CS 245 Notes

Angel Zhang

Spring 2021

Logic 1 Introduction

Valid

An argument is **valid** (or **correct**, or **sound**) if whenever the premises are true, the conclusion is also true.

Proposition

A **proposition** is a statement that is either *true* or *false*.

Equivalent ways of expressing an implication

The following are logically equivalent.

1. p → q.
2. If p then q.
3. Whenever p, then q.
4. p is sufficient for q.
5. p only if p.
6. p implies q.
7. q if p.
8. q whenever p.
9. q is necessary for p.
10. q is implied by p.

Logic 2 Propositional Language & Syntax

Atomic Formula

An **atomic formula**, or **atom**, is an expression in $\mathcal{L}^p$ with length one consisting of a variable only. The set of atomic formulas in $\mathcal{L}^p$ is denoted by $Atom(\mathcal{L}^p)$

Unique Readability

Every formula of $Form(\mathcal{L}^p)$ is of exactly one of the six forms: an atom, $\neg A$, $A \wedge B$, $A \vee B$, $A \to B$ or $A \leftrightarrow B$, and in each case, it is of that form in exactly one way.

Precedence Rules

- $\neg$ has precedence over $\wedge$
- $\wedge$ has precedence over $\vee$
- $\vee$ has precedence over $\to$
- $\to$ has precedence over $\leftrightarrow$

Logic 3 Propositional Language Semantics

Truth Valuations

- A **truth valuation** is a function t

$$t : Atom(\mathcal{L}^p) \to \{0, 1\}$$

  whose domain is the set of all propositional symbols and range is $\{0, 1\}$.
- A **truth table** list the values of a formula under all possible truth valuations.
- A truth valuation corresponds to a *single row* in the truth table.

Satisfiability

- A truth valuation t **satisfies** a formula A in $Form(\mathcal{L}^p)$ if and only if $A^t = 1$.
- We use the Greek letter Σ to denote any set of formulas.
- $\Sigma^t = 1$ if and only if each formula $B \in \Sigma$ has $B^t = 1$.

Satisfiable

A set of formulas $\Sigma \subseteq Form(\mathcal{L}^p)$ is **satisfiable** if and only if there exists a truth valuation t such that $\Sigma^t = 1$. If there is no truth valuation t such that $\Sigma^t = 1$ (or equivalently, if $\Sigma^t = 0$ for all truth valuations t), then the set Σ is **unsatisfiable**.

Notes

- Note that $\Sigma^t = 1$ means that under the truth valuation t, *all* the formulas of Σ are true.
- On the other hand, $\Sigma^t = 0$ means that for *at least one* formula $B \in \Sigma$, we have that $B^t = 0$.

- In particular, $\Sigma^t = 0$ does not necessarily mean that $C^t = 0$ for every formula C in Σ.

Tautologies and Contradictions

- A formula A is a **tautology** if and only if it is true under *all* possible truth valuations. For every truth valuation t, $A^t = 1$.
- A formula A is a **contradiction** if and only if it is false under *all* possible truth valuations. For every truth valuation t, $A^t = 0$.
- A formula that is neither a tautology nor a contradiction is called **contingent**.

Law of Excluded Middle

$p \vee \neg p$ is a tautology.

Law of Contradiction

$p \wedge \neg p$ is a contradiction.

Tautologies and Contradictions

A is a tautology if and only if $\neg A$ is a contradiction.

Tautological Consequence

Suppose $\Sigma \subseteq \mathit{Form}(\mathcal{L}^p)$ and $A \in \mathit{Form}(\mathcal{L}^p)$. We say that A is **tautological consequence** of Σ if and only if for every truth valuation t, if $\Sigma^t = 1$, then $A^t = 1$. We denote this as $\Sigma \models$ A.

Empty Set

$\varnothing \models A$ means that A is a tautology. For every truth valuation t, $\varnothing^t = 1$ is always vacuously true. Therefore $\varnothing \models A$ means that A is always true.

