ကွန်ပြူတာဘာသာဗေဒ

သတိ ။ စာကိုမကျက်ပါနှင့်။
နားလည်အောင်ဖတ်ပြီးစဉ်းစားပါ။

ဘာသာဗေဒ (Linguistic)

- ဘာသာစကားဟူသည် လောကအကြောင်းအရာ ခံစားဖြစ်ပျက်မှုများကို သင်္ကေတများဖြင့်လည်းကောင်း၊ အသံထွက်များဖြင့်လည်းကောင်း ဖော်ပြဆက်သွယ်ခြင်းဖြစ်သည်။

- ဘာသာစကားမှာ အခြေခံအားဖြင့် ၂ ပိုင်းရှိပါသည်။
  - လောကအကြောင်းအရာများ (ဝါ) အရာဝတ္ထုများ
  - ခံစားဖြစ်ပျက်မှုများ (ဝါ) ပြုမူလုပ်ဆောင်မှုများ

- သင်္ကရှိ၊ သင်္ကမဲ့၊ ဒြပ်ရှိ၊ ဒြပ်မဲ့ အရာများကို နာမ် (Noun) ဟုခေါ်ပြီး ပြုမူလုပ်ဆောင်မှုများကို ကြိယာ (Verb) ဟုခေါ်ပါသည်။

- ထို့ကြောင့် ဘာသာစကားဟူသည် နာမ် (Noun) နှင့် ကြိယာ (Verb) တို့ပေါင်းစပ်ဖွဲ့စည်းထားခြင်းဖြစ်ပါသည်။

Natural Languages

- လူသားတို့သည် ဘယ်အရာများသည် ဘယ်လိုဖြစ်ပျက်နေကြသည်ကို ဖော်ပြဆက်သွယ်ရာမှ ဘာသာစကား စတင်ဖြစ်ပေါ်လာပါသည်။
- တစ်နည်း ဘာသာစကားဟူသည် လူသားအချင်းချင်း ဖော်ပြဆက်သွယ်ဖို့ဖြစ်သည်။ သို့သော် ဘာသာစကားကို စက်ကိရိယာ (Machine)များကို ဖော်ပြဆက်သွယ်လို့ ရနိုင်မလား။
- ဒီမေးခွန်းကို Allan Turing က ပွင့်ပွင့်လင်းလင်းပဲ ဖြေသည်။ ရပါသည်။ ဒါကို Turing Test ဟုခေါ်သည်။
- Machine ကို လူတစ်ယောက်ပြောသလိုပြောကြည့်၊ Machine က လူတစ်ယောက်ပြန်ဖြေသလိုဖြေရင် Turing Test အောင် (Pass) သည်။
- ဒီလို လူ့ဘာသာစကားကို စက်ကိရိယာများကို ဖော်ပြဆက်သွယ်ခြင်းကို Natural Language Processing ဟု ဝိသေသပြုကြသည်။

Real World Model

- မိန်းကလေးတစ်ယောက် (နာမ်) ပြေးနေသည် (ကြိယာ)။
- ဒီစာကို ဖတ်ကြည့်ရင် မိန်းကလေးတစ်ယောက် ပြေးနေပုံပေါ်လာမည်။ သို့သော် သေသေချာချာ စဉ်းစားကြည့်။ ပြေးနေတဲ့မိန်းကလေးက အရပ်ရှည်လား၊ အရပ်ပုလား။ အသားဖြူလား၊ ညိုလား။ ဆံပင်ရှည်လား။
- ရေးထားတဲ့အထဲမှာ ဒါတွေတစ်ခုမှမပါ။ ဒါတောင် မိန်းကလေးတစ်ယောက် ပြေးနေပုံကို ပုံဖော်ကြည့်လို့ရသည်။
- ဘာသာစကားဟူသည် ဖော်ပြဆက်သွယ်ရုံသက်သက်မဟုတ်။ Real World Model တစ်ခုလည်းဖြစ်သည်။
- တစ်နည်း ဘာသာစကား (Language) ဟူသည့် အပြင်လောကကြီး (Real World) ကို စိတ်ထဲမှာ Mental Model တစ်ခုကို တည်ဆောက်နိုင်စွမ်းရှိသည်။ (စကားချပ်။ ပညာရှင်များက စာဖတ်ဖို့တိုက်တွန်းခြင်းသည် Mental Model ကိုပိုကောင်းအောင်တည်ဆောက်ဖို့ ဖြစ်သည်။ Mental Model ကောင်းလျှင်ကောင်းသလို စဉ်းစားဆင်ခြင်လာနိုင်သည်။)

