$	$B^v = 1$ *or* $C^v = 1$ *(or both)*
$(B \rightarrow C)$	$B^v = 0$ *or* $C^v = 1$ *(or both)*
$(B \leftrightarrow C)$	$B^v = C^v$
$\forall xB$	*for every* $\alpha \in \text{domain}(v)$, $B(u)^{v(u/\alpha)} = 1$
$\exists xB$	*there is some* $\alpha \in \text{domain}(v)$, *such that* $B(u)^{v(u/\alpha)} = 1$

See p89 in the text.

Lemma 12.4.10. *Let* A *be any formula of First-Order Logic. Let* v_1, v_2 *be any valuations, over the same domain* $\mathcal{D}$, *and such that*

- $a^{v_1} = a^{v_2}$, *for every individual symbol* a *in* A,
- $u^{v_1} = u^{v_2}$, *for every free variable symbol* u *in* A,
- $f^{v_1} = f^{v_2}$, *for every function symbol* f *in* A, *and*
- $F^{v_1} = F^{v_2}$, *for every relation symbol* F *in* A.

Then

$$A^{v_1} = A^{v_2}.$$

Proof. The proof is by Structural Induction on A, and is left as an exercise. See Lemma 12.4.7 for one ingredient. ■

12.4.7 Satisfiability and Validity

Validity and satisfiability of formulas have definitions analogous to the ones for propositional logic.

Definition 12.4.11. *A First-Order formula* A *is*

1. ***valid*** *if every valuation v satisfies* A*; that is, if* $A^v = 1$ *for every v,*
2. ***satisfiable*** *if some valuation v satisfies* A*; that is, if* $A^v = 1$ *for some v, and*
3. ***unsatisfiable*** *if no valuation v satisfies* A*; that is, if* $A^v = 0$ *for every v.*

Remarks:

1. In the past some students have suggested that we should shorthand " for every v" in part 1 of Definition 12.4.11 as $\forall v$. I strongly recommend against doing this. The universes over which $\forall$x inside A and $\forall v$ in the definition are making their assertions are **very different**. Thus confusion is likely if you write $\forall v$ here. The idea behind this suggestion points towards **second-order logic**, an enhancement to the first-order logic (a.k.a. Predicate Logic) which we are studying here.
2. The term "tautology" is not used in first-order logic.

Examples

1. Let A be the First-Order Formula $\exists x(F(x) \to G(x))$. Let $\mathcal{D} = \{a, b\}$.
   (a) **Problem:** Create a valuation v_1 over $\mathcal{D}$ such that $A^{v_1} = 1$.
   Solution: Define v_1 to select

$$\begin{aligned} F^{v_1} &= \{a\}, \text{ and} \\ G^{v_1} &= \{a, b\}. \end{aligned}$$

   Then we have $(F(u) \to G(u))^{v_1(u/a)} = 1$, so that, by $\exists$-satisfaction rule, $\exists x(F(x) \to G(x))^{v_1} = 1$.
   Remarks:
   i. $v_1(u/b)$ also witnesses that A is satisfied.
   ii. Other choices for F^{v_1}, G^{v_1} also work, e.g. $F^{v_1} = \emptyset, G^{v_1} =$ anything.
   iii. With the above choice for v_1 and the additional observation that $(F(u) \to G(u))^{v_1(u/b)} = 1$, we see that $(\forall x(F(x) \to G(x)))^{v_1} = 1$ too.
   iv. See also the next example.
   (b) **Problem:** Create a valuation v_2 over $\mathcal{D}$ such that $A^{v_2} = 0$.
   Solution: Define v_2 to select

$$\begin{aligned} F^{v_2} &= \{a, b\}, \text{ and} \\ G^{v_2} &= \emptyset. \end{aligned}$$

   Then we have

$$\begin{aligned} (F(u) \to G(u))^{v_2(u/a)} &= 0, \text{ and} \\ (F(u) \to G(u))^{v_2(u/b)} &= 0. \end{aligned}$$

   This shows that no domain element satisfies the inner formula, so that, by $\exists$-satisfaction rule, $\exists x(F(x) \to G(x))^{v_2} = 0$.
   Remarks:
   (a) Part a shows that A is satisfiable.
   (b) Part b shows that A is not valid.
2. Let B be the First-Order formula $\forall x(F(x) \to G(x))$. Let $\mathcal{D} = \{a, b\}$.
   (a) **Problem:** Create a valuation v_1 over $\mathcal{D}$ such that $B^{v_1} = 1$.
   Solution: Define v_1 to select

$$\begin{aligned} F^{v_1} &= \{a\}, \text{ and} \\ G^{v_1} &= \{a, b\}. \end{aligned}$$

   Then we have

$$\begin{aligned} (F(u) \to G(u))^{v_1(u/a)} &= 1, \text{ and} \\ (F(u) \to G(u))^{v_1(u/b)} &= 1, \end{aligned}$$

so that, by ∀-satisfaction rule, $\forall x(F(x) \rightarrow G(x))^{\nu_1} = 1$.

(b) **Problem:** Create a valuation ν_2 over $\mathcal{D}$ such that $B^{\nu_2} = 0$.
Solution: Define ν_2 to select

$$\begin{aligned} F^{\nu_2} &= \{a, b\}, \text{ and} \\ G^{\nu_2} &= \emptyset. \end{aligned}$$

Then we have

$$(F(u) \rightarrow G(u))^{\nu_2(u/a)} = 0,$$

so that, by ∀-satisfaction rule, $\forall x(F(x) \rightarrow G(x)))^{\nu_2} = 0$.

Remarks:

(a) Part a shows that B is satisfiable.
(b) Part b shows that B is not valid.

13 Lecture 13 - Logical Consequence

Outline

1. Logical Consequence
2. Subtleties of Logical Consequence
3. Examples of Logical Consequence
4. Translations from English into First-Order Logic
5. Translations from First-Order Logic into English
6. More Examples of Logical Consequence

13.1 Logical Consequence

Definition 13.1.1. *let* Σ *be a set of formulas of First-Order logic. Let* v *be a valuation. We say that* v ***satisfies*** Σ *(Notation:* $\Sigma^v = 1$*), if and only if* $A^v = 1$*, for every formula* $A \in \Sigma$*. Otherwise we say that* v ***does not satisfy*** Σ *(Notation:* $\Sigma^v = 0$*).*

Definition 13.1.2. *Suppose* Σ *is a set of First-Order formulas and* A *is a First-Order formula. We say that* Σ ***(logically) implies*** A*, (Notation:* $\Sigma \vDash A$*), if and only if, for every valuation* v*, we have*

$$\Sigma^v = 1 \textit{ implies } A^v = 1.$$

13.2 Subtleties of Logical Consequence

All the subtleties from Propositional Logic are still present here.

1. **The empty set $\emptyset$ is satisfied under any valuation,** v.
2. Therefore if $\emptyset \vDash C$, then C is a valid formula.
3. If Σ is not satisfiable, then $\Sigma \vDash C$, for **any** C.

13.3 Examples of Logical Consequence

1. **Example**: Show that

   $$\emptyset \vDash ((\forall x(A \to B)) \to ((\forall x A) \to (\forall x B))),$$

   i.e. prove that the formula $((\forall x(A \to B)) \to ((\forall x A) \to (\forall x B)))$ is **valid**, for **any** First-Order formulas A and B.
   - Towards a contradiction, suppose that there is a valuation v such that

     $$((\forall x(A \to B)) \to ((\forall x A) \to (\forall x B)))^v = 0.$$

   - Then by the $\to$-satisfaction rule, we must have $(\forall x(A \to B))^v = 1$ and $((\forall x A) \to (\forall x B))^v = 0$.
   - The second fact (again by the $\to$-satisfaction rule) gives $(\forall x A)^v = 1$ and $(\forall x B)^v = 0$.

- By the ∀-satisfaction rule, we have
  for every $a \in \text{domain}(\nu)$, $(A(u) \to B(u))^{\nu(u/a)} = 1$ and $A(u)^{\nu(u/a)} = 1$.
  (In simplified notation: for every $a \in \text{domain}(\nu)$, $(A(a) \to B(a))^{\nu} = 1$ and $A(a)^{\nu} = 1$; I will use the proper notation throughout, whereas the slides use the simplified notation.)
- Thus also $B(u)^{\nu(u/a)} = 1$ for every $a \in \text{domain}(\nu)$.
- Thus $(\forall x B)^{\nu} = 1$, a contradiction.

2. **Example:** Show that $\{(\forall x(\neg C))\} \vDash (\neg(\exists x C))$.
   - Suppose that $(\forall x(\neg C))^{\nu} = 1$.
   - By ∀-satisfaction rule, this means
     for every $d \in \text{domain}(\nu)$, $(\neg C(u))^{\nu(u/d)} = 1$.
   - By ¬-satisfaction rule, this is equivalent to
     for every $d \in \text{domain}(\nu)$, $C(u)^{\nu(u/d)} = 0$
     in other words
     there does not exist $d \in \text{domain}(\nu)$ such that $C(u)^{\nu(u/d)} = 1$.
   - This last is the definition of $(\neg(\exists x C))^{\nu} = 1$, as required.

   Remarks:
   (a) The tautological consequence in the other direction could be proved similarly.
   (b) Hence these two formulas are **logically equivalent**.

3. **Example:** Show that, in general,

$$\{((\forall x A) \to (\forall x B))\} \nvDash (\forall x(A \to B)).$$

   (That is, exhibit a choice of A and B such that the logical consequence does not hold, and prove that your choice is correct.)
   Key idea: $(B \to C)$ yields true whenever B is false.
   Let A be F(x). Let ν have domain $\{a, b\}$ and $F^{\nu} = \{a\}$. Then $((\forall x A) \to (\forall x B))^{\nu} = 1$ for any B. (Why?)
   - The reason why $((\forall x A) \to (\forall x B))^{\nu} = 1$ for any B:
     - We have $A(u)^{\nu(u/b)} = 0$.
     - This shows that $(\forall x A)^{\nu} = 0$.
     - Then we have $((\forall x A) \to (\forall x B))^{\nu} = 1$ for any B, the →-satisfaction rule.

   To obtain $(\forall x(A \to B))^{\nu} = 0$, we can use $(\neg F(x))$ for B. (Why?)
   - The reason why, to obtain $(\forall x(A \to B))^{\nu} = 0$, we can use $B = (\neg F(u))$:
     - Let $B = (\neg F(u))$.
     - Then $A(u)^{\nu(u/a)} = 1$ and $B(u)^{\nu(u/a)} = 0$.
     - Therefore $(A \to B)^{\nu(u/a)} = 0$.
     - This shows that $\forall x(A \to B)^{\nu} = 0$.

   Thus $\{((\forall x A) \to (\forall x B))\} \nvDash (\forall x(A \to B))$, as required. (Why?)
   - Just apply Definition 13.1.2, using the previous two facts.

4. **Example:** Prove the following Proposition.
   Proposition 13.3.1. *Let* A *be any First-Order formula* ***without*** *a free variable* u. *Let* ν *be any valuation. Then* $A^{\nu} = (\forall x A)^{\nu}$.

Proof. Let $\mathcal{D}$ be the domain of ν. Since u is not free in A, therefore $w^{\nu} = w^{\nu(w/a)}$, for every $a \in \mathcal{D}$ and for every w that occurs free in A.
Then by the Lemma 12.4.10, we have that $A^{\nu} = A^{\nu(w/a)}$, for any $a \in \mathcal{D}$, which establishes the desired result. ■

13.4 Translations from English into First-Order Logic

Problems: Translate the following English sentences into First-Order Logic. Use the following notation.

- S(u): u is a student.
- F(u): u is an professor.
- C(u): u is a course.
- M(u): u belongs to Math.
- U(u): u belongs to Computer Science.
- E(u, v): u is enrolled in v.
- T(u, v): u teaches v.

1. w is a course.
   Solution:
   $$C(w)$$
2. There exists a student y who is enrolled in a course u.
   Solution:
   $$\big(\exists y(S(y) \wedge (C(u) \wedge E(y, u)))\big)$$
3. Every Computer Science course is a Math course.
   Solution:
   $$\big(\forall x((C(x) \wedge U(x)) \to (C(x) \wedge M(x)))\big)$$
4. Not every Math course belongs to Computer Science.
   Solution:
   $$(\neg\, \big(\forall x((C(x) \wedge M(x)) \to U(x))\big))$$
   Or (logically equivalent)
   $$\big(\exists x(C(x) \wedge (M(x) \wedge (\neg U(x))))\big)$$
5. There is a student and there is a course such that the student is enrolled in the course.
   Solution:
   $$\Big(\exists x\,\big(\exists y(S(x) \wedge (C(y) \wedge E(x, y)))\big)\Big)$$
6. There is a student who is enrolled in some Computer Science course.
   Solution:
   $$\Big(\exists x\,\big(\exists y(S(x) \wedge (C(y) \wedge (U(y) \wedge E(x, y))))\big)\Big)$$
   Or (logically equivalent)
   $$\Big(\exists y\,\big(\exists x(S(x) \wedge (C(y) \wedge (U(y) \wedge E(x, y))))\big)\Big)$$

7. Some professor does not teach any course.
   Solution:
$$\big(\exists x(F(x) \land \big(\forall y(C(y) \to (\neg T(x,y))))\big)\big)$$
   Or (logically equivalent)
$$\big(\exists x\,\big(\forall y(F(x) \land (C(y) \to (\neg T(x,y))))\big)\big)$$
   Remarks:
   (a) Make certain you understand why we need $\to$ and not $\land$ here!
8. A professor cannot be a student.
   Solution:
$$\big(\forall x(F(x) \to (\neg S(x)))\big)$$
   Or (logically equivalent)
$$(\neg\,\big(\exists x(F(x) \land S(x))\big))$$

13.5 Translations from First-Order Logic into English

Problems: Translate the following First-Order Logic sentences into English. Use the same notation from the last batch of examples.

1. $$\big(\exists x(S(x) \land \big(\forall y(C(y) \to (\neg E(x,y))))\big)\big)$$
   Solution: Some student is not enrolled in any course.
2. $$\big(\exists x((C(x) \land U(x)) \land \big(\forall y((C(y) \land U(y)) \to (y = x)))\big)\big)$$
   Solution: There is exactly one Computer Science course.
3. $$(F(u) \land \big(\forall y((C(y) \land M(y)) \to T(u,y))\big))$$
   Solution: Professor u teaches every Math course.
4. $$((F(u) \land C(v)) \land T(u,v))$$
   Solution: Professor u teaches course v.