Tautological Equivalence

For two formulas we write $A \models\!\!\!\dashv B$ to denote $A \models B$ and $B \models A$. A and B are **tautologically equivalent** if $A \models\!\!\!\dashv B$.

Notes

- $A \to B$ is a formula. A valuation gives it a truth value.
- $A \models B$ if and only if the formula $A \to B$ is a tautology.

- $\varnothing \models A \to B$ means that $A \to B$ is a tautology.
- $A \leftrightarrow B$ is a formula. A valuation gives it a truth value.
- $A \models\!\!\!\!= \!\!\mid B$ if and only if $A \leftrightarrow B$ is a tautology.
- $\varnothing \models A \leftrightarrow B$ holds if and only if $A \leftrightarrow B$ is a tautology.

Replaceability

Let A be a formula which contains a subformula B. Assume that $B \models\!\!\!\!= \!\!\mid C$, and let A' be the formula obtained by simultaneously replacing in A some occurrences of the formula B by the formula C. Then $A' \models\!\!\!\!= \!\!\mid A$.

Duality

Suppose A is a formula composed only of atoms and the connectives $\neg, \vee, \wedge$ by the formation rules concerned these three connectives. Suppose $\Delta(A)$ results from simultaneously replacing in A all occurrences of $\wedge$ with $\vee$, all occurrences of $\vee$ with $\wedge$, and each atom with its negation. Then $\neg A \models\!\!\!\!= \!\!\mid \Delta(A)$.

Logic 4 Propositional Calculus

Literal

A formula is called a **literal** if it is of the form p or $\neg p$, where p is a proposition symbol. The two formulas p and $\neg p$ are called **complementary literals**.

Disjunctive Normal Form (DNF)

A disjunction with conjunctive clauses as its disjuncts is said to be in **Disjunctive Normal Form (DNF)**.

Conjunctive Normal Form (CNF)

A conjunction with disjunctive clauses as its conjuncts is said to be in **Conjunctive Normal Form (CNF)**.

Logic 5 Adequate Sets of Connectives

Number of distinct n-ary connectives

The number of distinct n-ary connectives is $2^{(2^n)}$

Adequate Set of Connectives

- Let S be a set of connectives. We say that S is **adequate** iff for every $n \geq 1$ and for every n-ary connective f, there exists a formula, A_S, written using only the connectives from S, such that $f(\mathrm{p}_1, \mathrm{p}_2, ..., \mathrm{p}_n) \models\mid \mathrm{A}_S$
- The set of **five standard connectives**, $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$, is adequate.
- The set $S_0 = \{\neg, \wedge, \vee\}$, is adequate.
- To show that a new set S of connectives is adequate, show that all connectives in $S_0 = \{\neg, \wedge, \vee\}$ are definable in terms of the new connectives in S.

Other Adequate Sets of Connectives

- $\{\neg, \wedge\}$ is an adequate set of connectives.
- $\{\neg, \vee\}$ is an adequate set of connectives.
- $\{\neg, \rightarrow\}$ is an adequate set of connectives.

Proving inadequacy

We show that one of the connectives in the adequate set $S_0 = \{\neg, \wedge, \vee\}$ *cannot* be defined by using the connectives in S.

Logic 6A Formal Deduction in Propositional Logic

Formal Deducibility

- A formula A is **formally deducible** from Σ, written as $\Sigma \vdash \mathrm{A}$, if and only if $\Sigma \vdash \mathrm{A}$ is generated by (a finite number of applications) the rules of the formal system.
- The sequence of rules generating $\Sigma \vdash \mathrm{A}$ is called a **formal proof**.

Logic 6B Formal Deduction in Propositional Logic

Consequence vs. Deducibility

- $\mathrm{A} \models \mathrm{B}$ iff $\mathrm{A} \rightarrow \mathrm{B}$ is a tautology.
- $\mathrm{A} \vdash \mathrm{B}$ iff $\varnothing \vdash \mathrm{A} \rightarrow \mathrm{B}$.

