CS 245 - by Alena Gusakov

Eason Li

2024 S

Contents

Definition 0.1: Proposition

A **Proposition** is a declarative sentence that is either true or false (but not both).

Example 0.1: Non-examples of Propositions

1. What time is it?
2. This sentence is False
3. Sentence Fragment: The dogs in the mark

   Something important to remind is that when it comes to the case when sentence fragment is the answer to a question, then it would become a "proposition".

Definition 0.2: Atomic and Compound

An **atomic** proposition is a proposition that cannot be breakdown into smaller propositions. **Compound** proposition are propositions that are not atomic.

Result 0.1

We denote the formal language of propositional logic as $\mathcal{L}^p$.

1 Propositional Logic

1.1 Lecture 1

Algorithm 1.1

We first break the sentence down into smaller atomic propositions.

1. $\neg$: not
2. $\wedge$: and
3. $\vee$: or
4. $\rightarrow$: implies
5. $\longleftrightarrow$: if and only if

Example 1.1

If it is sunny tomorrow, then I will play golf.

Solution: We can break it down into "it is sunny tomorrow" (S) and "I will play golf" (G), thus

$$(S \to G)$$

□

Example 1.2

He will eat an apple, or an orange, but not both.

Solution: Suppose we denote "eat an apple" as (A) and "eat an orange" as O, then we have

$$((A \vee O) \wedge (\neg(A \wedge O)))$$

□

Example 1.3

Nahdi eats a fruit if the fruit is an apple.

Solution: Denote "Nahdi eats an fruit" as (E) and "the fruit is an apple" as (A), then we have

$$(A \to E)$$

□

Example 1.4

Nahdi eats a fruit only if the fruit is an apple.

Solution: Denote "Nahdi eats an fruit" as (E) and "the fruit is an apple" as (A), then we have

$$(E \to A)$$

□

Example 1.5

If I ace CS 245, then I can get a job at Google, otherwise I will apply for the Geek Squad.

Solution: Denote "I ace..." as (A), "get Google job" as (G) and "Apply for Geek Squad" as (S), then

$$(A \to G) \wedge (\neg A \to S)$$

Remark: it is important to note that the solution

$$(A \to G) \vee S \equiv (\neg(A \to G) \to S)$$

is not correct, since the word "otherwise" only negates the atomic (A) rather than the whole statement, $(A \to G)$, thus we need to have the second part as

$$(\neg A \to S)$$

□

1.2 Unmarked Quiz 1

Result 1.1: Negation is not atomic

The proposition

$$\textit{I don't sing.}$$

is not atomic.

Discovery 1.1

We have

$$(p \vee q) \equiv (\neg p \to q)$$

Discovery 1.2

TFAE

1. $p \to q$
2. If p then q.
3. Whenever p, then q.
4. p is sufficient for q.
5. p only if q.
6. p implies q.
7. q if p.
8. q whenever p.
9. q is necessary for p.
10. q is implied by p.
11. Not p, unless q.
12. ...

Example 1.6

In this question, use these proposition symbols:

p: "I set my alarm."

q: "I decide to sleep in."

Select each correct translation for the propositional formula:

$$(\neg p \to q)$$

- ✓ A sufficient condition for my deciding to sleep in is that I don't set my alarm.
- ✓ I don't set my alarm only if I decide to sleep in.
- ✓ I set my alarm or I decide to sleep in.
- ✓ If I don't set my alarm, then I decide to sleep in.
- ✓ I set my alarm unless I decide to sleep in.

1.3 Lecture 2

Lecture 2 - Thursday, May 9

1.3.1 English Sentences with Logical Ambiguity

Example 1.7

1. Sidney will carry an unbrella unless it is sunny
2. Pigs can fly and the grass is red or the sky is blue

Definition 1.1

We view "unless" as *inclusive or*, that is, we would interpret the first sentence as

$$\text{carry an unbrella} \vee \text{it is sunny}$$

instead of

$$(\text{carry an unbrella} \vee \text{it is sunny}) \wedge (\neg(\text{carry an unbrella} \wedge \text{it is sunny}))$$

which would be the case when we have "otherwise".

Definition 1.2: Syntactically Correct Formulae

Let $\mathcal{P}$ be our set of proposition symbols. We define the set, $\text{Form}(\mathcal{L}^p)$, of syntactically correct formulae, inductively as follows:

1. A proposition symbol p in $\mathcal{P}$ is a formula.
2. If A is a formula in $\text{Form}(\mathcal{L}^p)$, then $(\neg A)$ is a formula in $\text{Form}(\mathcal{L}^p)$.
3. If A, B are formulae in $\text{Form}(\mathcal{L}^p)$, and $*$ is some binary logical connective, then $(A * B)$ is a formula in $\text{Form}(\mathcal{L}^p)$.

Theorem 1.1

Let $P : \text{Form}(\mathcal{L}^p) \to \{0, 1\}$ be some property defined on formulae in $\mathcal{L}^p$. For every syntactically correct formula A in $\text{Form}(\mathcal{L}^p)$, $P(A)$ holds, i.e. if $A \in \text{Form}(\mathcal{L}^p)$, then $P(A)$ is 1.

Lemma 1.1

Every formula in $\text{Form}(\mathcal{L}^p)$ has the same number of open patentheses as close parentheses.

Proof. Proof involves induction. For the induction step, we need to break it down into two cases: negation, and all other logic connectives (since negation would increase the number of open and close parentheses, while all other logic connectives have nothing to do with the number of both parentheses). □

Theorem 1.2: Unique Readability of Propositional Formulae

Every propositional formula A in $\text{Form}(\mathcal{L}^p)$ is exactly one of:

1. an atom
2. $(\neg B)$ for some $B \in \text{Form}(\mathcal{L}^p)$
3. $(B \wedge C)$ for some $B, C \in \text{Form}(\mathcal{L}^p)$
4. $(B \vee C)$ for some $B, C \in \text{Form}(\mathcal{L}^p)$
5. $(B \to C)$ for some $B, C \in \text{Form}(\mathcal{L}^p)$
6. $(B \leftrightarrow C)$ for some $B, C \in \text{Form}(\mathcal{L}^p)$ In each case A is of that form in exactly one way.

Definition 1.3: Proper Initial Segment

Suppose $A \in \text{Form}(\mathcal{L}^p)$, then the proper initial segment is a non-empty expression X such that $A = XY$ for Y is also a non-empty expression.

Lemma 1.2

Suppose $A \in \text{Form}(\mathcal{L}^p)$, every proper initial segment X of A has more open parentheses than close parentheses.

Proof. For the *base case*, suppose A is q for some propositional symbol q. Then q has no proper initial segment, thus this is vacuously true. Hence we can continue on our *induction step*:
Case 1: A is $(\neg B)$ for some $B \in \text{Form}(\mathcal{L}^p)$. Suppose every proper initial segment of B has more open than close parentheses (IH). After deviding this into four separate cases (`(` , `(¬` , `(¬`Z for Z a proper initial segment of B, and `(¬`B), we can complete the proof.
Case 2: A is $(B * C)$ for $B, C \in \text{Form}(\mathcal{L}^p)$ and $* \in \{\wedge, \vee, \to, \leftrightarrow\}$. Similarly, we can break it down into several smaller segments (in particular there are 6 subcases), and we could then complete our proof. □

1.4 Tutorial 1

Tutorial - Friday, May 10

Example 1.8

Denote "I eat healthily" as H and "I get good grades" as G, then the logic "I don't eat healthily whether or not I get good grades" can be expressed in logic as

$$(G \to \neg H) \wedge (\neg G \to \neg H)$$

1.5 Marked Quiz 1

Question 1

Select each item that represents a proposition.

Do not select those that are not propositions.

- ✓ ☑ The sentence "What time is it?" is a proposition.
- ✓ ☐ Tell me how many universities there are in Waterloo.
- ✓ ☐ Are there two universities in Waterloo?
- ✓ ☑ There are two universities in Waterloo.
- ✓ ☐ Table woman cat computer chair.
- ✓ ☑ The moon is made of green cheese.

Question 2

Select each atomic (simple) proposition.

Do not select those propositions that are not atomic (simple).

- ✓ ☐ I don't play tennis.
- ✓ ☑ I play tennis.
- ✓ ☐ I play tennis and I play golf.
- ✓ ☐ I play tennis unless I play golf.
- ✓ ☑ I play baseball.
- ✓ ☐ I play baseball if I play golf.

Question 3

For this question, use the following notation.

p : I feed my dog.
q : I walk my dog.
r : I give my dog a bath.
s : My dog is happy.

Match each English sentence with its correct translation into propositional logic.

✓ 7	If my dog is not happy, then I will not give her a bath.
✓ 3	If I walk my dog and give her a bath, then she will be happy.
✓ 2	My dog will be happy if I feed her.
✓ 9	I will walk my dog or give her a bath, but not both.
✓ 6	My dog won't be happy unless I walk her.

1. $s \rightarrow p$
2. $p \rightarrow s$
3. $(q \wedge r) \rightarrow s$
4. $q \wedge (r \rightarrow s)$
5. $q \rightarrow s$
6. $s \rightarrow q$
7. $(\neg s) \rightarrow (\neg r)$
8. $s \rightarrow r$
9. $(q \vee r) \wedge \neg(q \wedge r)$
10. $q \vee r$

1.6 Lecture 3

Lecture 3 - Thursday, May 14

Lemma 1.3

Let $A \in \text{Form}(\mathcal{L}^p)$, then the first symbol of A is either '(' or a proposition symbol.

Proof. For the base case, we have A is a proposition symbol, hence we are done. For the induction step, in either case, (A is in the form of $(\neg B)$ or in the form of $(B * C)$), A starts with '(', thus completing the proof. □

Definition 1.4: Syntactic Equality

Two formulae $A, B \in \text{Form}(\mathcal{L}^p)$ are said to be syntactically equal if they agree on every symbol. We denote this using "=" symbol.

Example 1.9

As an example,

$$(p \to (q \vee r)) = (p \to (q \vee r))$$
$$((p \wedge q) \wedge r) \neq (p \wedge (q \wedge r))$$

1.6.1 Proof of Unique Readability Theorem 1.2

Proof. Suppose we have $A \in \text{Form}(\mathcal{L}^p)$

1. Base case: A is some proposition symbol p
2. Indcution Step:
   (a) A is $(\neg B)$ for some $B \in \text{Form}(\mathcal{L}^p)$
   A cannot be an atom since atom does not start with a bracket. Moreover, A cannot be $(\neg B')$ for some $B' \neq B$ since otherwise we would have $(\neg B) = (\neg B')$ which is impossible. Finally, A cannot be in the form of $(B * C)$ either because B is either an atom or starts with '('.
   (b) A is $(B * C)$ for some $B, C \in \text{Form}(\mathcal{L}^p)$
   Trivially, A cannot be an atom since $p \neq (B * C)$ for all $p \in \text{Form}(\mathcal{L}^p)$. A cannot be in the form of $(\neg D)$ either, which has already been stated above. Last but not the least, the question becomes "Can A be in the form of $B'\$C'$" for $B', C' \in \text{Form}(\mathcal{L}^p), B' \neq B, C' \neq C$ and $\$ \in \{\wedge, \vee, \to, \leftrightarrow\}$. Now we have three cases:
      i. Suppose \$ is in the original position as $*$, then we have that it is impossible to have $B' \neq B$ and $C' \neq C'$. This is not the case.
      ii. Suppose that \$ is in B, then we know that B' is a proper initial segment of B, which means that the open parentheses of B' is greater than the close parentheses (1.2), thus $B' \notin \text{Form}(\mathcal{L}^p)$.
      iii. Similarly suppose $\$ \in C$, then B is a proper initial segment of B', and a similar (as the one above) proof follows.

□

Exercise:

1. Show that every formula in $\text{Form}(\mathcal{L}^p)$ has at least one proposition symbol.

   Proof. (a) Base case: A is some proposition ssymbol p, then A it self is a proposition symbole

   (b) Induction Step:

   i. A is in the form of $(\neg B)$ for some $B \in \text{Form}(\mathcal{L}^p)$, we know that B contains at least one proposition symbol, so $A \supseteq B$ also contains at least one proposition symbol.

   ii. A is in the form of $(B * C)$ for some $B, C \in \text{Form}(\mathcal{L}^p)$ and $* \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, we know that B and C contain at least one proposition symbol, so similarly, $A \supseteq B, C$ also contains at least one proposition symbol.

   thus completing the proof. □

2. Suppose A is a formula with m proposition symbols and n binary logical connectives (i.e. $\wedge$, $\vee$, $\rightarrow$, or $\leftrightarrow$). Show that $m = n + 1$.

   Proof. (a) Base case: A is some proposition ssymbol p, then we know that $m = 1$ and $n = 0$, thus $m = n + 1$.

   (b) Induction Step:

   i. A is in the form of $(\neg B)$ for some $B \in \text{Form}(\mathcal{L}^p)$, for B, we have $m_B = n_B + 1$, and we can also notice that $m_A = m_B$ and $n_A = n_B$.

   ii. A is in the form of $(B * C)$ for some $B, C \in \text{Form}(\mathcal{L}^p)$ and $* \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, we know that $m_B = n_B + 1$ and $m_C = n_C + 1$. Moreover, we can also notice that $m_A = m_B + m_C$ and $n_A = n_B + n_C + 1$, thus we have

   $$m_A = m_B + m_C = n_B + 1 + n_c + 1 = n_A + 1$$

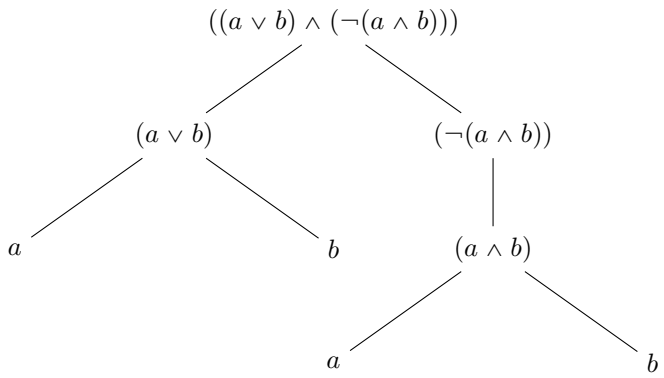
   thus completing the proof. □

Definition 1.5: Parse Tree

A **parse tree** of a formula is a visual representation of the syntactic structure of the formula.

Example 1.10

What is the parse tree for the following proposition: $((a \vee b) \wedge (\neg(a \wedge b)))$,



Definition 1.6: Generation Sequence

A **generation sequence** is an explicit series of steps for how to construct a syntactically correct formula according to the definition.

Example 1.11

$(p \to (q \leftrightarrow r))$ **Solution:**

1. q is in the core set
2. r is in the core set
3. Apply iff to (1) and (2) to get $(q \leftrightarrow r)$
4. p is in the core set
5. Apply implication to (4) and (3) to get $(p \to (q \leftrightarrow r))$

□

Example 1.12

$((p \wedge q) \to (\neg r))$ **Solution:**

1. p is in the core set
2. q is in the core set
3. r is in the core set ...

□

1.6.2 Precedence Rule

Result 1.2: Precedence Rule

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

Now we have:

$$\text{syntactically correct formula} \subseteq \text{propositional formula}$$

because for {propositional formula}, parentheses may be omitted.

Definition 1.7: Truth Evaluation

A truth valuation $t : \mathcal{P} \to \{0, 1\}$ is a function from the set of proposition symbols $\mathcal{P}$ to $\{0, 1\}$.

Example 1.13

Suppose $\mathcal{P} = \{p, q, r\}$ and define

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

This is not a truth evaluation since it does not map all the element in $\mathcal{P}$ to a value in $\{0, 1\}$.

Definition 1.8: Tautological Equivalence

Two formulae $A, B \in \text{Form}(\mathcal{L}^p)$ are **tautologically equivalent** if they have the same truth table, and we denote this as $A \models\!\models B$.

1.7 Lecture 4

Lecture 4 - Thursday, May 16

Lemma 1.4

Let $\mathcal{P}$ be a set of proposition symbols, and let t be a truth evaluation defined on $\mathcal{P}$. Let C be a propositional formula in $\text{Form}(\mathcal{L}^p)$. Then $C^t \in \{0, 1\}$.

Proof. 1. Base Case: C is p for some proposition symbol p, then $C^t = p^t \in \{0, 1\}$.

2. Induction step:

   (a) Case 1: C is $(\neg A)$ for some $A \in \text{Form}(\mathcal{L}^p)$, then

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

   (b) Case 2: $C = (A \wedge B)$ for some $A, B \in \text{Form}(\mathcal{L}^p)$, since $A^t \in \{0,1\}$ and $B^t \in \{0,1\}$, so

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

   (c) Similar proof follow...

□

1.7.1 Tautology

Definition 1.9: Tautology

A proposition formula C is called a **tautology**, or valid formula, if $C^t = 1$ for every truth evaluation t.

Definition 1.10: Satisfiable

A propositional formula C is called **satisfiable** if there is some truth valuation t such that $C^t = 1$.

Definition 1.11: Contradiction

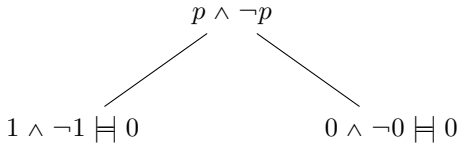
A propositional formula C is called a **contradiction**, or not satisfiable, if $C^t = 0$ for every truth valuation t.