13.6 More Examples of Logical Consequence

1. **Problem:** Prove that
$$\{\forall x F(x)\} \vDash \exists x F(x).$$
   Solution:
   - Let v be any valuation such that $\forall x F(x)^v = 1$.
   - Let $\mathcal{D}$ be the domain of v.

- Let $d \in \mathcal{D}$ be arbitrary. By definition, $\mathcal{D}$ is non-empty, so d always exists.
- By the ∀-satisfaction rule, we have

$$\mathrm{F(u)}^{\nu(\mathrm{u}/d)} = 1.$$

- Then by the ∃-satisfaction rule, we have

$$\exists \mathrm{xF(x)}^{\nu} = 1.$$

- Since ν was arbitrary, this completes the proof.

2. **Problem:** Prove that

$$\{\exists \mathrm{xF(x)}\} \not\models \forall \mathrm{xF(x)}.$$

Solution:

- We need to exhibit a valuation, ν, such that $\exists \mathrm{xF(x)}^{\nu} = 1$ and $\forall \mathrm{xF(x)}^{\nu} = 0$.
- Let ν be the valuation defined by
  - domain $\mathcal{D} = \{\mathrm{a}, \mathrm{b}\}$
  - $\mathrm{F}^{\nu} = \{\mathrm{a}\}$
- We have that

$$\mathrm{F(u)}^{\nu(\mathrm{u/a})} = 1.$$

- Then by the ∃-satisfaction rule, we have

$$\exists \mathrm{xF(x)}^{\nu} = 1.$$

  However we also have that

$$\mathrm{F(u)}^{\nu(\mathrm{u/b})} = 0.$$

- Therefore by the ∀-satisfaction rule, we have

$$\forall \mathrm{xF(x)}^{\nu} = 0.$$

14 Lecture 14 - First-Order Formal Deduction I

Outline

1. Proof Rules for Quantified formulas
2. Useful Theorems
3. Examples

14.1 Proof Rules for Quantified formulas

Remarks:

1. In this lecture we extend Formal Deduction for Propositional Logic to create the Formal Deduction proof system for First-Order Logic.
2. As in the Propositional case, proofs in First-Order Natural Deduction are 100% syntactic, 0% semantic.
3. All the proof rules for Formal Deduction for Propositional Logic work as before.
4. We will see during the next lecture that this new proof system is both **sound** and **complete**.

Proof Rules:

1. $(\forall -)$:
   If $\Sigma \vdash \forall x A(x)$,
   then $\Sigma \vdash A(t)$, for any term t.
2. $(\forall +)$:
   If $\Sigma \vdash A(u)$, u not occurring in Σ,
   then $\Sigma \vdash \forall x A(x)$.
3. $(\exists -)$:
   If $\Sigma, A(u) \vdash B$, u not occurring in Σ or B,
   then $\Sigma, \exists x A(x) \vdash B$.
4. $(\exists +)$:
   If $\Sigma \vdash A(t)$,
   then $\Sigma \vdash \exists x A(x)$, where A(x) results by replacing some (not necessarily all) occurrences of t in A(t) by x.
5. $(\approx -)$:
   If $\Sigma \vdash A(t_1)$,
   $\Sigma \vdash t_1 \approx t_2$,
   then $\Sigma \vdash A(t_2)$, where $A(t_2)$ results by replacing some (not necessarily all) occurrences of t_1 in $A(t_1)$ by t_2.
6. $(\approx +)$:
   $\varnothing \vdash u \approx u$.

Remarks:

1. To carefully define the ∀+ rule, we need to prove our formula A holds for an arbitrary

domain element u **with no additional assumptions about** u. If such an argument involves a formula in which a variable u is free, then it may impose additional undesired conditions on u. For example, if the argument involves Bird(u), then the conclusion, A, may hold only for domain elements which are birds, not necessarily to all domain elements.

2. Consider the example below:

$$\Sigma = \left\{\forall x F(x), \exists z G(u, z)\right\}.$$

   If G is the < relation, then the formula $\exists z G(u, z)$ asserts that "u is not the maximal element of the domain". If the desired conclusion A(u) is connected with u being maximal in the domain, then the presence of the earlier formula might tempt us to assume too much about u.
3. Similarly, $\exists-$ demands that u not occur in Σ or in B, so that we can safely conclude that Σ plus the ∃-formula suffices to establish B (without accidentally making additional assumptions).

14.2 Useful Theorems

Theorem (Replacement of equivalent formulas) 14.2.1. *Let* $A, B, C \in \text{Form}(\mathcal{L})$ *with* $B \vdash\dashv C$. *Let* A' *result from* A *by substituting some (not necessarily all) occurrences of* B *by* C. *Then* $A' \vdash\dashv A$.

- This is Theorem 3.5.10 in the text.
- It can be proved by structural induction on A.

Theorem (Complementation) 14.2.2. *Suppose* A *is a formula composed of atoms of* $\mathcal{L}$, *the connectives* ¬, ∨, ∧ *and the two quantifiers by the formation rules concerned, and* A' *is the formula obtained by exchanging* ∨ *and* ∧, ∃ *and* ∀, *and negating all atoms. Then* $A' \vdash\dashv \neg A$.

- This is Theorem 3.5.11 in the text.
- It can be proved by structural induction on A.

14.3 Examples

1. Prove that $\forall x F(x) \vdash \exists x F(x)$.
   Solution:

(1)	$\forall x F(x)$	$\vdash$	$\forall x F(x)$	(by (∈))
(2)	$\forall x F(x)$	$\vdash$	$F(t)$	(by (∀−), (1))
(3)	$\forall x F(x)$	$\vdash$	$\exists x F(x)$	(by (∃+), (2))

2. Prove that $\exists x F(x) \nvdash \forall x F(x)$.
   Solution: We will need the (contrapositive of) soundness of First-Order Formal Deduction to prove this fact, next time.
3. Prove

$$\emptyset \vdash \forall x(F(x) \to F(x))$$

Solution:

(1)	$F(u)$	$\vdash$	$F(u)$	(by $(\in)$)
(2)	$\varnothing$	$\vdash$	$(F(u) \to F(u))$	(by $(\to +)$, (1))
(3)	$\varnothing$	$\vdash$	$\forall x(F(x) \to F(x))$	(by $(\forall +)$, (2))
				(u not in $\varnothing$)

4. Prove

$$\{\forall x(F(x) \to G(x)), \forall x F(x)\} \vdash \forall x G(x)$$

Solution:

(1)	$\forall x(F(x) \to G(x)), \forall x F(x)$	$\vdash$	$\forall x(F(x) \to G(x))$	(by $(\in)$)
(2)	$\forall x(F(x) \to G(x)), \forall x F(x)$	$\vdash$	$(F(u) \to G(u))$	(by $(\forall -)$, (1))
(3)	$\forall x(F(x) \to G(x)), \forall x F(x)$	$\vdash$	$\forall x F(x)$	(by $(\in)$)
(4)	$\forall x(F(x) \to G(x)), \forall x F(x)$	$\vdash$	$F(u)$	(by $(\forall -)$, (3))
(5)	$\forall x(F(x) \to G(x)), \forall x F(x)$	$\vdash$	$G(u)$	(by $(\to -)$, (2), (4))
(6)	$\forall x(F(x) \to G(x)), \forall x F(x)$	$\vdash$	$\forall x G(x)$	(by $(\forall +)$, (5))
				(u not in $\forall x(F(x) \to G(x))$ or in $\forall x F(x)$)

5. Modify the proof in part 4 to show

$$\{\forall x(F(x) \to G(x)), \forall x F(x)\} \vdash \exists x G(x)$$

Solution: Exercise.

6. Prove

$$\{(\forall x F(x) \to \exists x H(x)), \forall x(F(x) \land G(x))\} \vdash \exists x H(x)$$

Solution: Exercise.

7. Prove

$$\{\exists x(F(x) \lor G(x))\} \vdash (\exists x F(x) \lor \exists x G(x))$$

Solution:

(1)	$F(u)$	$\vdash$	$F(u)$	(by $(\in)$)
(2)	$F(u)$	$\vdash$	$\exists x F(x)$	(by $(\exists +)$, (1))
(3)	$F(u)$	$\vdash$	$(\exists x F(x) \lor \exists x G(x))$	(by $(\lor +)$, (2))
(4)	$G(u)$	$\vdash$	$G(u)$	(by $(\in)$)
(5)	$G(u)$	$\vdash$	$\exists x G(x)$	(by $(\exists +)$, (4))
(6)	$G(u)$	$\vdash$	$(\exists x F(x) \lor \exists x G(x))$	(by $(\lor +)$, (5))
(7)	$(F(u) \lor G(u))$	$\vdash$	$(\exists x F(x) \lor \exists x G(x))$	(by $(\lor -)$, (3), (6))
(8)	$\exists x(F(x) \lor G(x))$	$\vdash$	$(\exists x F(x) \lor \exists x G(x))$	(by $(\exists -)$, (7))
				(u not in $\varnothing$ or in $(\exists x F(x) \lor \exists x G(x))$)

8. Prove

$$\{\exists x(P(x) \land Q(x))\} \vdash (\exists x P(x) \land \exists x Q(x))$$

Solution: Exercise.

9. Prove

$$\{\forall x(F(x) \to G(x)), \exists x F(x)\} \vdash \exists x G(x)$$

Solution:

(1)	$\forall x(F(x) \to G(x)), F(u)$	$\vdash$	$F(u)$	(by $(\in)$)
(2)	$\forall x(F(x) \to G(x)), F(u)$	$\vdash$	$\forall x(F(x) \to G(x))$	(by $(\in)$
(3)	$\forall x(F(x) \to G(x)), F(u)$	$\vdash$	$(F(u) \to G(u))$	(by $(\forall-)$, (2))
(4)	$\forall x(F(x) \to G(x)), F(u)$	$\vdash$	$G(u)$	(by $(\to -)$, (1), (3))
(5)	$\forall x(F(x) \to G(x)), F(u)$	$\vdash$	$\exists x G(x)$	(by $(\exists+)$, (4))
(6)	$\forall x(F(x) \to G(x)), \exists x F(x)$	$\vdash$	$\exists x G(x)$	(by $(\exists-)$, (5)) (u not in $\forall x(F(x) \to G(x))$ or in $\exists x G(x)$)

10. Prove

$$\{\exists y \forall x F(x, y)\} \vdash \forall x \exists y F(x, y)$$

Solution:

(1)	$\forall x F(x, w)$	$\vdash$	$\forall x F(x, w)$	(by $(\in)$)
(2)	$\forall x F(x, w)$	$\vdash$	$F(u, w)$	(by $(\forall-)$, (1))
(3)	$\forall x F(x, w)$	$\vdash$	$\exists y F(u, y)$	(by $(\exists+)$, (2))
(4)	$\exists y \forall x F(x, y)$	$\vdash$	$\exists y F(u, y)$	(by $(\exists-)$, (3)) (w not in $\varnothing$ or in $\exists y F(u, y)$)
(5)	$\exists y \forall x F(x, y)$	$\vdash$	$\forall x \exists y F(x, y)$	(by $(\forall+)$, (4)) (u not in $\exists y \forall x F(x, y)$)

11. Prove

$$\{(\forall x F(x) \lor \forall x G(x))\} \vdash \forall x(F(x) \lor G(x))$$

Solution: Exercise.

12. Prove

$$\varnothing \vdash \forall x((F(x) \to G(x)) \lor (G(x) \to F(x))).$$

You may use derived rules.
Solution: Exercise.

13. Prove

$$\{(a \approx b), (\forall x(F(x) \to G(x)), F(a)\} \vdash G(b)$$

Solution: Let $\Sigma = \{(a \approx b), \forall x(F(x) \to G(x)), F(a)\}$. Then we have

(1)	Σ	$\vdash$	$\forall x(F(x) \to G(x))$	(by $(\in)$)
(2)	Σ	$\vdash$	$F(a)$	(by $(\in)$)
(3)	Σ	$\vdash$	$(a \approx b)$	(by $(\in)$)
(4)	Σ	$\vdash$	$(F(a) \to G(a))$	(by $(\forall-)$, (1))
(5)	Σ	$\vdash$	$G(a)$	(by $(\to -)$, (2), (4))
(6)	Σ	$\vdash$	$G(b)$	(by $(\approx -)$, (3), (5))

DeMorgan Laws for Quantified Formulas

1. $(\neg \forall x A(x)) \vdash\dashv \exists x(\neg A(x))$ (Proved on Slide 29 on Logic14)
2. $(\neg \exists x A(x)) \vdash\dashv \forall x(\neg A(x))$ (Exercise on Slide 36 on Logic14)

Via soundness, we can rewrite both, replacing $\vdash\dashv$ with $\models\!\dashv$.

15 Lecture 15 - Individual Study Time

16 Lecture 16 - First-Order Formal Deduction II - Soundness and Completeness

Outline

1. Soundness
   (a) Rule $\forall-$ Is Sound
   (b) Rule $\exists+$ Is Sound
   (c) Rule $\forall+$ Is Sound
   (d) Rule $\exists-$ Is Sound
   (e) Rule $\approx -$ Is Sound
   (f) Rule $\approx +$ Is Sound
2. Completeness
3. More Examples
4. Introduction to Resolution for First-Order Logic

16.1 Soundness

Theorem 16.1.1. *Formal Deduction for First-Order Logic is* **sound**, *i.e. whenever* $\Sigma \vdash A$, *it follows that* $\Sigma \models A$.

Proof (Outline). Because Propositional Formal Deduction is sound, therefore it suffices to prove the soundness of $\forall-, \exists+, \forall+, \exists-, \approx -$ and $\approx +$.

16.1.1 Rule $\forall-$ Is Sound

Theorem 16.1.2. *The* $\forall-$ *proof rule is sound.*

Proof. Recall the $\forall-$ proof rule:
If $\quad \Sigma \vdash \forall x A(x)$,
then $\quad \Sigma \vdash A(t)$, for any term t.

- Suppose that $\Sigma \models \forall x A(x)$.
- We are finished if we can prove that $\Sigma \models A(t)$, for any term t.
- Let t be any First-Order term.
- Let v be any valuation such that $\Sigma^v = 1$.
- Let $\mathcal{D}$ be the domain of v.
- Then by our hypothesis $\forall x A(x)^v = 1$.
- By Proposition 12.4.6, $t^v \in \mathcal{D}$.
- By $\forall$-satisfaction, $A(u)^{v(u/t^v)} = 1$.
- In other words, $A(t)^v = 1$.

■

16.1.2 Rule ∃+ Is Sound

Theorem 16.1.3. *The ∃+ proof rule is sound.*

Proof. Recall the ∃+ proof rule:

If	Σ	$\vdash$	A(t),
then	Σ	$\vdash$	∃xA(x), where A(x) results by replacing some (not necessarily all) occurrences of t in A(t) by x.

