**CHAPTER 3**

**Orders**

* Preorder
  + Reflexive
  + Transitive
* Equivalence
  + Preorder that is also symmetric
* Weak Partial
  + Preorder that is also antisymmetric
* Strict Partial
  + Transitive, Asymmetric, Irreflexive
* Maximal element of S is an element such that for any x in a S, it is NOT true that M < x. IN other words, if something is a maximal element, nothing is greater than it. M must be in S itself.
* Maximum element of S is an element such that for any x in S, M >= x. M must be in S itself.
  + The difference between maximal and maximum? {1,2,3,4} has maximum 4, and maximal 4. The set {1,2,3,4,x,2x,abs(3x)} has maximal elements 4 and abs(3x), but not maximum.
* Maximum/maximal etc. elements only make sense for partials and things stricter than partials. Ditto with Upper bounds and supremum.
* Upper bound is just something that is thiccer than all the elements of a set. Least upper bound/supremum is something that is thiccer than all the elements of a set but smaller than all the other upper bounds.
* Note that preorders can be cyclical, whereas partial and strict orders cannot. Every binary relation R can be extended to a pre-order by taking the reflexive and transitive closure of R
* Weak total
  + Weak partial that is also total
* Strict total
  + Strict partial that is also trichotomous
* Reflexive: xRx
* Irreflexive: Never true that xRx
* Transitive: xRy and yRz means that xRz
* Symmetric: xRy means that yRx
* Asymmetric: xRy means that NOT(yRx)
* Antisymmetric: xRy and yRx means that x IS y
* Trichotomous: Either xRy or yRx
* Total: Either xRy or yRx or x IS y
* Examples: (Natural/Integers/Rationals/Reals, <=), are examples of weak total orders
* Well order
  + Strict total order such that all non empty subsets of S have a minimum element with respect to the given relation R
  + All well orders are Noetherian
    - Noetherian means no infinite descending sequences of the form aRbRcRd etc

**Logic**

* Logic contains common syntax, semantics and logical consequence
* First order is about quantification over individuals whereas higher order logic is about quantification over functions, predicates, etc. ore on slide 15-17 on predicate logic, propositional logic and sorts

**MSFOL**

* Sorts are basically just domains
* B are base types, basically just domains. The inductive definition is as follows:
  + Booleans
  + Base types
  + (Basetype1 🡪 Basetype2) is a basetype
  + (Btype1 x Btype2) is a basetype
  + Dumb Farmer inductive definition on slide 22
* Signatures
  + Formal language
    - Variables
    - Logical constants (and, universals, ors, etc)
      * These have the same meaning in EVERY interpretation of the language
    - Nonlogical constants (0 vector, <. etc)
      * Same meaning within a single interpretation of a language but (sometimes) different meanings when compared across other interpretations of said language
    - Punctuation symbols
      * Parentheses and shit
  + A signature of a formal language is just the set of nonlogical symbols of said formal language.
* MSFOL Signatures

A screenshot of a cell phone

Description automatically generated

* MSFOL Terms

A screenshot of a cell phone

Description automatically generated

* MSFOL Formulas

A screenshot of a cell phone screen with text

Description automatically generated

* Variable Binders
  + Basically something that just binds every occurrence of a variable x to some type B
* Bound vs Free variables
  + Bound if its in the subformula contained inside a universal quantifier
  + SigTerms and SigFormulas are closed if there are no free variables in it. A closed formula is called a SigSentence
  + More BS on slides 46-47
  + Variable capturing is free 🡪 bound
* Substitutions

A picture containing knife, table

Description automatically generated

A screenshot of a cell phone

Description automatically generated

* Mathematical Structures
  + Bruh

A screenshot of a cell phone

Description automatically generated

**CHAPTER 4**

**Theory of computation:** Study of the foundations of computable functions, basically what it means to be computable, what can/cannot be computed, computation power/mechanisms and classification of computable functions. Models of computation include Automata and grammars.