Result 1.3

It is important to notice that the opposite of *contradiction* is *satisfiable*.

Example 1.14: Evaluation Tree

Suppose we have $p \wedge \neg p$,



Definition 1.12: Σ

We denote the sets of propositional formulae using Σ

Definition 1.13: Satisfaction of a Set of Formulae

Let Σ be a set of propositional formulas in Form($\mathcal{L}^p$), and let t be a truth valuation. We say that t **satisfies** Σ if, for every $C \in \Sigma$, we have $C^t = 1$. We denote this as $\Sigma^t = 1$. If there is some $C \in \Sigma$ such that $C^t = 0$, then we say that t **does not satisfy** Σ. We denote this as $\Sigma^t = 0$.

Definition 1.14: Satisfiability of Set of Formulae

Let Σ be a set of formulas. We say that Σ is satisfiable if there exists some truth valuation t such that $\Sigma^t = 1$, and not satisfiable otherwise.

Discovery 1.3: Empty Set

$$\varnothing^t = 1$$

Definition 1.15: Tautological Consequence

If Σ tautologically implies a formula C, then we say that C is a tautological consequence of Σ. We denote this as $\Sigma \models C$.

1.7.2 Replacability Theorem and Duality Theorem

Theorem 1.3: Replacability Theorem

Let A be a formula containing a subformula B. Assume $B \models\!\models C$, 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 $A' \models\!\models A$.

Proof. Exercise □

Theorem 1.4: Duality Theorem

Suppose A is a formula composed only of atoms and the connectives $\neg, \vee, \wedge$, by the formation rules for these three connectives. Suppose we obtain $\Delta(A)$ from A by simultaneously

1. replacing all occurrences of $\wedge$ with $\vee$,
2. replacing all occurrences of $\vee$ with $\wedge$, and
3. replacing each atom with its negation.

Then $\Delta(A) \models\!\models \neg A$.

1.8 Unmarked Quiz 2

Example 1.15

Select all the ingredients that would be needed in a proof by structural induction that every propositional logic formula, C, has some property $R(C)$. Do not select the ingredients which would not be needed.

- ✓ $R(C)$ holds for every C of the form p, for any proposition symbol p.
- ▫ $R(C)$ holds for every C of the form $(\neg p)$, for any proposition symbol p.
- ▫ $R(C)$ holds for every C of the form $(p \wedge q)$, for any proposition symbols p and q.
- ✓ $R(C)$ holds for every C of the form $(\neg A)$, such that $R(A)$ holds.
- ✓ $R(C)$ holds for every C of the form $(A \wedge B)$, such that $R(A)$, $R(B)$ hold.

Example 1.16

Select the FIRST line that contains an error, or is simply inappropriate at that point.

To be proven: all natural numbers are even.

- ▫ Basis: 0 is even.
- ▫ Inductive Step: Let k be an arbitrary natural number.
- ▫ Inductive Hypothesis: The claim holds for all natural numbers k.
- ▫ To prove in the inductive step, namely that $k + 1$ is even:
- ▫ Observe that $k + 1 = (k - 1) + 2$.
- ✓ Since $k - 1 < k$, therefore the inductive hypothesis implies that $k - 1$ is even.
- ▫ Thus $k - 1 = 2m$, for some integer m, and $k + 1 = 2(m + 1)$ is also even.
- ▫ Therefore, by the principle of mathematical induction, all natural numbers are even.

1.9 Unmarked Quiz 3

Example 1.17

For the three formulas below, determine whether any of them are tautologically equivalent.

- ▫ $((p \vee q) \to (\neg r))$
- ▫ $(r \vee (p \wedge q))$
- ▫ $((\neg q) \to (r \to p))$
- ▫ All three formulas are equivalent.
- ✓ No two of the three are equivalent.

Example 1.18

For each of the formulas below, determine whether it is tautologically equivalent to the formula $(p \to (q \to (\neg r)))$.

- ✓ $(\neg(p \wedge (q \wedge r)))$
- ✓ $(q \to (p \to (\neg r)))$
- ✓ $(p \to (r \to (\neg q)))$
- ✓ $(r \to (q \to (\neg p)))$

1.10 Unmarked Quiz 4

Example 1.19

Let Σ be $\{p \to q, q \to p, q \to (\neg p), (\neg q) \to p\}$. Select all formulas that are tautologically implied by Σ.

- ✓ $p \wedge (\neg p)$
- ✓ $p \vee (\neg p)$
- ✓ $(\neg q) \to (\neg p)$
- ✓ $\neg p$
- ✓ $(\neg p) \to p$

Explanation: Σ is not satisfiable, thus it implies everything.

Example 1.20

Let P be the set of proposition symbols $\{p, q, r\}$. Let $\varnothing$ denote the empty set of propositional formulas. Select all the truth valuations, t, such that $\varnothing^t = 1$.

- ✓ $p^t = q^t = r^t = 0$

✓ $p^t = q^t = 0; r^t = 1$

✓ $p^t = q^t = 1; r^t = 0$

✓ $p^t = 1; q^t = r^t = 0$

✓ $p^t = q^t = r^t = 1$

Explanation: $\varnothing^t = 1$ for whatever t.

1.11 Tutorial 2

This is an interesting question.

Example 1.21

There is an island in which certain inhabitants, called knights, always tell the truth, and others, called knaves, always lie. It is assumed that every inhabitant of this island is either a knight or a knave.
Someone asks X: *"Are you a knight?"*
X replies: *"If I am a knight, then I will eat my hat."*
Prove that X will eat his hat.

Proof. Let p denote the event that "X is a knight" and q that "X will eat his hat".

1. **X is a knight**.
   Therefore $p^t = 1$ and $(p \to q)^t = 1$ (because X tells the truth). This further implies $q^t = 1$. Thus in this case X has to eat his hat.

2. **X is a knave**.
   Therefore $p^t = 0$ and $(p \to q)^t = 0$ (because X lies). However, $(p \to q)^t = 1$ no matter what the value of q^t is. Hence we have reached a contradiction (because $(p \to q)^t = 0 = 1$). Therefore this case cannot occur.

□

Example 1.22

Show that

$$\{(p_i \to p_{i+1}) : i \in \mathbb{N}\} \vDash (p_{77} \to p_{79}).$$

Proof. Let t be any truth valuation such that for every $i \in \mathbb{N}$, $(p_i \to p_{i+1})^t = 1$. Now, if $(p_{77})^t = 0$, then $(p_{77} \to p_{79})^t = 1$ regardless of the value of $(p_{79})^t$. Otherwise, if $(p_{77})^t = 1$, since $(p_{77} \to p_{78})^t = 1$, it must be that $(p_{78})^t = 1$. Similarly, since $(p_{78})^t = 1$ and $(p_{78} \to p_{79})^t = 1$, $(p_{79})^t = 1$. Lastly, since $(p_{77})^t = 1$ and $(p_{79})^t = 1$, therefore $(p_{77} \to p_{79})^t = 1$. □

1.12 Lecture 5

Lecture 5 - Thursday, May 23

1.12.1 Propositional Calculus Laws

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

Example 1.23: for Theorem (1.4)

Let $A = (p \wedge \neg q) \vee r$ and show that $\neg A \models\!\models \Delta(A)$.
Solution: We know that

$$\begin{aligned}\Delta(A) &\models\!\models (\neg p \vee \neg\neg q) \wedge \neg r \\ &\models\!\models (\neg p \vee q) \wedge \neg r\end{aligned}$$

The solution completes by writing out the truth table.

□

Definition 1.16: Literal, Disjuntion, and Conjuction

A **literal** is either a propositon symbol, or the negation of a propositional symbol.

A propositional formula made up only of literals and $\vee$, i.e. **disjunctions**, is called a **disjunctive clause**. In a disjunctive clause, the literals are called disjuncts. Example: $p \vee \neg q \vee r$

A propositional formula made up only of literals and $\wedge$, i.e. conjunctions, is called a **conjunctive clause**. In a conjunctive clause, the literals are called **conjuncts**. Example: $\neg p \wedge q \wedge \neg r$.

1.12.2 DNF & CNF

Definition 1.17: Disjunctive Normal Form

A formula that is written as a disjunction of conjunctive clauses is said to be in **Disjunctive Normal Form** (DNF). In particular, it is in the form of

$$(A_{11} \wedge \cdots \wedge A_{1n_1}) \vee \cdots \vee (A_{k1} \wedge \cdots \wedge A_{kn_k})$$

Algorithm 1.2: DNF Truth Table

Suppose we have some formula A with truth table

p	q	r	A
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

Then $A \models\models (p \wedge q \wedge r) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge \neg q \wedge r)$.

Example 1.24: Convert to DNF

Convert the following formula to DNF: $(\neg p \to q) \to (p \wedge (q \vee r))$.

$$
\begin{aligned}
&(\neg p \to q) \to (p \wedge (q \vee r)) \\
&\models\models \neg(\neg\neg p \vee \neg q) \vee (p \wedge (q \vee r)) && \textit{Step } 1 : \textit{Eliminate } \to, \leftrightarrow \\
&\models\models (\neg\neg\neg p \wedge \neg\neg q) \vee (p \wedge (q \vee r)) && \textit{Step } 2 : \textit{Push negation inwards} \\
&\models\models (\neg p \wedge q) \vee (p \wedge (q \vee r)) && \textit{Step } 3 : \textit{Eliminate} \neg\neg \\
&\models\models (\neg p \wedge q) \vee (p \wedge q) \vee (p \wedge r) && \textit{Step } 4 : \textit{Distributive laws}
\end{aligned}
$$

Discovery 1.4

Every propositional formula be converted to a tautologically equivalent formula that is in Disjunctive Normal Form.

Definition 1.18: Conjunctive Normal Form

A formula that is written as a conjunction of disjunctive clauses is said to be in **Conjunctive Normal Form** (CNF). In particular, it is in the form of

$$(A_{11} \vee \cdots \vee A_{1n_1}) \wedge \cdots \wedge (A_{k1} \vee \cdots \vee A_{kn_k})$$

Algorithm 1.3: CNF Truth Table

One method of obtaining CNF of a formula A is via its truth table, like we did for DNF. However, this process is a bit more involved:

1. Add an additional column for its negation $\neg A$
2. Write down DNF of $\neg A$ from the new column as before, and call this formula B.

3. Take the dual of B, i.e. $\Delta(B)$.

4. Eliminate double negations.

This will produce a formula that is the CNF of A. This is because $\Delta(B) \models\!\models \Delta(\neg A) \models\!\models \neg\neg A \models\!\models A$.

Example 1.25: Convert to CNF

Convert the following formula to CNF: $\neg((p \leftrightarrow \neg q) \wedge \neg r)$.

$$
\begin{aligned}
&\neg((p \leftrightarrow \neg q) \wedge \neg r) \\
&\models\!\models \neg(((p \wedge \neg q) \vee (\neg p \wedge \neg\neg q)) \wedge \neg r) && \textit{Step 1 : Eliminate } \rightarrow, \leftrightarrow \\
&\models\!\models (((\neg p \vee \neg\neg q) \wedge (\neg\neg p \vee \neg\neg\neg q)) \vee \neg\neg r) && \textit{Step 2 : Push negation inwards} \\
&\models\!\models (((\neg p \vee q) \wedge (p \vee \neg q)) \vee r) && \textit{Step 3 : Eliminate} \neg\neg \\
&\models\!\models (\neg p \vee q \vee r) \wedge (p \vee \neg q \vee r) && \textit{Step 4 : Distributive laws}
\end{aligned}
$$

Exercise: Convert $(p \wedge q) \rightarrow (r \vee s)$ into CNF.
Solution: We simply have

$$
\begin{aligned}
(p \wedge q) \rightarrow (r \vee s) &\models\!\models \neg(p \wedge q) \vee (r \vee s) \\
&\models\!\models \neg p \vee \neg q \vee r \vee s
\end{aligned}
$$

□

Exercise: Convert $\neg(p \vee q \vee r)$ into CNF.
Solution: We simply have

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

□

Exercise: Convert $p \rightarrow ((s \vee t) \wedge \neg(s \wedge t))$ into CNF.
Solution: We simply have

$$
\begin{aligned}
p \rightarrow ((s \vee t) \wedge \neg(s \wedge t)) &\models\!\models \neg p \vee ((s \vee t) \wedge \neg(s \wedge t)) \\
&\models\!\models \neg p \vee ((s \vee t) \wedge (\neg s \vee \neg t)) \\
&\models\!\models (\neg p \vee s \vee t) \wedge (\neg p \vee \neg s \vee \neg t)
\end{aligned}
$$

□

Definition 1.19: Arity of Functions

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

Example 1.26

1. $f(x) \coloneqq x^2$ is a *unary* function
2. $g(x, y) \coloneqq x * y$ is a *binary* function

Result 1.4

Our logical connectives are also functions. A logical connective with arity n is a function

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

Example 1.27

1. $\neg : \{0, 1\} \to \{0, 1\}$ is a unary function
2. $\wedge : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$ is a binary function

Definition 1.20: Adequate for Propositional Logic

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

Theorem 1.5

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

Proof. Let $f : \{0, 1\}^n \to \{0, 1\}$ be some logical connectives/functions of arity n.

1. Write truth table for f (see example 1.2).
2. Obtain formula A_{S_0} in DNF as described.

We are done, as by construction, $A_{S_0} \models\!\models f$ and only uses $\{\neg, \wedge, \vee\}$. □

Discovery 1.5

Claim: $\{\neg, \wedge\}$, $\{\neg, \vee\}$, and $\{\neg, \to\}$ are all adequate for propositional logic.

Proof. We have

$$\begin{aligned}(A \wedge B) &\models\!\models \neg\neg(A \wedge B) \\ &\models\!\models \neg(\neg A \vee \neg B)\end{aligned}$$

and the other cases follow the similar proof. □

1.12.3 Boolean Algebras

Definition 1.21: Boolean Algebras

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

1. dentity laws: $x + 0 = x$ and $x \cdot 1 = x$.
2. Complement laws: $x + \overline{x} = 1$, $x \cdot \overline{x} = 0$.
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 $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

1.13 Lecture 6

Lecture 6 - Tuesday, May 28

Example 1.28: Examples for Boolean Algebras

The set of subsets of a universal set $\mathcal{U}$, with the union operator $\cup$ as $+$, the intersection operator $\cap$ as $\cdot$, the set complementation operator ^{c} as $^-$, the empty set $\varnothing$ as 0, and the universal set $\mathcal{U}$ as 1, is a Boolean algebra.

Definition 1.22: Propositional Logic as Boolean Algebra

Four types of symbols in Form($\mathcal{L}^{p+}$):

1. Proposition symbols: p, q, r, p_1, etc;
2. Logical connectives: $\neg, \wedge, \vee, \wedge, \leftrightarrow$;
3. Punctuation: (and);
4. Identities: 0 and 1.

Construct Form($\mathcal{L}^{p+}$) using the same formation rules as Form($\mathcal{L}^{p}$), except we treat 0, 1 as proposition symbols syntactically.

Discovery 1.6

Observe that Form($\mathcal{L}^{p}$) $\subseteq$ Form($\mathcal{L}^{p+}$).

Exercise: Show that De Morgan's Laws hold in Boolean algebra:

1. $\overline{(x + y)} = \overline{x} \cdot \overline{y}$

2. $\overline{(x \cdot y)} = \overline{x} + \overline{y}$

Proof. Here we show the first equation: We have

$$\begin{aligned}(x+y)(\bar{x}\cdot\bar{y}) &= (x\bar{x}\bar{y}) + (y\bar{x}\bar{y}) \\ &= (0\bar{y}) + (0\bar{x}) \\ &= 0 = (x+y)\overline{(x+y)}\end{aligned}$$

Similarly,

$$\begin{aligned}(x+y) + (\bar{x}\cdot\bar{y}) &= (x+y+\bar{x})\cdot(x+y+\bar{y}) \\ &= (1+y)\cdot(1+x) \\ &= 1 = (x+y) + \overline{(x+y)}\end{aligned}$$

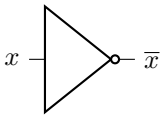
Therefore we can conclude that $\overline{(x+y)} = \overline{x}\cdot\overline{y}$. □

Definition 1.23

An n-variable **Boolean function** is a function $f : \{0,1\}^n \to \{0,1\}$. An electronic computer is made up of a number of **circuits**, each of which implements a Boolean function. The smallest building blocks of circuits are called **logic gates**. We will start with $\neg, \wedge, \vee$

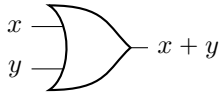
Definition 1.24: The NOT Gate

An inverter, or a NOT gate, is a logic gate that implements negation ($\neg$).



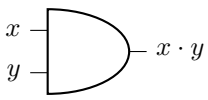
Definition 1.25: The OR Gate

The inputs to this gate are the values of two Boolean variables. The output is the Boolean Sigma + (denoting $\vee$) of their values.



Definition 1.26: The AND Gate

The inputs to this gate are the values of two Boolean variables. The output is the Boolean product (denoting $\wedge$) of their values.



1.13.1 Circuit

Definition 1.27: Combinational Circuits

Combinational logic circuits (sometimes called combinatorial circuits) are *memoryless* digital logic circuits whose output is a function of the present value of the inputs only.
A combinatorial circuits) is implemented as a combination of NOT gates, OR gates, and AND gates.

Example 1.29

Suppose our light fixture is controlled by 3 switches, and our circuit needs 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. Furthermore, when all switches are on (closed circuit), the light is on.