- Suppose that $\Sigma \vDash$ A(t), for some First-Order term t.
- We are finished if we can prove that $\Sigma \vDash \exists$xA(x), where A(x) results by replacing some (not necessarily all) occurrences of t in A(t) by x.
- Let v be any valuation such that $\Sigma^v = 1$.
- Let $\mathcal{D}$ be the domain of v.
- Suppose that A(u) results by replacing some (not necessarily all) occurrences of t in A(t) by u.
- Then by our hypothesis $\mathrm{A(u)}^{v(\mathrm{u}/\mathrm{t}^v)} = 1$.
- By Proposition 12.4.6, $\mathrm{t}^v \in \mathcal{D}$.
- By ∃-satisfaction, $\exists\mathrm{xA(x)}^v = 1$.

■

16.1.3 Rule ∀+ Is Sound

Theorem 16.1.4. *The ∀+ proof rule is sound.*

Proof. Recall the ∀+ proof rule:

If	Σ	$\vdash$	A(u), u not occurring in Σ,
then	Σ	$\vdash$	∀xA(x).

- Suppose that $\Sigma \vDash$ A(u), u not occurring in Σ.
- We are finished if we can prove $\Sigma \vDash \forall$xA(x).
- Let v be any valuation such that $\Sigma^v = 1$.
- Let $\mathcal{D}$ be the domain of v.
- Let $d \in \mathcal{D}$ be arbitrary.
- I claim that Lemma 12.4.10 implies that $\Sigma^{v(\mathrm{u}/d)} = \Sigma^v$.
- Let B $\in \Sigma$ be arbitrary.
- It suffices to prove that $\mathrm{B}^{v(\mathrm{u}/d)} = \mathrm{B}^v$.
- To apply Lemma 12.4.10 for formula B, we need to verify that $\mathrm{w}^{v(\mathrm{u}/d)} = \mathrm{w}^v$, for every free variable w in B.
- Because u does not occur in Σ, therefore w $\neq$ u (i.e. w and u are different free variable symbols).
- Hence by the definition of $v(\mathrm{u}/d)$, it is clear that $\mathrm{w}^{v(\mathrm{u}/d)} = \mathrm{w}^v$.
- Therefore Lemma 12.4.10 applies as stated.
- To summarize, $\Sigma \vDash$ A(u) and $\Sigma^{v(\mathrm{u}/d)} = \Sigma^v = 1$.

- Hence $A(u)^{v(u/d)} = 1$.
- Since $d \in \mathcal{D}$ was arbitrary, therefore by $\forall$-satisfaction, $\forall x A(x)^v = 1$.

■

16.1.4 Rule $\exists-$ Is Sound

Theorem 16.1.5. *The $\exists-$ proof rule is sound.*

Proof. Recall the definition of the $\exists-$ proof rule:

If	$\Sigma, A(u)$	$\vdash$	B, u not occurring in Σ or B,
then	$\Sigma, \exists x A(x)$	$\vdash$	B.

- Suppose that $\Sigma, A(u) \vDash B$, u not occurring in Σ or B.
- We are finished if we can prove that $\Sigma, \exists x A(x) \vDash B$.
- Let v be an arbitrary valuation such that $\Sigma^v = 1$ and $\exists x A(x)^v = 1$.
- Let $\mathcal{D}$ be the domain of v.
- There is some $d \in \mathcal{D}$ such that $A(u)^{v(u/d)} = 1$.
- Since u does not occur in Σ, therefore (similarly to the $\forall+$ case) $\Sigma^{v(u/d)} = \Sigma^v = 1$.
- Then since $\Sigma^{v(u/d)} = 1$ and $A(u)^{v(u/d)} = 1$, we have that $B^{v(u/d)} = 1$.
- Since u does not occur in B, therefore (similarly to the $\forall+$ case) $B^v = B^{v(u/d)} = 1$.

■

16.1.5 Rule $\approx -$ Is Sound

Theorem 16.1.6. *The $\approx -$ proof rule is sound.*

Proof. Recall the definition of the $\approx -$ proof rule:

If	Σ	$\vdash$	$A(t_1)$,
	Σ	$\vdash$	$t_1 \approx t_2$,
then	Σ	$\vdash$	$A(t_2)$, where $A(t_2)$ results by replacing some (not necessarily all) occurrences of t_1 in $A(t_1)$ by t_2.

- Suppose that $\Sigma \vDash A(t_1)$, and $\Sigma \vDash t_1 \approx t_2$.
- We are finished if we can prove $\Sigma \vDash A(t_2)$, where $A(t_2)$ results by replacing some (not necessarily all) occurrences of t_1 in $A(t_1)$ by t_2.
- Let v be any valuation such that $\Sigma^v = 1$.
- Then by our hypotheses, $A(t_1)^v = 1$ and $(t_1 \approx t_2)^v = 1$, in other words $t_1^v = t_2^v$.
- Then because $A(t_2)$ results by replacing some (not necessarily all) occurrences of t_1 in $A(t_1)$ by t_2, it follows that $A(t_2)^v = 1$.

■

16.1.6 Rule $\approx +$ Is Sound

Theorem 16.1.7. *The $\approx +$ proof rule is sound.*

Proof. Recall the definition of the $\approx +$ proof rule:
$\emptyset \quad \vdash \quad u \approx u.$

- We must prove that $\emptyset \models u \approx u$, i.e. that $(u \approx u)^{v} = 1$, for every valuation v.
- The above can be re-written as $u^{v} = u^{v}$, which clearly always holds.

■

16.2 Completeness

- The main ingredient in the proof of completeness is the same as in the Propositional case, namely that **every consistent set of formulas of First-Order logic is satisfiable**.
- We had to wave our hands to prove Completeness in the simpler Propositional case. In the First-Order case, things become so complicated that we cannot prove completeness even with hand-waving.
- We will state without proof that whenever $\Sigma \models A$, we also have that $\Sigma \vdash A$.

16.3 More Examples

1. **Problem:** Prove that $\{\exists x F(x)\} \nvdash \forall x F(x)$.
   Solution:
   - We showed earlier that $\{\exists x F(x)\} \not\models \forall x F(x)$.
   - By the contrapositive of soundness, we have $\{\exists x F(x)\} \nvdash \forall x F(x)$.

16.4 Introduction to Resolution for First-Order Logic

Remarks:

1. Many technical details are required here, which I will defer explaining fully until the next lecture.
2. I will do everything correctly, despite not explaining yet why everything is correct.
3. If possible, read all of the Logic15 slide deck before the next lecture.

Brief Explanation:

1. The resolution proof rule is the same in FOL as in propositional logic:

$$\{C \vee A, \neg A \vee D\} \vdash_{\text{Res}} C \vee D,$$

   where
   (a) A is an atomic formula, and
   (b) C and D are disjunctive clauses.

2. Like in the propositional case, we must convert our formulas into CNF before we start resolution.
3. By the time we start resolution, we must have "gotten rid" of all quantifiers:
   (a) $\underline{\forall:}$ Everything in sight is (quietly) $\forall$-quantified, so we won't write $\forall$ explicitly.
   (b) $\underline{\exists:}$
      i. If at the front, replace $\exists$y something(y) by something(a), for some individual symbol, a. This a is a **Skolem constant**.
      ii. If following one or more $\forall$ quantifiers, then replace $\forall$x$\exists$y something(x) something–else(y) by $\forall$x something(x) something – else(f(x)) (one of f argument per $\forall$). This f(x) is a **Skolem function**.

Examples:

1. Prove the following argument by Resolution.
   (a) Premise 1: $\exists y \forall x F(x, y)$
   (b) Conclusion: $\forall x \exists y F(x, y)$

   Solution:
   (a) Convert Premises Plus Negated Conclusion
      i. Premise 1:

$$\begin{aligned} & \exists y \forall x F(x, y) \\ \text{gives} \quad & \forall x F(x, a) \end{aligned}$$

      ii. Negated Conclusion:

$$\begin{aligned} & \neg \forall x \exists y F(x, y) \\ \vDash\!\!\!\!\Dashv \quad & \exists x \neg \exists y F(x, y) \text{ (DeMorgan)} \\ \vDash\!\!\!\!\Dashv \quad & \exists x \forall y \neg F(x, y) \text{ (DeMorgan)} \\ \text{gives} \quad & \forall y \neg F(b, y) \text{ (Skolem constant for x)} \end{aligned}$$

   (b) Resolve

1.	$F(x, a)$	premise
2.	$\neg F(b, y)$	negated conclusion
3.	$F(b, a)$	1 with $x := b$
4.	$\neg F(b, a)$	2 with $y := a$
5.	$\perp$	resolvent: 3,4

   (c) Lines 3 and 4 are **unification**, which allows us to set $\forall$-quantified variables to whatever we like. We set $\forall$-quantified variables to facilitate resolution in the resulting formulas.
2. Prove ths following argument by Resolution.
   (a) Premise 1: $\forall x(F(x) \rightarrow \exists y G(y))$
   (b) Premise 2: $\exists x F(x)$
   (c) Conclusion: $\exists y G(y)$

   Solution:

(a) Convert Premises Plus Negated Conclusion

i. Premise 1:

$$\begin{array}{rl} & \forall x(F(x) \rightarrow \exists y G(y)) \\ \boxminus & \forall x \exists y(F(x) \rightarrow G(y)) \text{ (move quantifier)} \\ \boxminus & \forall x \exists y(\neg F(x) \lor G(y)) \text{ (remove implication)} \\ \text{gives} & \forall x(\neg F(x) \lor G(f(x))) \text{ (Skolem function for y)} \end{array}$$

ii. Premise 2:

$$\begin{array}{rl} & \exists x F(x) \\ \text{gives} & F(a) \text{ (Skolem constant for x)} \end{array}$$

iii. Negated Conclusion:

$$\begin{array}{rl} & \neg \exists y G(y) \\ \boxminus & \forall y \neg G(y) \text{ (DeMorgan)} \end{array}$$

(b) Resolve

1.	$(\neg F(x) \lor G(f(x)))$	premise
2.	$F(a)$	premise
3.	$\neg G(y)$	negated conclusion
4.	$(\neg F(a) \lor G(f(a)))$	1 with $x := a$
5.	$G(f(a))$	resolvent: 2,4
6.	$\neg G(f(a))$	3 with $y := f(a)$
7.	$\bot$	resolvent: 5,6

17 Lecture 17 - First-Order Resolution

Outline

1. Resolution Proof Rule
2. Prenex Normal Form
3. ∃-Free Prenex Normal Form
4. Unification
5. Examples
6. Soundness And Completeness

17.1 Resolution Proof Rule

The resolution proof rule is the same in FOL as in propositional logic:

$$\{C \vee A, \neg A \vee D\} \vdash_{\text{Res}} C \vee D,$$

where

1. A is an atomic formula, and
2. C and D are disjunctive clauses.

17.2 Prenex Normal Form

Definition. A formula is in **prenex normal form** if it is of the form

$$\underbrace{Q_1 x_1\ Q_2 x_2 \cdots Q_n x_n}_{\text{prefix}}\ \underbrace{B}_{\text{matrix}}$$

where $n \geq 1$, Q_i is $\forall$ or $\exists$, for $1 \leq i \leq n$, and B is quantifier free.

Convention. If $n = 0$ (i.e. no quantifiers) then the formula is trivially in prenex normal form.

In our examples from the last lecture, we first turned every input formula into prenex normal form, by moving quantifiers to the front, using appropriate logical equivalences.

Algorithm for converting a formula in *Form*($\mathcal{L}$) **into prenex normal form**

Any formula in *Form*($\mathcal{L}$) is logically equivalent to (and can be converted into) a formula in prenex normal form (PNF). To find its logically equivalent formula in PNF:

1. Eliminate all occurrences of → and ↔ from the formula.
2. "Move all negations inward" such that, in the end, negations only appear as part of literals (**literal = an atom, or its negation**).
3. Standardize the variables apart (definition to follow), when necessary.
4. The prenex normal form can now be obtained by "moving" all quantifiers to the front of the formula.

In the following, we will describe the logical equivalences that can be used to accomplish the steps above.

Step 1 (**eliminate** $\rightarrow, \leftrightarrow$):

1. $A \rightarrow B \models\!\!\!\mid \neg A \vee B$.
2. $A \leftrightarrow B \models\!\!\!\mid (\neg A \vee B) \wedge (A \vee \neg B)$.
3. $A \leftrightarrow B \models\!\!\!\mid (A \wedge B) \vee (\neg A \wedge \neg B)$.

Step 2 (**move all negations inward, such that negations only appear as parts of literals**):

1. De Morgan's Laws.
2. Double negation: $\neg\neg A \models\!\!\!\mid A$.
3. $\neg \exists x A(x) \models\!\!\!\mid \forall x \neg A(x)$.
4. $\neg \forall x A(x) \models\!\!\!\mid \exists x \neg A(x)$.

Step 3

1. Recall that the symbol denoting a bound variable is just a place holder, and two occurrences of a symbol x in a formula do not necessarily refer to the same bound variable. For example, in $\forall x(A(x) \vee B(x)) \vee \exists x C(x)$, the first two occurrences of x refer to the variable in the scope of $\forall$, while the last occurence of x refers to a distinct variable, in the scope of $\exists$.
2. Renaming the variables in a formula such that distinct bound variables (variables bound by distinct quantifiers) have distinct names is called **standardizing the variables apart**.
3. To accomplish **Step 3** (standardize variables apart), we use the following theorem, which allows us to rename bound variables.

Theorem. **(Replaceability of bound variable symbols)** Let A be a formula in *Form*($\mathcal{L}$). Suppose that A' results from A by replacing in A some (not necessarily all) occurrences of $QxB(x)$ by $QyB(y)$, where $Q \in \{\forall, \exists\}$. Then $A \models\!\!\!\mid A'$ and $A \vdash\!\!\dashv A'$.

Example: Standardizing Variables Apart in $\forall x F(x) \wedge \exists x G(x)$ could lead to $\forall x_1 F(x_1) \wedge \exists x_2 G(x_2)$. The variable names and scopes are different now - there will be no possibility of confusion later.

Step 4 (**move all quantifiers to the front of the formula**) (Slide 10):

1. $A \wedge \exists x B(x) \models\!\!\!\mid \exists x(A \wedge B(x))$, x not occurring in A.
2. $A \wedge \forall x B(x) \models\!\!\!\mid \forall x(A \wedge B(x))$, x not occurring in A.
3. $A \vee \exists x B(x) \models\!\!\!\mid \exists x(A \vee B(x))$, x not occurring in A.
4. $A \vee \forall x B(x) \models\!\!\!\mid \forall x(A \vee B(x))$, x not occurring in A.