**Automaton:** Abstract computing machine

* **Finite automata:** Finite memory
* **Push down automata**: Finite memory, stack
* **Turing machine:** Unlimited memory

**Grammar**: Set of rules for generating the expressions of a given language

* **Regular**:Generate regular languages (Finite automata)
* **Context-free**: Generates context free languages (Push-down automata)
* **Unrestricted**: Generates recursively enumerable languages (Turings)
* **Context-sensitive:** Generates context-sensitive languages
* This means that Unrestricted ENC ContextF ENC Reg

The 4 grammars above are in the chomsky heiarchy

A screenshot of a social media post

Description automatically generated

The **alphabet** is simply that – a set of all the individual symbols that comprise a set. A **string** is a finite sequence of alphabet symbols. The empty string is denoted by epsilon. Note that 

and 

**String operations**: Concatenation, length (literally just length, and denoted by absolute value symbol), repetition (denoted by ^n symbol).

**Operation of sets of string:**

|  |  |  |
| --- | --- | --- |
| Union | A U B | {abc, bac} U {abc, bbb} = {abc, bac, bbb} |
| Intersection | A ∩ B | {abc, bac} ∩ {abc, bbb} = {abc} |
| Complement | Duh |  |
| Power | A^n = {eps} or A\*A\*A… n times | A^0 = {eps} A^5 = AAAAA |
| Astertate | A^0 U A^1 U A^2…. |  |
| Positive astertate | A^1 U A^2… |  |
| Concatenation | Same as cartesian product |  |

**Decision Problem:** Yes-or-no question regarding a specific input. Since this is essential an algorithm, decision problems are computable functions, tho the corresponding computable function may not exist. Note also that decision problems can be formalized as whether or not a string is a member of a particular set (slide 20)

**Decidability:** A decision problem is decidable if there is a computable function that solves it.

Church and Turing showed that there are undecidable decision problems, like EnUglygermanword and the halting problem.

**5. Push Down Automata and Context-Free Languages**

**Regular languages:** special sets of strings that can be specified by (1) DFA, (2) NFA, (3) NFA with e transition, (4) regular expression, (5) regular grammars

* Most sets of strings are not regular

**Context free languages:** specified by (1) context free grammar and (2) nondeterministic push down automata

* Most sets of strings are not context free

**L1 and L2 are CFL**

|  |  |
| --- | --- |
| **What’s not context free** | **What’s also context free** |
| Complementation ~L | Union L1 L2 |
| Intersection L1 L2 | Concatenation L1L2 |
|  | Asterate L\* |

**Context free grammar:** formal grammar in which production rules are of the form A -> a where A is a single nonterminal symbol and a is the string of terminals/nonterminals

* Considered “context free” when its production rules can be applied regardless of the context of a nonterminal
* No matter which symbol surrounds it, the single nonterminal on the left hand side can always be replaced by the right hand side
* G = (N, Σ, P, S) where

1. N = nonterminal symbol (A,B,C…)
2. Σ = terminal symbol (a,b,c)
3. N and Σ disjoint
4. P = N x (N U Σ)\* called production
5. S = start symbol

* (A,a) means A -> a
* Language generated by G, written L(G) is the set of all strings x Σ\* such that A close up of a clock

  Description automatically generated
* Two grammars are equivalent if L(G1) = L(G2)
* Introduced by Chomskey
* Nonterminal A is **useful** if there is derivation A picture containing object, clock

  Description automatically generated

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
|  | B is derivable from a in one step in G (B can be obtained from a by replacing some occurrence of a nonterminal A in a with some other terminal |
|  | a for any a |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Production** | **Meaning** |
| **– production** | Production of the form A -> |
| **Unit production** | Production of the form A -> B |

|  |  |
| --- | --- |
| **Special Types of Grammars** | **Description** |
| **Linear** | All its productions have at most one nonterminal on the right hand side.  A linear language is generated by some linear CFG  **Right Linear**: all its productions have the form A -> xB or A -> x  **Left Linear:** all its productions have the form A -> Bx or A -> x |
| **Regular** | It’s right linear or left linear.  A language is regular iff it is generated by a regular CFG  There are linear languages that are not regular |

**Backus-Naur Form (BNF):** context-free grammar for a Pascal-like programming language

* Consists of finite set of production rules
* Rules can be recursive
* Rule has exactly one nonterminal on the left hand side
* Essential technique for programming language specification
* BNF defines the syntax of a programming language
* Alternatives separated by |
* <x> nonterminal symbols, others called terminal symbol
* Each nonterminal generates a set of strings over a finite alphabet Σ
* **Derivation** of string w is a sequence of expressions starting with <stmt> and ending with w containing no nonternimals
* Each expression obtained from previous expression by replacing one nonterminal <x> with one of the alternatives on the right side of the production for <x>