Design a circuit that accomplishes this task, when there are three switches.

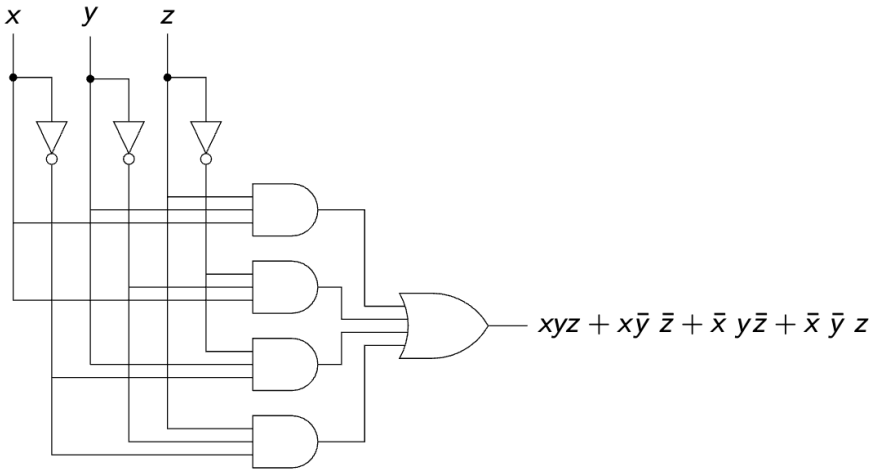
Solution: We have three switches, so we have three inputs: x, y, z. Let $x = 1$ if the first switch is closed, and $x = 0$ if it is open, similar for y, z. Let $F(x, y, z) = 1$ if the light is on, and $F(x, y, z) = 0$ if it's off. Since we specified that the light is on when all switches are closed, we have $F(1, 1, 1) = 1$. From there, we can fill in a truth table and find

$$F(x, y, z) \models\models (x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z)$$

Therefore we can find the circuit implementing the function

$$xyz + x\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z$$

.



□

Theorem 1.6: Circuit Minimization

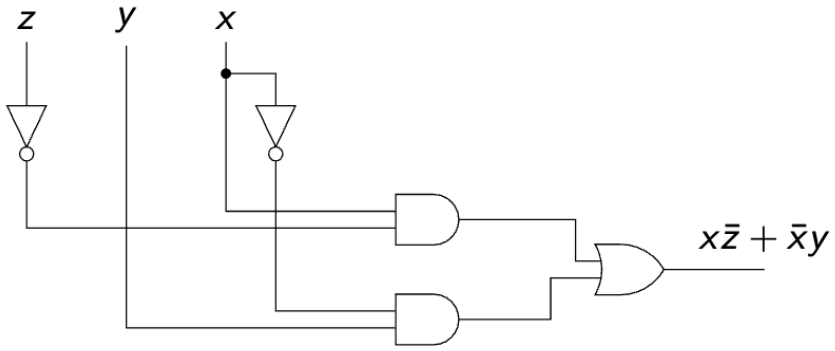
We can use tautological equivalence to simplify Boolean functions and the circuits that implement them.

Example 1.30

Minimize the circuit implementing the formula with the following truth table.

p	q	r	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

$$
\begin{aligned}
A &\models\models xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}y\bar{z} \\
&\models\models x\bar{z}y + x\bar{z}\bar{y} + \bar{x}yz + \bar{x}y\bar{z} && \text{(commutativity of } \cdot\text{)} \\
&\models\models (x\bar{z}y + x\bar{z}\bar{y}) + (\bar{x}yz + \bar{x}y\bar{z}) && \text{(associativity of } +\text{)} \\
&\models\models x\bar{z}(y + \bar{y}) + \bar{x}y(z + \bar{z}) && \text{(distributivity)} \\
&\models\models x\bar{z}(1) + \bar{x}y(1) && \text{(complement)} \\
&\models\models x\bar{z} + \bar{x}y && \text{(identity)}
\end{aligned}
$$



1.13.2 Code Analysis

Example 1.31

Consider the code fragment:

```
if (q1 or not q2) then
    if (not (q2 and q3)) then
        P1
    else
        if (q2 and not q3) then
            P2
        else
            P3
else
    P4
```

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

Now, we can observe under what combination of conditions q1, q2, q3 are various fragments of code executed.

Definition 1.28: Dead Code

Dead code is code that is never executed.

Example 1.32

P2 is dead code. In order for P2 to be executed, it has to be true that (q2 and not q3), AND (not (not (q2 and q3))), AND (q1 or not q2). The condition for P2 to be executed is to satisfy

$$(q1 \vee \neg q2) \wedge \neg\neg(q2 \wedge q3) \wedge (q2 \wedge \neg q3) \models\!\models (q1 \vee \neg q2) \wedge (q2 \wedge q3) \wedge (q2 \wedge \neg q3) \models\!\models 0$$

Since this condition can never be true, P2 is dead code.

1.14 Unmarked quiz 6 - Adequate Sets Boolean Algebra

Example 1.33

To prove that $\{\wedge, \vee\}$ is not adequate for propositional logic, we could argue that (select all that apply)

- ✓ $\rightarrow$ cannot be implemented using only $\{\wedge, \vee\}$.
- ▫ $\vee$ cannot be implemented using only $\{\wedge, \vee\}$.
- ✓ $\neg$ cannot be implemented using only $\{\wedge, \vee\}$.
- ▫ $\wedge$ cannot be implemented using only $\{\wedge, \vee\}$.
- ✓ A connective, f, of arity 100, cannot be implemented using only $\{\wedge, \vee\}$.

Example 1.34

elect all the Boolean expressions which are equivalent to $(a + \bar{a})(e + \bar{e})$.

- ✓ $\overline{a\bar{a}}$.
- ✓ $\overline{a\bar{a} + e\bar{e}}$.
- ✓ $\bar{a} + a$.
- ✓ $\bar{e} + e$.
- ✓ 1.

1.15 Unmarked quiz 7 - Circuit Design Code Analysis

Exercise: The minimum number of gates needed to implement the Boolean expression $a\bar{e} + \bar{a}e$ is $\underline{4}$

Proof. $a\bar{e} + \bar{a}e \models\!\models (a + e)\overline{(ae)}$. □

1.16 Tutorial 3

1.16.1 Adequate Sets of Logic Connectives

Prove that $\{\wedge, \rightarrow\}$ is **not** an adequate set of connectives for propositional logic.

Proof. STP it is not possible to implement $\neg$ using only $\wedge$ and $\rightarrow$ connective symbols.

Claim: Let $\mathcal{P}$ be a set of proposition symbols. Let A be a propositional formula over $\mathcal{P}$, constructed using only $\wedge$ and $\rightarrow$ connective symbols. Let t be the truth valuation that sets every proposition symbol in $\mathcal{P}$ to 1. Then $A^t = 1$.

Proof of Claim: The proof is by structural induction on A. Let $R(A)$ be the assertion that $A^t = 1$.

1. Base case:
   In this case A is a proposition symbol, p. Then by the definition of t, we have $A^t = p^t = 1$. Therefore $R(A)$ holds in the base case.

2. Induction step:
   There are two sub-cases:

   (a) A is $(B \wedge C)$, for some propositional formulas, B and C, constructed using only $\wedge$ and $\rightarrow$ connective symbols. The induction hypothesis is $R(B)$ and $R(C)$, in other words $B^t = 1 = C^t$. Then $A^t = (B \wedge C)^t = 1$. Therefore $R(A)$ holds in this case.

   (b) A is $(B \rightarrow C)$, for some propositional formulas, B and C, constructed using only $\wedge$ and $\rightarrow$ connective symbols. The induction hypothesis is $R(B)$ and $R(C)$, in other words $B^t = 1 = C^t$. Then $A^t = (B \rightarrow C)^t = 1$. Therefore $R(A)$ holds in this case.

By the principle of structural induction, $R(A)$ holds for any propositional formula A, constructed using only $\wedge$ and $\rightarrow$ connective symbols. □

1.16.2 Boolean Algebra

Simplify this Boolean algebra expression as much as possible: $(\bar{x}\bar{y})(x+y)$.

Proof. We have

$$\begin{aligned}
(\bar{x}\bar{y})(x+y) &\models\!\models (\bar{x}\bar{y})x + (\bar{x}\bar{y})y && (Distributivity)\\
&\models\!\models \bar{x}(\bar{y}x) + \bar{x}(\bar{y}y) && (Associativity)\\
&\models\!\models \bar{x}(\bar{y}x) + \bar{x}0 && (Complement)\\
&\models\!\models \bar{x}(\bar{y}x) + 0 && (Domination)\\
&\models\!\models \bar{x}(\bar{y}x) && (Identity)\\
&\models\!\models \bar{x}(x\bar{y}) && (Commutativity)\\
&\models\!\models (\bar{x}x)\bar{y} && (Associativity)\\
&\models\!\models 0\bar{y} && (Complement)\\
&\models\!\models 0 && (Domination)
\end{aligned}$$

□

1.17 Marked Quiz 2

Example 1.35

Let A be the propositional formula $(p \to q) \wedge r$. Let t be the truth valuation defined by $p^t = 0$, $q^t = 0$ and $r^t = 1$. Then $A^t = 1$.

✓ True

Example 1.36

Let A be the propositional formula $(p \wedge q) \wedge (\neg q \wedge \neg p)$. Then A is

✗ a tautology.

✗ contingent (i.e. satisfiable but not a tautology).

✓ unsatisfiable.

Example 1.37

Let A be a propositional formula. If A is satisfiable, then A is <u>never</u> a contradiction.

Example 1.38

Let Σ be $\{(p \vee q) \to r, (q \wedge r) \to p\}$. Let t be the truth valuation defined by $p^t = 0$, $q^t = 0$ and $r^t = 0$. Then $\Sigma^t = 1$.

✓ True

Example 1.39

Let t be a truth valuation. Then $\varnothing^t = 1$ *always* holds.

Example 1.40

$\varnothing \vDash q \wedge \neg q$.

✗ False

Example 1.41

$\{p \wedge \neg p\} \vDash q \wedge \neg q$.

✓ True

Example 1.42

Let Σ be a set of propositional formulas. Let t be a truth valuation. If $\Sigma^t = 1$, then Σ is *always* satisfiable.

1.18 Lecture 7

Lecture 7 - Thursday, May 30

Definition 1.29: Formal Deduction

Formal Deduction is a proof system for propositional logic.

Algorithm 1.4: Formal Deduction

A proof in formal deduction is a sequence of lines of the form $\Sigma \vdash A$ for some set Σ and some formula A. Each line of a proof has columns, which correspond to the following in order from left to right:

1. A line number,
2. a set of premise/asSigmaption formulas,
3. $\vdash$
4. a conclusion formula, and
5. a justification, either "by supposition" or by citing a proof rule of Formal Deduction based on previous lines in the proof.

The last line of the proof must be of the form $\Sigma \vdash C$, where C is the formula we want to prove from Σ.

Definition 1.30: Proof Rules

A **proof rule** takes one or more lines of a proof as input (where the lines correspond to formulas) and returns a new line of the proof as output.

1.18.1 Proof Rules

Algorithm 1.5: Proof Rules

1. $(\in)$: Premise axiom.

$$\text{If } A \in \Sigma, \text{ then } \Sigma \vdash A.$$

2. (Ref) : Reflexive rule.

$$A \vdash A.$$

3. $(+)$: Addition of premises rule.

$$\begin{array}{lccc} \text{If} & \Sigma_1 & \vdash & A, \\ \text{then} & \Sigma_1, \Sigma_2 & \vdash & A. \end{array}$$

Addition of premises rule $(+)$ is also called Monotonicity.

Algorithm 1.6: Negation Proof Rules

1. $(\neg -)$: NOT-elimination

$$\begin{array}{rccc} \text{If} & \Sigma, (\neg A) & \vdash & B, \\ & \Sigma, (\neg A) & \vdash & (\neg B), \\ \text{then} & \Sigma & \vdash & A. \end{array}$$

2. $(\neg +)$: NOT-introduction

$$\begin{array}{rccc} \text{If} & \Sigma, A & \vdash & B, \\ & \Sigma, A & \vdash & (\neg B), \\ \text{then} & \Sigma & \vdash & (\neg A). \end{array}$$

NOT-introduction $(\neg +)$ is also called Reductio Ad Absurdum.

Algorithm 1.7: Implication Proof Rules

1. $(\rightarrow -)$: IMP-elimination

$$\begin{array}{rccc} \text{If} & \Sigma & \vdash & (A \rightarrow B), \\ & \Sigma & \vdash & A, \\ \text{then} & \Sigma & \vdash & B. \end{array}$$

IMP-elimination $(\rightarrow -)$ is also called Modus Ponens.

2. $(\rightarrow +)$: IMP-introduction

$$\begin{array}{rccc} \text{If} & \Sigma, A & \vdash & B, \\ \text{then} & \Sigma & \vdash & (A \rightarrow B). \end{array}$$

IMP-introduction $(\rightarrow +)$ is also called Deduction Theorem.

Algorithm 1.8: And Proof Rules

1. $(\wedge -)$: AND-elimination

$$\begin{array}{rccc} \text{If} & \Sigma & \vdash & (A \wedge B), \\ \text{then} & \Sigma & \vdash & A, \\ & \Sigma & \vdash & B. \end{array}$$

2. $(\wedge +)$: AND-introduction

$$\begin{array}{rccc} \text{If} & \Sigma & \vdash & A, \\ & \Sigma & \vdash & B, \\ \text{then} & \Sigma & \vdash & (A \wedge B). \end{array}$$

Algorithm 1.9: Or Proof Rules

1. $(\vee -)$: OR-elimination

$$\begin{array}{lrcl} \text{If} & \Sigma, A & \vdash & C, \\ & \Sigma, B & \vdash & C, \\ \text{then} & \Sigma, (A \vee B) & \vdash & C. \end{array}$$

2. $(\vee +)$: OR-introduction

$$\begin{array}{lrcl} \text{If} & \Sigma & \vdash & A, \\ \text{then} & \Sigma & \vdash & (A \vee B), \\ & \Sigma & \vdash & (B \vee A). \end{array}$$

Algorithm 1.10: Iff Proof Rules

1. $(\leftrightarrow -)$: IFF-elimination

$$\begin{array}{lrcl} \text{If} & \Sigma & \vdash & (A \leftrightarrow B), \\ & \Sigma & \vdash & A, \\ \text{then} & \Sigma & \vdash & B. \end{array}$$

2. $(\leftrightarrow +)$: IFF-introduction

$$\begin{array}{lrcl} \text{If} & \Sigma, A & \vdash & B, \\ & \Sigma, B & \vdash & A, \\ \text{then} & \Sigma & \vdash & (A \leftrightarrow B). \end{array}$$

Example 1.43

Exercise: Prove the following: $A \to B, A \vdash B$

Proof.

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

□

Example 1.44

Exercise: Prove the following: $\neg\neg A \vdash A$.

Proof. We have

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

□

Example 1.45

Exercise: Prove the following: $A \vdash \neg\neg A$.

Proof. We have

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

□

Discovery 1.7: This is a proof of NOT-introduction $(\neg+)$

Proof. We have

(1)	Σ, A	$\vdash$	B	by supposition
(2)	Σ, A	$\vdash$	$\neg B$	by supposition
(3)	$\neg\neg A$	$\vdash$	A	see above
(4)	$\Sigma, \neg\neg A$	$\vdash$	A	$(+), (3)$
(5)	$\Sigma, \neg\neg A$	$\vdash$	Σ	by $(\in)$
(6)	$\Sigma, \neg\neg A$	$\vdash$	B	by $(Tr), (4), (5), (1)$
(7)	$\Sigma, \neg\neg A$	$\vdash$	$\neg B$	by $(Tr), (4), (5), (2)$
(8)	Σ	$\vdash$	$\neg A$	by $(\neg-), (7)$

□

1.19 Lecture 8

Lecture 8 - Tuesday, Jun 4

Algorithm 1.11: Inconsistency Rule

We have

$$A, \neg A \vdash B$$

Proof. We have

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

□

Exercise: Prove that $A \to B \vdash \neg B \to \neg A$.

Proof. We have

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

□

Exercise: Prove that $\neg A \to A \vdash A$.

Proof. We have

$$\begin{array}{llcll}
(1) & \neg A \to A, \neg A & \vdash & \neg A \to A & (\in) \\
(2) & \neg A \to A, \neg A & \vdash & \neg A & (\in) \\
(3) & \neg A \to A, \neg A & \vdash & A & (\to -), (1), (2) \\
(4) & \neg A \to A & \vdash & A & (\neg -), (2), (3)
\end{array}$$

□

Exercise: Prove that $\neg(A \to B) \vdash \neg B$.