These equivalences essentially show that if a formula A has a truth value that does not depend on x, then one is allowed to quantify, using any quantifier, over x.

More logical equivalences for Step 4 (Slide 11):

1. $\forall x A(x) \wedge \forall x B(x) \models\!\!\!\mid \forall x(A(x) \wedge B(x))$.
2. $\exists x A(x) \vee \exists x B(x) \models\!\!\!\mid \exists x(A(x) \vee B(x))$.

3. $\forall x \forall y A(x, y) \mathrel{\text{⟛}} \forall y \forall x A(x, y)$.
4. $\exists x \exists y A(x, y) \mathrel{\text{⟛}} \exists y \exists x A(x, y)$.
5. $Q_1 x A(x) \wedge Q_2 y B(y) \mathrel{\text{⟛}} Q_1 x Q_2 y(A(x) \wedge B(y))$, (x not occurring in B(y), and y not occurring in A(x)).
6. $Q_1 x A(x) \vee Q_2 y B(y) \mathrel{\text{⟛}} Q_1 x Q_2 y(A(x) \vee B(y))$, (x not occurring in B(y), and y not occurring in A(x)).

where $Q_1, Q_2 \in \{\forall, \exists\}$.

For example, under the conditions above, if $Q_1 = \forall$ and $Q_2 = \exists$, e.g.,

$$\begin{aligned} \forall x A(x) \wedge \exists y B(y) &\mathrel{\text{⟛}} \forall x \exists y(A(x) \wedge B(y)) \\ &\mathrel{\text{⟛}} \exists y B(y) \wedge \forall x A(x) \\ &\mathrel{\text{⟛}} \exists y \forall x(B(y) \wedge A(x)). \end{aligned}$$

This **only holds** if x does not occur in B(y), y does not occur in A(x).

Examples: See Slides.

17.3 ∃-Free Prenex Normal Form

Definition A sentence (formula without free variables) $A \in \mathit{Sent}(\mathcal{L})$ is said to be in **∃-free prenex normal form** if it is in prenex normal form and does not contain existential quantifier symbols.

Consider a sentence of the form $\forall x_1 \forall x_2 \cdots \forall x_n \exists y A$ where $n \geq 0$, and A is an expression, possibly involving other quantifiers.

1. Note that $\exists y A$ generates at least one individual for each n-tuple $(a_1, a_2, \ldots, a_n)$ in the domain.
2. In other words, the individual generated by $\exists y A$ is a function of $x_1, \ldots, x_n$, which can be expressed by using $f(x_1, x_2, \ldots, x_n)$.
3. The function f is called a **Skolem function**.
4. The function symbol for a Skolem function is a new function symbol, which must **not occur anywhere in** A.

Skolem functions for removal of ∃

- The skolemized version of $\forall x_1 \forall x_2 \cdots \forall x_n \exists y A$ is

$$(*) \qquad \forall x_1 \forall x_2 \cdots \forall x_n A'$$

  where $n \geq 0$, and A' is the expression obtained from A by substituting each occurrence of y by $f(x_1, x_2, \ldots, x_n)$.
  We did this in our second example, last time.

- **Example:** Let the domain be $\mathbb{Z}$, and consider $\forall x \exists y(x + y = 0)$. Each instance of x, say $x = d$, $d \in \mathbb{Z}$, generates a corresponding $y = -d$ that makes the formula true.
  - If we define $f(x) = -x$, we have that the skolemized version of the formula is $\forall x(x + f(x) = 0)$.
  - More generally, in $\forall x \exists y F(x, y)$, one has a different value of y generated, for each value of x. The skolemized version of $\forall x \exists y F(x, y)$ is $\forall x F(x, g(x))$.
  - Here, g(x) is the Skolem function "generating" a value $y = g(x)$, for each value of x.
- **Remark:** It can happen than $n = 0$, i.e. the formula has the shape $\exists yA$ (no $\forall$s at the front). A Skolem funtion, with arity 0, is just a constant, called a **Skolem constant**. In this special subcase, replace $\exists yA$ with A', where A' is constructed from A by replacing each y with an individual symbol, a.
  We did this in both examples, last time.

Algorithm for an $\exists$-free prenex normal form

- **Step 1.** Transform the input sentence $A_0 \in \mathit{Sent}(\mathcal{L})$ into a logically equivalent sentence A_1 in prenex normal form.
  Set $i = 1$.
- **Step 2.** Repeat until all the existential quantifiers are removed.
  - Assume A_i is of the form $A_i = \forall x_1 \forall x_2 \cdots \forall x_n \exists y A$ where A is an expression, possibly involving quantifiers.
  - If $n = 0$, then A_i is of the form $\exists yA$. Then $A_{i+1} = A'$, where A' is obtained from A by replacing all occurrences of y by the individual symbol c, where c is a symbol not occurring in A_i.
  - If $n > 0$, $A_{i+1} = \forall x_1 \forall x_2 \ldots \forall x_n A'$, where A' is the expression obtained from A by replacing all occurrences of y by $f(x_1, x_2, \ldots, x_n)$, where f is a new function symbol.
  - Increase i by 1.

17.4 Unification

In resolution we aim to reach the empty clause $\perp$ (a **contradiction**).

1. In propositional logic, it is impossible to derive a contradiction from a set of formulas, unless the same variable occurs more than once.
2. For instance, there is no way to derive a contradiction from the two formulas $p \wedge q \vee r$ and $\neg s$. The two formulas do not share variables, and the truth of the first has no bearing on the truth of the second.
3. Similarly, in first-order logic, one cannot derive a contradiction from two formulas A and B, unless A and B share complementary literals (literal = atom or its negation).
4. To obtain complementary literals, we may have to use a procedure called **unification**.
5. **Definition.** An **instantiation** is an assignment to a variable x_i of a quasi-term t_i' (defined as either an individual symbol, or a variable symbol, or a function symbol applied to individual symbols or variable symbols). We write $x_i := t_i'$.
6. **Definition.** Two formulas in first-order logic are said to **unify** if there are instantiations

that make the formulas in question identical. The act of unifying is called **unification**. The instantiation that unifies the formulas in question is called a **unifier**.
We unified in our examples from the last lecture.

17.5 Examples

From Logic15, Slides 36-38 Use resolution to prove that a relation $H \subseteq D \times D$ is reflexive if it is transitive and symmetric. (Assume that every individual of *D* is related, via H, to at least one other individual in *D*.)

Solution. Define H(x, y) as "x **is related to** y".

1. Every individual of *D* is related, via H, to at least one other individual in *D*: $\forall x \exists y H(x, y)$.
2. The relation H is transitive: $\forall x \forall y \forall z (H(x, y) \land H(y, z) \to H(x, z))$.
3. The relation H is symmetric: $\forall x \forall y (H(x, y) \to H(y, x))$.
4. We want to conclude from these premises that the relation H is reflexive, that is, $\forall x H(x, x)$.

To prove the theorem by resolution (which is essentially "proof by contradiction"), we negate the conclusion

$$\neg \forall x H(x, x) \mathrel{\text{⟚}} \exists x \neg H(x, x).$$

Pre-processing the input

1. $\underline{\text{Non-triviality assumption for H:}}$ $\forall x \exists y H(x, y)$ yields the formula in $\exists$-free PNF $\forall x H(x, f(x))$, where f is a Skolem function.
2. $\underline{\text{transitivity:}}$ $\forall x \forall y \forall z (H(x, y) \land H(y, z) \to H(x, z))$ yields:

$$\begin{aligned} & \forall x \forall y \forall z (H(x, y) \land H(y, z) \to H(x, z)) \\ \mathrel{\text{⟚}} \quad & \forall x \forall y \forall z (\neg (H(x, y) \land H(y, z)) \lor H(x, z)) \text{ (elim } \to) \\ \mathrel{\text{⟚}} \quad & \forall x \forall y \forall z (\neg H(x, y) \lor \neg H(y, z)) \lor H(x, z)) \text{ (DML)} \end{aligned}$$

3. $\underline{\text{symmetry:}}$ $\forall x \forall y (H(x, y) \to H(y, x))$ yields

$$\begin{aligned} & \forall x \forall y (H(x, y) \to H(y, x)) \\ \mathrel{\text{⟚}} \quad & \forall x \forall y (\neg H(x, y) \lor H(y, x)) \text{ (elim } \to) \end{aligned}$$

4. $\underline{\text{conclusion, reflexivity:}}$ $\forall x H(x, x)$. Negating this yields:

$$\begin{aligned} & \neg \forall x H(x, x) \\ \mathrel{\text{⟚}} \quad & \exists x \neg H(x, x) \text{ (DML)} \end{aligned}$$

which we concert into the formula in $\exists$-free PNF $\neg H(a, a)$, where a is an individual symbol (a Skolem function with zero arguments).

Remark: This example does not require standardizing variables apart.

By dropping all universal quantifiers we obtain the **clauses**:

$$H(x, f(x))$$
$$\neg H(x, y) \vee \neg H(y, z) \vee H(x, z)$$
$$\neg H(x, y) \vee H(y, x)$$
$$\neg H(a, a).$$

Performing resolution

1. $H(x, f(x))$	from premise 1
2. $\neg H(x, y) \vee \neg H(y, z) \vee H(x, z)$	from premise 2
3. $\neg H(x, y) \vee H(y, x)$	from premise 3
4. $\neg H(a, a)$	from the negation of conclusion
5. $\neg H(a, y) \vee \neg H(y, a) \vee H(a, a)$	2, $x := a, z := a$
6. $\neg H(a, y) \vee \neg H(y, a)$	resolve 4, 5
7. $H(a, f(a))$	1 with $x := a$
8. $\neg H(a, f(a)) \vee \neg H(f(a), a)$	6 with $y := f(a)$
9. $\neg H(f(a), a)$	resolve 7, 8
10. $\neg H(a, f(a)) \vee H(f(a), a)$	3, $x := a, y := f(a)$
11. $\neg H(a, f(a))$	resolve 9, 10
12. $\perp$	resolve 7, 11

Remarks:

1. We essentially followed a "set-of-support" approach, to decide what to resolve.
2. There is **no** algorithm in FOL, which is analogous to DPP in propositional logic. There are **too many** choices of unification for such an algorithm to exist.
3. You will get some practice at your FOL resolution skills, on A04.

17.6 Soundness and Completeness

First-order Resolution is Sound and Complete.

18 Lecture 18 - Turing Machines I

Outline

1. Introduction to Decidability
2. Decision Problems
   (a) Decidable
   (b) Undecidable
3. The Undecidability of the Halting Problem
4. Reductions
5. Examples

18.1 Introduction to Decidability

1. Intutively, something is **computable** if it can be calculated by a systematic procedure (aka an **algorithm**).
2. We say that a function is **computable** if there is an algorithm that a computer could execute to compute it.

Remarks:

1. Decidable problems and computable functions are closely connected.
2. For more details about decidable problems and computable functions, take CS 360/365 later on.
3. One might believe that, given enough resources and a sufficiently sophisticated program, a computer could solve any problem. However, this is not the case: $\underline{\text{undecidable problems exist}}$.

18.2 Decision Problems

Recall Definition 1.3.1

Definition 18.2.1. *A decision problem is* ***decidable*** *if there is an algorithm that, given an input to the problem,*

- *outputs yes (1) if the input has answer yes, and*
- *outputs no (0) if the input has answer no.*

A decision problem is ***undecidable*** *if it is not decidable.*

Remarks:

1. The algorithm **must always complete after finitely many steps**. If there is even one case in which the candidate algorithm will not finish in finitely many steps, then it is not actually an algorithm.
2. To prove that a decision problem is decidable, **write down an algorithm to decide it**.

3. A decision problem may be undecidable, and yet have particular choices of input for which the correct yes/no answer can be determined (e.g. validity of a First-Order formula). The existence of a special choice of input for which the question can be decided does not contradict the general result of undecidability. To say that a decision problem is undecidable is to say that no algorithm exists to give the correct yes/no answer **for every input**.

Examples:

1. These examples are **decidable**:
   (a) Given a formula A of Propositional logic, is A satisfiable?
      - **A:** The algorithm should be obvious.

   (b) **A Variation:** Given a formula A of Propositional logic, is A valid?
      - **A:** The algorithm should be obvious.

   (c) Given a positive integer n, is n prime?
      - **A:** An algorithm was given in Math 135.
2. By the end of the lecture, we will show that this example is undecidable:
   (a) Given a program `P`, and an input `I`, will program `P` terminate when run with input `I`? (the **Halting Problem**)

18.2.1 Decidable

- Many nice decision problems can be stated in terms of the question of membership in some set.
- Lots of non-trivial examples exist for subsets $S \subseteq \mathbb{N}$, where $\mathbb{N}$ denotes the Natural numbers.

Definition 18.2.2. *Let* $S \subseteq \mathbb{N}$ *be any subset. The* S-***membership problem*** *asks*

for an arbitrary $x \in \mathbb{N}$*, is* $x \in S$*?*

Definition 18.2.3. *A set* $S \subseteq \mathbb{N}$ *is called* ***decidable (resp undecidable)*** *if the* S-*membership problem is decidable (resp undecidable).*

Fact: Some Ss are decidable; others are undecidable.

Examples:

1. Suppose that $S \subseteq \mathbb{N}$ is decidable. Prove that the **complement**, $S^c = \mathbb{N} \backslash S = \{n \in \mathbb{N} \mid n \notin S\}$, is decidable.
   Proof: The following algorithm decides membership in S^c:
   (a) Let $x \in \mathbb{N}$ be given.
   (b) Use the decider for S to determine whether $x \in S$.
      - If $x \in S$ then return "no".
      - If $x \notin S$ then return "yes".

   Our constructed algorithm must finish in finitely many steps. It clearly gives the right answer. So we are done.

2. Suppose that S ⊂ ℕ is finite. Does it follow that S is decidable?
    - **A**: Yes. The required algorithm is obvious.
3. Suppose that S ⊂ ℕ is infinite. Does it follow that S is undecidable?
    - **A**: No. For a counterexample, let S be $\{y \in \mathbb{N} \mid y \equiv 0 \bmod 2\}$, the subset of even numbers. The algorithm to decide membership in S is obvious.
4. For examples of Ss which are undecidable, take CS 360/365.

18.2.2 Undecidable

1. Proving that a decision problem is undecidable is more difficult than proving that it is decidable.
2. Proving that no algorithm can exist to solve an arbitrary decision problem is not straightforward, using the definition alone.
3. To make such proofs rigourous, we will need to use a model of a computer called a **Turing Machine**.