**Chomky Normal Form (CNF):** all its production have the form A -> BC or A -> a

* For a given grammar, there could be more than one CNF

**CFG to CNF**

|  |  |
| --- | --- |
| **Steps** | **Explanation** |
| Original | S -> ASB  A -> aAS | a |  B -> SbS | A | bb |
| Eliminate start symbols | S0 -> S  S -> ASB  A -> aAS | a |  B -> SbS | A | bb |
| Eliminate null production A -> | S0 -> S  S -> ASB | SB  A -> aAS | aS | a  B -> SbS | A | | bb |
| Eliminate null production B -> | S0 -> AS | ASB | SB | S  S -> AS | ASB | SB | S  A -> aAS | aS | a  B -> SbS | A | bb |
| Eliminate unit production B -> A | S0 -> S  S -> AS | ASB | SB | S  A -> aAS | aS | a  B -> SbS | bb | aAS | aS | a |
| Eliminate unit production S0 -> S | S0 -> AS | ASB | SB | S  S -> AS | ASB | SB | S  A -> aAS | aS | a  B -> SbS | bb | aAS | aS | a |
| Eliminate unit production S -> S and S0 -> S | S0 -> AS | ASB | SB  S -> AS | ASB | SB  A -> aAS | aS | a  B -> SbS | bb | aAS | aS | a |
| Eliminate from RHS no more than two Non-terminals | S0 -> AS | PB | SB  S -> AS | QB | SB  A -> RS | XS | a  B -> TS | VV | US | XS | a  X -> a  Y -> b  V -> b  P -> AS  Q -> AS  R -> XA  T -> SY  U -> XA |

**Greibach Normal Form (GNF):** all its productions have the form A -> aB1…Bk where k >= 0

* For a given grammar, there could be more than one GNF
* GNF produces the same language as generated by CFG

**CFG to GNF**

|  |  |
| --- | --- |
| **Steps** | **Explanation** |
| Convert production rules to CNF, Eliminate left recursion (Original) | S -> XA | BB  B -> b | SB  X -> b  A -> a |
| Convert to GNF. B -> SB is not GNF so substitute S -> XA|BB in B -> SB | S -> XA|BB  B -> b|XAB|BBB  X -> b  A -> a |
| S -> XA and B -> XAB is not GNF. Substitute X -> b in production rules S -> XA and B -> XAB | S -> bA | BB  B -> b | bAB | BBB  X -> b  A -> a |
| Remove left recursion B -> BBB | S -> bA | BB  B -> bC | bABC  C -> BBC |  X -> b  A -> a |
| Remove null production C -> | S -> bA | BB  B -> bC | bABC | b | bABC  C -> BBC | BB  X -> b  A -> a |
| S -> BB is not in GNF. Substitute B -> bC | bABC | b | bAB in S-> BB | S -> bA | bCB | bABCB | bB | bABB  B -> bC | bABC | b | bAB  C -> BBC | BB  X -> b  A -> a |
| C -> BB is not in GNF. Substitute B -> bC|bABC|b|bAB in C->BB | S -> bA|bCB|bABCB|bB|bABB  B -> bC|bABC|b|bAB  C -> BBC  C -> bCB|bABCB|bB|bABB  X -> b  A -> a |
| C -> BBC is not in GNF. Substitute B -> bC|bABC|b|bAB in C -> BBC | S -> bA|bCB|bABCB|bB|bABB  B -> bC|bABC|b|bAB  C -> bCBC|bABCBC|bBC|bABBC  C -> bCB|bABCB|bB|bABB  X -> b  A -> a |

**Pumping Lemma for Context Free Languages**

|  |  |  |
| --- | --- | --- |
| **Types** | **Description** | **Uses** |
| **Regular languages** | Every sufficiently long string in a regular language contains a short substring that can be pumped. | Used as a proof for irregularity of a language (ie. If language is regular, it always satisfies pumping lemma) |
| **CFL** | Every sufficiently long string in a CFL can be divided into five segments such that the middle three segments are short and the second and fourth can be simultaneously pumped | Used to prove that a language is not CFL. If any one string does not satisfy its conditions, then the language is not CFL |