Proof. We have

$$\begin{array}{llcll}
(1) & \neg(A \to B), B, A & \vdash & B & (\in) \\
(2) & \neg(A \to B), B & \vdash & A \to B & (\to +), (1) \\
(3) & \neg(A \to B), B & \vdash & \neg(A \to B) & (\in) \\
(4) & \neg(A \to B) & \vdash & \neg B & (\neg +), (2), (3)
\end{array}$$

□

1.19.1 AND Practice

Exercise: Show that $(A \wedge B) \vdash A, B$

Proof. It suffices to show that $(A \wedge B) \vdash A$ and $(A \wedge B) \vdash B$. Notice that we have

$$\begin{array}{llcll}
(1) & (A \wedge B) & \vdash & A \wedge B & (\in) \\
(2) & (A \wedge B) & \vdash & A & (\wedge -), (1) \\
(3) & (A \wedge B) & \vdash & B & (\wedge -), (1)
\end{array}$$

□

Exercise: Show that $A, B \vdash (A \wedge B)$

Proof. Proof for this is similar as above. □

Exercise: Show that $\neg(A \wedge B) \vdash A \to \neg B$

Proof. Notice that we have

$$\begin{array}{llcll}
(1) & \neg(A \wedge B), A, B & \vdash & A & (\in) \\
(2) & \neg(A \wedge B), A, B & \vdash & B & (\in) \\
(3) & \neg(A \wedge B), A, B & \vdash & A \wedge B & (\wedge +), (1), (2) \\
(4) & \neg(A \wedge B), A, B & \vdash & \neg(A \wedge B) & (\in) \\
(5) & \neg(A \wedge B), A & \vdash & \neg B & (\neg +), (3), (4) \\
(6) & \neg(A \wedge B) & \vdash & A \to \neg B & (\to +), (5)
\end{array}$$

□

Exercise: Show that $A \to \neg B \vdash \neg(A \wedge B)$

Proof. Notice that we have

$$\begin{array}{lccc}
(1) & A \to \neg B,\ A \wedge B & \vdash & A \wedge B & (\in) \\
(2) & A \to \neg B,\ A \wedge B & \vdash & A & (\wedge -), (1) \\
(3) & A \to \neg B,\ A \wedge B & \vdash & B & (\wedge -), (1) \\
(4) & A \to \neg B,\ A \wedge B & \vdash & A \to \neg B & (\in) \\
(5) & A \to \neg B,\ A \wedge B & \vdash & \neg B & (\to -), (2), (4) \\
(6) & A \to \neg B & \vdash & \neg(A \wedge B) & (\neg +), (3), (5)
\end{array}$$

□

Exercise: Show that $(A \wedge B), (C \wedge D) \vdash (A \wedge D)$

Proof. □

1.20 Lecture 9

Lecture 9 - Thursday, Jun 6

1.20.1 OR Practice

Exercise: Show that $(A \to B) \vdash ((\neg A) \vee B)$.

Proof. We have

$$\begin{array}{lccc}
(1) & (A \to B), \neg((\neg A) \vee B),\ A & \vdash & A \to B & (\in) \\
(2) & (A \to B), \neg((\neg A) \vee B),\ A & \vdash & A & (\in) \\
(3) & (A \to B), \neg((\neg A) \vee B),\ A & \vdash & B & (\to -), (1), (2) \\
(4) & (A \to B), \neg((\neg A) \vee B),\ A & \vdash & ((\neg A) \vee B) & (\vee +), (3) \\
(5) & (A \to B), \neg((\neg A) \vee B),\ A & \vdash & \neg((\neg A) \vee B) & (\in) \\
(6) & (A \to B), \neg((\neg A) \vee B),\ \neg A & \vdash & \neg A & (\in) \\
(7) & (A \to B), \neg((\neg A) \vee B),\ \neg A & \vdash & ((\neg A) \vee B) & (\vee +), (6) \\
(8) & (A \to B), \neg((\neg A) \vee B),\ \neg A & \vdash & \neg((\neg A) \vee B) & (\in) \\
(9) & (A \to B), \neg((\neg A) \vee B) & \vdash & A & (\neg -), (7), (8) \\
(10) & (A \to B), \neg((\neg A) \vee B) & \vdash & \neg A & (\neg +), (4), (5) \\
(11) & (A \to B) & \vdash & ((\neg A) \vee B) & (\neg -), (9), (10)
\end{array}$$

□

Exercise: Show that $(\neg(A \wedge B)) \vdash ((\neg A) \vee (\neg B))$.

Proof. We have

$$\begin{array}{lccc}
(1) & \neg(A \wedge B) & \vdash & A \to \neg B & \text{Thm 2.6.8[5]} \\
(2) & A \to \neg B & \vdash & ((\neg A) \vee (\neg B)) & \text{Thm 2.6.1[5]} \\
(3) & \neg(A \wedge B) & \vdash & ((\neg A) \vee (\neg B)) & (Tr), (1), (2)
\end{array}$$

□

Exercise: Show that $((\neg A) \vee (\neg B)) \vdash (\neg(A \wedge B))$.

Proof. We have

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

□

Exercise: Show that $((\neg A) \vee B) \vdash (A \to B)$.

Proof. We have

(1)	$\neg A,\ A$	$\vdash$	B	*Inconsistency*
(2)	$B,\ A$	$\vdash$	B	$(\in)$
(3)	$\neg A \vee B,\ A$	$\vdash$	B	$(\vee -), (1), (2)$
(4)	$\neg A \vee B$	$\vdash$	$A \to B$	$(\to +), (3)$

□

Exercise: Show that $(A \vee B) \to C \vdash (A \to C) \wedge (B \to C)$.

1.20.2 IFF Practice

Exercise: Show that $\neg(A \leftrightarrow B) \vdash A \leftrightarrow \neg B$.
Exercise: Show that $\neg(A \leftrightarrow B) \vdash \neg A \leftrightarrow B$.
Exercise: Show that $A \leftrightarrow B, B \leftrightarrow C \vdash A \leftrightarrow C$.

1.20.3 Soundness and Completeness

Definition 1.31: Structural Induction on Formal Proofs

We can perform **structural induction on formal deduction proofs**. Suppose we have a formal deduction proof of $\Sigma \vdash C$, and we want to prove something about it. Then we have

1. *Base case*: The proof contains exactly one line of the form $C \vdash C$, with justification (Ref).
2. *Induction Step*: We consider the last line of the proof (our cases are the possible justification-s/rules), assume the induction hypothesis for the previous lines, and then show what we want to for the last line.

Theorem 1.7: Soundness Theorem

If $\Sigma \vdash C$, then $\Sigma \models C$.

Proof. We prove this using structural induction on formal proofs:

1. *Base Case:*
   We simply have $\Sigma \vdash C$ for $\Sigma = C$ because $C \vdash C$ (Ref).

2. *Induction Steps:*

   (a) $(+)$: $\Sigma \vdash C$ for $\Sigma = \Sigma_1 \cup \Sigma_2$. From induction hypothesis, we have $\Sigma_1 \models C$. Let t be truth valuation such that $(\Sigma_1 \cup \Sigma_2)^t = 1$, then Σ_1^t, which implies that $C^t = 1$, so we indeed have $\Sigma = \Sigma_1 \cup \Sigma_2 \models C$.

   (b) $(\neg -)$: By induction hypothesis, we have $\Sigma, \neg C \models A$ and $\Sigma, \neg C \models \neg A$. SFAC that there exists truth valuation t such that $\Sigma^t = 1$ and $C^t = 0$, thus $(\neg C)^t = 1$, which further implies that $A^t = 1$ and $(\neg A)^t = 1$, contradiction.

   (c) $(\rightarrow -)$: By induction hypothesis, we know that $\Sigma \models A \rightarrow C$ and $\Sigma \nvdash A$. Let t be any truth valuation such that $\Sigma^t = 1$. By tautological consequence and induction hypothesis, we know that $(A \rightarrow C)^t = A^t = 1$, then by $\rightarrow$-rule, we have $C^t = 1$, hence $\Sigma \models C$.

   (d) $(\rightarrow +)$: C is $(A \rightarrow B)$ for some $A, B \in \text{Form}(\mathcal{L}^p)$. By the induction hypothesis, we know that $\Sigma \cup \{A\} \models B$. Let t be any truth valuation such that $\Sigma^t = 1$. then we have two cases for A:

      i. $A^t = 1$, then $(\Sigma \cup \{A\})^t = 1$, so then $B^t = 1$, and then by $\rightarrow$-rule, $C^t = (A \rightarrow B)^t = 1$.

      ii. $A^t = 0$, then by $\rightarrow$-rule we have $C^t = (A \rightarrow B)^t = 1$.

   (e) $(\wedge -)$: C is A. By induction hypothesis, we have $\Sigma \models A \wedge B$. Let t be truth valuatino such that $\Sigma^t = 1$, then by induction hypothesis, $(A \wedge B)^t = 1$, so by $\wedge$-rule, $A^t = 1$, so $\Sigma \models C = A$.

   (f) $(\wedge +)$: IH tells us $\Sigma \models A$ and $\Sigma \models B$. Let t be any truth valuation such that $\Sigma^t = 1$. By definition of tautological consequence, $A^t = B^t = 1$, thus by $\wedge$-rule, $(A \wedge B)^t = 1$.

   (g) $(\vee -)$: IH_A tells us $\Sigma \cup \{A\} \models C$ and IH_B tells us $\Sigma \cup \{B\} \models C$. Let t be any truth valuation such that $(\Sigma \cup (A \vee B))^t = 1$, then $(A \vee B)^t = 1$, so by $\vee$-rule, $A^t = 1$ or $B^t = 1$ or both.

      i. If $A^t = 1$, then $(\Sigma \cup \{A\})^t = 1$, then by the definition of tautological consequence, $C^t = 1$.

      ii. If $B^t = 1$, then $(\Sigma \cup \{B\})^t = 1$, then by the definition of tautological consequence, $C^t = 1$.

   (h) $(\vee +)$: IH tells us $\Sigma \models A$. Let t be truth valuation such that $\Sigma^t = 1$, then by IH, $A^t = 1$. By $\vee$-rule, $(A \vee B)^t = (B \vee A)^t = 1$.

   (i) $(\leftrightarrow -)$: Consider two cases

      i. IH tells us that $\Sigma \models (A \leftrightarrow B)$ and $\Sigma \models A$. Let t be truth valuation such that $\Sigma^t = 1$, by definition of tautological consequence, $(A \leftrightarrow B)^t = A^t = 1$, then by $\leftrightarrow$-rule, we have $B^t = 1$.

      ii. IH tells us that $\Sigma \models (A \leftrightarrow B)$ and $\Sigma \models B$. Let t be truth valuation such that $\Sigma^t = 1$, by definition of tautological consequence, $(A \leftrightarrow B)^t = B^t = 1$, then by $\leftrightarrow$-rule, we have $A^t = 1$.

   (j) IH tells us that $\Sigma \cup \{A\} \models B$ and $\Sigma \cup \{B\} \models A$. Let t be truth valuation such that $\Sigma^t = 1$. Here we have several subcases:

      i. $A^t = B^t = 0$, then $(A \leftrightarrow B)^t = 1$.

      ii. $A^t = 0$, $B^t = 1$. Thus $(\Sigma \cup \{B\})^t = 1$, so by definition of tautological consequence, $A^t = 1$, which is a contradiction. This case cannot occur.

      iii. $A^t = 1$, $B^t = 0$. Thus $(\Sigma \cup \{A\})^t = 1$, so by definition of tautological consequence, $B^t = 1$, which is a contradiction. This case cannot occur.

      iv. $A^t = B^t = 1$, then $(A \leftrightarrow B)^t = 1$.

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

Thus by the principle of structural induction, we are done. □

1.21 Tutorial 5

See PDF

1.22 Lecture 10

Lecture 10 - Tuesday, Jun 11

Theorem 1.8: Finite Premises Theorem

If $\Sigma \vdash A$, then there exists a finite subset $\Sigma_0 \subseteq \Sigma$ such that $\Sigma_0 \vdash A$.

Proof. Prove this theorem via structural induction on $\Sigma \vdash A$. □

Example 1.46: Application of Soundness

Prove that $\{(A \to B)\} \nvdash (B \to A)$ using the soundness theorem.

Proof. We use the contrapositive of the soundness theorem (1.7):

$$\Sigma \nvDash C, \text{ then } \Sigma \nvdash C$$

Suppose we have a truth valuation such that

$$A^t = 0, \quad B^t = 1$$

thus we have $(A \to B) \nvDash (B \to A)$, hence $(A \to B) \nvdash (B \to A)$. □

Definition 1.32: Consistent 1

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

Definition 1.33: Consistent 2

Let Σ be a set of propositional formulae in Form($\mathcal{L}^p$). We call Σ consistent if, for every formula A such that $\Sigma \vdash A$, then $\Sigma \nvdash A$.

Theorem 1.9

The above two definitions are equivalent.

Proof. 1. ($\Longrightarrow$)

Assume there exists formula B such that $\Sigma \nvdash B$. Let A be any formula such that $\Sigma \vdash A$. SFAC that $\Sigma \vdash (\neg A)$. Then we have

(1)	Σ	$\vdash$	$A, \neg A$	*supposition*
(2)	$A, \neg A$	$\vdash$	B	*inconsistency*
(3)	Σ	$\vdash$	B	$(Tr), (1), (2)$

which is a contradiction.

2. ($\Longleftarrow$)

Assume that for all A such that $\Sigma \vdash A$, we have $\Sigma \nvdash (\neg A)$. SFAC that $\Sigma \vdash B$ for every formula B, then $\Sigma \vdash A$ and $\Sigma \vdash (\neg A)$, contradiction.

□

Lemma 1.5

Let Σ be a set of propositional formulae in 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. 1. ($\Longrightarrow$)

Assume $\Sigma \vDash A$, we have two cases for any truth valuation t:

(a) $\Sigma^t = 1$, then we have $A^t = 1$, thus $(\neg A)^t = 0$, so $(\Sigma \cup \{(\neg A)\})^t = 0$.

(b) $\Sigma^t = 0$, then $(\Sigma \cup \{(\neg A)\})^t = 0$.

2. ($\Longleftarrow$)

Assume that $\Sigma \cup \{(\neg A)\}$ is unsatisfiable, let t be truth valuation such that $\Sigma^t = 1$, then

(a) $A^t = 0$, then $(\neg A)^t = 1$. Contradiction.

(b) $A^t = 1$, then $(\neg A)^t = 0$. Done

□

Lemma 1.6

Let Σ be a set of propositional formulae in Form($\mathcal{L}^p$). Let A be a propositional formula, then $\Sigma \vdash A$ if and only if $\Sigma \cup \{(\neg A)\}$ is inconsistent.

Proof. 1. ($\Longrightarrow$)

Assume $\Sigma \vdash A$, then

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

This violates our definition for consistency.

2. ($\Longleftarrow$)
Assume that $\Sigma \cup \{(\neg A)\}$ is inconsistent, let B be any propositional formula, then

$$\begin{array}{lccll} (1) & \Sigma, \neg A & \vdash & B & \textit{inconsistency} \\ (2) & \Sigma, \neg A & \vdash & \neg B & \textit{inconsistency} \\ (3) & \Sigma & \vdash & A & (\neg -), (1), (2) \end{array}$$

□

Theorem 1.10: Completeness Theorem

If $\Sigma \models C$, then $\Sigma \vdash C$.

Theorem 1.11: Soundness of Propositional Formal Deduction

If Σ is satisfiable, then Σ is consistent.

Proof. 1. ($\Longrightarrow$)

$$\begin{aligned} \Sigma \text{ is satisfiable} &\Rightarrow \Sigma \text{ is consistent} \\ \equiv \Sigma \cup \{(\neg C)\} \text{ is inconsistent} &\Rightarrow \Sigma \cup \{(\neg C)\} \text{ is unsatisfiable} \\ \equiv \Sigma \vdash C &\Rightarrow \Sigma \models C \end{aligned}$$

2. ($\Longleftarrow$)

$$\begin{aligned} \Sigma \vdash C &\Rightarrow \Sigma \models C \\ \equiv \Sigma \cup \{(\neg C)\} \text{ is inconsistent} &\Rightarrow \Sigma \cup \{(\neg C)\} \text{ is unsatisfiable} \\ \equiv \Sigma \text{ is satisfiable} &\Rightarrow \Sigma \text{ is consistent} \end{aligned}$$

□

Theorem 1.12: Completeness of Propositional Formal Deduction

If Σ is consistent, then Σ is satisfiable.

Proof. Exercise. □

1.23 Unmarked Quiz 9 - Soundness and Completeness

If A and B are a formulas, then $\varnothing \vdash B \to (A \to A)$ is *always* a theorem.

Proof. $A \to A$ is a tautology. □

If A is a formula, then $A \to \neg A \vdash A$ is *sometimes* a theorem.

Proof. A could be a tautology, then the above implication is vacuously true. □

If A is a formula, then $\varnothing \vdash (A \to A) \to A$ is *sometimes* a theorem.

1.23.1 Resolution Proof System

Resolution follows the refutation principle; that is, it shows that ths negation of the conclusion is inconsistent with the premises. To prove that the argument $A_1, A_2, \ldots, A_n \vDash C$ is valid, one shows that the Set

$$\{A_1, A_2, \ldots, A_n, \neg C\}$$

is not satisfiable. In order to do this, we show that it is inconsistent.

Definition 1.34: Resolution

Resolution is the formal deduction rule

$$C \vee p, D \vee \neg p \vdash_r C \vee D$$

where C, D are (possibly empty) disjunctive clauses, and p is a proposition symbol.

Algorithm 1.12: Proving Argument Validity with Resolution

1. Convert everything in $\Sigma \cup \{\neg C\}$ to CNF;
2. Chop up all of the conjunctions in $\Sigma \cup \{\neg C\}$ into separate disjunctive clauses, e.g. $(p \vee q \vee r) \wedge (\neg p \vee q \vee \neg r)$ becomes $(p \vee q \vee r), (\neg p \vee q \vee \neg r)$, and put all of our separated disjunctive clauses into set S;
3. (a) Pick two clauses from S that contain complimentary literals and perform resolution on them to get resolvent $C \vee D$.
   (b) Add the new resolvent $C \vee D$ back into S.
4. Repeat step 3 until we output $\perp$ or can't perform it further (i.e. we don't have complimentary literals remaining).