18.3 The Undecidability of the Halting Problem

Theorem 18.3.1. *The Halting Problem is Undecidable.*

From now on, assume that all of our programs consume and produce natural numbers.

We will outline a proof of Theorem 18.3.1.

Why We Care: We use the undecidability of the Halting Problem to prove that other decision problems are undecidable.

Proof. • `https://www.youtube.com/watch?v=92WHN-pAFCs`
- We won't do the detailed proof in class.
- See the slides for the details.
- The proof in not examinable; you absolutely need to know the result however.

■

Remarks:

1. The desired contradiction occurs somewhere on the diagonal of this table:

inputs / programs	0	1	2	⋯
0	*			
1		*		
2			*	
⋮				

18.4 Reductions

A **reduction** is a technique for using the undecidability of the Halting Problem to argue that a new decision problem is undecidable.

Definition 18.4.1. *1. Suppose we have two decision problems,* P_1 *and* P_2*.*

2. *Suppose also that we have an* ***algorithm*** A*, that transforms inputs for (instances of)* P_1 *into inputs for (instances of)* P_2*, such that:*
   (a) *"Yes" instances of* P_1 *get mapped to "yes" instances of* P_2*,*
   (b) *"No" instances of* P_1 *get mapped to "no" instances of* P_2*, and*
   (c) *the algorithm* A *always takes finite time.*
3. *Then* A ***reduces*** P_1 *to* P_2*, and that* A *is a* ***reduction*** *from* P_1 *to* P_2*.*

Theorem 18.4.2. *If there is a reduction from* P_1 *to* P_2*, then*

1. *If* P_1 *is undecidable, then* P_2 *is also undecidable.*
2. *If* P_2 *is decidable, then* P_1 *is also decidable.*

Proof. See the slides. ■

Remarks:

1. The second statement is not really needed, as it is simply the contrapositive of the first. We state it explicitly because it can provide us with another means of proving that a decision problem is decidable.

Our Strategy For Proving Undecidability:

1. We will use part 1 of the Theorem, with P_1 being the Halting Problem, and P_2 being the new, unknown decision problem.
2. This week's tutorial notes will show some examples of this technique.
3. Occasionally, constructing a reduction "on the nose" is inconvenient. In such situations, a proof by contradiction may be more straighforward. See the second example below.

18.5 Examples

1. The `HaltOnEveryInput` decision problem asks
   Given any program Q, does Q halt for every input?
   Prove that the `HaltOnEveryInput` decision problem is undecidable.
   Solution:
   (a) We will exhibit a reduction from the Halting Problem to the `HaltOnEveryInput` problem.
   (b) Let (M, w) be any candidate for the Halting Problem.
   (c) Construct a new program, Q to carry out this algorithm:
      i. Given any input, x,
      ii. Ignore the input, x, and run M on input w.

iii. If M halts on input w, then make Q halt on x.
iv. M may also run forever on input w; in this case, by construction, Q also runs forever on x.

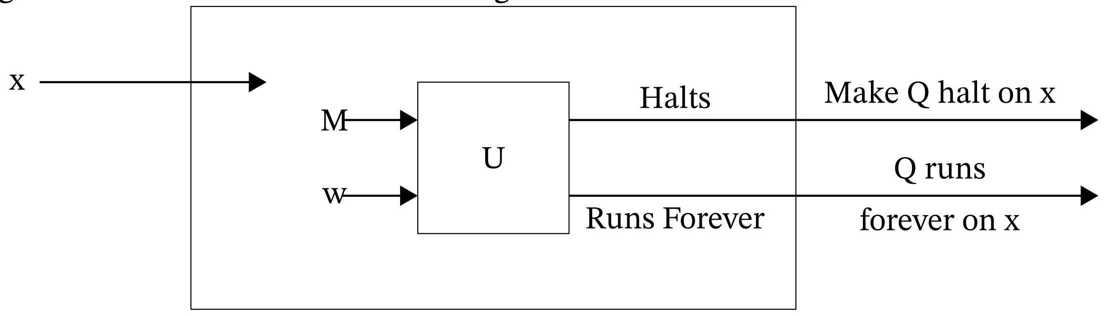
This is a **terminating algorithm** for constructing Q from any (M, w). By construction,

i. If (M, w) halts ("yes for P_1"), then Q halts for **every** input x ("yes for P_2"), and
ii. If (M, w) runs forever ("no for P_1"), then Q runs forever for **every** input x ("no for P_2").

(d) This shows that we have constructed a reduction from the Halting Problem to the `HaltOnEveryInput` problem.
(e) Therefore the `HaltOnEveryInput` decision problem is also undecdiable.

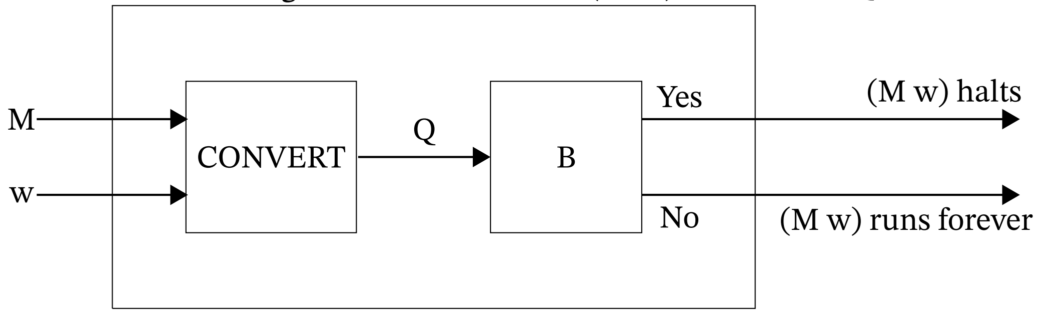
Solution 2: We present a solution using diagrams to explain the constructions involved, as in the slides.

Towards a contradiction, suppose that `HaltOnEveryInput` is decidable. Let B be a decider for `HaltOnEveryInput` (B accepts a Turing machine Q as input, halts for every Turing machine input, and gives the correct answer). Let (M, w) be any candidate for the Halting Problem. Construct the Turing machine Q, according to the following diagram, where U is the universal Turing Machine:



Note, the machine Q halts on every x, if and only if M halts on input w.

Now construct the following Turing Machine to decide the Halting of (M, w), where CONVERT is a Turing Machine which uses (M, w) to construct Q, as above:



Since we have constructed a decider for the Halting Problem, which is known to be undecidable, therefore we have a contradiction. Therefore `HaltOnEveryInput` is undecidable.

2. The `LoopOnZero` decision problem asks

Given any program Q, does Q run forever when processing the particular input n = 0?

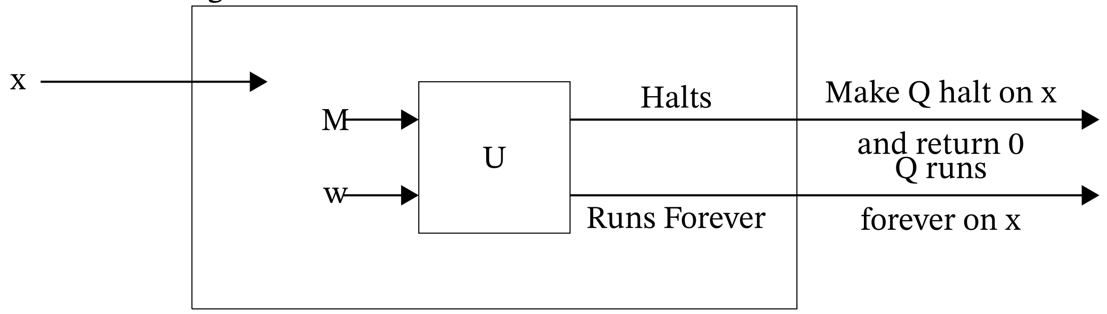
Prove that the `LoopOnZero` decision problem is **undecidable**.

Solution:

(a) Towards a contradiction, suppose that `LoopOnZero` is decidable, with a decider, B.
(b) Let (M, w) be any candidate for the Halting Problem.
(c) Construct a new program, Q, which will
   i. ignore its input, x, and run M with input w, and
   ii. if M halts when run with input w, then halt and return 0.
(d) By construction, Q halts **for all inputs** (including 0) if and only if M halts when run with input w.
(e) Now feed the constructed Q into B, and return the **opposite** answer for whether (M, w) halts.
(f) This provides a decider for the Halting Problem.
(g) Since the Halting Problem is undecidable, therefore we have a contradiction.
(h) This completes the proof.

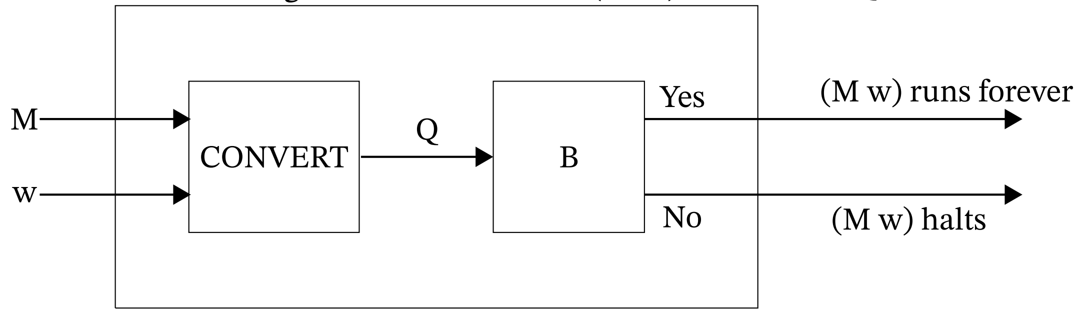
Solution 2: We present a solution using diagrams to explain the constructions involved, as in the slides.

Towards a contradiction, suppose that `LoopOnZero` is decidable. Let B be a decider for `LoopOnZero` (B accepts a Turing machine Q as input, halts for every Turing machine input, and gives the correct answer). Let (M, w) be any candidate for the Halting Problem. Construct the Turing machine Q, according to the following diagram, where U is the universal Turing Machine:



Note, the machine Q halts on every x (including 0), if and only if M halts on input w.

Now construct the following Turing Machine to decide the Halting of (M, w), where CONVERT is a Turing Machine which uses (M, w) to construct Q, as above:



Since we have constructed a decider for the Halting Problem, which is known to be undecidable, therefore we have a contradiction. Therefore `LoopOnZero` is undecidable.

3. See the slides for several other examples of using reductions to prove that new decision problems are undecidable.
4. **Satisfiability in First-Order Logic is Undecidable**
   (a) This is **not** proved in the slides.
   (b) We defer the proof for now.
   (c) I hope we will have time to prove it later.
   (d) Obviously, the proof will not be examinable.

19 Lecture 19 - Turing Machines II

Outline

1. Turing Machines
   (a) Diagrams
2. Languages Recognized by a Turing Machine
3. Computations Performed by a Turing Machine
4. Turing Machine Examples
5. Turing Machine Enhancements
6. Algorithm Complexity (Optional)

19.1 Turing Machines

A **Turing Machine** $T = (S, F, I, f, s_0)$ consists of

1. S: a finite set of **states** of the finite control unit
2. $F \subseteq S$: a subset of **final states**
3. I: a finite set of **tape symbols**, containing a **blank symbol**, B
   (a) A Turing machine does its work by reading / writing on an imaginary tape. The tape is infinitely long in both directions. A finite number of cells are filled with non-B contents to start; all the rest of the cells are filled with B symbols.
4. $f: S \times I \longrightarrow S \times I \times \{L, R\}$ is the **transition function**, where L: "left" and R: "right".
5. $s_0 \in S$ is the **start state**.

Brief Explanation of Transition Function: One Move the Turing Machine Can Make
If $f(s, x) = (s', x', D)$, then in state s, reading tape symbol x, our Turing machine will:

1. change its state to s',
2. write the symbol x' in the current cell, over-writing x, and
3. move the tape head one cell to the right if $D = R$, or one cell to the left if $D = L$.

If the (partial) function f is undefined for the pair (s, x) (f need **not** be total), then the Turing Machine T will **halt**.

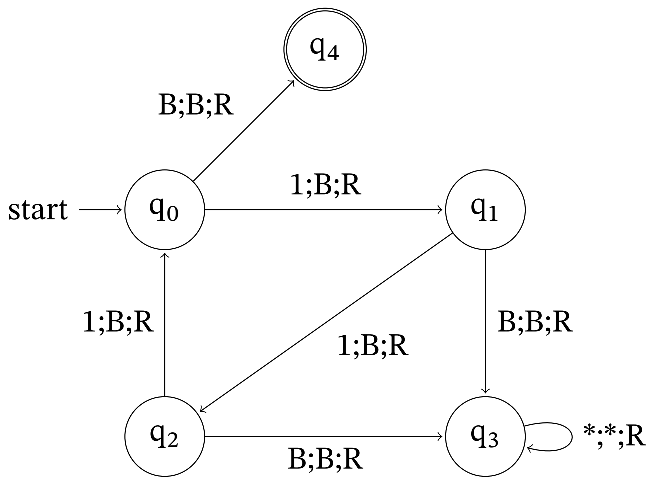
Important Remark:

- There are many variations of the definition of a Turing machine.
- If you take CS 360/365, you will see a variation of our definition.
- **All the variations of the definition of a Turing machine are equivalent in terms of their computing power.**
- You must use the correct definition for the context in which you are working.

19.1.1 Diagrams

1. We often represent a Turing machine using a diagram.

2. For example, below is a diagram of a Turing machine that
   (a) accepts its input string if and only if it consists of a number of 1s whose length is congruent to 0 modulo 3,
   (b) has tape alphabet is {B, 1}, where B is a blank symbol, and
   (c) starts execution with the tape head pointing to the leftmost 1, if there is one, and to one of the infinitely many B symbols otherwise.
3. We'll discuss how a Turing machine accepts/recognizes a language, in the next section.



19.2 Language Recognized by a Turing Machine

1. TMs can be used to accept (recognize) languages (sets of words over some alphabet, Σ).
2. The input string, w, is placed on the tape, with the tape head ponting to its leftmost character (rest of the tape is B).
3. The TM processes, according to its specification, one step at a time.
4. There are three possible outcomes:
   (a) Run forever.
   (b) Halt (i.e. no transition defined for current state, tape symbol):
      i. halting state is final: the TM **accepts** the input word.
      ii. halting state is not final: the TM **rejects** the input word.
5. Gathering up all words accepted by the TM creates the **language accepted by the Turing Machine** T, denoted by L(T): the set of strings $w \in \Sigma^*$ such that

$$s_0 w \Longrightarrow^* \alpha s_f \beta$$

   where
   (a) s_f is a final state,
   (b) α, β are tape strings in I^*, and
   (c) $\Longrightarrow^*$ denotes repeatedly many appplications (including zero times) of the steps of the Turing Machine transition function.
6. We write the details of a TM computation as a sequence of configurations, each of which displays

(a) the entire non-B tape contents,
(b) with the state inserted **immediately before** the cell where the tape head is currently pointing (see the slides for examples).