- Real World Model တစ်ခုမှာ
  - Abstraction (မိန်းကလေးတစ်ယောက် ပြေးနေသည်)
  - Generalization (လူသည် သေမျိုးဖြစ်သည်)
  - Substitution (မောင်မောင်သည် တနေ့ သေမည်ဖြစ်သည်)
  - Recursion (မလုပ်လို့ မဖြစ်တဲ့ အလုပ်တွေကို လုပ်လို့ဖြစ်အောင် လုပ်ကိုလုပ်ရမည်)
  - Condition (မိုးရွာလျှင် ရေချိုးမည်)
- **Real World Model တစ်ခုသည် ပြောဆိုဆက်သွယ်ဖို့ (Communication) သာမက ပုံဖော်တွေးတောဖို့ (Conceptualization) အတွက်ပါဖြစ်သည်။**

Simple or Precise

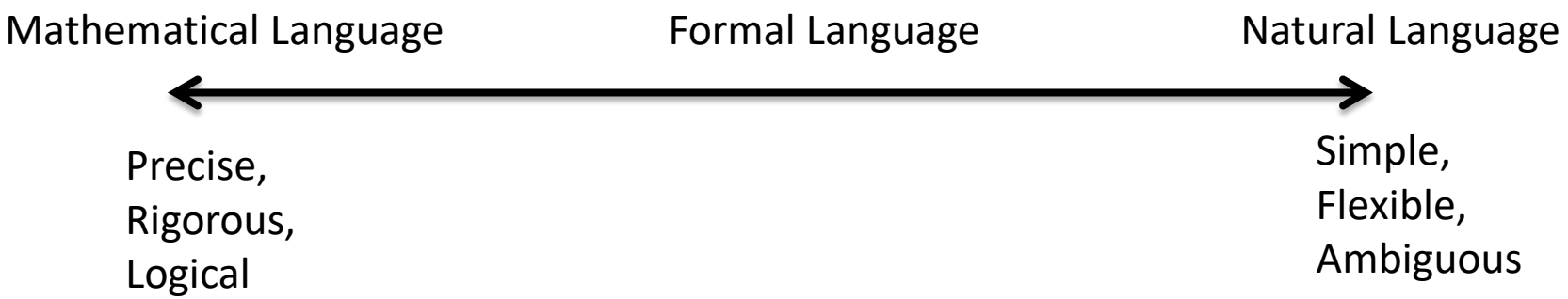
- Real World Model ဟူသည် လောကကြီးအား အရေးကြီးတာကိုပဲ ပိုမိုရိုးရှင်းစွာဖော်ပြခြင်းဖြစ်သည်။
- Real World Model ကို ဖော်ပြရာမှာ
  - ပိုမိုရိုးရှင်းစွာဖော်ပြမလား (More Simply)
  - ပိုမိုတိကျစွာဖော်ပြမလား (More Precisely)
- နေတော်တော်ပူသည်။ (သို့သော် ဘယ်လောက်ပူလဲ။ ၄၅ ဒီဂရီလား။ ၃၅ ဒီဂရီလား)။ လူအများကြီးလာသည်။ (ဘယ်လောက်လဲ)
- လူ့ဘာသာစကားသည် ရိုရှင်းခြင်း (Simple) ကို အဓိကထားသည်။ ပြဿနာသည် သိပ်မတိကျခြင်း (Vague and Ambiguous) ဖြစ်သည်။
- လူနှင့်လူခြင်း ပြောဆိုဆက်သွယ်ရာမှာ ပြဿနာမရှိသော်လည်း Machine ကို ပြောရင် ပြဿနာဖြစ်လာသည်။ ဒါကြောင့် Natural Language များကို အခုထိ သိပ်အသုံးမတွင်ကျယ်သေးတာဖြစ်သည်။
- ဒါက ဘာလို့လဲဆိုတော့ Machine မှာ Common Sense လို့ခေါ်တဲ့ ဆက်စပ်စဉ်းစားနိုင်စွမ်း သိပ်မရှိသေး။ လူ့ဘာသာစကားကိုနားလည်ဖို့သည် ဆက်စပ်စဉ်းစားနိုင်စွမ်းရှိမှဖြစ်သည်။
- ပုံမှန်လူ့ဘာသာစကားများက လူလူချင်းပြောဆိုဆက်သွယ်ဖို့ အဓိကဖြစ်၍ ရိုးရှင်းခြင်းကို ဦးစားပေးသည်။ သိပ်မတိကျသော်လည်း ပြဿနာမရှိ။
- သို့သော် ပိုပြီးတိတိကျကျ စဉ်းစားတွေးခေါ်ဖို့ လိုအပ်လာသည့်အခါ စဉ်းစားတွေးခေါ်ဖို့ ဘာသာစကားတစ်ခု သက်သက်လိုအပ်လာသည်။

