Discrete Notes

Propositional Logic

Proposition: a declarative sentence that is either true or false

Examples:

```
The Moon is made of green cheese.
Trenton is the capital of New Jersey.
Toronto is the capital of Canada.
1 + 0 = 1
0 + 0 = 2
```

Examples that are not propositions

```
Sit down!
What time is it?
x + 1 = 2
x + y = z
```

Propositional Variables: p, q, r, s, ...

The proposition that is always true is denoted by **T** and the proposition that is always false is denoted by **F**.

Compound Propositions

- Propositions can be combined in several ways to create **compound propositions**
- This can be made by using **Propositional Formulas**

Name	Symbol	Logic	Definition	Example
Proposition	p	p	A declarative statement	I **will** go shopping today
Negation	¬p	**not** p	Contradiction of a statement	I will **not** go shopping today
Conjunction	p ∩ q	p **and** q	A compound proposition that is true **if and only if all** of its component propositions are true.	I will go to work **and** I will go shopping today
Disjunction	p ∪ q	p **or** q	A compound proposition that is true **if and only if at least one** of a number of alternatives is true.	John is at the library **or** John is studying
Exclusive Or	p ⊕ q	p **xor** q	A compound proposition that is true if and only if at least one of a number of alternatives is true exclusively.	This summer, I am going to London, **or** I am going to Paris
Conditional/Implication	p ⇒ q	**if** p **then** q	An if-then statement in which p is a hypothesis and q is a conclusion.	**If** I do my homework, **then** I get an allowance.
Biconditional	p ⇔ q	p **if and only if** q	True whenever both parts have the same truth value	I will take out the trash **if and only if** you do the dishes

Different Ways of Expressing p ⇒ q
if p, **then** q
if p, q
q **unless** ¬p
q **if** p
q **whenever** p
q **follows from** p
p **implies** q
p **only if** q
q **when** p
p **is sufficient for** q
q **is necessary for** p
a necessary condition for p **is** q
a sufficient condition for q **is** p

Different Ways of Expressing Biconditional p ⇔ q
p **is necessary and sufficient for** q
if p **then** q , and conversely
p **iff** q

More conditional statements related to p ⇒ q ("It raining is a sufficient condition for my not going to town.":

Name	Symbol	Logic	Example
Converse	q ⇒ p	**if** p **then** q	If I do not go to town, then it is raining.
Contrapositive	¬q ⇒ ¬p	**if** not q **then** not p	If it is not raining, then I will go to town.
Inverse	¬p ⇒ ¬q	**if** not p **then** not q	If I go to town, then it is not raining.

- Always first write statements in if then form
- Contrapositive will have the same truth values as p ⇒ q

Truth Tables

- A representation of all the combinations of logical operators can be done through a tool called a "Truth Table" which can tell us the values of the propositions
- Rows: Need a row for every possible combination of values for the atomic propositions.
- Columns:
    - Need a column for the compound proposition (usually at far right)

    - Need a column for the truth value of each expression that occurs in the compound proposition as it is built up.
- Truth Tables:
    - Proposition vs Negation

p	**¬p**
T	F
F	T

    - Conjunction

p	**q**	**p ∩ q**
T	T	T
T	F	F
F	T	F
F	F	F

    - Disjunction

p	**q**	**p ∪ q**
T	T	T
T	F	T
F	T	T
F	F	F

    - Exclusive Or

p	**q**	**p ⊕ q**
T	T	F
T	F	T
F	T	T
F	F	F

    - Conditional

p	**q**	**p ⇒ q**
T	T	T
T	F	F
F	T	T
F	F	T

- Biconditional

p	q	p ⇔ q
T	T	T
T	F	T
F	T	T
F	F	T

Precedence of Logical Operators

Operator	Precedence
¬	1
∩ ∪	2 3
⇒ ⇔	4 5

Compound Propositions

- Rows: Need a row for every possible combination of values for the compound propositions which in turn would be $2^{\text{propositions}}$.
- Columns: Need a column for each proposition variable, need a column for the truth value of each expression that occurs in the compound proposition as it is built up, and need a column for the compound proposition.

Applications of Propositional Logic

Translating English Sentences

1. **Identify atomic propositions**
2. **Determine appropriate logical connectives**

ex. If I go to the store **or** the movies, I **won't** do my homework.

Atomic propositions:

p: I go to the store

q: I go to the movies

r: I do my homework

Logical connectives:

if, or, won't

Representation: (p ∪ q) ⇒ ¬r

ex. The automated reply **can't** be sent **when** the system is full

Atomic propositions:

p: The system is full

q: The automated reply can be sent

Logical connectives:

can't, when

Representation: $p \Rightarrow \neg q$

Translating Propositions

ex.

q: You can ride the rollercoaster

r: You are under 4 feet tall

s: You are older than 16 years old

Translate: $(r \cup \neg s) \Rightarrow \neg q$

Might be helpful to write shell: if r or not s, then not q

If you are under 4 feet tall **or** **not** older than 16 years old **then** you can **not** ride the rollercoaster

Propositional Equivalences

Tautology: A compound proposition that is always true

Contradiction: A compound proposition that is always false

Contingency: A compound proposition that is neither a tautology nor a contradiction

Logical Equivalences

Equivalence	Name
$p \wedge T \equiv p$ $p \vee F \equiv p$	Identity laws
$p \vee T \equiv T$ $p \wedge F \equiv F$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan’s laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv T$ $p \wedge \neg p \equiv F$	Negation laws

Logical Equivalences Involving Conditional Statements	Logical Equivalences Involving Biconditional Statements.
$p \rightarrow q \equiv \neg p \vee q$	$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$
$p \rightarrow q \equiv \neg q \rightarrow \neg p$	$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
$p \vee q \equiv \neg p \rightarrow q$	$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$p \wedge q \equiv \neg(p \rightarrow \neg q)$	$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$
$\neg(p \rightarrow q) \equiv p \wedge \neg q$	
$(p \rightarrow q) \wedge (p \rightarrow r) \equiv p \rightarrow (q \wedge r)$	
$(p \rightarrow r) \wedge (q \rightarrow r) \equiv (p \vee q) \rightarrow r$	
$(p \rightarrow q) \vee (p \rightarrow r) \equiv p \rightarrow (q \vee r)$	
$(p \rightarrow r) \vee (q \rightarrow r) \equiv (p \wedge q) \rightarrow r$	

Examples

1. $\neg(p \vee (\neg p \wedge q)) \equiv \neg p \wedge \neg q$

Original Statement	Result	Reasoning
¬(p ∨ (¬p ∧ q))	¬p ∧ **¬**(¬p ∧ q)	2nd De Morgan law
	¬p ∧ [**¬**(**¬**p) ∨ ¬q]	1st De Morgan Law
	¬p ∧ (p ∨ ¬q)	Double Negation Law
	(**¬p** ∧ **p**) ∨ (¬p ∧ ¬q)	Second Distributive Law
	F ∨ **(¬p** ∧ **¬q)**	2nd Negation Law
	(¬p ∧ **¬q)** ∨ **F**	Commutative Law
	¬p ∧ ¬q	Identity Law

bolded portions are the places where changes changes were made

2. (p ∧ q) → (p ∨ q) is a tautology or (p ∧ q) → (p ∨ q) ≡ T

Original Statement	Result	Reasoning
(p ∧ q) → (p ∨ q	¬(p ∧ q) ∨ (p ∨ q)	p → q ≡ ¬p ∨ q
	(¬p ∨ ¬q) ∨ (p ∨ q)	1st De Morgan Law
	(¬p ∨ p) ∨ (¬q ∨ q)	Commutative Law + Associative law
	T ∨ T	2nd Negation Law
	T	Domination law

3. ¬(¬p ∨ q) ≡ ¬q ∧ p)

Original Statement	Result	Reasoning
¬(¬p ∨ q)	¬(¬p) ∧ ¬q	1st De Morgan Law
	p ∧ ¬q	Double Negation Law
	¬q ∧ p)	Commutative Law

Predicates and Quantifiers

Predicate Logic Components

- **Variables: x,y,z** these are the subjects of the statement(s)
- **Predicates:** A property the variable can have
  - ex. "x < 2" and "x + y = z"
  - **Propositional Function**: P(x), where P is the predicate and x is the variable
- **Quantifiers**

Common Number Sets

Number Set	Description
N	{0,1,2,3,...}, the set of **natural numbers**
Z	{...,−2,−1,0,1,2,...}, the set of **integers**
$\mathbf{Z^+}$	{1,2,3,...}, the set of **positive integers**
Q	{p/q | p ∈ **Z**, q ∈ **Z**, and q != 0}, the set of **rational numbers**
R	the set of **real numbers**
$\mathbf{R^+}$	the set of **positive real numbers**

Quantifiers

- Universal Quantifier ∀
  - ∀xP(x) tells us that the proposition P(x) must be true for **all** values of x in the domain of the discourse/universe
- Existential Quantifier ∃
  - ∃xP(x) tells us that the proposition P(x) must be true for **some** value(s) of x in the domain of the discourse/universe

Statement	When True?	When False?
∀xP(x)	P(x) is true for every x.	There is an x for which P(x) is false.
∃xP(x)	There is an x for which P(x) is true.	P(x) is false for every x

Translating and Negating with Quantifiers

De Morgan's Laws for Quantifiers

Negation	Equivalent Statement
¬∃xP(x)	∀x¬P(x)
¬∀xP(x)	∃x¬P(x)

ex. "There is an honest politician" and "All Americans eat cheeseburgers"

Let H(x) represent "x is honest" for domain of all politicians

Therefore ∃xH(x)

Negation: ¬∃xH(x) = ∀x¬H(x) = Every politician is dishonest

Let C(x) represent "x eat cheeseburgers" for domain of all Americans

Therefore: ∀xC(x)

Negation: ¬∀xC(x) = ∃x¬C(x) = Not every American eats cheeseburgers

ex. "Every student in this class has visited Canada or Mexico"

Domain: Students in this class

Let M(x) represent "x has visited Mexico"

Let C(x) represent "x has visited Canada"

Therefore ∃x(C(x) ∨ M(x))

Domain: All people

Let M(x) represent "x has visited Mexico"

Let C(x) represent "x has visited Canada"

Let S(x) represent "x is a student in this class"

Therefore ∀x(S(x) → (C(x) ∨ M(x)))

Nested Quantifiers

ex. Let Q(x,y) denote “x + y = 5.” Domain is all real numbers.