Theorem 1.13: Soundness of Resolution

The resolvent is tautologically implied by its parent clauses, which makes resolution a sound rule of formal deduction.

Proof. Let C, D be disjunctive clauses, let p be propositional symbol. By resolution, we have

$$C \vee p, D \vee \neg p \vdash_r C \vee D$$

we have to show that

$$C \vee p, D \vee \neg p \vDash_r C \vee D$$

Consider cases for C and D:

1. *Case 1*: If at least one of C and D is non-empty, and t is the truth valuation such that $(C \vee p)^t = (D \vee \neg p)^t = 1$, so we have
   (a) $p^t = 0$, then $C^t = 1$ by $\vee$-rule, then we have $(C \vee D)^t = 1$ by $\vee$-rule again.

(b) $p^t = 1$, then $(\neg p)^t = 0$, so $D^t = 1$ by $\vee$-rule, then we have $(C \vee D)^t = 1$ by $\vee$-rule again.

2. *Case 2*: If C and D are empty, then resolvent of p and $\neg p$ is $\bot$. In this case we have $p, \neg p \models \bot$. Therefore, we have tautological consequence.

□

Algorithm 1.13: Set-of-support Strategy

The set of support strategy has an additional condition for its resolution proof rule: every resolution step must use at least one formula from the set of support.

Theorem 1.14: Completeness of Resolution

Resolution with the set-of-support strategy is complete.

Algorithm 1.14: Davis-Putnam Procedure

Any disjunctive clause uniquely corresponds to a set of literals:

Example 1.47

$p \vee \neg q \vee r$ corresponds to $\{p, \neg q, r\}$.
$\neg s \vee q$ corresponds to $\{\neg s, q\}$.

If clauses are represented as sets, one can write the resolvent, on p, of two clauses $C \cup \{p\}$ and $D \cup \{\neg p\}$, when neither C nor D is empty, as

$$[(C \cup \{p\}) \cup (D \cup \{\neg p\})]\ \{p, \neg p\}$$

When C and D are both empty, we define the resolvent as the empty clause $\bot$.

Result 1.5

* If the output of DPP is the empty clause, $\{\bot\}$, this indicates that both p and $\neg p$ were produced. This implies that the set of clauses obtained by pre-processing the premises and negation of the conclusion of the original argument is inconsistent, hence not satisfiable. The latter implies that the argument is valid.
* If the output of DPP is an empty set of clauses, $\varnothing$, this means that no contradiction can be derived from the initial set of clauses, which means that the set of clauses is consistent, hence satisfiable, implying that the original argument is not valid.

1.24 Marked Quiz 3

Example 1.48

1. If A and B are a formulas, then $\varnothing \vdash B \to (A \to A)$ is always a theorem.
2. Let Σ be a set of propositional formulas. Let A be a propositional formula. If $\Sigma \cup \{ |A\}$ is unsatisfiable, then $\Sigma \vDash A$ always holds.
3. If A is a tautology, then $\varnothing \vdash A$ is always a theorem.
4. Let Σ be a set of propositional formulas. Let A be a propositional formula. If Σ is consistent, then $\Sigma \vdash A$ sometimes holds.
5. Let Σ be a set of propositional formulas. Let A be a propositional formula. If Σ is inconsistent, then $\Sigma \vdash A$ always holds.

2 Midterm

2.1 DNF and CNF

Result: As an example, $p \vee q \vee r$ qualifies as both CNF and DNF.

2.2 Formal Deduction (Problems I found the Hardest)

Exercise: Show that $p \wedge (q \vee r) \vdash ((p \wedge q) \vee (p \wedge r))$.
Exercise: Show that $A \vee B \vdash (A \wedge C) \vee (C \to B)$.

Proof. Here I only provide a sketch of the proof. We have

$$B, C \vdash B \quad \Rightarrow \quad B \vdash C \to B \quad \Rightarrow \quad B \vdash (A \wedge C) \vee (C \to B)$$

Thus we want to show that $A \vdash (A \wedge C) \vee (C \to B)$. Notice

$$\begin{aligned} A, C \vdash (A \wedge C) \quad &\Rightarrow \quad A, C \vdash (A \wedge C) \vee (C \to B) \\ A, \neg C \vdash (C \to B) \quad &\Rightarrow \quad A, \neg C \vdash (A \wedge C) \vee (C \to B) \end{aligned}$$

Therefore we can conclude that $A \vdash (A \wedge C) \vee (C \to B)$ (Why?). Therefore we can further conclude that $(A \vee B) \vdash (A \wedge C) \vee (C \to B)$. □

3 First-order Logic

Translate the following sentence from English to First-order Logic:

"For all integers, adding 0 returns the same integer." Equivalently, "For every integer x, x + 0 = x".

The answer would be

$$\forall x(I(x) \to (x + 0 = x))$$

where $I(x) := x$ is an integer.

Definition 3.1: Domain

Semantically, a domain (or universe) is a non-empty set of elements that our first-order formulas make assertions about. Some common domains include $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, any finite set, *any non-empty set*.

Example 3.1

Let the domain (universe) be C, and 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 an even number.
5. $Odd(u)$: u is an odd number

Translate the following sentences into formulas in first-order logic:

1. Not all real numbers are rational numbers.

$$\exists x(R(x) \land \neg Q(x)) \quad \text{or} \quad (\neg\forall x(R(x) \to Q(x)))$$

2. Some real numbers are not rational numbers.

$$\exists x(R(x) \land \neg Q(x))$$

3. Every natural number is either odd or even.

$$\forall x(N(x) \to Even(x) \lor Odd(x))$$

4. No natural number is both odd and even.

$$\forall x(N(x) \to \neg(Even(x) \land Odd(x))) \quad \text{or} \quad \neg\exists x(N(x) \land Even(x) \land Odd(x))$$

Definition 3.2: Relations

The building blocks of first order logic are relations (a.k.a. predicates) on some set X.

Definition 3.3: k-ary Relation

Let X be a non-empty set, and let $k \geqslant 1$. A k-ary relation, or k-ary predicate on X is any set of k-tuples of elements of X. The arity of the relation is k.

Definition 3.4: Bound Variables and Free Variables

A bound variable is a variable that comes with a quantifier. A free variable u is a variable that is not bound.

Definition 3.5: Symbols in First-order Logic

1. Individual symbols. Usually $a, b, a_1, b_2, \ldots, c_1, c_2, \ldots$. Sometimes $0, 1, \pi$, or anything appropriate. These are constants with names.
2. Variable symbols. Usually $x, y, z, \ldots, x_1, x_2, \ldots, y_1, y_2, \ldots$
   (a) We use x, y, z for bound variables.
   (b) We use u, v, w for free variables.
3. Function symbols. Usually $f, g, h, \ldots, f_1, f_2, \ldots$ Sometimes $sum(u, v)$, or anything descriptive. (comes with an arity, k)
4. Relation symbols. $F, G, \ldots, F_1, F_2, \ldots, G_1, G_2, \ldots$. Sometimes $Equal(u, v)$, or anything descriptive. (comes with an arity, k)
5. Logical Connectives. $\neg, \wedge, \vee, \rightarrow$, and $\leftrightarrow$
6. Quantifiers. $\forall$ and $\exists$
7. Punctuation. '(', ')', and ','.

3.1 Lecture 13

Lecture 13 - Thursday, Jun 20

Example 3.2

$\forall x(I(x) \rightarrow (x + 0 = x))$. Identify the different parts of the translation.

Proof. Here, we have $I(x)$ is a relation, $x + 0$ is a function, and $x + 0 = x$ is a relation. □

Definition 3.6: Term

A term is a placeholder for a domain element. An atomic term is either an individual symbol or a variable symbol. Examples of atomic terms: u, v, 0, 1. A non-atomic term is an expression built using some combination of individual symbols, variable symbols, and function symbols.

Algorithm 3.1: Structural Induction on Terms

1. *Base Case:*
   (a) Individual symbols;
   (b) Variables
2. *Induction step*: We exhaust all the possible combinations of non-atomic terms from the set.

Definition 3.7: Atomic Formula

An **atomic formula** is an expression in First-order Logic that is written using a combination of terms and relations.

Example 3.3: Some examples of atomic formulas include

$$S(u) \quad F(u,v) \quad (u > 0) \quad F(f(u),0)$$

Definition 3.8: Syntactically Correct Formula in First-order Logic

Let $Atom(\mathcal{L})$ be our set of atomic formulas. We define the set $\text{Form}(\mathcal{L}^p)$ of syntactically correct formulas in first-order logic, or just formulas, inductively as follows:

1. An atomic formula is a formula, i.e. $Atom(\mathcal{L}) \subseteq \text{Form}(\mathcal{L}^p)$
2. If A is a formula in $\text{Form}(\mathcal{L}^p)$, then $(\neg A)$ is a formula in $\text{Form}(\mathcal{L}^p)$.
3. If A, B are formulas in $\text{Form}(\mathcal{L}^p)$, and $*$ is some binary logical connective, then $(A * B)$ is a formula in $\text{Form}(\mathcal{L}^p)$.
4. If $A(u)$ is in $\text{Form}(\mathcal{L}^p)$ such that x does not occur in $A(u)$, then then we have the following:

$$\forall x A(x) \in \text{Form}(\mathcal{L}^p) \qquad \exists x A(x) \in \text{Form}(\mathcal{L}^p)$$

Definition 3.9: Sentence

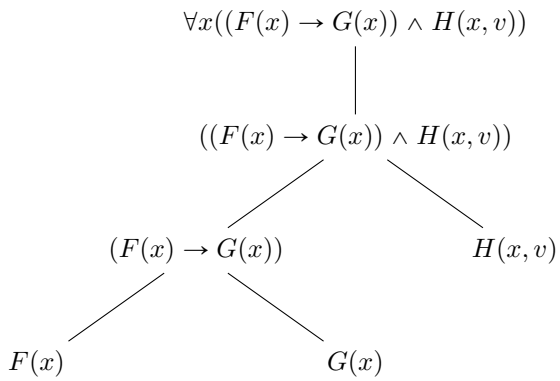
A formula with no free variables is called a **closed formula** or a **sentence**.

Definition 3.10: Parse Tree in First-order Logic

Just like what we had for propositional logic, we have the following example to illustrate a parse tree for first-order logic:

Example 3.4

Draw the parse trees for the following formula in First-order Logic: $\forall x((F(x) \to G(x)) \wedge H(x, v))$.



Definition 3.11: Simple formula

A formula is either

1. **simple** if it is of the form $F(t_1, \ldots, t_k)$ for some k-ary relation F and terms $t_1, \ldots, t_k$, or
2. constructed from simple formulas using connectives and/or quantifiers.

Example 3.5: Example of non-simple formulae

1. $\forall x(I(x) \to (x + 0 \approx x))$;
2. $\forall x f(f(x), 0)$;
3. $(S(u) \wedge E(u, v))$.

3.1.1 Semantics of First-Order Logic

Definition 3.12: Interpretation

An **interpretation** of a First-Order formula selects a domain D, then maps

1. each individual symbol to a domain element,
2. each k-ary function symbol to a total function $D^k \to D$, and
3. each k-ary relation symbol to a k-ary relation on$D$

Definition 3.13: Total Function

Consider function f^v:

$$f^v : D \times \cdots \times D \to D$$

where the inputs to the function come from D^k, the k-fold product of copies of D. To say that f^v is a total function of arity k defined on D means

1. every tuple $(d_1, \ldots, d_k) \in D^k$ is a legal input to f^v, and
2. $f^v(d_1, \ldots, d_k) \in D$ for every tuple $(d_1, \ldots, d_k) \in D^k$.

Definition 3.14: Assignment

An **assignment** uses a selected domain D from an interpretation, then maps each free variable symbol to a domain element.

Definition 3.15: Valuation

A **valuation** is an interpretation plus an assignment.

Proposition 3.1

Let t be any term of First-Order Logic. Let v be any valuation, with domain D. Then $t^v \in D$.

Lemma 3.1

Let t be any term of First-Order Logic. Let v_1, v_2 be any valuations, over the same domain D, such that

1. $a^{v_1} = a^{v_2}$ for every individual symbol a
2. $u^{v_1} = u^{v_2}$ for every individual symbol u
3. $f^{v_1} = f^{v_2}$ for every function symbol f in t, and
4. $F^{v_1} = F^{v_2}$ for every relation symbol F in A.

Then

$$t^{v_1} = t^{v_2}$$

Definition 3.16: Satisfiability

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.

Note: We don't use the word "tautology" in first-order logic!

Example 3.6

Let A be the First-Order Formula $\exists x(F(x) \to G(x))$. Let $D = \{a, b\}$.

1. Create a valuation v_1 over D such that $A^{v_1} = 1$.
2. Create a valuation v_2 over D such that $A^{v_2} = 0$.

Proof. We can have

1. v_1 could be $D = \{a, b\}$, $F = \{a\}$, and $G = \{a, b\}$, then we have
$$(F(u) \to G(u))^{v_1(u/a)} = 1$$
2. v_2 could be $D = \{a, b\}$, $F = \{a, b\}$, and $G = \varnothing$, then we have
$$(F(u) \to G(u))^{v_1(u/a)} = 0$$

□

Example 3.7

Let B be the First-Order Formula $\forall x(F(x) \to G(x))$. Let $D = \{a, b\}$.

1. Create a valuation v_1 over D such that $B^{v_1} = 1$.
2. Create a valuation v_2 over D such that $B^{v_2} = 0$.

Proof. We can have

1. v_1 could be $D = \{a, b\}$, $F = \{a, b\}$, and $G = \{a, b\}$, then we have
$$(F(u) \to G(u))^{v_1(u/a)} = (F(u) \to G(u))^{v_1(u/b)} = 1$$
2. v_2 could be $D = \{a, b\}$, $F = \{a\}$, and $G = \{b\}$, then we have
$$(F(u) \to G(u))^{v_1(u/a)} = 0$$

□

3.2 Lecture 14

Lecture 14 - Tuesday, Jun 25

3.2.1 Precedence Rule in First-Order Logic

Theorem 3.1: Precedence Rule in First Order Logic

The precedence rules we had in propositional logic stay the same (recall Precedence Rule in Propositional Logic (5)). A quantifier binds to the first syntactically correct formula to its immediate right.

Example 3.8

Show whether the following is valid, unsatisfiable, or satisfiable but not valid.

$$\forall x F(x) \land \forall x G(x) \to \forall x (F(x) \land G(x))$$

Proof. Let v be valuation with domain $\mathcal{D}$, here we have two cases:

1. *Case 1*: $(\forall x F(x) \land \forall x G(x))^v = 0$
   We simple have the whole formula $A^v = 1$ by $\to$-rule;

2. *Case 2*: $(\forall x F(x) \land \forall x G(x))^v = 1$
   By $\land$-rule, we know that $(\forall x F(x))^v = 1$ and $(\forall x G(x))^v = 1$. Take any $\alpha \in \mathcal{D}$, by $\forall$-rule, we have both $F(u)^{v(u/\alpha)} = 1$ and $G(u)^{v(u/\alpha)} = 1$, thus $(F(x) \land G(x))^{v(u/\alpha)} = 1$ by $\land$-rule. Therefore,

   $$\forall x (F(x) \land G(x))^v = 1$$

   Now by $\to$-rule, we indeed have the formula $A^v = 1$.

Therefore the original formula is valid. □

Example 3.9

Show whether the following is valid, unsatisfiable, or satisfiable but not valid.

$$\forall x R(x) \to R(a)$$

Proof. Let v be valuation with domain $\mathcal{D}$, here we have two cases:

1. *Case 1*: $(\forall x R(x))^v = 0$
   We simple have the whole formula $A^v = 1$ by $\to$-rule;

2. *Case 2*: $(\forall x R(x))^v = 1$
   There must be some $\alpha \in D$ since D is non-empty (by Definition (3.1)). By our valuation v, there is something in D that a maps to, denoted as a^v. Then by $\forall$-rule, we have $R(u)^{v(u/a^v)} = 1$, which tells us that $R^v(a^v) = 1$, thus the formula $A^v = 1$.

Therefore the original formula is valid. □

Example 3.10

Show whether the following is valid, unsatisfiable, or satisfiable but not valid.

$$\exists x (F(x) \land \neg G(x)) \land (\exists x F(x) \to \forall x G(x))$$

Proof. Let v be valuation with domain $\mathcal{D}$, here we have two cases:

1. *Case 1*: $\exists x(F(x) \wedge \neg G(x))^v = 0$
   We simple have the whole formula $A^v = 0$ by $\wedge$-rule;

2. *Case 2*: $\exists x(F(x) \wedge \neg G(x))^v = 1$
   By $\exists$-rule, there exists $\alpha \in \mathcal{D}$ such that $(F(u) \wedge \neg G(u))^{v(u/\alpha)} = 1$. By $\wedge$-rule, we know that both $F.^{v(u/\alpha)} = 1$ and $(\neg G(u))^{v(u/\alpha)} = 1$. By $\neg$-rule, we have $G(u)^{v(u/\alpha)} = 0$, which further suggests that $\forall x G(x)^v = 0$. Thus we have $(\exists x F(x) \to \forall G(x))^v = 0$ by $\to$-rule. Therefore, the formula $A^v = 0$.