Mathematical Languages

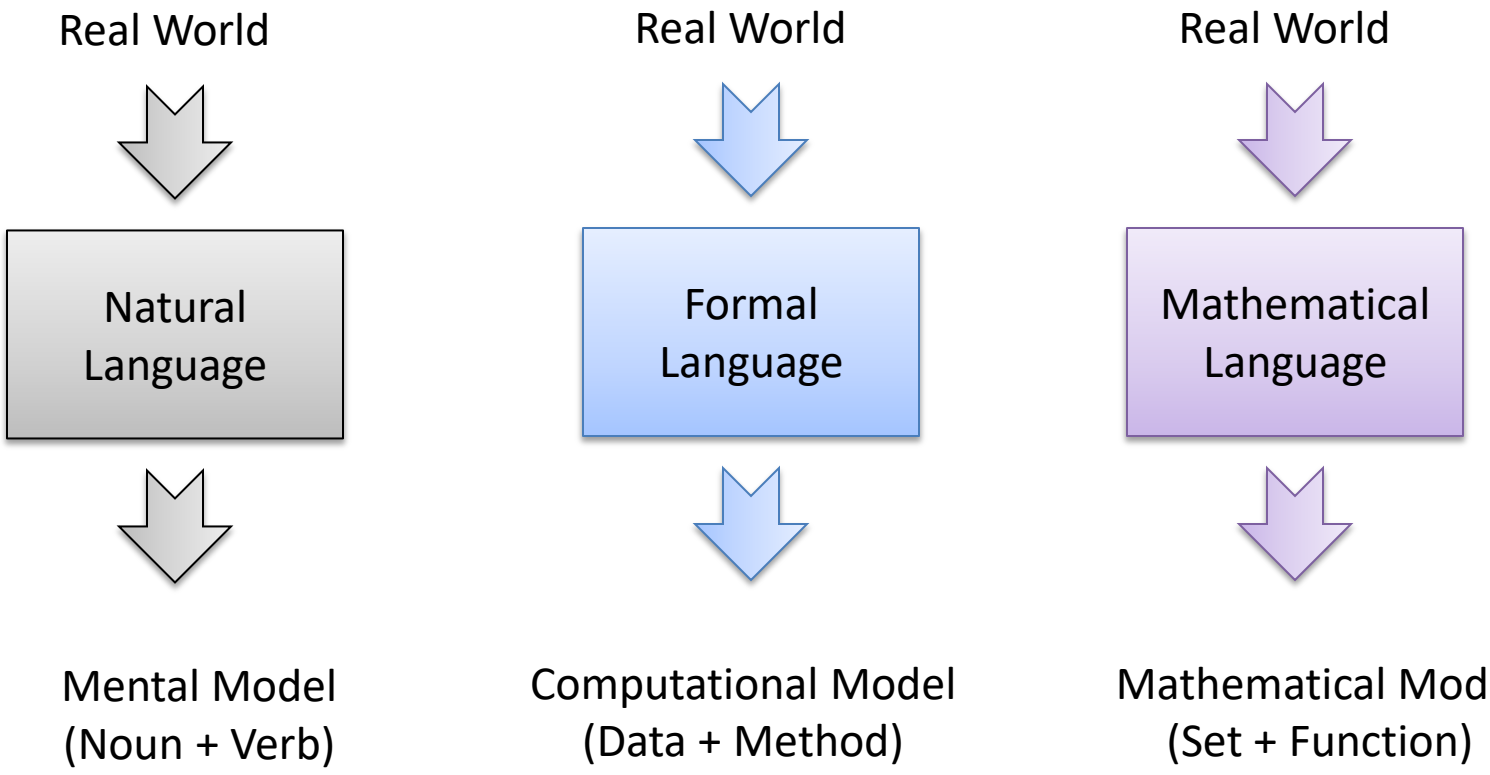
- ပုံမှန်လူ့ဘာသာစကားများအပြင် စဉ်းစားတွေးခေါ်ဖို့ တိတိကျကျ ဖော်ပြနိုင်သော ဘာသာစကားတစ်ခုလဲ ပေါ်ထွက်လာသည်။
- ဒီဘာသာစကားကို သခ်ၤာဟု ခေါ်သည်။
- သခ်ၤာ သည် ဘာသာစကားတစ်ခုဖြစ်သည်။ သို့သော် သူက ပြောဆိုဆက်သွယ်ဖို့မဟုတ်။ တိတိကျကျ စဉ်းစားတွေးခေါ်ဖို့ ဖြစ်သည်။
- သခ်ၤာသည် ဘာသာစကားတစ်ခုဖြစ် သည့်အတိုင်း သူ့မှာလည်း အရာဝတ္ထုများ (Nouns) နှင့် ပြုမူလုပ်ဆောင်မှုများ (Verbs) ဖြင့်ပေါင်းစပ်ဖွဲ့စည်းထားသည်။
- သခ်ၤာ၏ အရာဝတ္ထုများကို အစုများ (Sets) ဖြင့် ဖော်ပြပြီး ပြုမူလုပ်ဆောင်မှုများကို ဖန်ရှင်များ (Functions) ဖြင့် ဖော်ပြသည်။
- Mathematical Language မှာ Add (ပေါင်းခြင်း) သည် လုပ်ဆောင်မှု (Function) တစ်ခုဖြစ်ပြီး ကိန်းများ၏ ဆက်သွယ်ချက် (Relations) ကို ဖော်ပြပေးသည်။
- A Adds B => A + B (Mathematical Language)
- Natural Language မှာ Eat (အစာစားခြင်း) သည် လုပ်ဆောင်မှု (Verb) တစ်ခုဖြစ်ပြီး အရာဝတ္ထုများ၏ ဆက်သွယ်ချက် (Relations) ကို ဖော်ပြပေးသည်။
- A Eats B => A Eat B (Natural Language)
- Mathematical Language ၏အဓိကသည် Real World ကို Mathematical Model တည်ဆောက်ဖို့ဖြစ်သည်။