19.3 Computation Performed by a Turing Machine

A Turing machine, T, **computes the (partial) function** f(x), if

1. given any input, x, to f on the tape at the start of computation,
2. If T halts, then at the time of halting, the output f(x) will be written on the tape, with the tape head pointing at its first character.
3. For any xs which cause T to run forever, f(x) is **not defined**.

When all of the above happens, we say that f is a **computable function** with T a TM that **computes** f. Some authors demand that T always halt before we call f computable; we permit partial fs here.

See the slides for examples.

Church-Turing Thesis:

> Any problem that can be solved with an algorithm can be solved by a Turing Machine.

Remarks:

1. This says that we can use "algorithms" and "Turing machines" interchangeably.
2. It also says that algorithms need not always terminate, because there exist Turing machines that do not always halt.
3. From now on, we will specify a **terminating algorithm** or a **total TM** if we demand that our TM terminates for every input.

19.4 Turing Machine Examples

See the slides.

19.5 Turing Machine Enhancements

See the slides. This material is not examinable.

19.6 Algorithm Complexity (Optional)

See the slides. This material is not examinable.

20 Lecture 20 - Peano Arithmetic I

Outline

1. Properties of Equality
   (a) Reflexivity
   (b) Symmetry
   (c) Transitivity
2. Introduction to Peano Arithmetic

20.1 Properties of Equality

20.1.1 Reflexivity

We have

(1)	$\varnothing$	$\vdash$	$u \approx u$	(by $(\approx +)$)
(2)	$\varnothing$	$\vdash$	$\forall x(x \approx x)$	(by $(\forall+)$, (1), u not occurring in $\varnothing$)

20.1.2 Symmetry

We have

(1)	$\varnothing$	$\vdash$	$u \approx u$	(by $(\approx +)$)
(2)	$u \approx v$	$\vdash$	$u \approx v$	(by $(\in)$)
(3)	$u \approx v$	$\vdash$	$u \approx u$	(by $(+)$, (1))
(4)	$u \approx v$	$\vdash$	$v \approx u$	(by $(\approx -)$, (2), (3))
(5)	$\varnothing$	$\vdash$	$(u \approx v \to v \approx u)$	(by $(\to +)$, (4))
(6)	$\varnothing$	$\vdash$	$\forall y(u \approx y \to y \approx u)$	(by $(\forall+)$, (5), v not occurring in $\varnothing$)
(7)	$\varnothing$	$\vdash$	$\forall x \forall y(x \approx y \to y \approx x)$	(by $(\forall+)$, (6), u not occurring in $\varnothing$)

Explanation of the application of $(\approx -)$ on line (4):

- Σ is $\{u \approx v\}$
- t_1 is u
- t_2 is v
- $A(t_1)$ is $u \approx u$
- $A(t_2)$ is $v \approx u$

Remarks:

1. I am being extra verbose here, as this is our first opportunity to use the $(\approx -)$ rule. You do not need to include this type of explanation in your own proofs.

2. Soon, we'll need formulas like $(t_1 \not\approx t_2)$. Strictly following our syntax rules would require us to write $(\neg(t_1 \approx t_2))$ instead of $(t_1 \not\approx t_2)$. We'll write $\not\approx$, abusing notation slightly.

20.1.3 Transitivity

We have

(1)	$(u \approx v \wedge v \approx w)$	$\vdash$	$(u \approx v \wedge v \approx w)$	(by $(\in)$)
(2)	$(u \approx v \wedge v \approx w)$	$\vdash$	$u \approx v$	(by $(\wedge-)$,(1))
(3)	$(u \approx v \wedge v \approx w)$	$\vdash$	$v \approx w$	(by $(\wedge-)$,(1))
(4)	$(u \approx v \wedge v \approx w)$	$\vdash$	$u \approx w$	(by $(\approx -)$, (2), (3))
(5)	$\emptyset$	$\vdash$	$((u \approx v \wedge v \approx w) \to u \approx w)$	(by $(\to +)$, (4))
(6)	$\emptyset$	$\vdash$	$\forall z((u \approx v \wedge v \approx z) \to u \approx z)$	(by $(\forall+)$, (5), w not occurring in $\emptyset$)
(7)	$\emptyset$	$\vdash$	$\forall y \forall z((u \approx y \wedge y \approx z) \to u \approx z)$	(by $(\forall+)$, (6), v not occurring in $\emptyset$)
(8)	$\emptyset$	$\vdash$	$\forall x \forall y \forall z((x \approx y \wedge y \approx z) \to x \approx z)$	(by $(\forall+)$, (7), u not occurring in $\emptyset$)

Explanation of the application of $(\approx -)$ on line (4):

- Σ is $\{(u \approx v \wedge v \approx w)\}$
- t_1 is v
- t_2 is w
- $A(t_1)$ is $u \approx v$
- $A(t_2)$ is $u \approx w$

Remarks:

1. Since $\approx$ is reflexive, symmetric and transitive, therefore $\approx$ is an **equivalence relation**.
2. It is easy to prove (see this week's tutorial for details) that all three results above apply to any terms in place of the bound variables.

Two Useful Facts, To Be Proved During This Week's Tutorial:

Theorem(EQSubs). Let r(u) be a term that contains u as a free variable, and let t_1, t_2 be terms. Let $r(t_i)$ denote r where all instances of u have been replaced by t_i. For any set Σ of first-order logic formulas, we have that $\Sigma \vdash t_1 \approx t_2$ implies $\Sigma \vdash r(t_1) \approx r(t_2)$.

Proof.

(1)	Σ	$\vdash$	$t_1 \approx t_2$	(by supposition)
(2)	$\emptyset$	$\vdash$	$r(t_1) \approx r(t_1)$	(by reflexivity of equality of terms)
(3)	Σ	$\vdash$	$r(t_1) \approx r(t_1)$	(by $(+)$, (2))
(4)	Σ	$\vdash$	$r(t_1) \approx r(t_2)$	(by $(\approx -)$, (1), (3))

■

Explanation of the application of $(\approx -)$ on line (4):

- Σ is Σ
- $A(t_1)$ is $r(t_1) \approx r(t_1)$
- t_1 is t_1
- t_2 is t_2

Remark: Every result proved below with EQSubs can be proved, in more lines, using $(\approx -)$.

Extended transitivity

Theorem (EQTrans(k)). Let $k \geq 1$ be a natural number, Σ be a set of first-order logic formulas, and $t_1, t_2, \dots, t_{k+1}$ be terms. If $\Sigma \vdash t_i \approx t_{i+1}$ for all $1 \leq i \leq k$, then $\Sigma \vdash t_1 \approx t_{k+1}$.

Proof. The proof is by induction on k.

Base ($k = 1$)**:** The conclusion $\Sigma \vdash t_1 \approx t_2$ is immediate from the setup.

Induction ($k > 1$)**:** The induction hypothesis is that $\Sigma \vdash t_1 \approx t_k$. Then since we assume that $\Sigma \vdash t_k \approx t_{k+1}$, by transitivity of equality of terms, we have $\Sigma \vdash t_1 \approx t_{k+1}$, as required. ■

20.2 Introduction to Peano Arithmetic

1. Fix the domain as $\mathbb{N}$, the natural numbers.
2. Interpret the individual symbol 0 as zero and the unary function symbol s as **successor** ($s(n) = n + 1$).
3. Thus each number in $\mathbb{N}$ has a term: 0, $s(0)$, $s(s(0))$, $s(s(s(0)))$, ...

Zero and successor satisfy the following axioms.

PA1 $\forall x \neg(s(x) \approx 0)$.
"Zero is not a successor."

PA2 $\forall x \forall y (s(x) \approx s(y) \rightarrow x \approx y)$.
"Nothing has two predecessors."

("PA" stands for Peano Axioms, named for Giuseppe Peano.) Further axioms characterize $+$ (addition) and $\times$ (multiplication).

PA3 $\forall x (x + 0 \approx x)$.
Adding zero to any number yields the same number.

PA4 $\forall x \forall y (x + s(y) \approx s(x + y))$.
Adding a successor yields the successor of adding the number.

PA5 $\forall x (x \times 0 \approx 0)$.
Multiplying by zero yields zero.

PA6 $\forall x \forall y (x \times s(y) \approx (x \times y) + x)$.
Multiplication by a successor.

The six axioms above define $+$ and $\times$ for any particular numbers.
They do not, however, allow us to reason adequately about all numbers.
For that, we use an additional axiom: **induction**.

PA7 For each formula $A(u)$ and variable x,

$$((A(0) \wedge \forall x(A(x) \rightarrow A(s(x)))) \rightarrow \forall x A(x))$$

is an axiom.

The formula $A(u)$ represents the "property" to be proved, for a free u.

To prove $\forall x A(x)$, we can

1. prove the base case $A(0)$, and
2. prove the inductive case $\forall x(A(x) \rightarrow A(s(x)))$.

The axiom PA7 is just the translation of the Principle of Mathematical Induction into First-Order Logic.

These axioms imply all of the familiar properties of the natural numbers.

For example,

Theorem 20.2.1.

$$\varnothing \vdash_{PA} \forall x(\neg(s(x) \approx x))$$

"No natural number equals its successor".

Proof. Let $A(u)$ be $\neg(s(u) \approx u)$.

Base Case: The base case is $\varnothing \vdash A(0)$, which in this proof is is

$$\varnothing \vdash \neg(s(0) \approx 0).$$

We have

(1)	$\varnothing$	$\vdash$	$\forall x \neg(s(x) \approx 0)$	(by PA1)
(2)	$\varnothing$	$\vdash$	$(\neg(s(0) \approx 0))$	(by ($\forall-$), (1))

Inductive Case: The inductive case is

$$\varnothing \vdash \forall x(A(x) \rightarrow A(s(x))),$$

which in this proof is

$$\varnothing \vdash \forall x(\neg(s(x) \approx x) \rightarrow \neg(s(s(x)) \approx s(x))).$$

We have

(1)	$\emptyset$	$\vdash$	$\forall x\forall y(s(x) \approx s(y) \to x \approx y)$	(by PA2)
(2)	$\emptyset$	$\vdash$	$\forall y(s(s(u)) \approx s(y) \to s(u) \approx y)$	(by $(\forall-)$, (1))
(3)	$\emptyset$	$\vdash$	$(s(s(u)) \approx s(u) \to s(u) \approx u)$	(by $(\forall-)$, (2))
(4)	$(\neg(s(u) \approx u)), s(s(u)) \approx s(u)$	$\vdash$	$(s(s(u)) \approx s(u) \to s(u) \approx u)$	(by $(+)$, (3))
(5)	$(\neg(s(u) \approx u)), s(s(u)) \approx s(u)$	$\vdash$	$s(s(u)) \approx s(u)$	(by $(\in)$)
(6)	$(\neg(s(u) \approx u)), s(s(u)) \approx s(u)$	$\vdash$	$s(u) \approx u$	(by $(\to -)$, (4), (5))
(7)	$(\neg(s(u) \approx u)), s(s(u)) \approx s(u)$	$\vdash$	$\neg(s(u) \approx u)$	(by $(\in)$)
(8)	$(\neg(s(u) \approx u))$	$\vdash$	$\neg(s(s(u)) \approx s(u))$	(by $(\neg+)$, (6), (7))
(9)	$\emptyset$	$\vdash$	$(\neg(s(u) \approx u)) \to (\neg(s(s(u)) \approx s(u)))$	(by $(\to +)$, (8))
(10)	$\emptyset$	$\vdash$	$\forall x((\neg(s(x) \approx x)) \to (\neg(s(s(x)) \approx s(x))))$	(by $(\forall+)$, (9), u not occurring in $\emptyset$)

Putting It All Together: The PA7 axiom is

$$\emptyset \vdash ((A(0) \land \forall x(A(x) \to A(s(x)))) \to \forall x A(x)),$$

which in this proof is:

$$\emptyset \vdash (((\neg(s(0) \approx 0)) \land \forall x((\neg(s(x) \approx x)) \to (\neg(s(s(x)) \approx s(x))))) \to \forall x(\neg(s(x) \approx x))).$$

We have

(1)	$\emptyset$	$\vdash$	$((\neg(s(0) \approx 0)) \land \forall x((\neg(s(x) \approx x)) \to (\neg(s(s(x)) \approx s(x)))))$ $\to \forall x(\neg(s(x) \approx x))$	(by PA7)
(2)	$\emptyset$	$\vdash$	$(\neg(s(0) \approx 0))$	(by Base Case)
(3)	$\emptyset$	$\vdash$	$\forall x((\neg(s(x) \approx x)) \to (\neg(s(s(x)) \approx s(x))))$	(by Induction Case)
(4)	$\emptyset$	$\vdash$	$((\neg(s(0) \approx 0)) \land \forall x((\neg(s(x) \approx x)) \to (\neg(s(s(x)) \approx s(x)))))$	(by $(\land+)$, (2), (3))
(5)	$\emptyset$	$\vdash$	$\forall x(\neg(s(x) \approx x))$	(by $(\to -)$, (1), (4))

■

Remarks:

1. Just as in MATH 135, applying POMI (for us, applying PA7) does not need to be as detailed as I have made it here.
2. Proving the base case and the induction case is enough. Then we can just say, by PA7, that the desired result follows.

21 Lecture 21 - Peano Arithmetic II

Outline

1. Commutativity of $+$
2. Soundness and Completeness of Peano Arithmetic

21.1 Commutativity of $+$

.

Remarks:

1. We know that $+$ in $\mathbb{N}$ is commutative.
2. But the Peano Axioms do not say this explicitly.
3. Before we can safely use the commutativity of $+$, we need to prove it!

Theorem: Addition in Peano Arithmetic is commutative; that is,

$$\varnothing \vdash_{\mathrm{PA}} \forall x \forall y(x + y \approx y + x).$$

(Notation "$\varnothing \vdash_{\mathrm{PA}}$" or "PA $\vdash$" means "provable [in *FD*] using the $\approx$ and PA axioms". For the rest of this lecture, $\vdash$ means $\vdash_{\mathrm{PA}}$.)

How can we prove this result?

We must use induction (PA7). The key first step: choose a good formula, A, for the induction property.