- ∃y∀xQ(x,y)
  - For all real numbers x there exists a real number y such that x + y = 5
  - TRUE
- ∀x∃yQ(x,y)
  - There exists a real number y such that for all real numbers x, x + y = 5
  - FALSE, y = 0 , x = 2, 2 + 0 != 5

Quantification of Two Variables

Statement	When True?	When False?
∀x∀yP(x,y) ∀y∀xP(x,y)	P(x,y) is true for every pair x,y	There is a pair x,y for which P(x,y) is false.
∀x∃yP(x,y)	For every x there is a y for which P(x,y) is true.	There is an x such that P(x,y) is false for every y.
∃x∀yP(x,y)	There is an x for which P(x,y) is true for every y.	For every x there is a y for which P(x,y) is false.
∃x∃yP(x,y) ∃y∃xP(x,y)	There is a pair x,y for which P(x,y) is true.	P(x,y) is false for every pair x,y.

Negation

ex. Let P(x,y) denote (x = -y). Find the negation of ∀x∃yP(x,y)

- No negation to precede a quantifier

¬(∀x∃yP(x,y)) = ∃x¬(∃yP(x,y)) = ∃x∀y¬P(x,y) = There exists a real number x such that for all real numbers y, x != -y

Translation

ex. Translate "The sum of two positive integers is always positive"

- For all positive integers x and y, x+y > 0
- Let P(x,y) denote x+y > 0
- ∀x∀y(P(x,y)), domain Z^+

ex. Let E(x,y) denote "x sent y an email" and T(x,y) denote "x sent y a text." Domain: students in class

a. Every student in the class sent an email to Joe

- ∀xE((x != Joe) → E(x, Joe))

b. There is a student in class who has not received a text or email from any other student in class

- ∃y∀x((x != y) → ¬(E(x,y) ∨ T(x,y))) = ∃y∀x((x != y) → (¬E(x,y) ∧ ¬T(x,y)))

ex. There is a man that has taken a flight on every airline in the world

- Let P(x, y) denote x has taken flight y
- Let Q(y, z) denote y is a flight on airline z
- ∃m∀a∃f(P(m, f) ∧ Q(f, a)) | m = man, a = airline, and f = flight

Rules of Inference

Argument: a sequence of propositions p_1, p_2 … p_n

- All but the final proposition in the argument are called **premises**
- The final proposition is called the **conclusion**
- An argument is valid if the truth of all its premises implies that the conclusion is true.
- p_1, p_2 … p_n and conclusion q is valid, when $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow q$ is a tautology.

Rule of Inference	Tautology	Name
p $p \to q$ $\therefore q$	$(p \wedge (p \to q)) \to q$	Modus Ponens
$\neg q$ $p \to q$ $\therefore \neg p$	$(\neg q \wedge (p \to q)) \to \neg p$ $\equiv (\neg q \wedge (\neg q \to \neg p)) \to \neg p$	Modus Tollens
$p \to q$ $q \to r$ $\therefore p \to r$	$((p \to q) \wedge (q \to r)) \to (p \to r)$	Hypothetical Syllogism (Transitive Property)
$p \vee q$ $\neg p$ $\therefore q$	$((p \vee q) \wedge \neg p) \to q$ * p or q occurred but p didn't occur implies that q occurred	Disjunctive Syllogism
p $\therefore p \vee q$	$p \to (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \to p$ $(p \wedge q) \to q$	Simplification
p q $\therefore p \wedge q$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$ * if p = T then ¬p = F, which makes q = T and r can be T or F * if p = F then ¬p = T, which makes q = T or F and r = T	Resolution

ex. From the proposition $p \wedge (p \to q)$, show that q is a conclusion

Statement	Reason
$p \wedge (p \to q)$	Premise
P	Simplification on 1: $(p \wedge q) \to p$
$p \to q$	Simplification on 2: $(p \wedge q) \to q$
q	Modus Ponens on 2 and 3: $(p \wedge (p \to q)) \to q$

ex. From the propositions p, $p \to \neg q$, and $\neg q \to \neg r$, show that $\neg r$ is a conclusion

Statement	Reason
p	Premise
$p \rightarrow \neg q$	Premise
$\neg q$	Modus Ponens on 1 and 2: $(p \wedge (p \rightarrow q)) \rightarrow q$
$\neg q \rightarrow \neg r$	Premise
$\neg r$	Modus Ponens on 3 and 4: $(p \wedge (p \rightarrow q)) \rightarrow q$

Proofs

Useful Definitions

- **Even integer:** The integer n is even if there exists an integer k such that n = 2k
- **Odd integer:** The integer n is odd if there exists an integer k such that n = 2k + 1
- **Rational number**: The real number r is rational if there exist integers p and q with q != 0 such that r = p/q.

Direct Proof

- In a direct proof, we assume the antecedent is true, then use rules of inference, axioms, definitions and/or previously proven theorems to show the consequent is true.

ex. If n is an odd integer, the n^2 is odd

- **Assume** n is an odd integer
  - Then n = 2k+1, $k \in Z$ by definition of an odd integer
  - Then square both sides: $(n)^2 = (2k+1)^2 \rightarrow n^2 = 4k^2 + 4k + 1$
  - $n^2 = 2(2k^2 + 2k) + 1$
  - $n^2 = 2r + 1$ where $r = 2k^2 + 2k$, $r \in Z$
  - Therefore n^2 is odd

ex. The sum of two even integers is even

- If I add two even integers, then the sum is even
- **Assume** a and b are even integers
  - then a = 2k, $k \in Z$ and b = 2m, $m \in Z$ by definition of an even integer
  - a + b = 2k + 2m
  - a + b = 2(k + m)
  - a + b = 2r where r = k + m, $r \in Z$

Proof by Contraposition

- A type of indirect proof that makes use of the fact that $p \rightarrow q$ is equivalent to its contrapositive $\neg q \rightarrow \neg p$. So we assume $\neg q$ is true, then work to prove $\neg p$ is true.

ex. If n is an integer and 3n+2 is odd, then n is odd

- $p \rightarrow q \equiv \neg q \rightarrow \neg p$
- Assume n is even. So n = 2k, k ∈ Z by definition of an even integer
    - 3n+2 = 3(2k)+2
    - 3n+2 = 6k+2
    - 3n+2 = 2(3k+1)
    - 3n+2 = 2r where r = 3k + 1, r ∈ Z
    - Therefore 3n + 2 is even when n is odd and then by contraposition 3n + 2 is odd when n is odd

Sets

- **Set**: unordered collection of objects
- **Elements**: An object in the set
- a ∈ A to denote that a is an element of the set A.
- a∉ A denotes that a is not an element of the set A

Important Sets

Number Set	Description
N	{0,1,2,3,...}, the set of **natural numbers**
Z	{...,−2,−1,0,1,2,...}, the set of **integers**
$\mathbf{Z^+}$	{1,2,3,...}, the set of **positive integers**
Q	{p/q | p ∈ **Z**, q ∈ **Z**, and q != 0}, the set of **rational numbers**
R	the set of **real numbers**
$\mathbf{R^+}$	the set of **positive real numbers**

Set Notation

Roster Notation: Used with discrete sets (countable)

- S = {1, 2, 3, 4, 5}
- Not roster notation: 0 <= x <= 1

Set-Builder Notation: Can be used for any type of set

- S = {x | x ∈ N, 1 <= x <= 5}
- | = "such that"

Interval Notation: Used with continuous sets

- B = {x |0 <= x <= 1} = **[0, 1]**

Universal Set: The set of all elements under consideration

Empty Set: { } or ∅

Set Relationships

Set Equality

- Sets are equal if and only if they have the same elements
- ∀x(x ∈ A ↔ x ∈ B, Notation: A = B if A and B are equal sets

Subsets

- The set A is a subset of B if and only if every element of A is also an element of B.
- ∀x(x ∈ A → x ∈ B), Notation: A ⊆ B
    - **Showing that A is a Subset of B:** To show that A ⊆ B, show that if x belongs to A then x also belongs to B.
    - **Showing that A is Not a Subset of B**: To show that A ⊈ B, find a single x ∈ A such that x ∉ B.
    - **Showing Two Sets are Equal:** To show that two sets A and B are equal, show that A ⊆ B and B ⊆ A.
- **Proper subset:** When we wish to emphasize that a set A is a subset of a set B but that A != B, we write A ⊂ B

Cardinality

- The number of distinct elements of a set
- A = {1, 2, 3, 3, 3, 4}, Cardinality Notation: |A| = 4

Power Sets

- The set of all subsets of a set
- A = {0, 1, 2}, Power Set Notation: P(A) = {∅, {0}, {1}, {2}, {0, 1}, {0, 2}, {1, 2}, {0, 1, 2}}
- Cardinality of a power set of a set with n elements is 2^n

Cartesian Product

- The set of ordered pair (a, b) where a ∈ A and b ∈ B, resulting from A x B
- Notation: A × B = {(a,b) | a ∈ A ∧ b ∈ B}
- A = {0, 1}, B = {a, b, c}
    - A x B = {(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)}
    - B x A = {(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)}

Truth Set

- A truth set of P is the set of elements in x in D such that P(x) is true
- Notation: {x ∈ D | P(x)}

Set Operations

Union of Sets

- Let A and B be sets. The union of the sets A and B, denoted by A ∪ B, is the set that contains those elements **that are either in A or in B, or in both.**
- **A ∪ B** = {x | x ∈ A ∨ x ∈ B}