Double-negation elimination ($\neg\neg-$)

For every formula A, one can prove $\neg\neg\mathrm{A} \vdash \mathrm{A}$.

Empty Set of Premises

- When $\varnothing \vdash \mathrm{A}$ holds, A is said to be **formally provable** (from nothing).
- Two examples of formally provable formulas are **non-contradiction** and **excluded middle**. That is,

$$\varnothing \vdash \neg(\mathrm{A} \wedge \neg \mathrm{A})$$
$$\varnothing \vdash \mathrm{A} \vee \neg \mathrm{A}$$

Transitivity of Deducibility (Tr)

$$\text{If } \Sigma \vdash \mathrm{A}_1, ..., \mathrm{A}_n \text{ and } \mathrm{A}_1, ..., \mathrm{A}_n \vdash \mathrm{A}, \text{ then } \Sigma \vdash \mathrm{A}.$$

Syntactic Equivalence

For two formulas A and B, we write $\mathrm{A} \dashv\vdash \mathrm{B}$ iff $\mathrm{A} \vdash \mathrm{B}$ and $\mathrm{B} \vdash \mathrm{A}$. They are said to be **syntactically equivalent**.

Finiteness of Premise Set

If $\Sigma \vdash \mathrm{A}$, then there is some finite $\Sigma_0 \subseteq \Sigma$ such that $\Sigma_0 \vdash \mathrm{A}$.

Soundness and Completeness

- Every statement that one can prove should actually be correct. This property is called **soundness**.
- One should be able to prove, within the system, every correct statement. This property is called **completeness** of the formal system.

Soundness and Completeness of the Formal Deduction System

- The formal system of Natural Deduction (the eleven rules) is **sound** for propositional logic. That is, for all Σ and A,

$$\text{if } \Sigma \vdash \mathrm{A}, \text{ then } \Sigma \models \mathrm{A}.$$

- The formal system of Natural Deduction (the eleven rules) is **complete** for propositional logic. That is, for all Σ and A,

$$\text{if } \Sigma \models \mathrm{A}, \text{ then } \Sigma \vdash \mathrm{A}.$$

Logic 6C Formal Deduction in Propositional Logic

Inconsistent Sets

- *Definition.* A set of formulas Σ is **inconsistent** iff there exists a formula A such that $\Sigma \vdash \text{A}$ and $\Sigma \vdash \neg\text{A}$.
- *Lemma.* A set Σ is inconsistent iff for every formula B, $\Sigma \vdash \text{B}$.

Consistent Sets

- *Definition.* A set of formulas Σ is **consistent** iff it is not inconsistent. That is, there exists no formula A such that $\Sigma \vdash \text{A}$ and $\Sigma \vdash \neg\text{A}$.
- *Lemma.* A set Σ is inconsistent iff there exists a formula B such that $\Sigma \nvdash \text{B}$.

Proofs and Inconsistency

Lemma. Let Σ be a set of propositional formulas, and A be a propositional formula. Then

$$\Sigma \vdash \text{A if and only if } \Sigma \cup \{\neg\text{A}\} \text{ is inconsistent.}$$

Equivalently,

$$\Sigma \nvdash \text{A if and only if } \Sigma \cup \{\neg\text{A}\} \text{ is consistent.}$$

An analogy in Semantics

Lemma. Let Σ be a set of propositional formulas, and A be a propositional formula. Then

$$\Sigma \models \text{A if and only if } \Sigma \cup \{\neg\text{A}\} \text{ is unsatisfiable.}$$

Equivalently,

$$\Sigma \not\models \text{A if and only if } \Sigma \cup \{\neg\text{A}\} \text{ is satisfiable.}$$

Maximal Consistent Sets

- *Definition.* A consistent set Σ is **maximal** if for every formula A, either $\text{A} \in \Sigma$ or $\neg\text{A} \in \Sigma$.
- *Lemma.* Let Σ be a maximal consistent set. Then for every A, $\Sigma \vdash \text{A}$ iff $\text{A} \in \Sigma$

Soundness of Propositional Formal Deduction

Theorem. Σ is satisfiable if and only if Σ is consistent.