Choosing A(u) to be $\forall y(u + y \approx y + u)$ yields the base case

$$\varnothing \vdash \forall y(0 + y \approx y + 0)$$

and the inductive case

$$\varnothing \vdash \forall x(\forall y(x + y \approx y + x) \rightarrow \forall y(s(x) + y \approx y + s(x))).$$

Then PA7 plus one application of $(\land+)$ plus one application of $(\rightarrow -)$ yield the desired formula

$$\varnothing \vdash_{\mathrm{PA}} \forall x \forall y(x + y \approx y + x).$$

How should we prove $\forall y(0 + y \approx y + 0)$?

We must use induction again, to prove the base case of the main result. To control the complication, let's make it a lemma:

Lemma 21.1.1. *Peano Arithmetic has a proof of*

$$\varnothing \vdash_{\mathrm{PA}} \forall y(0 + y \approx y + 0).$$

Proof. The proof is by induction (PA7) with free variable v replacing y, and B(v) as $0+v \approx v+0$.

Basis: Prove

$$\varnothing \vdash_{PA} 0+0 \approx 0+0.$$

This is immediate from reflexivity of $\approx$.

Inductive step: Prove

$$\varnothing \vdash_{PA} \forall y\big(0+y \approx y+0 \rightarrow 0+s(y) \approx s(y)+0\big).$$

Pick v free. Assume $0+v \approx v+0$ (this is the antededent of the above implication, with the bound variable y replaced by a free v). Then, informally,

$$\begin{aligned} 0+s(v) &\approx s(0+v) && \text{PA4} \\ &\approx s(v+0) && \text{Assumption + EQSubs} \\ &\approx s(v) && \text{PA3 + } (\forall-) \text{ + EQSubs} \\ &\approx s(v)+0 && \text{PA3 + } (\forall-) \text{ + symmetry of } \approx . \end{aligned}$$

- There will be a few more of these informal arguments throughout the lecture.
- We could formalize each of them, if needed.
- We will demonstrate this, by formally proving the above informal argument here.

From the above informal proof of $0+s(v) \approx s(v)+0$, write a formal proof of

$$0+v \approx v+0 \vdash_{PA} 0+s(v) \approx s(v)+0.$$

(1)	$\varnothing$	$\vdash$	$\forall x(x+0 \approx x)$	(by [PA3])
(2)	$\varnothing$	$\vdash$	$\forall x\forall y(x+s(y) \approx s(x+y))$	(by [PA4])
(3)	$\varnothing$	$\vdash$	$\forall y(0+s(y) \approx s(0+y))$	(by $(\forall-)$, (2))
(4)	$\varnothing$	$\vdash$	$0+s(v) \approx s(0+v)$	(by $(\forall-)$, (3))
(5)	$\varnothing$	$\vdash$	$v+0 \approx v$	(by $(\forall-)$, (1))
(6)	$\varnothing$	$\vdash$	$s(v+0) \approx s(v)$	(by EQSubs, (5))
(7)	$\varnothing$	$\vdash$	$s(v)+0 \approx s(v)$	(by $(\forall-)$, (1))
(8)	$0+v \approx v+0$	$\vdash$	$0+s(v) \approx s(0+v)$	(by (+), (4))
(9)	$0+v \approx v+0$	$\vdash$	$v+0 \approx v$	(by (+), (5))
(10)	$0+v \approx v+0$	$\vdash$	$s(v+0) \approx s(v)$	(by (+), (6))
(11)	$0+v \approx v+0$	$\vdash$	$s(v)+0 \approx s(v)$	(by (+), (7))
(12)	$0+v \approx v+0$	$\vdash$	$0+v \approx v+0$	(by $(\in)$)
(13)	$0+v \approx v+0$	$\vdash$	$s(0+v) \approx s(v+0)$	(by EQSubs, (12))
(14)	$0+v \approx v+0$	$\vdash$	$s(v) \approx s(v)+0$	(by Symm of $\approx$, (11))
(15)	$0+v \approx v+0$	$\vdash$	$0+s(v) \approx s(v)+0$	(by EQTrans, (8), (13), (10), (14))

Now, complete this to a proof of

$$\varnothing \vdash \forall y(0+y \approx y+0 \rightarrow 0+s(y) \approx s(y)+0).$$

(16)	$\varnothing$	$\vdash$	$0+v \approx v+0 \to 0+s(v) \approx s(v)+0$	(by $(\to +)$, (15))
(17)	$\varnothing$	$\vdash$	$\forall y(0+y \approx y+0 \to 0+s(y) \approx s(y)+0)$	(by $(\forall+)$, (16)) (v does not occur in $\varnothing$)

Now using PA7, $(\wedge+)$ and $(\to -)$ completes the proof of the inductive step of the Lemma:

(1)	$\varnothing$	$\vdash$	$((0+0=0+0) \wedge \forall y((0+y=y+0) \to (0+s(y)=s(y)+0))))$ $\to \forall y(0+y=y+0)$	(by [PA7])
(2)	$\varnothing$	$\vdash$	$(0+0=0+0)$	(by Base Case)
(3)	$\varnothing$	$\vdash$	$\forall y((0+y=y+0) \to (0+s(y)=s(y)+0)))$	(by Induction Case)
(4)	$\varnothing$	$\vdash$	$((0+0=0+0) \wedge \forall y((0+y=y+0) \to (0+s(y)=s(y)+0))))$	(by $(\wedge+)$, (2), (3))
(5)	$\varnothing$	$\vdash$	$\forall y(0+y=y+0)$	(by $(\to -)$, (1), (4))

This completes the proof of the Lemma for the base case. ■

The next Lemma provides the content for the induction step of the induction case.

Lemma 21.1.2. *For each free variable* u*,*

$$\{\forall y(u+y \approx y+u)\} \vdash_{PA} \forall y(s(u)+y \approx y+s(u)).$$

Why This Suffices:

- Once we establish the above, one invocation of $(\to +)$, followed by one invocation of $(\forall+)$ will produce the desired inductive step.

Proof. Strategy: induction on free variable v in place of y, where formula B(v) is $s(u)+v \approx v+s(u)$ (keeping u free throughout).

Basis:

$$\varnothing \vdash_{PA} s(u)+0 \approx 0+s(u).$$

The target follows from $\forall y(0+y \approx y+0)$ plus $(\forall-)$ plus the symmetry of $\approx$.

Induction:

$$\varnothing \vdash_{PA} \forall y\big(s(u)+y \approx y+s(u) \to s(u)+s(y) \approx s(y)+s(u)\big).$$

Let a free v replace y. Assuming $s(u)+v \approx v+s(u)$ yields, informally,

$$\begin{aligned} s(u)+s(v) &\approx s\big(s(u)+v\big) && \text{PA4} \\ &\approx s\big(v+s(u)\big) && \text{Assumption + EQSubs} \\ &\approx s(s(v+u)) && \text{PA4 + EQSubs.} \end{aligned}$$

The premise of the Lemma implies $u+s(v) \approx s(v)+u$; thus, informally,

$$\begin{aligned} s(v)+s(u) &\approx s(s(v)+u) && \text{PA4} \\ &\approx s(u+s(v)) && \text{premise + symmetry of} \approx \text{+ EQSubs} \\ &\approx s(s(u+v)) && \text{PA4 + EQSubs.} \end{aligned}$$

The premise of the Lemma also implies $u + v \approx v + u$. Then by symmetry and transitivity of $\approx$, plus EQSubs twice, we obtain

$$s(u) + s(v) \approx s(v) + s(u),$$

so that the desired equation holds, as required.

Now applying PA7, $(\wedge+)$ and $(\rightarrow -)$ completes the proof of the lemma. ■

Now for the main proof, taking A to be $\forall y(u + y \approx y + u)$, We have

(1)	$\emptyset$	$\vdash$	$(\forall y(0 + y \approx y + 0) \wedge \forall x((\forall y(x + y \approx y + x)) \rightarrow ((\forall y(s(x) + y \approx y + s(x))))))$ $\rightarrow \forall x(\forall y(x + y \approx y + x))$	(by [PA7])
(2)	$\emptyset$	$\vdash$	$\forall y(0 + y \approx y + 0)$	(by Base Case)
(3)	$\emptyset$	$\vdash$	$\forall x((\forall y(x + y \approx y + x)) \rightarrow ((\forall y(s(x) + y \approx y + s(x)))))$	(by Induction C
(4)	$\emptyset$	$\vdash$	$(\forall y(0 + y \approx y + 0) \wedge \forall x((\forall y(x + y \approx y + x)) \rightarrow ((\forall y(s(x) + y \approx y + s(x))))))$	(by $(\wedge+)$, (2), (3
(5)	$\emptyset$	$\vdash$	$\forall x(\forall y(x + y \approx y + x))$	(by $(\rightarrow -)$, (1), (

Remarks:

1. Now that we have proved the commutativity of $+$, we can freely use it, when needed!
2. The other familiar properties of addition and multiplication have similar proofs.
3. One can continue: divisibility, primeness, etc.

21.2 Soundness and Completeness of Peano Arithmetic

PA is sound in $\mathbb{N}$:

1. Formal Deduction for First-Order Logic is sound.
2. PA augments First-Order FD by adding the 7 Peano Axioms, in the semantic context of domain $\mathbb{N}$, with the usual $0, +, \cdot$ and s as the successor function.
3. In the semantic context of the arithmetic of $\mathbb{N}$, each Peano axiom obviously holds.
4. Hence any result proved in PA must hold in the semantic context of the arithmetic of $\mathbb{N}$.

PA is not complete in $\mathbb{N}$:

1. Gödel's Incompleteness Theorem tells us that no axiomatization of the arithmetic of $\mathbb{N}$ can be both
   (a) consistent, and
   (b) complete.
2. In other words, for any axiomatization of the arithmetic of $\mathbb{N}$, either
   (a) there are true statements that are not provable, or
   (b) some provable statements are not true.
   PA falls into the first case.
3. Exhibiting an example of a true but unprovable statement is well beyond the scope of CS 245.

22 Lecture 22 - Program Verification I (Introduction/Assignments/Co

Outline

1. Introduction to Program Verification
2. Assignments
3. Conditionals

22.1 Introduction to Program Verification

Remarks:

1. We will work in a small imperative programming language (`C`-like).

Definition 22.1.1. *A **Hoare triple** is a triple* $(\!|P|\!)\ C\ (\!|Q|\!)$ *composed of*

1. P*, a **precondition**, a First-Order formula,*
2. C*, some code, and*
3. Q*, a **postcondition**, another First-Order formula*

Definition 22.1.2. *A **specification** of a program* C *is a Hoare triple with* C *as its middle element.*

Definition 22.1.3. *A Hoare triple is **satisfied under partial correctness** (a.k.a. **partially correct**) if, whenever execution starts in a **state** satisfying precondition* P*, and terminates, it follows that the **state** after execution satisfies postcondition* Q*.*

Definition 22.1.4. *The **state** of a program at a given moment is the list of the values of each of its variables at that moment.*

Examples:

1. The Hoare triple
   $(\!|(x = x_0)|\!)$
   `x = 1;`
   $(\!|(x = 1)|\!)$
   is satisfied under partial correctness.
2. The Hoare triple
   $(\!|(x = x_0)|\!)$
   `x = 0;`
   $(\!|(x = 1)|\!)$
   is **not** satisfied under partial correctness.

Remarks:

1. We will always be given a specification for a program C to start. Our job will be to prove that the specification is satisfied under partial correctness. Our technique will be
   (a) Annotate the program, according to the (global) pre- and post-conditions, and the proof rule that applies to each line of code.

   (b) Prove any implications that arise from the annotation, where two assertions occur on consecutive lines. We will write these proofs in "Math 135 style".
      i. In past runnings of the course, we have made these proofs formal, using Peano Arithmetic.
2. We use **logical variables** (like x_0 in the earlier examples) to keep track of the starting values of variables, as required. This is crucial, if one of the variables gets overwritten during execution.
3. Although we usually perform the annocation from bottom to top, it must read correctly top to bottom when it is complete.

Recall the definition of **partial correctness** (Definition 22.1.3).

Definition 22.1.5. *A Hoare triple* $(\!|P|\!)\ C\ (\!|Q|\!)$ *is **satisfied under total correctness** (a.k.a. **totally correct**) if it is satisfied under partial correctness, and whenever* C *starts in a state obeying* P*, it follows that* C *terminates.*

In other words, $\boxed{\text{total correctness} = \text{partial correctness} + \text{termination}}$.

Remarks:

1. Total correctness is always our goal.
2. Partial and total correctness are the same thing, **until we introduce loops**, i.e. until termination is no longer guaranteed.
3. Suppose that a program `C` contains no loops, and that a specification of that program is satisfied under partial correctness. Then it follows immediately that the specification will also be satisfied under total correctness.
4. To show that a specification of a program `C`, with a while-loop, is satisfied under total correctness, we will
   (a) prove the specificaton of `C` is satisfied under partial correctness, and
   (b) provide a separate proof that the program `C` always terminates, when run starting in a state obeying the given pre-condition.
5. If a program `C` never terminates, then **any** specification of `C` is always (vaccuously) satisfied under partial correctness.
6. The setup for proving partial correctness (N.B. **not** total correctness) is often useful for arguing termination also, later.

Useful Rules For Program Annotations

Rule of "Precondition strengthening":

$$\frac{P \to P' \qquad (\!|P'|\!)\ C\ (\!|Q|\!)}{(\!|P|\!)\ C\ (\!|Q|\!)}\ \textit{(implied)}$$

Rule of "Postcondition weakening":

$$\frac{(\!|P|\!)\ C\ (\!|Q'|\!) \qquad Q' \to Q}{(\!|P|\!)\ C\ (\!|Q|\!)}\ \textit{(implied)}$$

$$\frac{(\!|P|\!)\ C_1\ (\!|Q|\!),\quad (\!|Q|\!)\ C_2\ (\!|R|\!)}{(\!|P|\!)\ C_1;\ C_2\ (\!|R|\!)}\ \textit{(composition)}$$

Using composition will require us to find a **midcondition** Q, as above.

In our examples, Q usually arises from applying the assignment rule.

In general, choosing Q can be very difficult.

22.2 Assignments

- The Assignment rule is

  $(\!|Q[E/x]|\!)$
  x = E;
  $(\!|Q|\!)$ assignment.

  where Q[E/x] denotes a copy of Q, with all **free** xs replaced by Es.
- The assignment rule has no premises and is therefore an **axiom** of our proof system.