**Context-Sensitive Grammars**

G = (N, Σ, P, S) where

* N = nonterminal symbols
* Σ = terminal symbols
* N and Σ are disjoint
* A picture containing knife

  Description automatically generated
* S = Start symbol
* S -> is allowed if S doesn’t appear on right side of any production

**Language, Grammar, Automation Table**

**A screenshot of a cell phone

Description automatically generated**

**NFA As machine**

* Made up of (1) input tape, (2) memory (state), (3) program
* Tape is read left to right
* Read only
* Memory is finite
* Program is nondeterministic

**NPDA as machine**

* Made up of (1) input tape, (2) memory (state, stack) , (3) program
* Tape is read left to right
* Read only
* Memory is unbounded but accessed as a stack
* Program is nondeterministic

**Nondeterministic Push-Down Automata (NPDA)**

M = (Q, Σ,  , , s, ㅗ, F) where

* Q = states
* Σ = input alphabets
*  = stack alphabet
*  = transition relation
* s = start state
* ㅗ = initial stack symbol
* F = final state
* **Configuration:** member of Q x Σ\* x \* that designates M’s current state, unread input string, and stack components
* NFA is accepted if it’s either (1) accepted by final state, or (2) accepted by empty stack

**Deterministic Push-Down Automata**

* Similar to NPDA except exactly one transition applies to each configuration
* For each input symbol, one can determine the state to which the machine will move

**Parsing:** Let L be a programming language. A parser for L is a program that takes a program in L as input and returns a parse tree of the program if the program is syntactically correct and otherwise identifies the syntax errors in the program

* It recognizes what strings are programs in L
* If the syntax of L is specified by a CFG then a parser for L can be implemented as a DPDA which is usually generated automatically from the grammar for L

**List of Theorems from Slide 5**

|  |  |
| --- | --- |
| **Concept** | **Theorems** |
| Context Free Grammar | Every CFL not containing is generated by some CFG that has no useless non-terminals, -productions, or unit productions |
| Chomsky Normal Form | Every CFL not containing is generated by some CFG in Chomsky Normal Form |
| Greibach Normal Form | Every CFL not containing is generated by some CFG in GNF |
| Regular Language | A language is regular iff it is generated by a regular CFG.  There are linear languages that are not regular |
| NPDA |  |
| Equivalence of CFG and NPDA | Let G be CFG. Then there is an NPDA M such that L(G) = L(M).  Let M be NPDA. Then there is a CFG G such that L(G) = L(M).  CFGs generate and NPDSs accept the same class of languages – the class of context free languages |
| DPDA | Every DCFL is CFL but not every CFL is DCFL  DCFL are closed under complements (unlike CFLs) |

**6. Turing Machines and Computability**

**Effective method:** method for solving family of problems in mechanical ways. It satisfies:

* Consists of series of steps where each step results from executing precise instruction
* Each instruction expressed by finite number of symbols
* Number of possible instructions is finite
* Finishes after a finite number of steps
* Solves every problem in the family
* Requires no ingenuity to succeed

**Models of Computation:** model that describes how an output is computed from an input.

* Ex. Lambda calculus, turing machines, general recursive functions
* All these models are computationally equivalent

**Church Turing Thesis:** any model of computation equivalent to those listed above captures exactly our intuition of what an effective method is

* Implies that any effective method can be implemented in any model of computation equivalent to those listed above

**Great Limitation Theorems**

* It states that there are limits on what can be proved, defined, and computed

|  |  |
| --- | --- |
| **Theorems** | **Descriptions** |
| **First Incompleteness Theorem** | No consistent, sufficiently strong, recursively axiomatizable proof system can prove all the truths of natural number arithmetic |
| **Second Incompleteness Theorem** | ;; prove its own consistency |
| **Undefinability of Truth** | Truth cannot be defined in any sufficiently strong theory |
| **Undecidability of First-Order Logic** | Validity is undecidable in first-order logic |