Logic 7 Resolution for Propositional Logic

Overview

Resolution theorem proving is a method of formal derivation (formal deduction) that has the following features:

- The only formulas allowed in resolution theorem proving are *disjunctions of literals.*
- There is essentially only one rule of formal deduction, **resolution**.

Resolution

Resolution is the formal deduction rule

$$\mathrm{C} \vee \mathrm{p}, \mathrm{D} \vee \neg \mathrm{p} \vdash_{\mathrm{r}} \mathrm{C} \vee \mathrm{D}$$

where $\mathrm{C} \vee \mathrm{p}$ and $\mathrm{D} \vee \neg\mathrm{p}$ are called **parent clauses**, and we say that we are resolving the two parent clauses over p. $\mathrm{C} \vee \mathrm{D}$ is called the **resolvent**. The resolvent of p and $\neg$p is called the **empty clause** and is denoted by $\{\}$.

$$\mathrm{p}, \neg \mathrm{p} \vdash_{\mathrm{r}} \{\}$$

In the context of resolution, the empty clause $\{\}$ is a notation signifying that the contradiction $\mathrm{p}\wedge\neg\mathrm{p}$ was reached. By definition, the empty clause is **unsatisfiable**.

Soundness of Resolution Formal Deduction

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

The Davis-Putnam Procedure (DPP)

- If the output of DPP is the empty clause, $\{\}$, then this indicates that both p and $\neg$ p were produced. This implies that the set of clauses btained by pre-processing the premises and negation of the conclusion of the argument was **unsatisfiable**, that is, the argument (theorem) is **valid**.
- If the output of DPP is no clause, $\varnothing$, then no contradiction was found, and teh original argument (theorem) was **invalid**.

Soundness and Completeness of DPP

Let S be a finite set of clauses. Then S is not satisfiable iff the output of DPP on input S is the empty clause $\{\}$. DPP is **sound** and **complete**.

Logic 10 First-Order Logic

Domain

- The **domain** (or **universe of discourse**) is the collection of all persons, ideas, symbols, data structures and so on, that affect the logical argument under consideration.
- A domain is always non-empty.

Individuals

- The elements of the domain are called **individuals**.
- An individual can be a person, a number, a data structure, etc.

Relations

- Generally, relations make statements about individuals.
- In each statement, there is a list of individuals, called **argument list**.
- The entries of the argument list are called **arguments**.
- If all arguments of a relation are individuals in the domain, then the resulting atomic formula must be either true or false.

Arity

- The number of elements in the argument list of a relation is called the **arity** of the relation.
- The arity of a relation is fixed.
- A relation with arity n is often called an n**-ary relation symbol**, or an n**-place relation symbol**.
- A relation with arity n is often called an n**-ary relation symbol**, or an n**-place relation symbol**.

Variables

- Variables range over the domain.
- Variables are often chosen from the end of the alphabet.
- x, y and z, with or without subscripts, suggest **bound variable** names.
- u, v and w, with or without subscripts, suggest **free variable** names.

Atomic Formulas

- A relation name, followed by an argument list in parentheses is called an **atomic formula**.
- Atomic formulas take true/false values
- Atomic formulas can be combined by logical connectives, like propositions.
- Examples: Human(Socrates) and Mortal(Socrates).

Universal Quantifier

- ∀xA(x) is true iff A(u) is true for all possible values of u in the domain.
- ∀x is called the **universal quantifier** and A(x) is called the **scope** of the quantifier.
- The variable x is said to be **bound** by the quantifier.

Existential Quantifier

- ∃xA(x) is true iff A(u) is true for at least one value u in the domain.
- ∃x is called the **existential quantifier** and A(x) is called the **scope** of the quantifier.
- The variable x is said to be **bound** by the quantifier.