Therefore the original formula is unsatisfiable. □

Definition 3.17: Logical Consequence

Suppose Σ is a set of First-Order Logic formulas, and let C be a First-Order Logic formula. We say that Σ (logically) implies C if and only if, for every valuation v, we have

$$\Sigma^v = 1 \text{ implies } C^v = 1$$

Example 3.11

Prove $\{\forall x(\neg C)\} \models (\neg \exists x C)$.

Proof. Just yappppp. □

Example 3.12

Prove $(\forall x A(x) \to \forall x B(x)) \not\models \forall x(A(x) \to B(x))$.

Proof. Let v be valuation with domain $\mathcal{D}$ such that $(\forall x A(x) \to \forall x B(x))^v = 1$, then by $\to$-rule, we have

1. *Case 1*: $\forall x A(x)^v = 0$
   It is possible to have $\forall x B(x)^v = 0$. Thus define $\mathcal{D} = \{a, b, c\}$, $A = \{a, b\}$, and $B = \{c\}$. Hence there is some element (a) of $\mathcal{D}$ such that
   $$(A(u) \to B(u))^{v(u/a)} = 0$$
   but we have $(\forall x A(x) \to \forall x B(x))^v = 1$ by $\to$-rule.

2. *Case 1*: $\forall x B(x)^v = 1$

□

Result 3.1

1. The empty set $\varnothing$ is satisfied under any valuation v.

2. If $\varnothing \models C$, then C is a valid formula.

3. If Σ is not satisfiable, then $\Sigma \vDash C$ for any C.

3.3 Lecture 15

Lecture 15 - Thursday, Jun 27

Example 3.13

Prove $\varnothing \vDash (\forall x(A \to B) \to (\forall xA \to \forall xB))$.

Proof. SFAC that there is some valuation v such that $(\forall x(A \to B) \to (\forall xA \to \forall xB))^v = 0$, so by $\to$-rule, we have

$$\forall x(A \to B)^v = 1 \qquad \text{and} \qquad (\forall xA \to \forall xB)^v = 0$$

Furthermore, by $\to$-rule again, we have

$$\forall xA^v = 1 \qquad \text{and} \qquad \forall xB^v = 0$$

but by $\forall$-rule, we know that there exists $\alpha \in D$ such that we have

$$B(u)^{v(u/\alpha)} = 0 \qquad \text{and} \qquad A(u)^{v(u/\alpha)} = 1$$

But this contradicts the fact that $\forall x(A \to B)^v = 1$ because now we have

$$(A(u) \to B(u))^{v(u/\alpha)} = 0$$

□

Example 3.14

Prove $\{\forall x(\neg C)\} \vDash (\neg \exists xC)$.

Proof. Suppose v is a valuation such that $\forall x(\neg C)^v = 1$, then by $\forall$-rule, we have $(\neg C(u))^{v(u/\alpha)} = 1$ for any $\alpha \in D$, thus by $\exists$-rule, we have

$$C(u)^{v(u/\alpha)} = 0 \qquad \text{for any } \alpha \in D$$

By $\exists$-rule, $\exists xC^v = 0$, hence $(\neg \exists xC)^v = 1$. □

Example 3.15

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

Proof. STP some valuation v such that

$$(\forall xA \to \forall xB)^v = 1 \qquad \text{and} \qquad \forall x(A \to B)^v = 0$$

Let A be $F(u)$, we define v as following:

1. $D = \{a, b\}$ and $F^v = \{a\}$

Hence we know that $F(u)^{v(u/b)} = 0$, so by $\forall$-rule, we have $\forall x A^v = 0$, thus by $\rightarrow$-rule, we have $(\forall x A \rightarrow \forall x B)^v = 1$. We then define $B = \neg F(u)$, then we have for $a \in D$,

$$A(u)^{(u/a)} = 1 \qquad \text{and} \qquad B(u)^{(u/a)} = 0$$

Then by $\rightarrow$-rule,

$$(A(u) \rightarrow B(u))^{v(u/a)} = 0 \;\; \Rightarrow \;\; \forall x (A \rightarrow B)^v = 0$$

as desired. □

3.3.1 Formal Deduction in First Order Logic

We can extend our Formal Deduction proof system to create the Formal Deduction proof system for First-order Logic

Algorithm 3.2: Forall Proof Rules

1. ($\forall-$): Forall-Elimination

$$\begin{array}{rccc} \text{If} & \Sigma & \vdash & \forall x(Ax), \\ \text{then} & \Sigma & \vdash & A(t), \text{ for any term } t. \end{array}$$

2. ($\forall+$): Forall-Introduction

$$\begin{array}{rccc} \text{If} & \Sigma & \vdash & A(u), \; u \text{ not occurring in } \Sigma \\ \text{then} & \Sigma & \vdash & \forall x A(x). \end{array}$$

Algorithm 3.3: Exists Proof Rules

1. ($\exists-$): Exists-Elimination

$$\begin{array}{rccc} \text{If} & \Sigma, A(u) & \vdash & B, \; u \text{ not occurring in } \Sigma \\ \text{then} & \Sigma, \exists x A(x) & \vdash & B. \end{array}$$

2. ($\exists+$): Exists-Introduction

$$\begin{array}{rccc} \text{If} & \Sigma & \vdash & A(t), \\ \text{then} & \Sigma & \vdash & \exists x A(x), \text{ where } A(x) \text{ results by replacing some} \\ & & & \text{(not necessarily all) occurrences of } t \text{ in } A(t) \text{ by } x \end{array}$$

Algorithm 3.4: Equal Proof Rules

1. ($\approx -$): Equals-Elimination

$$\begin{array}{rccc} \text{If} & \Sigma & \vdash & A(t_1) \\ & \Sigma & \vdash & t_1 \approx t_2 \\ \text{then} & \Sigma & \vdash & A(t_2), \text{ where } A(t_2) \text{ results by replacing some} \\ & & & \text{(not necessarily all) occurrences of } t_1 \text{ in } A(t_1) \text{ by } t_2. \end{array}$$

2. $(\approx +)$: Equals-Introduction

$$\varnothing \quad \vdash \quad u \approx u$$

Example 3.16: Forall-Practice

Prove $\varnothing \vdash \forall x(F(x) \to F(x))$.

Proof. We have

$$\begin{array}{llll}
(1) & F(u) \vdash & F(u) & (\in) \\
(2) & \varnothing \vdash & F(u) \to F(u) & (\to +), (1) \\
(3) & \varnothing \vdash & \forall x(F(x) \to F(x)) & (\forall +), u \text{ not in } \varnothing, (2)
\end{array}$$

□

Example 3.17: Forall-Practice

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

Proof. We have

$$\begin{array}{llll}
(1) & \forall x(F(x) \to G(x)), \forall x F(x) \vdash & \forall x(F(x) \to G(x)) & (\in) \\
(2) & \forall x(F(x) \to G(x)), \forall x F(x) \vdash & \forall x F(x) & (\in) \\
(3) & \forall x(F(x) \to G(x)), \forall x F(x) \vdash & F(u) \to G(u) & (\forall -), (1) \\
(4) & \forall x(F(x) \to G(x)), \forall x F(x) \vdash & F(u) & (\forall -), (2) \\
(5) & \forall x(F(x) \to G(x)), \forall x F(x) \vdash & G(u) & (\to -), (3), (4) \\
(6) & \forall x(F(x) \to G(x)), \forall x F(x) \vdash & \forall x G(x) & (\forall +), (5)
\end{array}$$

□

Example 3.18: Exists-Practice

Prove $\{\exists x(F(x) \vee G(x))\} \vdash \exists x F(x) \vee \exists x G(x)$.

Proof. We have

$$\begin{array}{llll}
(1) & F(u) \vdash & F(u) & (\in) \\
(2) & F(u) \vdash & \exists x F(x) & (\exists +), (1) \\
(3) & F(u) \vdash & \exists x F(x) \vee \exists x G(x) & (\vee +), (2) \\
(4) & G(u) \vdash & G(u) & (\in) \\
(5) & G(u) \vdash & \exists x G(x) & (\exists +), (4) \\
(6) & G(u) \vdash & \exists x F(x) \vee \exists x G(x) & (\vee +), (5) \\
(7) & F(u) \vee G(u) \vdash & \exists x F(x) \vee \exists x G(x) & (\vee -), (3), (6) \\
(8) & \exists x(F(x) \vee G(x)) \vdash & \exists x F(x) \vee \exists x G(x) & (\exists -), (7)
\end{array}$$

□

Example 3.19: Equals-Practice

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

Proof. We have

$$\begin{array}{lllll}
(1) & (a \approx b), \forall x(F(x) \to G(x)), F(a) & \vdash & \forall x(F(x) \to G(x)) & (\in) \\
(2) & (a \approx b), \forall x(F(x) \to G(x)), F(a) & \vdash & F(a) \to G(a) & (\forall -), (1) \\
(3) & (a \approx b), \forall x(F(x) \to G(x)), F(a) & \vdash & F(a) & (\in) \\
(4) & (a \approx b), \forall x(F(x) \to G(x)), F(a) & \vdash & G(a) & (\to -), (2), (3) \\
(5) & (a \approx b), \forall x(F(x) \to G(x)), F(a) & \vdash & a \approx b & (\in) \\
(6) & (a \approx b), \forall x(F(x) \to G(x)), F(a) & \vdash & G(b) & (\approx -), (4), (5)
\end{array}$$

□

Theorem 3.2: Replacement Theorem

Let $A, B, C \in \text{Form}(\mathcal{L}^p)$ with $B \dashv\vdash C$. Let A' result from A by substituting some (not necessarily all) occurrences of B by C, then $A' \dashv\vdash A$.

Theorem 3.3: Complementation Theorem

Suppose A is a formula composed of

1. atoms of L,
2. the connectives $\neg$, $\vee$, $\wedge$,
3. and the two quantifiers $\forall$, $\exists$ using the appropriate formation rules.

Suppose A' is the formula obtained from A by

1. exchanging $\vee$ and $\wedge$,
2. exchanging $\exists$ and $\forall$,
3. and negating all atoms.

Then $A' \dashv\vdash \neg A$.

Example 3.20: Forall and Exists Practice

Prove $\forall x F(x) \vdash \exists x F(x)$.

Proof. We have

$$\begin{array}{lllll}
(1) & \forall x F(x) \vdash \exists x F(x) & \vdash & \forall x F(x) \vdash \exists x F(x) & (\in) \\
(2) & \forall x F(x) \vdash \exists x F(x) & \vdash & F(t) & (\forall -), (1) \\
(3) & \forall x F(x) \vdash \exists x F(x) & \vdash & \exists x F(x) & (\exists +), (2)
\end{array}$$

□

Example 3.21: Forall and Exists Practice

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

Example 3.22: Forall and Exists Practice

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

3.4 Lecture 16

Lecture 16 - Tuesday, Jul 2

3.4.1 Soundness of First-Order Logic

Theorem 3.4

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

STP the soundness for the six new rules we introduced because we have already proven (1.7). Recall Proposition (3.1) and Lemma (3.1),

1. $(\forall -)$ is sound:
   IH: $\Sigma \vDash \forall x A(x)$. 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 IH, $\forall x A(x)^v = 1$. Let $\mathcal{D}$ be domain of v, by Propositional (3.1), we have that $t^v \in \mathcal{D}$. By $\forall$-rule, we have that $A(u)^{v(u/t^v)} = 1$. Therefore, $\Sigma \vDash A(t)$.

2. $(\forall +)$ is sound:
   IH: $\Sigma \vDash A(u)$ for u not occurring in Σ. Let V be a valuation such that $\Sigma^v = 1$. Let $\mathcal{D}$ be the domain. Let $d \in \mathcal{D}$ be arbitrary. Claim: $\Sigma^v = \Sigma^{v(u/d)}$ by Lemma (3.1). Let $B \in \Sigma$, STP $B^v = B^{v(u/d)}$. If w is free variable in B, we have that $w \neq u$, so $w^v = w^{v(u/d)}$ by definition $v(u/d)$. By Lemma (3.1), we have $B^v = B^{v(u/d)}$, then $\Sigma^v = \Sigma^{v(u/d)}$, so we have $\Sigma^{v(u/d)} = 1$. By our IH, we have $A(u)^{v(u/d)} = 1$. By $\forall$-rule, $\forall x A(x)^v = 1$.

3. $(\exists -)$ is sound:
   IH: $\Sigma, A(u) \vDash B$ for u not occurring in Σ or 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) $B^v = B^{v(u/d)} = 1$.

4. $(\exists +)$ is sound:
   IH: $\Sigma \vDash A(t)$, for some First-Order term t. 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 $A(u)^{v(u/t^v)} = 1$. By Proposition (3.1), $t^v \in \mathcal{D}$. By $\exists$-satisfaction, $\exists x A(x)^v = 1$.

5. $(\approx -)$ is sound:
   IH: $\Sigma \vDash A(t_1)$, and $\Sigma \vDash t_1 \approx t_2$. 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 $A(t_2)^v = 1$.

6. $(\approx +)$ is sound:
   We must prove that $\varnothing \vDash u \approx u$, 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. □

Definition 3.18: Consistency

A set Σ of First-order formulas is consistent if there is no First-order formula A such that $\Sigma \vdash A$ and $\Sigma \vdash \neg A$.

3.4.2 Completeness of First-Order Logic

Theorem 3.5

Let Σ be a set of first-order formulas, let A be a first-order formula. If $\Sigma \vDash A$, it follows that $\Sigma \vdash A$.

Example 3.23

We proved in class that $\{\forall x F(x)\} \vdash \exists x F(x)$, but the other direction is not true. We need to use soundness to show that $\{\exists x F(x)\} \nvdash \forall x F(x)$.

Proof. STP $\{\exists x F(x)\} \nvdash \forall x F(x)$ then the result follows by the contrapositive of Soundness. Let v be the following valuation: $\mathcal{D} = \{0, 1\}$ and $F = \{0\}$. By $\exists$-rule, we know that $\exists x F(x)^v = 1$, but by $\forall$-rule, we have $\forall x F(x)^v = 0$. □

3.4.3 First Order Resolution

Algorithm 3.5: Skolemization

Skolemization is the following process of removing quantifiers (informal explanation):

1. $(\forall)$: Everything is by default implicitly $\forall$-quantified, so if a first-order formula only contains $\forall$, we remove it.

2. $(\exists)$: For existence, we have a couple cases:

   (a) If $\exists x$ is at the front of a formula, we replace occurrences of its bound variable x with a constant/ individual symbol a, we call this a Skolem constant.

   (b) If $\exists y$ follows a $\forall x$, then this means the variable y bound to the $\exists$ is dependent on which x we're talking about. Then we replace y with $f(x)$, where f is a Skolem function.

Example 3.24

As an example, $\exists x F(x)$ gives us $F(a)$. For $\forall x \exists y F(x, y)$, we replace y with $f(x)$, so it gives us $\forall x F(x, f(x))$.

Algorithm 3.6: Unification

Another aspect of FOL Resolution is unification: informally, if we have a formula $\forall x F(x)$, and we've removed the $\forall$ quantifier to get $F(x)$, we can "plug in" a Skolem constant a in place of x, i.e. we can set

$$F(a) \textit{ with } x := a$$

as a line in our resolution proof.

Example 3.25

Prove the following argument by Resolution.

1. Premise 1: $\exists y \forall x F(x, y)$,
2. Conclusion: $\forall x \exists y F(x, y)$.

Proof. Convert the premise(s) and negated conclusion into CNF:

1. Premise 1: $\exists y \forall x F(x, y)$ gives $\forall x F(x, a)$ (Skolem constant for y)
2. Negated Conclusion:

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

Resolution on $\{F(x, a), \neg F(b, y)\}$

(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)	$\bot$	resolvent 3, 4

Lines 3 and 4 are unification steps. □

Example 3.26

Prove the following argument by Resolution.

1. Premise 1: $\forall x (F(x) \to \exists y G(y))$,
2. Premise 2: $\exists x F(x)$,

3. Conclusion: $\exists y G(y)$.

Proof. 1. Premise 1:

$$\begin{aligned} & \forall x(F(x) \to \exists y G(y)) & \\ \models\!\models\ & \forall x \exists y(F(x) \to G(y)) & \text{(move quantifier)} \\ \models\!\models\ & \forall x \exists y(\neg F(x) \vee G(y)) & \text{(remove implication)} \\ \text{gives}\ & \forall x(\neg F(x) \vee G(f(x))) & \text{(Skolem constant for } y\text{)} \end{aligned}$$

2. Premise 2:

$$\exists x F(x) \ \text{ gives } \ F(a) \quad \text{(Skolem constant for } x\text{)}$$

3. Negated Conclusion:

$$\neg \exists y G(y) \models\!\models \forall y \neg G(y) \quad \text{(DeMorgan)}$$

Resolution on $\{(\neg F(x) \vee G(f(x))), F(a), \neg G(y)\}$

(1)	$(\neg F(x) \vee G(f(x)))$	premise
(2)	$F(a)$	premise
(3)	$\neg G(y)$	negated conclusion
(4)	$(\neg F(a) \vee G(f(a)))$	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

Lines 3 and 4 are unification steps. □

3.5 Lecture 17

Lecture 17 - Thursday, Jul 4

Alena sick.

3.5.1 PNF

Definition 3.19: Prenex Normal Form

A formula is in **Prenex Normal Form** if it is of the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n B$$

where $n \geqslant 1$, each Q_i is $\forall$ or $\exists$, and B does not contain any quantifiers.