Intersection of Sets

- Let A and B be sets. The intersection of the sets A and B, denoted by A ∩ B, is the set containing those elements **in both A and B.**
- **A ∩ B** = {x | x ∈ A ∧ x ∈ B}
- If A ∩ B = ∅ then A and B are said to be disjoint

Complement

- Let A be a set. The complement of set A with respect to U is U - A.
- ¬A = {x | x ∈ U | x !∈ A}

Set Identities

Identity	Name
A ∩ U = A A ∪ ∅ = A	Identity laws
A ∪ U = U A ∩ ∅ = ∅	Domination laws
A ∪ A = A A ∩ A = A	Idempotent laws
¬(A) = A	Complementation law
A ∪ B = B ∪ A A ∩ B = B ∩ A	Commutative laws
A ∪ (B ∪ C) = (A ∪ B) ∪ C A ∩ (B ∩ C) = (A ∩ B) ∩ C	Associative laws
A ∪ (B ∩ C) = (A ∪ B) ∩ (A ∪ C) A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)	Distributive laws
¬(A ∩ B) = ¬A ∪ ¬B ¬(A ∪ B) = ¬A ∩ ¬B	De Morgan’s laws
A ∪ (A ∩ B) = A A ∩ (A ∪ B) = A	Absorption laws
A ∪ ¬A = U A ∩ ¬A = ∅	Complement laws

Proving Set Identities

- Prove each set in the identity is a subset of the other

ex. ¬(A ∩ B) = ¬A ∪ ¬B

- Need to show that ¬(A ∩ B) ⊆ ¬A ∪ ¬B and that ¬A ∪ ¬B ⊆ ¬(A ∩ B)

Steps for ¬(A ∩ B) ⊆ ¬A ∪ ¬B	**Reasoning**
x ∈ ¬(A ∩ B)	By assumption
x !∈ A ∩ B	Definition of complement
¬((x ∈ A) ∧ (x ∈ B))	Definition of intersection
¬(x ∈ A) ∨ ¬(x ∈ B)	De Morgan's Law for Propositional Logic
x !∈ A ∨ x !∈ B	Definition of a negation
x ∈ ¬A ∨ x ∈ ¬B	Definition of complement
x ∈ ¬A ∪ ¬B	Definition of a union

Steps for ¬A ∪ ¬B ⊆ ¬(A ∩ B)	**Reasoning**
x ∈ ¬A ∪ ¬B	By assumption
x ∈ ¬A ∨ x ∈ ¬B	Definition of a union
x !∈ A ∨ x !∈ B	Definition of complement
¬(x ∈ A) ∨ ¬(x ∈ B)	Definition of a negation
¬((x ∈ A) ∧ (x ∈ B))	De Morgan's Law for Propositional Logic
¬(x ∈ A ∩ B)	Definition of intersection
x ∈ ¬(A ∩ B)	Definition of complement

Sequences

- An ordered list of elements created by a function mapping the integers to a set S
- Notation: a_n represents the nth term

Arithmetic Sequence

- A sequence formed by adding the initial term, a, and the product of the common difference, d, and the term number, n.
- $\{a_n\} = a, a + d, a + 2d, \ldots, a + nd$ | $\mathbf{a_n = a + dn}$

ex. Let a = 5 and d = 2. Find the first 5 terms of $\{a_n\}$

$a_n = 5 + 2n$: {5, 7, 9, 11, 13}; $a_4 = 5+2(4) = 13$

ex. Find a and d for the sequence {7, 4, 1, -2, ...}

$a_n = a + dn$

$a_n = 7 + dn$; $a_1-a_0 = 4 - 7 = -3 = d$

$a_n = 7 - 3n$

Geometric Sequence

- A sequence formed by multiplying the initial term, a, by the common ratio to the nth power, r^n
- $\{a_n\} = a, ar, ar^2, \ldots, ar^n$ | $\mathbf{a_n = ar^n}$

ex. Let a = 4 and r = 3. Find the first 5 terms of the geometric progression.

$a_n = 4(3)^n = \{4, 12, 36, 108, 324\}$; $a_3 = 4(3)^3 = 36$

ex. Find a and r for $\{a_n\} = \{3, 3/2, 3/4, 3/8\}$

$a_n = ar^n$

$a_n = 3r^n$; $a_1/a_0 = (3/2)/3 = 1/2 = r$

$a_n = 3(1/2)^n$

Recurrence Relation

- **An equation** that expresses a_n in terms of one ore more of the previous terms of the sequence. **Initial conditions** are required to specify terms that preceded the first term where the relation takes effect.

ex. $a_n = a_{n-1} + 2a_{n-2}$ for n >= 2 where $a_0 = 2$ and $a_1 = 5$

$a_2 = a_1 + 2a_0 = 5 + 2(2) = 9$

$a_3 = a_2 + 2a_1 = 9 + 5(2) = 19$

ex. Let a_n be the sequence that satisfies the recurrence relation $a_n = a_{n-1} + 6$ for n >=1

a. If $a_0 = 3$, find a_1, a_2, a_3

$a_1 = 3 + 6 = 9$

$a_2 = 9 + 6 = 15$

$a_3 = 15 + 6 = 21$

$a_n = a + dn$: $a_n = 3 + 6n$

ex. Fibonacci Sequence: 0, 1, 1, 2, 3,4, 8, 13, 21, 34... Express the recurrence relation.

$f_0 = 0$

$f_1 = 1$

$f_n = f_{n-1} + f_{n-2}$ for n >= 2

ex. Let a_n be the sequence that satisfies the recurrence relation $a_n = a_{n-1} + 3$ for n>=1 with $a_0 = 2$

a = 2, d = 3 therefore $a_n = 2 + 3n$

Some Useful Sequences

nth Term	First 10 Terms
n^2	1,4,9,16,25,36,49,64,81,100,...
n^3	1,8,27,64,125,216,343,512,729,1000,...
n^4	1,16,81,256,625,1296,2401,4096,6561,10000,...
2^n	2,4,8,16,32,64,128,256,512,1024,...
3^n	3,9,27,81,243,729,2187,6561,19683,59049,...
n!	1,2,6,24,120,720,5040,40320,362880,3628800,...
f_n	1,1,2,3,5,8,13,21,34,55,89,...

Summation and Mathematical Induction

Sigma Notation

- To express the sum of the terms of the sequence $a_n = \{a_m, a_{m+1}, ..., a_n\}$, we write:

  $$\sum_{i=m}^{n} a_i = \sum_{i=m}^{n} a_i = \sum_{m \le i \le n} a_i$$

  which represents $a_m + a_{m+1} + a_{m+2} + a_n$

ex. Use sigma notation to express the sum of the first 100 terms of the sequence a_i where $a_i = 1/i$ for i = 1, 2, ...

$$\sum_{i=m}^{n} a_i = \sum_{i=1}^{100} 1/i$$

ex. What is the value of $\sum_{i=5}^{9} i^2$

= 25 + 36 + 49 + 64 + 81 = 255

Summation Properties

- $\sum_{k=1}^{n} ca_k = c\sum_{k=1}^{n} a_k$
- $\sum_{k=1}^{n} c = cn$
- $\sum_{k=1}^{n} (a_k \pm b_k) = \sum_{k=1}^{n} a_k \pm \sum_{k=1}^{n} b_k$
- $\sum_{k=1}^{n} a_k = \sum_{k=1}^{j} a_k + \sum_{k=j+1}^{n} a_k$ for $i \le j < j+1 < n$

Summation Formulas

- $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
- $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4}$
- $\sum_{k=1}^{\infty} x^k = \frac{1}{1-x}$
- $\sum_{k=1}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2}$
- ex. $\sum_{k=n}^{2n} k = \sum_{k=1}^{2n} k - \sum_{k=1}^{n-1} k$
- Geometric progression: $\sum_{k=0}^{n} ar^k = \frac{ar^{n+1} - a}{r-1}$

Mathematical Induction

Proving Equalities

- To prove P(x) is true for x ∈ Z^+, where P(x) is a propositional function, we complete two steps:
  1. **Basis step** - verify P(1) is true
  2. **Inductive step** - verify if P(k) is true, then P(k + 1) is true ∀k ∈ Z^+
     - Inductive Hypothesis: P(k) is true
     - Must show: P(k) → P(k+1)
  3. **Conclusion**: P(x) is true ∀k ∈ Z^+

ex. $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$, show 1 + 2 + 3 + ... + n = $\frac{n(n+1)}{2}$

Let P(n): 1 + 2 + 3 + ... + n = $\frac{n(n+1)}{2}$

Basis Step:

P(1): n = 1

1 = $\frac{1(1+1)}{2}$ = 1

Inductive Step: P(k) → P(k+1)

Inductive Hypothesis: P(k) = 1 + 2 + 3 + ... + k = $\frac{k(k+1)}{2}$

Must Show P(k+1) = 1 + 2 + 3 + ... + k + (k+1) = $\frac{(k+1)((k+1)+1)}{2} = \frac{(k+1)(k+2)}{2}$

LHS	RHS	Steps
1 + 2 + 3 + ... + k +(k+1) =	$\frac{k(k+1)}{2}$+(k+1)	Add (k+1) to both sides of P(k)
=	$\frac{k(k+1)}{2} + \frac{2(k+1)}{2}$	Get common denominator
=	$\frac{k(k+1)+2(k+1)}{2}$	Get common denominator
=	$\frac{(k+1)(k+2)}{2}$	Factor by grouping
	$\frac{(k+1)(k+2)}{2}$	= Result of P(k+1)

ex. Use mathematical induction to show that for all non-neg integers n, $1 + 2 + 2^2 + ... + 2^n = 2^{n+1} - 1$

non-negative integers means integers $n \geq 0$

Let P(n): $1 + 2 + 2^2 + ... + 2^n = 2^{n+1} - 1$

Basis Step:

P(0): $n = 2^0$

$2^0 = 1 = 2^{0+1} - 1 = 2 - 1 = 1$

Inductive Step: P(k) → P(k+1)