Formal Languages

- ကွန်ပြူတာ (Machine)များကို ပြောဆိုဆက်သွယ်ဖို့ လိုအပ်ချက်ကြောင့် ကွန်ပြူတာဘာသာစကားများ ပေါ်ထွက်လာသည်။
- ဒီဘာသာစကားများကို Formal Languages များဟု ဝိသေသပြုကြသည်။
- Formal Language သည် ဘာသာစကားတစ်ခုဖြစ် သည့်အတိုင်း သူ့မှာလည်း အရာဝတ္ထုများ (Data) နှင့် ပြုမူလုပ်ဆောင်မှုများ (Procedures or Methods) ဖြင့်ပေါင်းစပ်ဖွဲ့စည်းထားသည်။
- Formal Languages များသည် Mathematical Languages များလောက်လဲ Rigorous မဖြစ်၊ Natural Languages များထက်လဲ ပိုတိကျသည်။
- Formal Language ၏အဓိကသည် Real World ကို Computational Model တည်ဆောက်ဖို့ဖြစ်သည်။



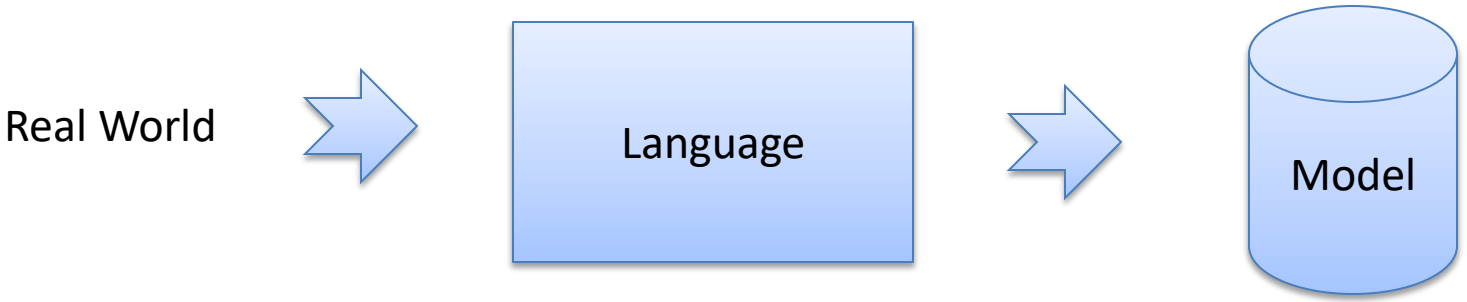
Modeling



Model ဆိုတာ Abstraction of Real World ကိုခေါ်တာဖြစ်ပါသည်။

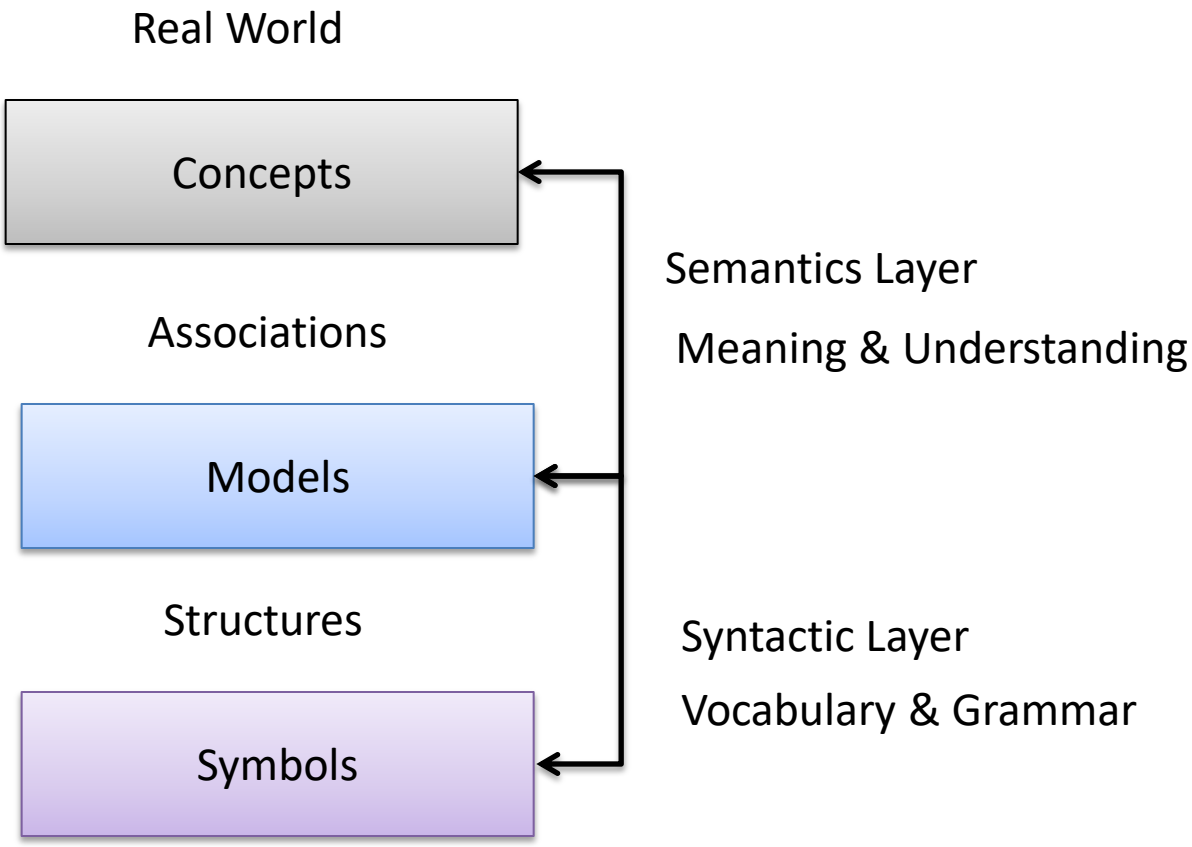
Modeling

Input (Analysis). For example: Reading, Listening.