Algorithm 3.7

1. Eliminate all operators except $\{\neg, \wedge, \vee\}$ using substitution;
2. (a) Push negation inwards using De Morgan's Laws;
   (b) Eliminate double negations;

3. Standardize variables apart;

4. Move all quantifiers to the front of the formula;

   (a) $\forall x \forall y A(x,y) \models\!\models \forall y \forall x A(x,y)$.

   (b) $\exists x \exists y A(x,y) \models\!\models \exists y \exists x A(x,y)$.

   (c) $Q_1 x A(x) \wedge Q_2 y B(y) \models\!\models Q_1 x Q_2 y (A(x) \wedge B(y))$, ($x$ not occurring in $B(y)$, and y not occurring in $A(x)$), ($\wedge$ can be replaced with $\vee$ and this still holds).

Example 3.27: PNF Example

As an example:

$$
\begin{aligned}
& ((\exists x F(x) \wedge \forall y(\neg(G(y) \rightarrow F(y)))) \wedge \exists x(\neg G(x))) \\
\models\!\models\ & ((\exists x F(x) \wedge \forall y(\neg(\neg G(y) \vee F(y)))) \wedge \exists x(\neg G(x))), (\rightarrow -elim), \#1 \\
\models\!\models\ & ((\exists x F(x) \wedge \forall y(\neg\neg G(y) \wedge \neg F(y)))) \wedge \exists x(\neg G(x)), (DML), \#2 \\
\models\!\models\ & ((\exists x F(x) \wedge \forall y(G(y) \wedge \neg F(y)))) \wedge \exists x(\neg G(x)), (\neg\neg), \#2 \\
\models\!\models\ & ((\exists x F(x) \wedge \forall y(G(y) \wedge \neg F(y)))) \wedge \exists z(\neg G(z)), \#3 \\
\models\!\models\ & (\exists x \forall y(F(x) \wedge (G(y) \wedge \neg F(y))) \wedge \exists z(\neg G(z))), \#4 \\
\models\!\models\ & \exists z(\exists x \forall y(F(x) \wedge (G(y) \wedge \neg F(y))) \wedge (\neg G(z))), \#4
\end{aligned}
$$

Definition 3.20

A sentence (formula without free variables) is said to be in **$\exists$-free Prenex Normal Form** if it is in prenex normal form and does not contain existential quantifier symbols.

Result 3.2

Note: When we make a Skolem function for an $\exists$ quantifier, we have to make sure we give it a function symbol that isn't used anywhere else in the formula.

This means that if we have multiple $\exists$ quantifiers in a sentence, each one has to be given a distinct Skolem function.

Theorem 3.6: Skolemization for removal of $\exists$

Suppose f is a function symbol that does not appear in A. The Skolemized version of $\forall x_1 \forall x_2 \cdots \forall x_n \exists y A$ is

$$\forall x_1 \forall x_2 \cdots \forall x_n A'$$

where $n \geqslant 0$, and A' is the expression obtained from A by substituting each occurrence of y by $f(x_1, x_2, \cdots, x_n)$.

3.6 Lecture 18

Lecture 18 - Tuesday, Jul 9

3.6.1 Decision Problem

Definition 3.21: Decidable

decision problem is **decidable** if there is an algorithm that completes in finitely many steps that, given an input to the problem,

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

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

Definition 3.22: Algorithm

Algorithm has to complete in finitely many steps, otherwise it is not an algorithm.

Example 3.28: These examples are decidable:

1. Given a formula A of Propositional logic, is A satisfiable?
   A: The algorithm should be obvious.
2. **A Variation**: Given a formula A of Propositional logic, is A valid?
   A: The algorithm should be obvious.
3. Given a positive integer n, is n prime?
   A: An algorithm was given in Math 135.

Definition 3.23: S-membership Problem

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

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

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

Example 3.29: S-membership Problem

1. Suppose $S \subseteq \mathbb{N}$ is decidable. Is the complement $S^C = \mathbb{N} \backslash S = \{n \in \mathbb{N} : n \notin S\}$ decidable?
2. Suppose $S \subset \mathbb{N}$ is finite. Does it follow that S is decidable?
3. Suppose $S \subset \mathbb{N}$ is infinite. Does it follow that S is undecidable?

3.6.2 Halting Problem

Definition 3.24: Halting Problem

The Halting Problem is the following decision problem:

Given any program M and input x, will M finish running (halt), or run forever on x?

We can use the Halting Problem to prove undecidability of a decision problem via reduction.

Theorem 3.7

The Halting Problem is undecidable

Proof. The outline of the proof is here: https://www.youtube.com/watch?v=92WHN-pAFCs. □

Result 3.3

You will not need to know the proof for this class, but you will need to know the result.

Algorithm 3.8: Reduction

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

Definition 3.25: Reduction

Suppose that we have two decision problems, P1 and P2. Suppose also that we have an algorithm A that transforms inputs for (instances of) P1 into inputs for (instances of) P2 such that

1. "Yes" instances of P1 get mapped to "yes" instances of P2;
2. "No" instances of P1 get mapped to "no" instances of P2;
3. the algorithm A always takes a finite amount of time.

Then A **reduces** P1 to P2 and that A is a **reduction** from P1 to P2.

Theorem 3.8

If there is a reduction of from P_1 to P_2, then

1. If P_1 is undecidable then P_2 is undecidable;
2. If P_2 is decidable then P_1 is decidable.

Example 3.30: Reduction Problem: HaltOnEveryInput

The `HaltOnEveryInput` decision problem asks:

Given any program Q, does Q halt on every input?

Proof. Let (M, w) be any candidate for the halting problem. Construct a new program, Q, to carry out the following algorithm:

1. Given any input x,
2. Ignore the input x, and run M on w.
3. If M halts on w, then make Q halt on x.
4. If M runs forever on w, then by construction, Q runs forever on x.

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

1. If (M, w) halts ("yes for P_1"), then Q halts for every input x ("yes for P_2");
2. If (M, w) runs forever ("no for P_1"), then Q runs forever for every input x ("no for P_2").

This constructs a reduction from the Halting Problem to the `HaltOnEveryInput` problem. Therefore, the `HaltOnEveryInput` problem is undecidable. □

Example 3.31: Reduction Problem: LoopOnZero

The `LoopOnZero` decision problem asks:

Given any program Q, does Q run forever when processing the particular input $n = 0$?

Proof. Suppose for contradiction that LoopOnZero is decidable. Let (M, w) be any candidate for the halting problem. Construct a new program, Q, to carry out the following algorithm:

1. ignore its input, x, and run M with input w, and
2. if M halts when run with input w, then halt and return 0.

By construction, Q halts for all inputs (including 0) if and only if M halts when run with input w. Now feed the constructed Q into B, and return the *opposite* answer for whether (M, w) halts. This provides a decider for the Halting Problem. Since the Halting Problem is undecidable, therefore we have a contradiction. □

Theorem 3.9

Satisfiability in First-Order Logic is Undecidable

Proof. The proof is not examinable. □

3.7 Lecture 19

Lecture 19 - Thursday, Jul 11

3.7.1 Turring Machine

Definition 3.26: Turring Machine

A **Turing Machine** $T = (S, I, f, s_0)$ consists of

1. S: a finite set of states of the finite control unit;
2. I: a finite set of tape symbols, containing a blank symbol B;
3. $s_0 \in S$, the start state;
4. $f : S \times I \to S \times I \times \{L, R\}$, the transition function.

A Turing machine does its work by reading / writing on an imaginary tape. The tape is infinitely long in both directions.

The start state, s_0, is that a finite number of cells are filled with non-B contents, and all the rest of the cells are filled with the blank symbol B

Definition 3.27: Transition Function

The **transition function** describes what the Turing machine does given a state and tape symbol.

Let $s \in S$ be a state, and $x \in I$ be a tape symbol. Then if $f(s, x) = (s', x', D)$, our Turing

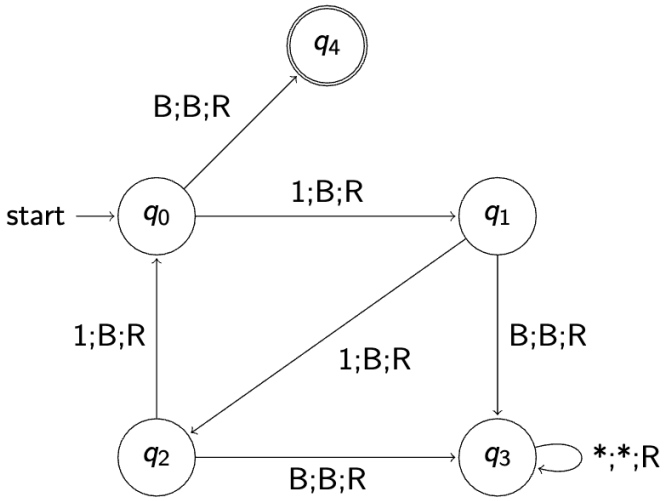
Machine

1. changes its state to s';
2. writes the symbol x' in the current cell on the tape, overwriting x;
3. move the tape head one cell to the right if $D = R$, or one cell to the left if $D = L$.

If the function f is undefined for a pair (s, x) (f need not be a total function) and the Turing Machine reaches that state and symbol, then the Turing Machine will halt.

Example 3.32

We often represent Turing Machines with diagrams.



This is a diagram for a Turing Machine that

1. has tape alphabet $\{B, 1\}$ where B is a blank symbol;
2. starts execution with the tape head pointing to the leftmost 1, if there is one, and to one of infinitely many B symbols otherwise, and;
3. accepts its input string if and only if it consists of a number of 1s that is congruent to 0 modulo 3.

Definition 3.28

$q4$ is often known as **acceptance state**, which indicates that the turring machine has halted.

Discovery 3.1

In the tuple, $\boxed{x; x'; D}$:

1. x is the symbol under the tape head;
2. x' is the overwriting symbol;
3. D is the direction of the tape head movement.

Result 3.4

TMs can be used to accept (recognize) languages (sets of words over some alphabet Σ).

The input string, w, is put on the tape, and the tape head starts at the leftmost character of w. (Recall that the rest of the tape is filled blank symbol B).

To define acceptance of a string, we need to define the concept of a final state.

Definition 3.29: Final State

A **final state** of a Turing Machine $T = (S, I, f, s_0)$ is any state $s_f \in S$ for which f has no next state defined for any symbol (i.e. no outgoing transitions).

Notation: When we draw a TM diagram, we typically denote final states as nodes with a double border.

Definition 3.30: Language Accepted by the Turing Machine T

The set of all possible words that a TM can accept is called the **language accepted by the Turing Machine** T, denoted by $L(T)$.

More formally, the language $L(T)$ accepted by Turing Machine T is defined as the set of strings $w \in \Sigma^*$ such that

$$s_0 w \implies^* \alpha s_f \beta$$

where

1. s_f is a final state;
2. α, β are tape strings in I^*, and;
3. $\implies^*$ denotes repeatedly many applications (including zero times) of the steps of the Turing Machine transition function.

The following examples use the earlier example diagram, which was meant to accept a string of 1s exactly when the number of 1s in the string was a multiple of 3.

Example 3.33

For string 111, we have the following computation:

$$\begin{aligned} & q_0111 \\ \rightarrow & MBq_111 \\ \rightarrow & MBBq_21 \\ \rightarrow & MBBBq_0B \\ \rightarrow & MBBBBq_4B \end{aligned}$$

There is no outgoing transition defined for (q_4, B). Therefore our machine halts. Since q_4 is a final state (no outgoing transitions), therefore our machine halts and accepts the input 111.
Remark: M here indicates that that position is the furthest we could reach.

Example 3.34

For string 11, we have the following computation:

$$\begin{aligned} & q_011 \\ \rightarrow & MBq_11 \\ \rightarrow & MBBq_2B \\ \rightarrow & MBBBq_3B \\ \rightarrow & MBBBBq_3B \\ \rightarrow & MBBBBBq_3B \ldots \text{ and so on } \ldots \end{aligned}$$

We can see that we are in an infinite loop here. Therefore our machine runs forever. Our machine does not accept the input word 11.

Definition 3.31: Computations Performed by a Turing Machine

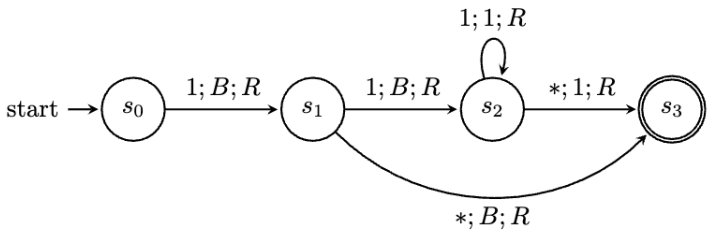
A Turing Machine, T, **computes the (partial) function** $f(x)$ if, given any input, x, to f on the tape at the start of computation,

1. If T halts, then at the time of halting, the output $f(x)$ will be written on the tape, with the tape head pointing to the first character;
2. For any x which cause T to run forever, $f(x)$ is *not defined*.

When all above happens, we say that f is a computable function with T, a TM that computes f.
Remark: Note: Some authors require that T always halts before we call f computable, but we permit partial f's here.

Example 3.35: A TM that adds two natural numbers

The transition function: $(s_0, 1, s_1, B, R)$, $(s_1, *, s_3, B, R)$, $(s_1, 1, s_2, B, R)$, $(s_2, 1, s_2, 1, R)$, $(s_2, *, s_3, 1, R)$.



3.7.2 Church-Turing Thesis

Theorem 3.10

Any problem that can be solved with an algorithm can be solved by a Turing Machine.

Result 3.5

This says that we can use "algorithms" and "Turing machines interchangeably.
It also says that algorithms need not always terminate, because there exist Turing machines that do not always halt.

From now on, we will specify a terminating algorithm or a total TM if we demand that our TM terminates for every input.

4 Peano Arithmetic

4.1 Lecture 20

Lecture 20 - Tuesday, Jul 16

Definition 4.1: Equivalence Relation

A relation that is reflexive, symmetric, and transitive is called an equivalence relation.

4.1.1 $\approx$ is an equivalence relation

Example 4.1: $\approx$ is an equivalence relation

$\approx$ is an equivalence relation.

Proof. 1. *Reflexivity*:

$$\begin{array}{llll} (1) & \varnothing & \vdash & u \approx u & (\approx +) \\ (2) & \varnothing & \vdash & \forall x(x \approx x) & (\forall +) \end{array}$$

2. *Symmetry*:

$$\begin{array}{llll} (1) & \varnothing & \vdash & v \approx v & (\approx +) \\ (2) & v \approx w & \vdash & v \approx w & (\in) \\ (3) & v \approx w & \vdash & v \approx v & (+), (1) \\ (4) & v \approx w & \vdash & w \approx v & (\approx -), (3), (2) \\ (5) & \varnothing & \vdash & v \approx w \to w \approx v & (\to +), (4) \\ (6) & \varnothing & \vdash & \forall y(v \approx y \to y \approx v) & (\forall +), (5) \\ (7) & \varnothing & \vdash & \forall x \forall y(x \approx y \to y \approx x) & (\forall +), (6) \end{array}$$

3. *Transitivity*:

$$\begin{array}{llll} (1) & (u = v) \wedge (v = w) & \vdash & (u = v) \wedge (v = w) & (\in) \\ (2) & (u = v) \wedge (v = w) & \vdash & (u = v) & (\wedge -), (1) \\ (3) & (u = v) \wedge (v = w) & \vdash & (v = w) & (\wedge -), (1) \\ (4) & (u = v) \wedge (v = w) & \vdash & (u = w) & (\approx -), (2), (3) \\ (5) & \varnothing & \vdash & (u = v) \wedge (v = w) \to (u = w) & (\to +), (4) \\ (6) & \varnothing & \vdash & \forall z((u = v) \wedge (v = z) \to (u = z)) & (\forall +), (5) \text{ [w not elsewhere]} \\ (7) & \varnothing & \vdash & \forall y \forall z((u = y) \wedge (y = z) \to (u = z)) & (\forall +), (6) \text{ [v not elsewhere]} \\ (8) & \varnothing & \vdash & \forall x \forall y \forall z((x = y) \wedge (y = z) \to (x = z)) & (\forall +), (7) \text{ [u not elsewhere]} \end{array}$$

□

Theorem 4.1: Equivalence Substitution

Suppose r is a term and v is free variable, then if $\Sigma \vdash t_1 \approx t_2$, then $\Sigma \vdash r(t_1) \approx r(t_2)$.

Definition 4.2: Peano Arithmetic

Peano Arithmetic is a way for us to build up a theory of the natural numbers $\mathbb{N}$ from scratch (i.e. rigorously).

4.1.2 Peano Axioms

Result 4.1: Peano Axioms

"PA" stands for Peano Axioms, named for Giuseppe Peano.

1. PA1 $\forall x \neg(s(x) \approx 0)$.
   "Zero is not a successor."
2. PA2 $\forall x \forall y(s(x) \approx s(y) \to x \approx y)$.
   "Nothing has two predecessors."
3. PA3 $\forall x(x + 0 \approx x)$.
   Adding zero to any $n \in \mathbb{N}$ yields n.

4. PA4 $\forall x \forall y(x + s(y) \approx s(x + y))$.
   If $n, m \in \mathbb{N}$, then adding the successor of m to n yields the successor of $n + m$.