Inductive Hypothesis: $1 + 2 + 2^2 + ... + 2^k = 2^{k+1} - 1$

Must Show P(k+1): $1 + 2 + 2^2 + ... + 2^k + 2^{k+1} = 2^{(k+1)+1} - 1 = 2^{k+2} - 1$

LHS	RHS	Steps
$1 + 2 + 2^2 + ... + 2^k + 2^{k+1} =$	$2^{k+1} - 1 + 2^{k+1}$	Add 2^{k+1} to both sides of P(k)
=	$2(2^{k+1})-1$	Combine like terms
=	$2^1(2^{k+1})-1$	Properties of exponents
=	$2^{k+2}-1$	Properties of exponents
	$2^{k+2}-1$	= Result of P(k+1), P(n) is true for all $n \geq 0$, n ∈ Z

ex. Prove $1 + 3 + 5 + ... + (2n - 1) = n^2$ for first n positive odd integers

Let P(n): $1 + 3 + 5 + ... + (2n - 1) = n^2$

Basis Step:

P(1) = 2n-1

$2(1) - 1 = 1 = 1^2 = 1$

Inductive Step: P(k) → P(k+1)

Inductive Hypothesis: $1 + 3 + 5 + ... + (2k - 1) = k^2$

Must Show P(k+1): $1 + 3 + 5 + ... + (2k - 1) + (2k+1) = (k+1)^2$; (2k+1) as that 2 more (odd integer)

LHS	RHS	Steps
1 + 3 + 5 + ... + (2k - 1) + (2k+1) =	$k^2 + (2k+1)$	Add (2k+1) to both sides of P(k)
=	$k^2 + 2k + 1$	Expand and organize the terms
=	$(k+1)^2$	Factor
	$(k+1)^2$	= Result of P(k+1)

Proving Inequalities

ex. Prove $n < 2^n$ $\forall n \in Z^+$ using mathematical induction

Let P(n): $n < 2^n$

Basis Step:

P(1) = n

$1 < 2^1 \equiv 1 < 2$

Inductive Step: P(k) $\rightarrow$ P(k+1)

Inductive Hypothesis: $k < 2^k$

Must Show P(k+1): $k+1 < 2^{k+1}$

LHS	RHS	Steps
k +1 <	$2^k + 1$	Add 1 to both sides of P(k)
<	$2^k + 2^k$	As P(1) showed that $1 < 2^k$ $\forall n \in Z^+$ so replace 1 with 2^k
<	$2(2^k)$	Factor
<	$2^1(2^k)$	Properties of exponents
<	2^{k+1}	Properties of exponents
	2^{k+1}	= Result of P(k+1)

ex. Prove $2^n < n!$ $\forall n \in Z^+$ and $n \geq 4$

Let P(n): $2^n < n!$

Basis Step:

P(4) = 2^n

$2^4 < 4! \equiv 16 < 4 * 3 * 2 * 1 \equiv 16 < 24$

Inductive Step: P(k) $\rightarrow$ P(k+1)

Inductive Hypothesis: $2^k < k!$

Must Show P(k+1): $2^{k+1} < (k+1)!$

LHS	RHS	Steps
$2^k * 2^1 <$	$k! * 2^1$	Multiply 2^1 to both sides of P(k)
$2^{k+1} <$	$k! * (k+1)$	(k+1) > 2 so replace 2 with (k+1)
<	k(k+1)!	Formatting
<	(k+1)!	k! = k * k-1 ..., so (k+1)! = (k+1) * k * (k-1) ...
	(k+1)!	= Result of P(k+1)

ex. Prove $3^n > 2n$ for all integer $n \geq 1$

Let P(n): $3^n > 2n$

Basis Step:

P(1) = 3^n

$3^1 > 2(1) \equiv 3 > 2$

Inductive Step: P(k) $\rightarrow$ P(k+1)

Inductive Hypothesis: $3^k > 2k$

Must Show P(k+1): $3^{k+1} > 2(k+1)$

LHS	RHS	Steps
$3^k * 3^1 >$	$2k * 3^1$	Multiply 3^1 to both sides of P(k)
$3^{k+1}>$	6k	Simplification
>	2(k+1)	Since 6k > 2(k+1) for $k \geq 1$
>	2(k+1)	= Result of P(k+1)

Note: $6k > 2(k+2) \equiv 6k > 2k + 2 \equiv k > \frac{1}{2}$

ex. Prove $1 + \frac{1}{4} + \frac{1}{9} + \ldots + \frac{1}{n^2} \leq 2 - \frac{1}{n}$ for integer $n \geq 1$

Let P(n): $1 + \frac{1}{4} + \frac{1}{9} + \ldots + \frac{1}{n^2} \leq 2 - \frac{1}{n}$

Basis Step:

P(1) = $\frac{1}{n^2}$

$\frac{1}{1^2} \leq 2 - \frac{1}{n} \equiv 1 \leq 2 - 1 \equiv 1 \leq 1$

Inductive Step: P(k) $\rightarrow$ P(k+1)

Inductive Hypothesis: $1 + \frac{1}{4} + \frac{1}{9} + \ldots + \frac{1}{k^2} \leq 2 - \frac{1}{k}$

Must Show P(k+1): $1 + \frac{1}{4} + \frac{1}{9} + \ldots + \frac{1}{k^2} + \frac{1}{(k+1)^2} \leq 2 - \frac{1}{k+1}$

LHS	RHS	Steps
$1+\frac{1}{4}+\frac{1}{9}+\ldots+\frac{1}{k^2}+\frac{1}{(k+1)^2} \leq$	$2-\frac{1}{k}+\frac{1}{(k+1)^2}$	Add $\frac{1}{(k+1)^2}$ to both sides of P(k)
$\leq$	$2-\frac{1}{k+1}$	Since $2-\frac{1}{k}+\frac{1}{(k+1)^2} \leq 2-\frac{1}{k+1}$ for $k \geq 1$ *
	$2-\frac{1}{k+1}$	= Result of P(k+1)

Note: $2-\frac{1}{k}+\frac{1}{(k+1)^2} \leq 2-\frac{1}{k+1} \equiv \frac{1}{(k+1)^2} \leq \frac{1}{k(k+1)}$

Strong Induction and Well-Ordering

The Well-Ordering Principle

- Every non-empty set of non-negative integers has a least element.
  - If Z^+ then least element will be n = 1
  - If non-negative integers then least element will be n = 0

Using Induction

ex. Prove that every amount of postage of 0.12 dollars or more can be formed using 0.04 dollars and 0.05 dollars stamps.

Let P(n) be the statement that postage of n-cents can be formed using 4-cent and 5-cent stamps if $n \geq 12$

Basis Step: Show postage of 12-cents can be made

12-cents = 3(4)-cent stamps

Inductive Step: Show if P(k) is true, then P(k+1) is true for $k \geq 12$

Inductive Hypothesis: We can have postage of k-cents using 4- and 5-cent stamps

Must Show P(k+1): I can make postage of k+1 cents using 4- and 5-cent stamps

Case 1: I have used one or more 4-cent stamps

If I have used a 4-cent stamp for k-cent postage, then I can replace my 4-cent stamp with a 5-cent stamp. ∴ k+1 postage was formed.

Case 2: I have NOT used a 4-cent stamp

If no 4-cent stamp was used, then I have used at least 3 5-cent stamps because $n \geq 12$, so I can replace 3 5-cent stamps with 4 4-cent stamps ∴ k+1 postage was formed.

Using Strong Induction

ex. Prove that every amount of postage of 0.12 dollars or more can be formed using 0.04 dollars and 0.05 dollars stamps.

Let P(n) be the statement that postage of n-cents can be formed using 4-cent and 5-cent stamps if $n \geq 12$

Basis Step: Show postage of 12, 13, 14, and 15-cents can be made

12-cents = 3(4)-cent stamps

13-cents = 2(4)-cent stamps + 1(5)-cent stamp

14-cents = 1(4)-cent stamps + 2(5)-cent stamps

15-cents = 3(5)-cent stamps

Inductive Step: Show if P(j) is true, for $12 \leq j \leq k$, where $k \geq 15$, then P(k+1) is true.

Inductive Hypothesis: P(k-3) is true

Must Show P(k+1):

If I can make postage of k-3 cents, then I can make postage of k+1 cents by adding a 4-cent stamp

Recursive Definitions and Structural Induction

Well Formed Formulae Rules

1. A well-formed formulae is proposition that has one clear definition of the logic it is trying to prove.
2. Any proposition is a well formed formulae: P, Q, R
3. If α is a well formed formulae:
   - $\neg\alpha$ is a well formed formulae: ¬(P), ¬(P ∧ Q)
4. if α and β are well formed formulae:
   - ($\alpha \wedge \beta$) is a well formed formulae
   - ($\alpha \vee \beta$) is a well formed formulae
   - $\alpha \rightarrow \beta$ is a well formed formulae
   - $\alpha \leftrightarrow \beta$ is a well formed formulae

ex. Is the following a well formed formula: (p → (q ∨ T)) → q ∨ r

No, as there is an absence of parentheses around q ∨ r so there is no clear definition of the logic

Recursively Defined Functions

Basis Step: Specifies the value of the function for the first terms(s)

- Initial conditions

Recursive Step: Gives a rule for finding subsequent values using a previous value(s) beginning at those defined in the basis step.