**Robinson Arithmetic**

A screenshot of a cell phone

Description automatically generated

* Q (Robinson arithmetic) is a theory of natural number arithmetic
* Without induction schema, Q is a very weak theory
* The limitation theories apply
* Metatheorems about Q:
  + Q is finitely axiomatizable
  + Q is incomplete and every consistent recursively axiomatizable extension of Q is incomplete (Q is essentially incomplete)
  + Consistency is not provable in Q nor in any consistent recursively axiomatizable extension of Q
  + Truth is not definable in Q
  + Q is undecidable and every consistent extension of Q is undecidable (Q is essentially undecidable)

**Turing Machines**

* Has the following components: (1) tape (ㅏ,a0,a1…-,-,-,…), (2) state, (3) program (transition function)
* Tape is two way, read/write, semi-infinite
* Tape is used for input, output, and memory
* Input string is finite
* Tape is unbounded, sequentially accessed memory
* Program is deterministic
* Takes the current tape symbol and state as input, and produces a new tape symbol, state, and left/right tape position change as output

**A screenshot of a cell phone

Description automatically generated**

**TM Terms**

|  |  |
| --- | --- |
| **Terms** | **Definitions** |
| **Accepts** | Reach accept state |
| **Rejects** | Reach reject state |
| **Halts** | If it either accepts or rejects |
| **Loops** | Neither accepts nor rejects |
| **Total** | Halt on all inputs |
| **L(M)** | Set of strings accepted by M |
| **Recursively Enumerable** | If a language is (M) for some TM M |
| **Recursive** | If a language is L(M) for some total TM M |

**Equivalent Definitions of Turing Machines**

1. With two-way tapes
2. With multiple tapes
3. With two-dimensional tapes
4. With multiple heads
5. Nondeterministic Turing machines

**Turing machines can be used in different ways to:**

1. Decide a decision problem
2. Semidecide a decision problem
3. Compute a total function
4. Compute a partial function
5. Enumerate a set of values

**Decidability Terms:** for a language L …

|  |  |
| --- | --- |
| **Terms** | **Definitions** |
| **Decidable** | If there is some TM that accepts every string in L and rejects every string not in L |
| **Semi-decidable** | If there is some TM that accepts every string in L, and either rejects or loops on every string not in L |
| **Undecidable** | There is no TM M that halts on every input and L(M) = L |

**Universal Turing Machine**

* TM that simulates arbitrary TM on arbitrary input
* Reads both the description of the machine to be simulated as well as the input to that machine from its own tape
* It’s universal in the sense that, for any problem that can be solved by TM, you could either use a TM that directly solves that problem, or you could use UTM and give it the description of a TM that directly solves the problem
* It’s like an interpreter for all TM

**Diagonalization and Halting Problem**

* Halting problem is the problem of deciding whether a given Turing machine will halt on given input
* Halting problem is semi decidable, and undecidable
* Diagonalization is employed to reach a contradiction that forces us to abandon an assumption
* Assumption: existence of TM that decides the halting problem
* Diagonalization is applied by building a table A such that Aij says whether the i-th TM halts on input j.
* Some Program D:

A screenshot of a cell phone

Description automatically generated

**Reduction:**

* A Turing reduction from a problem A to a problem B is a reduction which solves A, assuming the solution to B is already known
* An algorithm that could be used to solve A if it had available to it a subroutine for solving B
* A function computable by an oracle machine with an oracle B
* Can be applied to both decision problems and function problems

**Undecidable problems:**

* Techniques for showing that problems are undecidable: (1) diagonalization, (2) reduction
* If x A? is an undecidable problem, then we can show that x B? is undecidable by reducing A to B
* Reduction emphasizes that, if B is recursive, then A must be recursive (contradiction)

**Chapter 6 Theorems**

|  |  |
| --- | --- |
| **Concept** | **Theorems** |
| Recursive and recursively enumerable sets | If A is recursive, then ~A is also recursive  If A and ~A are r.e. then A is recursive  The decision problem x A is:   * Decidable iff A is recursive * Semidecidable iff A is r.e. * Undecidable iff A is nonrecursive   If A is r.e. then there is a Turing machine that will enumerate the members of A |
| Halting | The halting problem is undecidable |
|  |  |
|  |  |