Bound and free variables

- The variable appearing in the quantifier is said to be **bound**.
- Any variable that is not bound is said to be **free**.

Logic 11 First-Order Logic Syntax

Terms over $\mathcal{L}$

$Term(\mathcal{L})$ is the smallest class of expressions of L closed under the following formation rules:

1. Every individual symbol is a term of $\mathcal{L}$.
2. Every free-variable symbol is a term of $\mathcal{L}$.
3. If $t_1, t_2, ..., t_n$ are terms of $\mathcal{L}$, and f is an n-ary function symbol, then $f(t_1, t_2, ..., t_n)$ is a term of $\mathcal{L}$.

Atoms of $\mathcal{L}$

An expression of $\mathcal{L}$ is an **atom** in $Atom(\mathcal{L})$ iff it has one of the following two forms:

1. If $t_1, t_2, \ldots, t_n$ are terms in $Term(\mathcal{L})$, then $F(t_1, t_2, \ldots, t_n)$ is an atom.
2. If t_1 and t_2 are terms in $Term(\mathcal{L})$, then $\approx(t_1, t_2)$ is an atom.

Logic 12 First-Order Logic Semantics

Valid and Satisfiability

- A formula A is **valid** if *every* interpretation and valuation satisfy A. That is, if $\mathrm{A}^v = 1$ for every v.
- A formula A is **satisfiable** if *some* interpretation and valuation satisfy A. That is, if $\mathrm{A}^v = 1$ for some v.
- A formula A is **unsatisfiable** if *no* interpretation and valuation satisfy A. That is, if $\mathrm{A}^v = 0$ for every v.

Logical Consequence

- Suppose Σ is a set of formulas and v a valuation. We define that $\Sigma^v = 1$ (v **satisfies** Σ) if for every $\mathrm{A} \in \Sigma$, $\mathrm{A}^v = 1$.
- Suppose Σ is a set of formulas and v a valuation. Then A is a **logical consequece** of Σ, or Σ entails A, written as $\Sigma \models \mathrm{A}$, iff for every valuation v, we have $\Sigma^v = 1$ implies $\mathrm{A}^v = 1$
- $\varnothing \models \mathrm{A}$ means that A is **valid**.

Satisfiability in First-Order Logic

Suppose Σ is a set of formulas in L.

- $\Sigma \subseteq Form(L)$ is **satisfiable** iff there is some interpretation I such that $\Sigma^v = 1$.
- When $\Sigma \subseteq Form(L)$ and $\Sigma^v = 1$, we say that I **satisfies** Σ, or I is a **model** of Σ, or Σ is **true** in I.

Logic 14 First-Order Logic Deducibility

Formal Deducibility fo Predicate Logic

- Formal deducibility in predicate logic is an extension of that in propositional logic.
- The 11 rules of formal deduction for propositional logic are *included* in predicate logic, but the formulas occurring in them are now first-order formulas.

Formal Deducibility

Suppose Σ is a set of formulas in $\mathcal{L}$ and A is a formula in $\mathcal{L}$. We say that A is **formally deducible** from Σ in predicate logic iff the sequent

$$\Sigma \vdash \text{A}$$

can be generated by the 17 rules of formal deduction. The use of these 17 rules is called **Natural Deduction** for predicate logic.

Replacement of Equivalent Formulas

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

Complementation

Suppose A is a formula composed of atoms of $\mathcal{L}^p$, the connectives $\neg, \wedge, \vee$ and the two quantifiers by the formation rules concerned, and A$'$ is the formula obtained by exchanging $\vee$ and $\wedge$, $\exists$ and $\forall$, and negating all atoms. Then A$'$ $\vdash\dashv$ $\neg$A.

Soundness and Completeness

Let $\Sigma \subseteq$ *Form*($\mathcal{L}$) and A $\in$ *Form*($\mathcal{L}$). Then $\Sigma \models$ A if and only if $\Sigma \vdash$ A. The theorem states that the formal natural deduction for predicate logic is **sound** and **complete**.