5. PA5 $\forall x(x \times 0 \approx 0)$.
   Multiplying by zero yields zero.

6. PA6 $\forall x \forall y(x \times s(y) \approx (x \times y) + x)$.
   Multiplication by a successor.

7. PA7 For each formula $A(u)$ and variable x,

$$((A(0) \wedge \forall x(A(x) \to A(s(x)))) \to \forall x A(x))$$

Discovery 4.1

The axiom PA7 is a translation of the Principle of Mathematical Induction into First-order Logic.

Algorithm 4.1

To prove $\forall x A(x)$, we must

1. prove the base case $A(0)$, and;
2. prove the inductive case $\forall x(A(x) \to A(s(x)))$.

Theorem 4.2: no natural number equals its successor

$$\varnothing \vdash_{PA} \forall x(\neg s(x) \approx x)$$

Proof. 1. *Base Case*: Prove $\varnothing \vdash \neg s(0) \approx 0$:

(1)	$\varnothing$	$\vdash$	$\forall x \neg s(x) \approx 0$	PA1
(2)	$\varnothing$	$\vdash$	$\neg s(0) \approx 0$	$(\forall -), (1)$

2. *Inductive Case*: $\varnothing \vdash \forall x(\neg(s(x) \approx x) \to \neg(s(s(x)) \approx s(x)))$:

(1)	$\neg(s(v) \approx v), s(s(v)) \approx s(v)$	$\vdash$	$s(v) \approx v$	PA2
(2)	$\neg(s(v) \approx v), s(s(v)) \approx s(v)$	$\vdash$	$\neg(s(v) \approx v)$	$(\in)$
(3)	$\neg(s(v) \approx v)$	$\vdash$	$\neg(s(s(v)) \approx s(v))$	$(\neg +), (1), (2)$
(4)	$\varnothing$	$\vdash$	$(\neg(s(v) \approx v) \to \neg(s(s(v)) \approx s(v)))$	$(\to +), (3)$
(5)	$\varnothing$	$\vdash$	$\forall x(\neg(s(x) \approx x) \to \neg(s(s(x)) \approx s(x)))$	$(\forall +), (4)$

I jumped steps in line (1). Therefore, by PA7, we have

$$\varnothing \vdash (((\neg(s(0) \approx 0)) \wedge \forall x((\neg(s(x) \approx x)) \to (\neg(s(s(x)) \approx s(x))))) \to \forall x(\neg(s(x) \approx x)))$$

Combined with our base case and inductive case, we have

$$\varnothing \vdash \forall x(\neg s(x) \approx x)$$

as wanted. □

Example 4.2

Show that the successor function, with our chosen axioms, behaves like $s(n) = n + 1$, i.e. show that

$$\varnothing \vdash_{PA} \forall x(x + s(0) \approx s(x))$$

Note: We don't use the symbol 1 in our formal deduction, but we know that the successor of 0 is 1, so we represent 1 as $s(0)$.

4.2 Lecture 21

Lecture 21 - Thursday, Jul 18

Theorem 4.3: Extended EqTrans

Let $k \geqslant 1$ be a natural number, Σ be a set of first-order logic formulas, and $t_1, t_2, \ldots, t_{k+1}$ be terms. If $\Sigma \vdash t_i \approx t_{i+1}$ for all $1 \leqslant i \leqslant k$, then $\Sigma \vdash t_1 \approx t_{k+1}$.

Proof. The proof is by induction on k. □

Lemma 4.1

Peano Arithmetic has a proof of

$$\varnothing \vdash_{PA} \forall y(0 + y \approx y + 0)$$

Proof. By PA7,

1. *Base Case*:

   $$\varnothing \vdash 0 + 0 \approx 0 + 0$$

   This is immediate from reflexivity of $\approx$.

2. *Induction Step*:

   $$\varnothing \vdash \forall y(0 + y \approx y + 0 \to 0 + s(y) \approx s(y) + 0)$$

(1)	$\varnothing$	$\vdash$	$\forall x(x + 0 \approx x)$	[PA3]
(2)	$\varnothing$	$\vdash$	$\forall x \forall y(x + s(y) \approx s(x + y))$	[PA4]
(3)	$\varnothing$	$\vdash$	$\forall y(0 + s(y) \approx s(0 + y))$	$(\forall -), (2)$
(4)	$\varnothing$	$\vdash$	$0 + s(v) \approx s(0 + v)$	$(\forall -), (3)$
(5)	$\varnothing$	$\vdash$	$v + 0 \approx v$	$(\forall -), (1)$
(6)	$\varnothing$	$\vdash$	$s(v + 0) \approx s(v)$	$EQSubs, (5)$
(7)	$\varnothing$	$\vdash$	$s(v) + 0 \approx s(v)$	$(\forall -), (1)$
(8)	$0 + v \approx v + 0$	$\vdash$	$0 + s(v) \approx s(0 + v)$	$(+), (4)$
(9)	$0 + v \approx v + 0$	$\vdash$	$v + 0 \approx v$	$(+), (5)$
(10)	$0 + v \approx v + 0$	$\vdash$	$s(v + 0) \approx s(v)$	$(+), (6)$
(11)	$0 + v \approx v + 0$	$\vdash$	$s(v) + 0 \approx s(v)$	$(+), (7)$
(12)	$0 + v \approx v + 0$	$\vdash$	$0 + v \approx v + 0$	$(\in)$
(13)	$0 + v \approx v + 0$	$\vdash$	$s(0 + v) \approx s(v + 0)$	$EQSubs, (12)$
(14)	$0 + v \approx v + 0$	$\vdash$	$s(v) \approx s(v) + 0$	$Symmof \approx, (11)$
(15)	$0 + v \approx v + 0$	$\vdash$	$0 + s(v) \approx s(v) + 0$	$EQTrans, (8), (13), (10), (14)$

Now, complete this to a proof of

$$\varnothing \vdash \forall y(0 + y \approx y + 0 \to 0 + s(y) \approx s(y) + 0)$$

(16)	$\varnothing$	$\vdash$	$0 + v \approx v + 0 \to 0 + s(v) \approx s(v) + 0$	$(\to +), (15)$
(17)	$\varnothing$	$\vdash$	$\forall y(0 + y \approx y + 0 \to 0 + s(y) \approx s(y) + 0)$	$(\forall +), (16)(v$ does not occur in $\varnothing)$

□

Lemma 4.2

For each free variable u,

$$\{\forall y(u + y \approx y + u)\} \vdash_{PA} \forall y(s(u) + y \approx y + s(u))$$

Proof. 1. *Base Case*:

$$\varnothing \vdash_{PA} s(u) + 0 \approx 0 + s(u)$$

The target follows from $\forall y(0 + y \approx y + 0)$ plus $(\forall -)$ plus the symmetry of $\approx$.

2. *Induction*;

$$\varnothing \vdash_{PA} \forall y(s(u) + y \approx y + s(u) \to s(u) + s(y) \approx s(y) + s(u))$$

Let a free v replace y. Assuming $s(u) + v \approx v + s(u)$ yields, informally

$$\begin{array}{lll} s(u) + s(v) & \approx s(s(u) + v) & \text{PA4} \\ & \approx s(v + s(u)) & \text{Assumption + EQSubs} \\ & \approx s(s(v + u)) & \text{PA4 + EQSubs.} \end{array}$$

The premise of the Lemma implies $u + s(v) \approx s(v) + u$; thus, informally,

$$\begin{array}{lll} s(v) + s(u) & \approx s(s(v) + u) & \text{PA4} \\ & \approx s(u + s(v)) & \text{premise + symmetry of } \approx \text{ + EQSubs} \\ & \approx s(s(u + v)) & \text{PA4 + EQSubs.} \end{array}$$

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. □

4.2.1 Commutativity of + in Peano Arithmetic

Theorem 4.4: Commutativity of + in Peano Arithmetic

Addition in Peano Arithmetic is commutative; that is,

$$\varnothing \vdash_{PA} \forall x \forall y (x + y \approx y + x)$$

Proof. The proof the by induction using Lemma (4.1) as out Base Case and Lemma (4.2) as our induction step. In particular, we have

(1)	$\varnothing$	$\vdash$	$(\forall y(0 + y \approx y + 0) \wedge \forall x((\forall y(x + y \approx y + x))$	
			$\rightarrow ((\forall y(s(x) + y \approx y + s(x)))))) \rightarrow \forall x(\forall y(x + y \approx y + x))$	[PA7]
(2)	$\varnothing$	$\vdash$	$\forall y(0 + y \approx y + 0)$	by Base Case
(3)	$\varnothing$	$\vdash$	$\forall x((\forall y(x + y \approx y + x)) \rightarrow ((\forall y(s(x) + y \approx y + s(x)))))$	by Induction Step
(4)	$\varnothing$	$\vdash$	$(\forall y(0 + y \approx y + 0) \wedge \forall x((\forall y(x + y \approx y + x))$	
			$\rightarrow ((\forall y(s(x) + y \approx y + s(x)))))$	$(\wedge+), (2), (3)$
(5)	$\varnothing$	$\vdash$	$\forall x(\forall y(x + y \approx y + x))$	$(\rightarrow -), (1), (4)$

□

4.2.2 Associativity of + in Peano Arithmetic

Theorem 4.5: Associativity of + in Peano Arithmetic

Addition in Peano Arithmetic is associative, i.e.

$$\varnothing \vdash_{PA} \forall x \forall y \forall z ((x + y) + z \approx x + (y + z))$$

5 Hoare Logic

5.1 Lecture 22

Lecture 22 - Tuesday, Jul 23

5.1.1 Hoare Triple

Definition 5.1: Hoare Triple

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.

A specification of a program C is a Hoare triple with C as its middle element.

Definition 5.2: State

The **state** of a program at a given moment is the list of values of each of its variables at that moment.

Definition 5.3: Partial Correctness

A Hoare triple is **satisfied under partial correctness** (aka partially correct) if, whenever execution starts in a state satisfying precondition P, and terminates, it follows that the state after execution satisfies postcondition Q.

Example 5.1

The Hoare triple

$$(\!|x \approx x_0|\!) \tag{1}$$
$$x = 1; \tag{2}$$
$$(\!|x_0 = 1|\!) \tag{3}$$

is satisfied under partial correctness.

Definition 5.4: Total Correctness

A Hoare Triple $(\!|P|\!)C(\!|Q|\!)$ is **satisfied under total correctness** (aka totally correct) if it is satisfied under partial correctness, and whenever C starts in a state obeying P, it follows that C terminates.

Discovery 5.1

Partial and total correctness are the same thing until we introduce loops, i.e. until termination is no longer guaranteed.

Result 5.1

total correctness = partial correctness + termination.

Algorithm 5.1

To show that a specification of a program C, with a while-loop, is satisfied under total correctness, we will

1. prove the specificaton of C is satisfied under partial correctness, and;
2. provide a separate proof that the program C always terminates, when run starting in a state obeying the given pre-condition.

5.1.2 Program Annotation Rules

Theorem 5.1: Program Annotation Rules: Pre- and Post-Conditions

1. Rule of "Precondition strengthening":

$$\frac{P \to P' \qquad (\!|P'|\!)\, C\, (\!|Q|\!)}{(\!|P|\!)\, C\, (\!|Q|\!)}\,(implied)$$

2. Rule of "Postcondition weakening":

$$\frac{(\!|P|\!)\, C\, (\!|Q'|\!) \qquad Q' \to Q}{(\!|P|\!)\, C\, (\!|Q|\!)}\,(implied)$$

Theorem 5.2: Program Annotation Rules: Composition

Composition Rule:

$$\frac{(\!|P|\!)\, C_1\, (\!|Q|\!) \qquad (\!|P|\!)\, C_2\, (\!|Q|\!)}{(\!|P|\!)\, C_1;\ C_2\, (\!|Q|\!)}\,(composition)$$

Theorem 5.3: Program Annotation Rules: Assignment

The assignment rule is

$$\begin{array}{l} (\!|Q[E/x]|\!) \\ x = E \\ (\!|Q|\!) \end{array}$$

where $Q[E/x]$ denotes a copy of Q, with all free x's replaced by E's.

Result 5.2

The assignment rule has no premises and is therefore an axiom of our proof system.

Example 5.2: Swap x and y

Prove

$$\begin{array}{l}(\!|x \approx x_0 \wedge y \approx y_0|\!) \\ t = x; \\ x = y; \\ y = t; \\ (\!|x \approx y_0 \wedge y \approx x_0|\!)\end{array}$$

Proof. **Exercise:** Prove

$$\varnothing \vdash x \approx x_0 \wedge y \approx y_0 \rightarrow y \approx y_0 \wedge x \approx x_0$$

Annotated program:

$$\begin{array}{ll}(\!|x \approx x_0 \wedge y \approx y_0|\!) & \\ (\!|y \approx y_0 \wedge x \approx x_0|\!) & \text{implied}(a)[\text{Exercise}] \\ t = x; & \\ (\!|y \approx y_0 \wedge t \approx x_0|\!) & \text{assignment} \\ x = y; & \\ (\!|x \approx y_0 \wedge t \approx x_0|\!) & \text{assignment} \\ y = t; & \\ (\!|x \approx y_0 \wedge y \approx x_0|\!) & \text{assignment}\end{array}$$

□

Theorem 5.4: Program Annotation Rules: Conditionals

1. if-then-else:
$$\frac{(\!|P \wedge B|\!)C_1(\!|Q|\!) \qquad (\!|P \wedge \neg B|\!)C_2(\!|Q|\!)}{(\!|P|\!)\ if\ (B)\ C_1\ else\ C_2(\!|Q|\!)}(if\text{-}then\text{-}else)$$
2. if-then (without else):
$$\frac{(\!|P \wedge B|\!)C(\!|Q|\!) \qquad (P \wedge \neg B) \rightarrow Q}{(\!|P|\!)\ if\ (B)\ C\ (\!|Q|\!)}(if\text{-}then)$$

5.2 Lecture 23

Lecture 23 - Thursday, Jul 25

5.2.1 More Program Annotation Rules

Theorem 5.5: Program Annotation Rules: Partial while

Partial while": does not (yet) require termination.

$$\frac{(\!|I \wedge B|\!)C(\!|I|\!)}{(\!|I|\!)\ while\ (B)\ C\ (\!|I \wedge \neg B|\!)}(partial\text{-}while)$$

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.

5.2.2 Example: Factorial

Example 5.3

Example to compute x!: Prove that the following specification is satisfied under total correctness.

```
(|x ⩾ 0|)
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
(|y ≈ x!|)
```

Proof.

```
(|x ⩾ 0|)
(|1 ≈ 0!|)                          implied (a)
y = 1;
(|y ≈ 0!|)                          assignment
z = 0;
(|y ≈ z!|)                          assignment
while (z != x) {
    (|y ≈ z! ∧¬(z ≈ x)              partial-while
    (|y * (z+1) ≈ (z+1)!|)          implied (b)
    z = z + 1;
    (|y * z ≈ z!|)                  assignment
    y = y * z;
    (|y ≈ z!|)                      assignment
}
(|y ≈ z! ∧ (z ≈ x)|)               partial while
(|y ≈ x!|)                          implied (c)
```

where implied (a) and implied (b) follow from the definition of factorial. Notice that this only proves partial correctness. In order to show total correctness, we need to show that the loop terminates eventually. Suppose that we start in a state obeying the precondition $x \geqslant 0$. 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?

```
(|x ⩾ 0|)
y = 1;
z = 0;                  At start of loop: x - z ⩾ 0 ✓
while (z != x) {
    z = z + 1;          x - z decreases by 1  ✓
    y = y * z;          x - z unchanged
}
(|y ≈ x!|)
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. □

Definition 5.5: Checkmarks

The three checkmarks on the previous slide indicate the three key features of a loop variant.
In the presence of a loop variant having these three features, termination is clear.

5.2.3 Example: Exponentiation

Example 5.4

Prove that the following specification is satisfied under total correctness.

```
(|n ⩾ 0∧a ⩾ 0|)
s = 1;
i = 0;
while (i < n) {
    s = s * a;
    i = i + 1;
}
(|s = a^n|)
```

Proof. Our loop invariant would be $s \approx a^i \wedge i \leqslant n$.

```
(|n ⩾ 0∧a ⩾ 0|)
(|1 = a^0∧0 ⩽ n|)                    implied (a)
s = 1;
(|s = a^0∧0 ⩽ n|)                    assignment
i = 0;
(|s = a^i∧i ⩽ n|)                    assignment
```

```
while (i < n) {
    (|s = a^i ∧ (i ⩽ n) ∧ i < n|)        partial-while
    (|s * a = a^(i+1) ∧ (i + 1 ⩽ n)|)    implied (b)
    s = s * a;
    (|s = a^(i+1) ∧ (i + 1 ⩽ n)|)        assignment
    i = i + 1;
    (|s = a^i ∧ (i ⩽ n)|)                assignment
}
(|s = a^i ∧ (i ⩽ n) ∧ ¬(i < n)|)         partial-while
(|s = a^n|)                              implied (c)
```

□

Result 5.3: Total Correctness Problem

Is an arbitrary Hoare triple $(\!|P|\!)C(\!|Q|\!)$ satisfied under total correctness?

Theorem 5.6

The Total Correctness Problem is undecidable.

Result 5.4: Partial Correctness Problem

Is an arbitrary Hoare triple $(\!|P|\!)C(\!|Q|\!)$ satisfied under partial correctness?

Theorem 5.7

The Partial Correctness Problem is undecidable.

5.3 Lecture 24

Lecture 24 - Friday, Jul 30

Just Review...