- Function

ex. If f is defined recursively by $f(0) = 2$ and $f(n+1) = 3f(n) - 1$, find $f(1)$, $f(2)$, $f(3)$ and $f(4)$

- $f(n+1) = 3f(n) - 1 \equiv f(n) = 3(f(n-1)) - 1$

  Basis Step: $f(0) = 2$

  Recursive Step: $f(n) = 3(f(n-1)) - 1$

  $f(1) = 3f(0) - 1 = 3(2) - 1 = 5$

  $f(2) = 3f(1) - 1 = 3(5) - 1 = 14$

  $f(3) = 3f(2) - 1 = 3(14) - 1 = 41$

  $f(4) = 3f(3) - 1 = 3(41) - 1 = 122$

ex. Give a recursive definition for a^n for $a \in \mathbb{R}$ and non-negative and $n \in \mathbb{Z}^+$

$a^0 = 1$

$a^1 = 1 * a$

$a^2 = a * a$

$a^3 = a^2 * a$

$a^4 = a^3 * a$

$\therefore$

Initial conditions(s): $a^0 = 1$

Recursive Function: $a^n = a^{n-1} * a \; for \; n \geq 1$

ex. Give a recursive definition of the sequence $\{a_n\}, n = 1, 2, 3 \ldots$ if $a_n = 2n + 1$

$a^0 = 2(0) + 1 = 0 + 1 = 1$

$a^1 = 2(1) + 1 = 2 + 1 = 3$

$a^2 = 2(2) + 1 = 4 + 1 = 5$

$a^3 = 2(3) + 1 = 6 + 1 = 7$

$a^4 = 2(4) + 1 = 8 + 1 = 9$

$\therefore$

Initial conditions(s): $a_0 = 1$

Recursive Function: $a_n = a_{n-1} + 2$ for $n \geq 1$

Recursively Defined Sets

Basis Step: Specifies an initial collection of elements

Recursive Step: Gives rules for forming new elements from those already in the set

ex. Let S be a subset of the integers defined recursively by:

Basis Step: $7 \in S$

Recursive Step: If $x \in S$ and $y \in S$, then $x + y \in S$

List the elements of S produced by the first 5 applications of the recursive definition

```
1. 7
2. 7+7 = 14
3. 7+14 = 21
4. 7+21 = 28
5. 7+28 = 35
```

Mathematical Induction

Let A be the set of all integers divisible by 7 $\therefore$ A = 7n. To prove A = S, we must show $A \subseteq S$ and $S \subseteq A$

$A \subseteq S$ Proof:

Let P(n) be the statement $7n \in S$ (Derived from A = 7n. To prove A = S).

Basis Step: $n = 1 \therefore 7 * 1 = 7 \in S$

Inductive Step: P(k) $\rightarrow$ P(k+1)

Inductive Hypothesis: $7k \in S$

Must Show P(k+1):

If $7k \in S$ then $7k + 7 \in S$ since $7, 7k \in S$

$\therefore 7k + 7 = 7(k + 1) \in S$ and $A \subseteq S$

$S \subseteq A$ Proof:

Basis Step: $A = 7n \therefore 7 * 1 \in A = 7 \in A$

From the recursive step $x + y \in S$, and since $7/x$ and $7/y$, then $x = 7a$ and $y = 7b$ for some integers a and b.

$x + y = 7a + 7b = 7(a + b) \therefore$ using $A = 7n, x + y \in A$

Structural Induction

Basis Step: Show that the result holds for all elements specified in the basis step of the recursive definition.

Recursive Step: Show that if the statement is true for all elements used to construct new elements in the recursive step of the definition, the result hold for these new elements.

Recursive Algorithms

- An algorithm is called recursive if it solve a problem by reducing it to an instance of the same problem with a smaller input

ex. Give a recursive algorithm for computing the greatest common divisor of two non-negative integers a and b with a < b.

```
procedure gcd(a,b: non-negative integers, a<b)
if a = 0
    then return b
else
    return gcd(b mod a, a)
```

ex. Give a recursive algorithm for computing n!, where n is a non-negative integer

```
procedure factorial(n: non-negative integer)
if n = 0
    then return 1
else
    return n*factorial(n-1)
```

ex. Give a recursive algorithm for computing a^n , where a is a non-zero real number and n is a non-negative integer

```
procedure power(a is non-zero real number, n non-negative integer)
if n = 0
    then return 1
else
    return a*power(a,n-1)
```

Inductive Proof:

Basis Step: $a^0 = 1$ for every non-zero real number a and power(a,0) = 1

Inductive Step: Assume power(a,k) = a^k for all $a \neq 0$

Must Show: power(a,k+1) = a^{k+1}

LHS	RHS	Steps
power(a,k+1)	a*power(a,k)	
=	$a*a^k$	power(a,k) = a^k
=	a^{k+1}	Properties of exponents

$\therefore$ Algorithm is correct

Program Correctness

A program, or program segment, S is said to be partially correct with respect to the initial assertion p and the final assertion q if whenever p is true for the input values of S and S terminates, then q is true for the output values of S. The notation p{S}q indicates that the program, or program segment, S is partially correct with respect to the initial assertion p and the final assertion q.

Base Case: p{S}q

- p is true for input values of S
- S executes and terminates
- q is true for output values of S
- $\therefore p\{S\}q$ is true

ex. Prove that the program segment is correct with respect to the initial assertion x = 0 and the final assertion z = 1.

```
y := 1
z := x + y
```

p: x = 0 is true

S executes and terminates: y = 1 and z = 0 + 1 = 1

q: z = 1

$\therefore p\{S\}q$ is true and program segment is correct

Program with subprograms: p{S_1;S_2}q

- For p{S_1}q
  - p is true for input values of S_1
  - S_1 executes and terminates
  - q is true for output values of S_1
- For p{S_2}q
  - q is true for input values of S_2
  - S_2 executes and terminates
  - r is true for output values of S_2
- $\therefore p\{S_1; S_2\}q$ is true

p{if condition then S}q

- For (p ∧ condition){S}q
  - p is true for input values of S AND condition is also true
  - S executes and terminates
  - q is true for output values of S
- For (p ∧ ¬condition) → q
  - p is true for input values of S AND condition is false
  - S does NOT execute
  - q is true
- $\therefore p\{if\ condition\ then\ S\}q$ is true

p{if condition then S_1 else S_2}q

- For (p ∧ condition){S_1}q
  - p is true for input values of S = S_1;S_2 AND condition is also true
  - S_1 executes and terminates
  - q is true for output values of S_1
- For (p ∧ ¬condition) → q
  - p is true for input values of S = S_1;S_2 AND condition is false
  - S_2 executes and terminates
  - q is true
- $\therefore p\{if\ condition\ then\ S_1\ else\ S_2\}q$ is true

ex. Verify that the program segment is correct with respect to the initial assertion y = 3 and the final assertion z = 6.

```
x := 2
z := x + y
if y > 0 then
    z := z + 1
else
    z := 0
```

p: y = 3 ∴ x = 2 and z = 5 and condition is true 3 > 0

S_1 executes and terminates: z = 5 + 1 = 6

q: z = 6

$\therefore p\{if\ condition\ then\ S_1\ else\ S_2\}q$ is true and program segment is correct

Loop Invariants: p{while condition S}(¬ condition ∧ p)

- p is a loop invariant if (p ∧ condition){S}p is true
  - p is true prior to first iteration of the loop
  - If p is true before an iteration of the loop, it remains true before the next iteration
- Suppose p is proven to be a loop invariant
  - p is true for input values of S
  - Condition is true and S executes and terminates
  - p is true AND condition is false
- ∴ p{while condition S}(¬ condition ∧ p)

Languages and Grammars

Formal Language: A language that is specified by a well-defined set of rules of syntax

Formal Grammar:

- A formal grammar G is any compact, precise definition of a language L.
- A grammar implies an algorithm that would generate all legal sentences of the language.

Phase-structure Grammars

Vocabulary: A vocabulary (or alphabet) **V** is a finite, nonempty set of elements called symbols.

Word: A word (or sentence) over V is a string of finite length elements of V .

Empty String λ: The empty string or null string, denoted by λ, is the string **containing no symbols**.

S**et of Words & Set of Language:**

- The set of all words over V is denoted by V^* .
- A language over V is a subset of V^* .

Phrase-Structure Grammar:

- A phrase-structure grammar **G = (V,T,S,P)** consists of:
  - a vocabulary **V**
  - a subset **T** of V consisting of **terminal symbols**
  - a **start symbol S from V**
  - a finite set of **productions P**.
- The set V − T is denoted by N.
- Elements of N are called nonterminal symbols.
- Every production in P must contain at least one nonterminal on its left side.

Derivability

Let $G = (V, T, S, P)$ be a phrase-structure grammar. Let $w_0 = lz_o r$ (that is, the concatenation of l, z_o, and r) and $w_1 = lz_1 r$ be strings over V. If $z_o \to z_1$ is a production of G, we say that w_1 is *directly derivable* from w_0 and we write $w_0 \Rightarrow w_1$. If $w_0, w_1, \ldots, w_n$ are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \ldots, w_{n-1} \Rightarrow w_n$, then we say that w_n is *derivable from* w_0, and we write $w_0 \overset{*}{\Rightarrow} w_n$. The sequence of steps used to obtain w_n from w_0 is called a *derivation*.

Language Generated by G/Language of G/L(G):

- The set of all strings of terminals that are derivable from the starting state S
- $L(G) = \{w \in T^* | S^* \Rightarrow w\}$

General Example

ex. Let G = (V,T,S,P) be the phrase-structure grammar with V = {0,1,A,B,S}, T = {0,1}, and set of productions P consisting of S → 0A,S → 1A,A → 0B,B → 1A,B → 1.

1. Show that 10101 belongs to the language generated by G.

   S → 1A = 1A

   A → 0B = 10B

   B → 1A = 101A

   A → 0B = 1010B

   B → 1 = **10101**

2. Show that 10110 does not belong to the language generated by G.

   S → 1A = 1A

   A → 0B = 10B

   B → 1A = 101A

   P does not contain any rules that allow two 1s to be adjacent to each other.

3. What is the language generated by G?

   Since S → 0A and S → 1A we know that strings must start with either 0 or 1

   Using production rules we know that we will next get 101A or 001A and that the string terminates with only B → 1.

   $\therefore \{0(01)^n | n \geq 1\} \cup \{1(01)^n | n \geq 1\}$

Finding a phrase-structure grammar

ex. Find a phrase-structure grammar for each of these languages.