Logic 15 First Order Logic Resolution

Prenex Normal Form (PNF)

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

$$Q_1x_1Q_2x_2 \dots Q_nx_nB$$

where $n \geq 1$, Q_i is the universal quantifier $\forall$ or the existential quantifier $\exists$, for $1 \leq i \leq n$, and the expression B is *quantifier free*. The convention is that a formula with no quantifiers ($n = 0$) is regarded as a *trivial case* of a prenex normal form.

Examples of Prenex Normal Form

- $\forall x \forall y \neg(F(x) \rightarrow G(y))$
- $\forall x \exists y H(x, y)$
- $H(u, v)$

Converting to PNF

Any formula in $\mathit{Form}(\mathcal{L})$ is logically equivalent to (and can be converted into) a formula in PNF. The following steps are needed to convert a formula to an equivalent one in PNF

1. Eliminate all $\to$ and $\leftrightarrow$ from the formula
2. Move all negations inward, such that negations on ly appera as part of leterals.
3. Standardize the variables apart
4. Move all quantifiers to the front.

$\exists$-free Prenex Normal Form

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

Skolem Function

- Consider a sentence of the form $\forall x_1 x_2 \dots \forall x_n \exists y A$ where $n \geq 0$, and A is an expression, possibly involving other quantifiers.
- Note that $\exists A$ generates at least one individual for each n-tuple $(a_1, a_2, \dots, a_n)$ in the domain.
- In other words, the individual generated by $\exists y A$ is a *function* of $x_1, \dots, x_n$. It can be expressed by using $f(x_1, x_2, \dots, x_n)$. The function f is called a **Skolem function**.
- The function symbol for a Skolem function is a new function symbol, which must not occur anywhere in A.

Notes on Skolemization

Note that the sentence obtained by using Skolem functions is *not*, in general, logically equivalent to the original sentence.

Dropping the Universal Quantifier

When working with formulas in $\exists$-free prenex normal form (e.g. in resolution for first-order logic), all variables are implicitly considered to be universally quantified).

Valid Argument & Satisfiability

Let Σ be a set of sentences, and A be a sentence. The argument $\Sigma \models \text{A}$ is **valid** if and only if the set

$$C_{\neg A} \cup \left[\bigcup_{B \in \Sigma} C_B \right]$$

is *not satisfiable*. In other words, an argument in first-order logic is valid (the conclusion is a logical consequence of the premises) if and only if the set of clauses consisting of the union of

- the set of clauses obtained from each premise B in Σ, and
- the set of clauses generated by the *negation* of the conclusion A

is *not satisfiable*

Instantiation

An **instantiation** is an assignment to a variable x_i of a quasi-tern t'_i (defined as either an individual symbol, or a vriable symbol, or a function symbol)

Unification

Two formulas in first-order logic are said to **unify** if there are instantiations that make the formulas in question identical. The act of unifying is called **unification**. The instantiation that unifies the formulas in question is called a **unifier**.

Steps of Proving by Resolution

To automatically prove using resolution that a theorem is a valid logical argument, we perform the following steps:

1. Convert each formula to PNF
2. Replace $\exists$ (the existential quantifiers) by Skolem functions.
3. Drop $\forall$ (the universal quantifiers).
4. Convert the quantifier-free sentences to CNF.

If the set of clauses thus obtained is *not satisfiable*, then the theorem is a *valid* logical argument.

Sound and Completeness of Resolution in First-Order Logic

A set S of clauses in first-order logic is **not satisfiable** if and only if there is a resolution that derives the *empty clause,* $\{\}$ (a contradiction) from S. The resolution is *sound* and *complete*, if resolution with input S outputs the empty clause, then the S is not satisfiable (soundness), and if the set S is not satisfiable, then resolution with input S outputs the empty clause (completeness).