Remarks:

1. If we wish to show that Q holds in the state after the assignment x = E;, we must show that Q[E/x] holds before the assignment (so that the Hoare triple given in the statement of the assignment rule is satisfied under partial correctness).
2. Several explanations may be required to understand this rule.
3. At first sight, it looks as if the rule has been stated in reverse; one might expect that, if Q holds in a state in which we perform the assignment x = E;, then surely Q[E/x] holds in the resulting state, i.e. we just replace x by E. In other words, one might suspect that the rule should actually be

   $(\!|Q|\!)$
   x = E;
   $(\!|Q[E/x]|\!)$???.

   This is wrong. It is true that the assignment x = E; replaces the value of x in the starting state by E, but that does not mean that we replace occurrences of x in an **assertion** on the starting state by E.

Examples:

1. For example, let Q be (x = 6) and E be 5. Then
   $(\!|(x = 6)|\!)$
   ```
   x = 5;
   ```
   $(\!|(5 = 6)|\!)$
   is **not** satisfied under partial correctness: given a state in which (x = 6), the execution of x = 5; results in a state in which (x = 5). But Q[E/x] is the formula (5 = 6) which holds in **no** state.

2. Here is the same setup (i.e. the same Q and E), with the correct assignment rule. The Hoare triple
   $(\!|(5 = 6)|\!)$
   `x = 5;`
   $(\!|(x = 6)|\!)$
   is satisfied under partial correctness, in a very trivial way. We can never satisfy the precondition.
3. The right way to understand the assignment rule is to think about what you would have to know about the precondition in order to prove that Q holds as a post-condition. Since Q will in general be asserting something about the value of x, whatever it asserts about that value must have been true of E, since in the resulting state the value of x is E. Thus, Q with E in place of x which says whatever Q says about x but applied to E must be true in the precondition.
4. Another example: The Hoare triple
   $(\!|(1 > 0)|\!)$
   `x = 1;`
   $(\!|(x > 0)|\!)$
   is satisfied under partial correctness. The precondition is always satisfied. We might as well think of it as 1. Since $x > 0$ will hold after assigning `x = 1;`, therefore this Hoare triple is satisfied under partial correctness.

Example From the Slides (Swap x **and** y**)**

$(\!|x = x_0 \wedge y = y_0|\!)$
$(\!|y = y_0 \wedge x = x_0|\!)$ implied(a) *[proof required]*
`t = x;`
$(\!|y = y_0 \wedge t = x_0|\!)$ assignment
`x = y;`
$(\!|x = y_0 \wedge t = x_0|\!)$ assignment
`y = t;`
$(\!|x = y_0 \wedge y = x_0|\!)$ assignment
Finally, show $x = x_0 \wedge y = y_0$ implies $y = y_0 \wedge x = x_0$.

Remarks:

1. We need **precondition strengthening** here to prove that the annotation provides a Hoare triple which is satisfied under partial correctness.
2. The proof witnessing $\emptyset \vdash (x = x_0 \wedge y = y_0) \rightarrow (y = y_0 \wedge x = x_0)$ is left as an (easy) exercise.
3. We need to give a clear explanation of why the implication holds, in Math 135 style.
4. The annotation plus the proof of implied(a) establishes the **partial correctness** of the given specification.
5. Since no while loop is present, this also establishes **total correctness**.

22.3 Conditionals

`if-then-else`:

$$\frac{(\!|P \land B|\!)\ C_1\ (\!|Q|\!) \qquad (\!|P \land \neg B|\!)\ C_2\ (\!|Q|\!)}{(\!|P|\!)\ \text{if } (B)\ C_1 \text{ else } C_2\ (\!|Q|\!)}\ \text{(if-then-else)}$$

See the corresponding template on slide 54.

`if-then` (without `else`):

$$\frac{(\!|P \land B|\!)\ C\ (\!|Q|\!) \qquad (P \land \neg B) \to Q}{(\!|P|\!)\ \text{if } (B)\ C\ (\!|Q|\!)}\ \text{(if-then)}$$

See the corresponding template on slide 55.

Example (from the slides - Computing max of x and y)

```
(|true|)
if (x > y)
      (|x > y|)                                   if-then-else
      (|(x > y ∧ x ≈ x) ∨ (x ≤ y ∧ x ≈ y)|)         implied (a)
      max = x;
      (|(x > y ∧ max ≈ x) ∨ (x ≤ y ∧ max ≈ y)|)     assignment
} else {
      (|¬(x > y)|)                                if-then-else
      (|(x > y ∧ y ≈ x) ∨ (x ≤ y ∧ y ≈ y)|)         implied (b)
      max = y;
      (|(x > y ∧ max ≈ x) ∨ (x ≤ y ∧ max ≈ y)|)     assignment
 }
(|(x > y) ∧ max ≈ x) ∨ (x ≤ y ∧ max ≈ y)|)          if-then-else
```

It remains to prove the two implications arising from the above annotation.

(a) Prove $\varnothing \vdash x > y \to (x > y \land x \approx x) \lor (x \leq y \land x \approx y)$.

1. $x > y$ implies itself.
2. $x \approx x$ is trivially implied.
3. Thus $x > y$ implies $x > y \land x \approx x$.
4. Then by $(\lor+)$, $x > y$ implies $(x > y \land x \approx x) \lor (x \leq y \land x \approx y)$.

(b) Prove $\varnothing \vdash \neg(x > y) \to (x > y \land y \approx x) \lor (x \leq y \land y \approx y)$.

1. Rewrite $\neg(x > y)$ as $x \leq y$, by basic algebra.
2. $x \leq y$ implies itself.
3. $y \approx y$ is trivially implied.
4. Thus $x \leq y$ implies $x \leq y \land y \approx y$.
5. Then by $(\lor+)$, $x \leq y$ implies $(x > y \land y \approx x) \lor (x \leq y \land y \approx y)$.

Remarks:

1. The annotation plus the proofs of implied(a) and implied(b) establishes the partial correctness of the given specification.
2. Since no while loop is present, this also establishes **total correctness**.

Example (From the slides - Ensuring max **is at least** x**)**

```
(|true|)
if ( max < x )
     (|max < x|)      if-then
     (|x ≥ x|)        implied (a)
     max = x ;
     (|max ≥ x|)      assignment
}
     (|max ≥ x|)      if-then
```

Proof of implied (a):

$$\varnothing \vdash \text{max} < x \rightarrow x \geq x.$$

- Clearly $x \geq x$ is a valid formula and so the required implication holds.

Proof of implied (b): Show $\varnothing \vdash ((P \wedge (\neg B)) \rightarrow Q)$, which in this example is

$$\varnothing \vdash \neg(\text{max} < x) \rightarrow \text{max} \geq x.$$

- The hypothesis, $\neg(\text{max} < x)$ can be simplified to $\text{max} \geq x$ by basic algebra.
- Then the conclusion, $\text{max} \geq x$, clearly follows.

Remarks:

1. The annotation plus the proofs of implied(a) and implied(b) establishes the partial correctness of the given specification.
2. Since no while loop is present, this also establishes **total correctness**.

23 Lecture 23 - Program Verification II (Loops/Examples)

Outline

1. Loops
2. Example - Factorial
3. Example - Exponentiation
4. Undecidability of Program Verification

23.1 Loops

Once while-loops enter the picture, partial and total correctness are different.

"Partial while": do not (yet) require termination.

$$\frac{(\!|I \wedge B|\!)\ C\ (\!|I|\!)}{(\!|I|\!)\ \text{while (B) C}\ (\!|I \wedge \neg B|\!)}\ (\text{partial-}\mathtt{\textit{while}})$$

where I is a **loop invariant**, i.e. a formula expressing a relationship among program variables, such that

1. I holds before we execute the loop for the first time, and
2. I holds after any number of iterations of the loop code C.

See the corresponding template on slide 68.

Remarks:

1. I is not forced on us, as P and Q were in earlier examples. We have to choose I ourselves.
2. There is no algorithm for choosing I. There are some rules of thumb, demonstrated in the examples that follow.
3. I must satisfy #1 and #2 above, AND it must be strong enough to prove the final implication arising from the annotation of the while-loop.
4. In practice, we often make an initial choice for I which makes the loop teplate work, but fails to prove the final implication. In this situation, we must strengthen I, preserving the loop behaviour, to enable ourselves to prove the post-condition Q at the end of the annotation.
5. A good choice of I is often useful in proving termination, later.

23.2 Example - Factorial

Example to compute x! Prove that the following specification is satisfied under total correctness.

$(\!| x \geq 0 |\!)$
```
y = 1;
z = 0;
while (z != x) {
     z = z + 1;
     y = y * z;
}
```
$(\!| y \approx x! |\!)$

First Goal: The key ingredient in the program is a while-loop. Hence we need the partial-while template. Therefore our first goal is to select a loop invariant, I.
Suggested Technique: Make a table of states, with one row for each time that we reach the start of the loop code.

At the `while` statement:

x	y	z	$z \not\approx x$
5	1	0	true
5	1	1	true
5	2	2	true
5	6	3	true
5	24	4	true
5	120	5	false

From the states and the post-condition, a candidate loop invariant is $y \approx z!$

Remarks:

1. $y \geq z$ and $x \geq 0$ are also possible loop invariants, **but not useful** because they do not enable us to prove the final implication.
2. If we had chosen a weaker invariant, e.g. $x \geq 0$, then our final implication, implied(c) would read:

$(\!| x \geq 0 \wedge z \approx x |\!)$	partial-while
$(\!| y \approx x! |\!)$	implied (c)

   This implication is **not provable**: e.g. let

   $$\begin{aligned} x &= 1 \\ y &= 0 \\ z &= 1. \end{aligned}$$

   This state satisfies $x \geq 0 \wedge z \approx x$ but does not satisfy $y \approx x!$.

Here is the annotated program.

```
$\llparenthesis x \geq 0 \rrparenthesis$
$\llparenthesis 1 \approx 0! \rrparenthesis$                       implied (a)
y = 1;
$\llparenthesis y \approx 0! \rrparenthesis$                       assignment
z = 0;
$\llparenthesis y \approx z! \rrparenthesis$                       assignment
while (z != x) {
     $\llparenthesis y \approx z! \land z \neq x \rrparenthesis$         partial-while
     $\llparenthesis y(z+1) \approx (z+1)! \rrparenthesis$   implied (b)
     z = z + 1;
     $\llparenthesis yz \approx z! \rrparenthesis$                  assignment
     y = y * z;
     $\llparenthesis y \approx z! \rrparenthesis$                   assignment
}
$\llparenthesis y \approx z! \land z \approx x \rrparenthesis$             partial-while
$\llparenthesis y \approx x! \rrparenthesis$                       implied (c)
```

Proof of implied (a):

$$\emptyset \vdash x \geq 0 \to 1 \approx 0!.$$

This result follows by the definition of 0!.

Proof of implied (b):

$$\emptyset \vdash (y \approx z! \land \neg(z \approx x)) \to (z+1)y \approx (z+1)!.$$

If $y \approx z!$, then we can multiply both sides by $(z+1)$, to obtain $y(z+1) \approx (z+1)z! \approx (z+1)!$.

Proof of implied (c):

$$\emptyset \vdash (y \approx z! \land z \approx x) \to y \approx x!.$$

From $z \approx x$, we obtain $z! \approx x!$. Then since $y \approx z!$, by the transitivity of equality, we obtain $y \approx z! \approx x!$.

This completes the proof of partial correctness.

Proof of Termination: Suppose that we start in a state obeying the precondition $x \geq 0$. The factorial code from earlier has a **loop guard** of $z \not\approx x$, which is equivalent to $x - z \not\approx 0$.

What happens to the value of $x - z$ during execution?

```
$\llparenthesis x \geq 0 \rrparenthesis$
y = 1;
z = 0;              At start of loop: $x - z \geq 0$✓
while (z != x) {
     z = z + 1;     $x - z$ decreases by 1 ✓
     y = y * z;     $x - z$ unchanged
}
$\llparenthesis y \approx x! \rrparenthesis$
```

Thus the value of $x - z$ will eventually (after finitely many steps) reach 0.
The loop then exits and the program terminates. ✓

This completes the proof of total correctness.

Remarks:

1. The three checkmarks above indicate the three key features of a **loop variant**. In the presence of a loop variant having these three features, termination is clear.
2. Exhibiting a loop variant is sufficient, but not necessary, to establish termination.
3. Any other convincing argument for termination after some finite number of steps is also acceptable.
4. If $x < 0$ to start, then the factorial program will go into an infininte loop, i.e. **it will not terminate**.
5. It **does matter** for proving termination that we assume we start in a state **obeying the given precondition**.
6. A general proof of termination must be written based on the formula B in the while statement (however complicated B may be).
7. We can see that our program (Turing machine) halts for some inputs, and not for others. This does not violate the undecidability of the Halting Problem. Saying that the Halting Problem is undecidable says that no algorithm exists to deside the question **for every input**. It is not to say that we cannot decide correctly for certain special cases.

23.3 Example - Exponentiation

Prove that the following specification is satisfied under total correctness.

$(\!|n \geq 0 \wedge a \geq 0|\!)$

```
s = 1;
i = 0;
while (i < n) {
     s = s * a;
     i = i + 1;
}
```

$(\!|s \approx a^n|\!)$

Solution:

1. Partial Correctness
   (a) See the Logic18 slide deck for a proof of partial correctness.
   (b) Pay particular attention to the strenghening of the original loop invariant, $s \approx a^i$, to obtain the final loop invariant, $s \approx a^i \wedge i \leq n$.
   (c) This strengthening is required to make the final implication arising from the annotation provable.
2. Total Correctness: We verify that $n - i$ is a loop variant.
   (a) Before the loop starts, we have

    i. $n \geq 0$, and
    ii. $i = 0$, and hence
  $n - i \geq 0$.
  (b) Each time through the loop,
    i. n remains fixed, and
    ii. i increases by 1, so that
  $n - i$ decreases by 1.
  (c) We exit the loop when $n - i = 0$.
This shows that $n - i$ is a correct loop variant, so termination follows.

23.4 Undecidability of Program Verification

Total Correctness Problem: Is an arbitrary Hoare triple $(\!|P|\!)C(\!|Q|\!)$ satisfied under total correctness?

Theorem 23.4.1. *The Total Correctness Problem is undecidable.*

Proof. See the Logic18 slides. ■

Partial Correctness Problem: Is an arbitrary Hoare triple $(\!|P|\!)C(\!|Q|\!)$ satisfied under partial correctness?

Theorem 23.4.2. *The Partial Correctness Problem is undecidable.*

Proof. See the Logic18 slides. ■

24 Lecture 24 - Review and Wrap-Up

Outline

1. Review and Wrap-Up
2. Student Perception Surveys

24.1 Review and Wrap-Up

Stuff.

24.2 Student Perception Surveys

`https://perceptions.uwaterloo.ca`