1. The set consisting of the bit strings 0, 1, and 11

   Let G = (V,T,S,P) be the phrase-structure grammar with V = {0,1,S}, T = {0,1}, and set of productions P consisting of S → 0, S → 1, S → 11.

2. The set of bit strings containing only 1s

   Let G = (V,T,S,P) be the phrase-structure grammar with V = {1,S,A}, T = {1}, and set of productions P consisting of S → 1A, A → 1A, A → λ.

3. The set of bit strings that start with 0 and end with 1

Let G = (V,T,S,P) be the phrase-structure grammar with V = {0,1,S,A}, T = {0,1}, and set of productions P consisting of S → 0A1, A → 0A, A → 1A, A → λ.

4. The set of bit strings that consist of a 0 followed by an even number of 1s

   Let G = (V,T,S,P) be the phrase-structure grammar with V = {0,1,S,A}, T = {0,1}, and set of productions P consisting of S → 0A, A → 11A, A → λ.

Constructing a phrase-structure grammar to generate a set

Construct phrase-structure grammars to generate each of these sets.

1. $\{0^n | n \geq 0\}$

   Let G = (V,T,S,P) be the phrase-structure grammar with V = {0,S}, T = {0}, and set of productions P consisting of S → 0S, S → λ

2. $\{1^n 0 | n \geq 0\}$

   Let G = (V,T,S,P) be the phrase-structure grammar with V = {0,1,S,A}, T = {0,1}, and set of productions P consisting of S → A0, A → A1, A → λ

3. $\{(000)^n | n \geq 0\}$

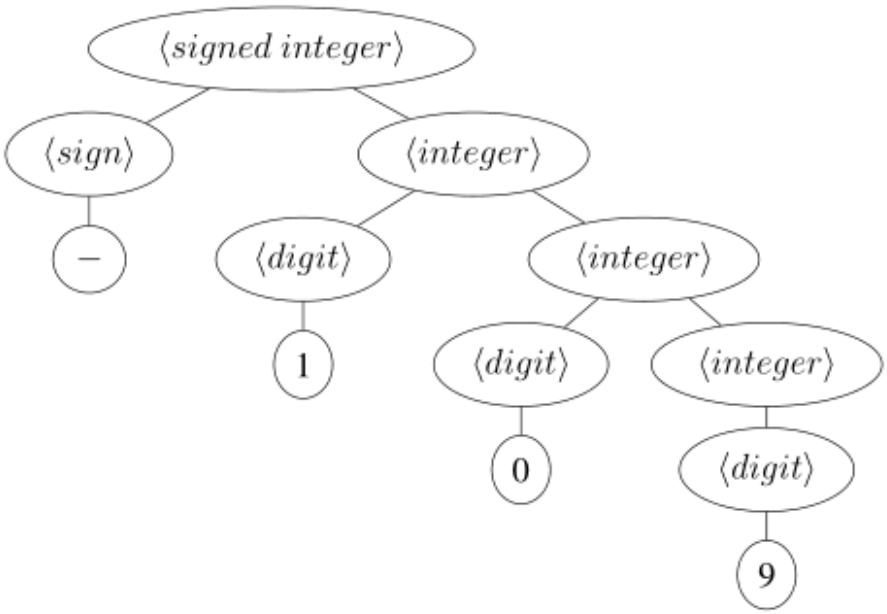
   Let G = (V,T,S,P) be the phrase-structure grammar with V = {0,S}, T = {0}, and set of productions P consisting of S → 000S, S → λ.

Derivation Trees

- **Root**: Starting symbol
  - **Internal Vertices**: Nonterminal symbols
    - **Leaves**: Terminal symbols

Construct a derivation tree given grammar

Construct a derivation tree for −109



Backus-Naur Form

- Instead of listing all the productions separately, we can combine all those with the same nonterminal symbol on the left-hand side into one statement.
- Instead of using the symbol →in a production, we use the symbol ::=

- We enclose all nonterminal symbols in brackets, $\langle\rangle$, and we list all the right-hand sides of productions in the same statement, separating them by bars, $|$.

Give production rules in Backus-Naur form

Give production rules in Backus-Naur form for an identifier if it can consist of

1. one to four uppercase or lowercase letters beginning with an uppercase letter.

   $\langle identifier\rangle ::== \langle ucletter\rangle|\langle ucletter\rangle\langle letter\rangle|\langle ucletter\rangle\langle letter\rangle\langle letter\rangle|\langle ucletter\rangle\langle letter\rangle\langle letter\rangle\langle letter\rangle$

   $\langle letter\rangle ::== \langle ucletter\rangle|\langle lcletter\rangle$

   $\langle ucletter\rangle ::== A|B|C...|Z$

   $\langle lcletter\rangle ::== a|b|c...|z$

2. a lowercase letter, followed by a digit or an underscore, followed by three or four alphanumeric characters (lower or uppercase letters and digits).

   $\langle identifier\rangle ::== \langle lcletter\rangle\langle digitorus\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle|$

   $\langle lcletter\rangle\langle digitorus\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle$

   $\langle digitorus\rangle ::== \langle digit\rangle|\_$

   $\langle alphanumeric\rangle ::== \langle letter\rangle|\langle digit\rangle$

   $\langle letter\rangle ::== \langle ucletter\rangle|\langle lcletter\rangle$

   $\langle ucletter\rangle ::== A|B|C...|Z$

   $\langle lcletter\rangle ::== a|b|c...|z$

   $\langle ucletter\rangle ::== A|B|C...|Z$

   $\langle digit\rangle ::== 0|1|2...|9$

Finite-State Machines with Output

A finite-state machine M = (S,I,O,f,g,s_0) consists of

- a finite set S of states
- a finite input alphabet I
- a finite output alphabet O
- a transition function f (f : S × I → S)
- an output function g (g : S × I → O)
- an initial state s 0

Mealy machines: outputs correspond to transitions between states

- If length of input is n then output length is n

Moore machine: output is determined only by the state

- If length of input is n then output length is n+1

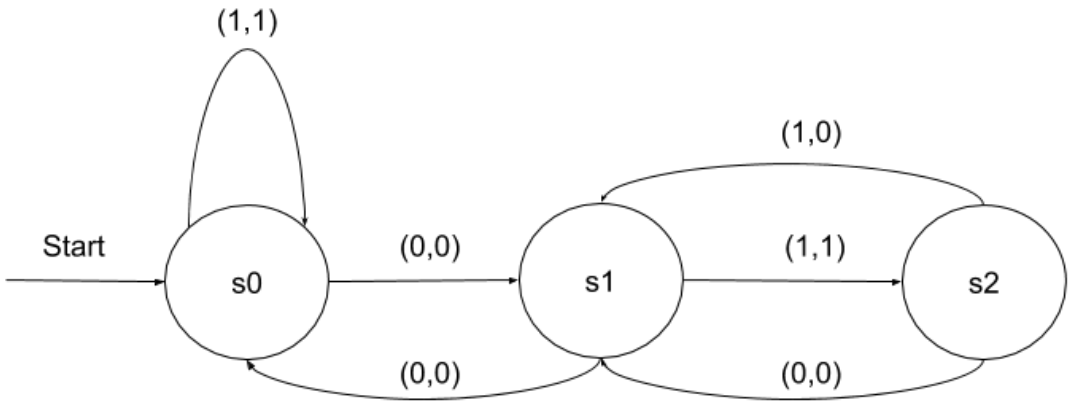
State Diagrams from State Tables

ex. Draw the state diagrams for the finite-state machines with these state tables.

1.

	f	f	g	g
State	**0**	**1**	**0**	**1**
s_0	s_1 (0,0)	s_0 (1,1)	0	1
s_1	s_0 (0,0)	s_2 (1,1)	0	1
s_2	s_1 (0,0)	s_1 (1,0)	0	0

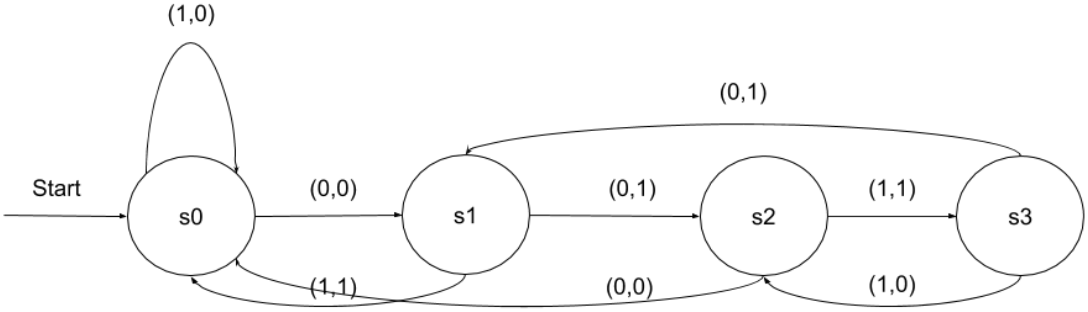
*Add pairs to table to make things easier



2.

	f	f	g	g
State	**0**	**1**	**0**	**1**
s_0	s_1 (0,0)	s_0 (1,0)	0	0
s_1	s_2 (0,1)	s_0 (1,1)	1	1
s_2	s_0 (0,0)	s_3 (1,1)	0	1
s_3	s_1 (0,1)	s_2 (1,0)	1	0

*Add pairs to table to make things easier



Output generated from input string, given a sequence

ex. Find the output generated from the input string 01110 for the finite-state machine with the state table from previous example

1. Table 1: $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow s_1$

Solution:

s_0: 0 → **0** →s_1: 1 → **1**→ s_2: 1 → **0** → s_1: 1 → 1 → s_2: 0 → 0 → s_1

Output: 01010

2. Table 2: $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_0 \rightarrow s_0 \rightarrow s_1$

Solution:

s_0: 0 → **0** →s_1: 1 → **1**→ s_0: 1 → **0** → s_0: 1 → **0** → s_0: 0 → **0** → s_1

Output: 01000

Constructing finite state machine when given only a description

Construct a finite-state machine with output that produces a 1 if and only if the last 3 input bits read are 0s.