Logic 16 Turing Machines

Turing Machine

A **Turing Machine** $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ consists of

- Q, the finite set of *states* of the control unit.
- Σ, the finite set of *input symbols*, also called the alphabet. $\Sigma \subseteq \Gamma$
- Γ, the finite set of *tape symbols.*
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the *transition function*, where L and R stands for the directions left and right.
- $q_0 \in Q$ is the transition function, where L and R stand for the directions left and right.
- B is the *blank symbol.* $B \in \Gamma$, $B \notin \Sigma$.
- $F \subseteq Q$ is the set of *accepting states.*

Result of Computation

If we run a Turing Machine M with input x, the following outcomes are possible.

1. M reaches a state q and symbol a such that $\delta(q, a)$ is undefined. In this case, we say that M **halts** on input x. If $q \in F$, we say that M **accepts** x. Otherwise, we say that M **rejects** x.
2. M attempts to move left from the first (leftmost) cell on the tape. In this case, we say that M crashes on input x.
3. M continues making transitions forever. In this case, we say that M **runs forever** on input x, or M **loops** on input x.

Languages

Let M be a Turing machine. with input alphabet Σ. Suppose that on every input $x \in \Sigma^*$, M with input x halts. Then M **decides** the language of strings over Σ that lead M to accept, that is, the set

$$\{x \in \Sigma^* \mid M \text{ accepts input } x\}$$

If M does not halt on every input, then M does not decide any language at all.

Decidable

A language is **decidable** if there is a Turing machine that decides it. If there is no such Turing machine, the language is **undecidable**.

Examples of Decidable and Undecidable Languages

- The set of all strings is *decidable.*
- The empty set is *decidable.*
- $Form(\mathcal{L}^p)$, the set of propositional formulas, is *decidable.*
- The set of propositional tautologies is *decidable.*
- $Form(\mathcal{L})$, the set of formulas of first-order predicate logic, is *decidable.*
- The set of universally valid formulas of first-order predicate logic is *undecidable.*

Logic 16u Undecidability

Universal Turing Machine

A **universal Turing machine** is a TM that, given as input a discription of a TM <. and an input x for M, performs the actions of M on x, and produces the same result.

Halting Problem

The **Halting Problem** is

$$\text{Given } \langle M\rangle \text{ and } x, \text{ does } M \text{ with input } x \text{ halt?}$$

As a language, this becomes HALT $= \{\langle M\rangle, x \mid \langle M\rangle$ halts on x$\}$.
A machine decides a language if it accepts every member of the language, and rejects every non-member of the language.

Theorem (Turing)

The Halting Problem is undecidable.

Logic 16r Reductions

Reductions

Suppose that P_1 and P_2 are computational problems. A **reduction** from P_1 to P_2 is an algorithm (i.e., a Turing machine) that

1. Always halts, on any input
2. Given an instance x_1 of prblem P_1, produce an instance x_2 of problem P_2, such that in every case, the answer to x_1 (as an instance of P_1) is the same as the answer to x_2 (as an instance of P_2)

For a reduction R, we will denote its output on an instance x_1 by $R(x_1)$. In the case that P_1 and P_2 are formal decision problems (languages), the condition for R to be a reduction is that for all x_1, $x_1 \in P_1$ iff $R(x_1) \in P_2$ and also that R must always halt.

Algorithms via Reductions

Given problems P_1 and P_2 and an algorithm to solve problem P_2, we want to find an algorithm to solve problem P_1. We can create a reduction R from P_1 to P_2. If the reduction R is correct, and the algorithm for P_2 is correct, then the specified algorithm is correct for P_1.

Undecidability via Reductions

Given problems P_1 and P_2 and the fact that problem P_1 is undecidable, we want to show that problem P_2 is undecidable. We can create a reduction R from P_1 to P_2. If the reduction R is correct, an algorithm that decides P_2 would provide an algorithm to decide P_1. Since P_1 is undecidable, this is a contradiction and therefore no algorithm can decide P_2.