Create possible valid and invalid inputs:

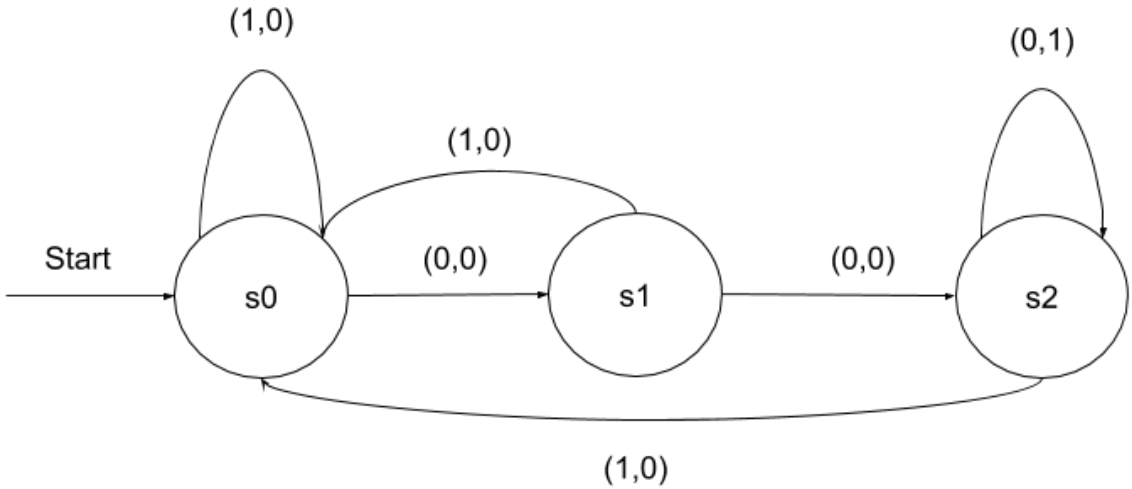
Inputs that will produce 1: 000, 1000, 101000

Inputs that will not produce 1: 111, 1001, 0010

Create Table:

	f	f	g	g
State	**0**	**1**	**0**	**1**
s_0	s_1 (0,0)	s_0 (1,0)	0	0
s_1	s_2 (0,0)	s_0 (1,0)	0	0
s_2	s_2 (0,1)	s_0 (1,0)	1	0

Construct Diagram:



Finite-State Machines with No Output

Set of Strings

Suppose that A and B are subsets of V^*, where V is a vocabulary.

- The concatenation of A and B, denoted by AB, is the set of all strings of the form xy, where x is a string in A and y is a string in B.

Suppose that A is a subset of V^*.

- Then the **Kleene closure** of A, denoted by A^* , is the set consisting of concatenations of arbitrarily many strings from A.

String Concatenation

ex. Let A = {0,11} and B = {00,01}. Find each of these sets:

1. AB

   AB = {000, 001, 110, 1101}

2. BA

   BA = {000, 010, 0011, 0111}

3. A^2

   A^2 = {00, 011, 110, 1111}

Describe the elements

ex. Describe the elements of the set A^* for these values of A.

1. {10}

   Solution: $\{(10)^n | n = 0,1,2,...\}$

2. {111}

   Solution: $\{1^{3n} | n = 0,1,2,...\}$

3. {0,01}

   Solution: The set of strings where every 1 is immediately preceded by a 0.

Determining if a string is part of a set

ex. Determine whether the string 11101 is in each of these sets.

1. $\{0,1\}^*$

   Yes

2. $\{1\}^*\{0\}^*\{1\}^*$

   Yes

   Possible other sets produced: 101, 1101, 110001, 110001111

3. $\{11\}\{0\}^*\{01\}$

   No

   Possible other sets produced: 11001, 110001, 1100001

4. $\{11\}^*\{01\}^*$

   No

   Possible other sets produced: 1101, 111101, 11110101

5. $\{111\}^*\{0\}^*\{1\}$

Yes

Possible other sets produced: 11111101, 111111001

6. {11,0}{00,101}

   Yes

   Possible other sets produced: 1100, 000, 0101

Finite-State Automata

Finite-state automata are finite-state machines with no output.
A finite-state automaton M = (S,I,f,s_0,F) consists of

- a finite set S of states
- a finite input alphabet I
- a transition function f (f : S × I → S)
- an initial state s_0
- a finite set F of final states (or accepting states)

When drawing a diagram, make a node for garbage collection for invalid inputs.

If s_0 is an accepted state then the automata recognizes empty strings.

Constructing a deterministic finite-state automaton when given only a description

ex. Construct a deterministic finite-state automaton that recognizes the set of all bit strings beginning with 01.

1. Create possible valid and invalid inputs:

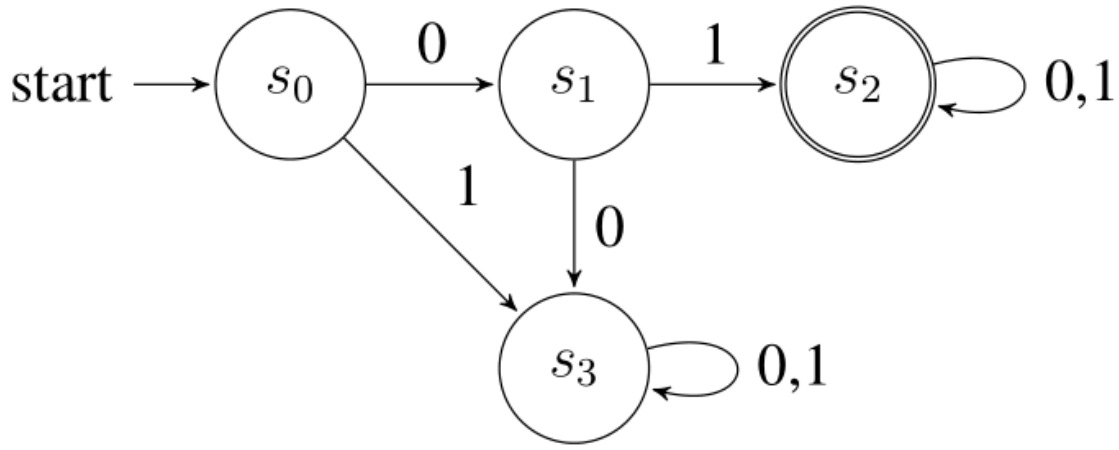
   Inputs that will be recognized: 01, 010, 011, 0110, 0111

   Inputs that won't be recognized: 10, 11, 11001, 001

   Create Table:

	f	**f**
State	**0**	**1**
s_0	s_1	s_3
s_1	s_2	s_3
s_2	s_2	s_2
s_3	s_3	s_3

   Construct Diagram:



2. Create possible valid and invalid inputs:

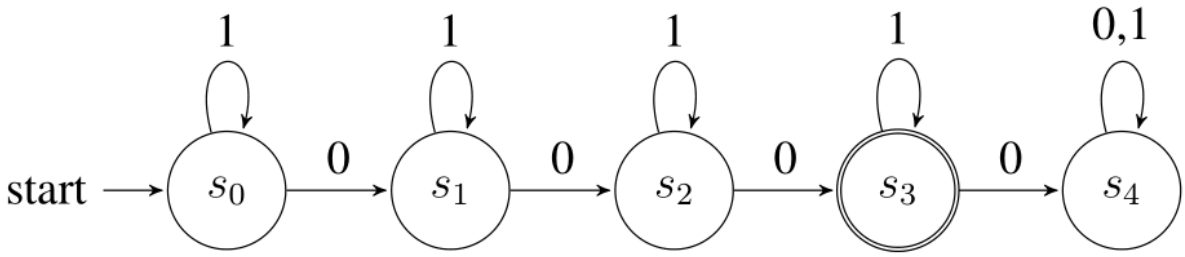
   Inputs that will be recognized: 000, 1000, 10001, 101010

   Inputs that won't be recognized: 10, 11, 11001, 001

   Create Table:

	f	f
State	**0**	**1**
s_0	s_1	s_0
s_1	s_2	s_1
s_2	s_3	s_2
s_3	s_4	s_3
s_4	s_4	s_4

   Construct Diagram:



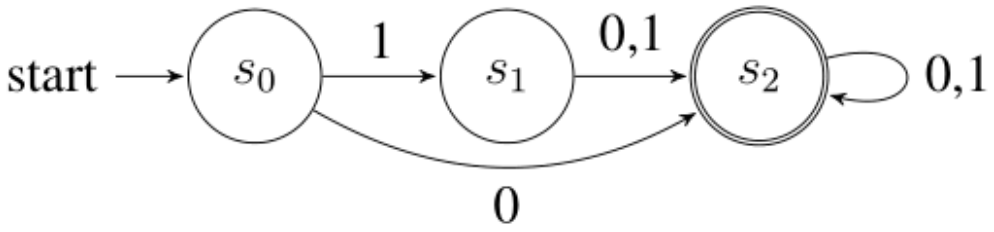
Language Recognition by Finite-State Machines

A string x is said to be recognized or accepted by the machine M = (S,I,f,s_0,F) if it takes the initial state s_0 to a final state, that is f(s_0,x) is a state in F. The language recognized or accepted by the machine M, denoted by L(M), is the set of all strings that are recognized by M. **Two finite-state automata are called equivalent if they recognize the same language.**

Finding recognized language given a finite-state automaton

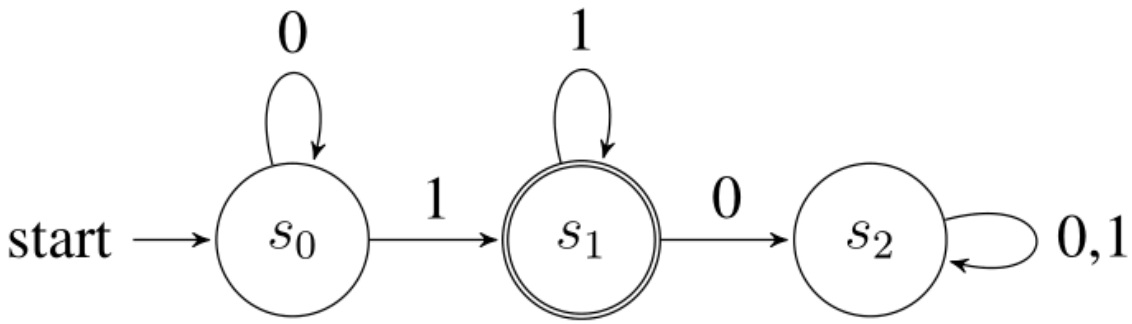
ex. Find the language recognized by the given deterministic finite-state automaton.

1.



Solution: {0,10, 11}{0,1}*

2.



Solution: {0}*{1}{1}*

- Initial 0 input will cause loop until a follow up 1 is added

Nondeterministic Finite State Automata

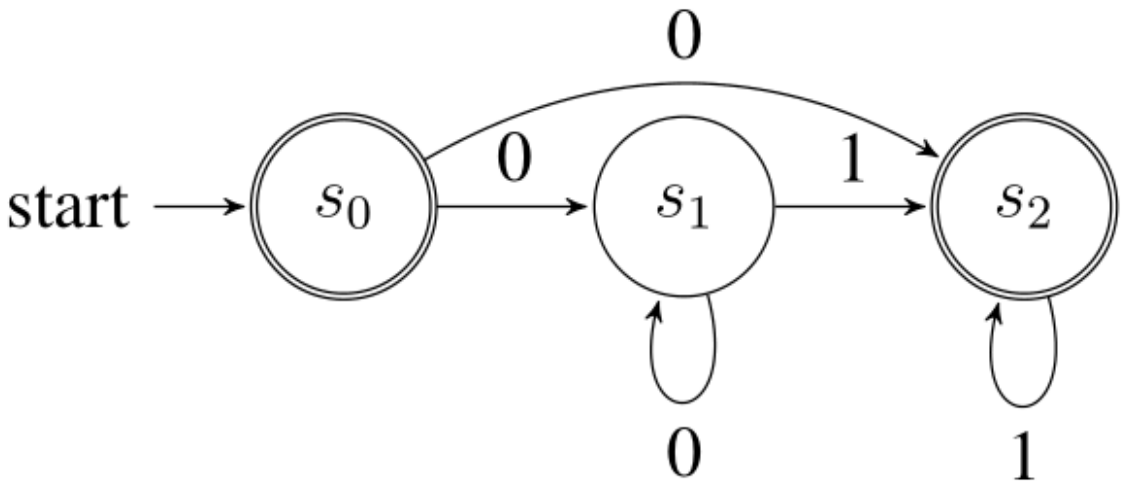
A nondeterministic finite-state automaton M = (S,f,I,s_0,F) consists of

- a finite set S of states
- a finite input alphabet I
- a transition function f (f : S × I → P(S))
- an initial state s_0
- a finite set F of final states

- If the language L is recognized by a nondeterministic finite-state automaton M_0 , then L is also recognized by a deterministic finite-state automaton M_1

Finding recognized language given a nondeterministic finite-state automaton

ex. Find the language recognized by the given nondeterministic finite-state automaton.



Accepts empty strings (λ)

Accepts $\{0^n1^m | n,m \geq 1\}$

Accepts $\{01^m | m \geq 0\}$

$\therefore \{\lambda\} \cup \{0^n 1^m | n, m \geq 1\} \cup \{01^m | m \geq 0\}$

Turing Machines

A Turing machine consists of a finite state control unit and a tape divided into cells that expands infinitely in both directions.

The control unit:

- is in one state of finitely many different states at any one step
- has read and write capabilities on the tape as the control unit moves left and right on the tape

A Turing machine is the most general model of computation. They can model all computations that are performed on a computer.

Formal Definition of Turing Machine

A Turing machine T = (S,I,f,s_0) consists of

- a finite set S of states
- an alphabet I containing the blank symbol B
- a partial function f (f : S × I → S × I × {R,L})
    - The five-tuple (state, symbol, state, symbol, direction) corresponding to the partial function are called transition rules
- a starting state s_0

Transition in Turing Machines

- If the control unit is in state s and if the partial function f is defined for the pair (s,x) with **f(s,x) = (s_0 ,x_0 ,d) (corresponding to the five-tuple (s,x,s_0,x_0,d))**, the control unit will

1. **Enter the state s_0**
2. Write the symbol **x_0 in the current cell, thus erasing x**
3. Move **right one cell if d = R**, or **left one cell if d = L**

- **If the partial function f is undefined for the pair (s,x), then the Turing machine T will halt.**

- At the initial state s_0, the control head is positioned either
  - over the **leftmost nonblank symbol on the tape**
  - over **any cell if the tape is all blank**

Recognizing Sets

A **final state of a Turing machine T is a state that is not the first state** in any five-tuple in the description of T using five-tuples.

Let V be a subset of an alphabet I. A Turing machine T = (S,I,f,s_0) recognizes a string x in V^* if and only if T, starting in the initial position when x is written on the tape, halts in a final state. T is said to recognize a subset A of V^* if x is recognized by T if and only if x belongs to A.

Determining final tape from given five-tuples and initial tape

ex. Let T be the Turing machine defined by the five-tuples: (s_0 ,0,s_1 ,1,R), (s_0 ,1,s_1 ,0,R), (s_0,B,s_1,0,R), (s_1,0,s_2 ,1,L), (s_1,1,s_1,0,R), and (s_1,B,s_2,0,L). For each of these initial tapes, determine the final tape when T halts, assuming that T begins in initial position.

To make life easier put tuples into a vertical layout or table:

(s_0,0,s_1,1,R)

(s_0,1,s_1,0,R)

(s_0,B,s_1,0,R)

(s_1,0,s_2 ,1,L)

(s_1,1,s_1,0,R)

(s_1,B,s_2,0,L)

Initial State	Symbol (0, 1)	Destination State	Symbol (0, 1)	Direction (L or R)
s_0	0	s_1	1	R
s_0	1	s_1	0	R
s_0	B	s_1	0	R
s_1	0	s_2	1	L
s_1	1	s_1	0	R
s_1	B	s_2	0	L

1. B B 0 0 1 1 B B

   s_0: B B **0** 0 1 1 B B

   s_1: B B 1 **0** 1 1 B B

   s_2: B B **1** 1 1 1 B B

   1111
2. B B 1 0 1 B B B

   s_0: B B **1** 0 1 B B B

s_1: B B 0 **0** 1 B B B

s_2: B B **0** 1 1 B B B

011

3. B B B B B B B B

   s_0: **B** B B B B B B B

   s_1: 0 **B** B B B B B B

   s_2: 0 0 B B B B B B

   00

Turing machine output when given five-tuples and input

ex. What does the Turing machine described by the five-tuples (s_0,0,s_0,0,R), (s_0,1,s_1,0,R), (s_0,B,s_2,B,R), (s_1,0,s_1,0,R), (s_1,1,s_0,1,R), and (s_1,B,s_2,B,R) do when given:

To make life easier put tuples into a vertical layout or table:

(s_0,0,s_0,0,R)

(s_0,1,s_1,0,R)

(s_0,B,s_2,B,R)

(s_1,0,s_1,0,R)

(s_1,1,s_0,1,R)

(s_1,B,s_2,B,R)

1. 1 1

   s_0: **1** 1 B B B B B

   s_1: 0 **1** B B B B B

   s_0: 0 1 **B** B B B B

   s_2: 0 1 B **B** B B B

   01

2. an arbitrary bit string

   This machine will read in a string and flip every other 1, starting with the first 1, in the bit string.

Constructing a Turing machine when given a description

ex. Construct a Turing machine with tape symbols 0, 1, and B that, when given a bit string as input, replaces the first 0 with a 1 and does not change any of the other symbols on the tape.

if (s_0, 0) then (s_1, 1) then move R

if (s_0, 1) then (s_0, 1) then move R

$\therefore$ (s_0, 0, s_1, 1, R), (s_0, 1, s_0, 1, R)

ex. Construct a Turing machine with tape symbols 0, 1, and B that, when given a bit string as input, replaces all but the leftmost 1 on the tape with 0s and does not change any of the other symbols on the tape.

if (s_0, 0) then (s_0, 0) then move R

if (s_0, 1) then (s_1, 1) then move R

if (s_1, 0) then (s_1, 0) then move R

if (s_1, 1) then (s_1, 0) then move R

$\therefore$ (s_0, 0, s_0, 1, R), (s_0, 1, s_1, 1, R), (s_1, 0, s_1, 0, R), (s_1, 1, s_1, 0, R),

ex. Construct a Turing machine that recognizes the set of all bit strings that end with a 0.

if (s_0, 0) then (s_1, 0) then move R

if (s_0, 1) then (s_0, 1) then move R

if (s_1, 0) then (s_1, 0) then move R

if (s_1, 1) then (s_0, 1) then move R

if (s_1, B) then (s_2, B) then move R

$\therefore$ (s_0, 0, s_1, 0, R), (s_0, 1, s_0, 1, R), (s_1, 0, s_1, 0, R), (s_1, 1, s_0, 1, R), (s_1, B, s_2, B, R)

ex. Construct a Turing machine that recognizes the set of all bit strings that contain an even number of 1s.

if (s_0, 0) then (s_0, 0) then move R

if (s_0, 1) then (s_1, 1) then move R

if (s_1, 0) then (s_1, 0) then move R

if (s_1, 1) then (s_0, 1) then move R

if (s_0, B) then (s_2, B) then move R

$\therefore$ (s_0, 0, s_0, 0, R), (s_0, 1, s_1, 1, R), (s_1, 0, s_1, 0, R), (s_1, 1, s_0, 1, R), (s_0, B, s_2, B, R)