Summary on Reduction

Given a reduction from P_1 to P_2, we have

- If P_2 is decidable, then P_1 is decidable.
- If P_1 is undecidable, then P_2 is undecidable.

Note that these two are contrapositives of each other, and the converse statements do not hold.

Logic 16l Undecidability in Logic

Gödel's Incompleteness Theorem

Let Γ be a set of formulas, such that membership in Γ is decidable. Then there are two cases:

1. $PA \cup \Gamma$ is inconsistent, or
2. There is a formula A such that $PA \cup \Gamma \vdash A$ and $PA \cup \Gamma \nvdash \neg A$

In other words, for a given set of premises Γ, one of the following holds.

- Γ is useless, because it contradicts basic facts of arithmetic, or
- Γ is undecidable, which means that given a formula, we cannot determine whether it belongs to Γ or not, or
- There are facts about arithmetic that are not implied by Γ.

Logic 17 Peano Arithmetic

Peano Arithmetic (PA) Axioms

Non-logical symbols:

- constant 0
- functions +, ·, s (successor)
- equality predicate (= or $\approx$)

Axioms

- PA1: $\forall x(s(x) \not\approx 0)$
- PA2: $\forall x \forall y(s(x) \approx s(y)) \to (x \approx y))$
- PA3: $\forall x(x + 0 = x)$
- PA4: $\forall x \forall y(x + s(y) = s(x + y)$.
- PA5: $\forall x(x \cdot 0 = 0)$
- PA6: $\forall x \forall y(x \cdot s(y) = x \cdot y + x)$
- PA7: For each formula A(u) with free variable u,

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

Logic 18 Program Verification

Program Specification

A **program specification** is an (informal or formal) definition of what the program is expected to do.

Hoare Triple

A **Hoare triple** is an assertion ⦇ P ⦈ C ⦇ Q ⦈. The ⦇ P ⦈ is called the **precondition**, the C is called **program** or **code**, and the ⦇ Q ⦈ is called the **postcondition**.

Partial Correctness

A Hoare triple ⦇ P ⦈ C ⦇ Q ⦈ is **satisfied under partial correctness**, if and only if for every state s that satisfies the precondition P, if the execution of the program C starting from state s terminates in state s′, then the state s′ satisfies the post condition Q.

Notes on Partial Correctness

The program

while true { x = 0; }

satisfies all specifications under partial correctness. It is an endless loop and never terminates, but partial correctness only says what must happen *if the program terminates.*

Total Correctness

A Hoare triple $(\!| P |\!)\ C\ (\!| Q |\!)$ is **satisfied under total correctness**, if and only if for every state s that satisfies the precondition P, the execution of the program C starting from state s terminates, and the state s' satisfies the post condition Q.

Logical Variables

Sometimes the pre- and postconditions require additional variables that do not appear in the program. hense are called **logical variables** (or **auxiliary variables**). For a Hoare triple, its set of logical variables consists of those variables that are free in P or Q, and do not occur in C.

Loop Invariants

A **loop invariant** is an assertion (condition) that is true bot *before* and *after* execution of the body of a loop. It needs to be true *before* the while loop begins and true *after* the while loop ends. It expresses a relationship among the variables used within the body of the loop. Some of these variables will have their values changed within the loop. It may or may not be useful in proving termination.

Proving Termination (for Total Correctness)

Only while loops can be responsible for non-termination in our programming language. To prove termination, for each while while loop in the program, identify an integer expression which is *always* non-negative and whose value *decreases* every time through the while loop.

Total Correctness Problem

The **Total Correctness Problem**: "Is a given Hoare triple $(\!| P |\!)\ C\ (\!| Q |\!)$ satisfied under total correctness?" is *undecidable.* Proof: Reduce the Blank Tape Halting Problem.

Partial Correctness Problem

The **Partial Correctness Problem**: "Is a given Hoare triple $(\!| P |\!)\ C\ (\!| Q |\!)$ satisfied under partial correctness?" is *undecidable.* Proof: Reduce the Blank Tape Halting Problem.