Output (Synthesis). For example: Writing, Speaking.

Linguistic Layers



Learning Languages

- ဘယ်ဘာသာစကားကို သင်သင်၊ သခ်ၤာ (Math) ပဲဖြစ်ဖြစ်၊ အင်္ဂလိပ်စာ (English) ပဲဖြစ်ဖြစ်၊ ပရိုဂရမ်မင်း (Programming) ပဲဖြစ်ဖြစ် Concepts (Meanings), Models နှင့် Syntax တွေက အရေးကြီးပါသည်။

- Syntax က Symbols တွေ၊ Structures တွေကို သုံးပြီး Model တစ်ခုကို တည်ဆောက်ပါသည်။ Syntax မှာ ဝေါဟာရများ (Vocabulary) နဲ့ Grammar များပါဝင်ပါသည်။

  - ကျွန်တော် ပန်းသီးကို ကြိုက်တယ်။
  - I like apple.
  - 나는 사과를 좋아한다
  - F: x ->y => f(x, y) where F is affinity function with x = {I} and y = {apple}
  - var Like = fruit == 'Apple' && person == 'I';

- Semantics မှာက Syntax တွေ၊ Model တွေကို Real World နဲ့ Associate လုပ်ပြီး Concept (Meaning) ကို ရှာဖွေပါတယ်။

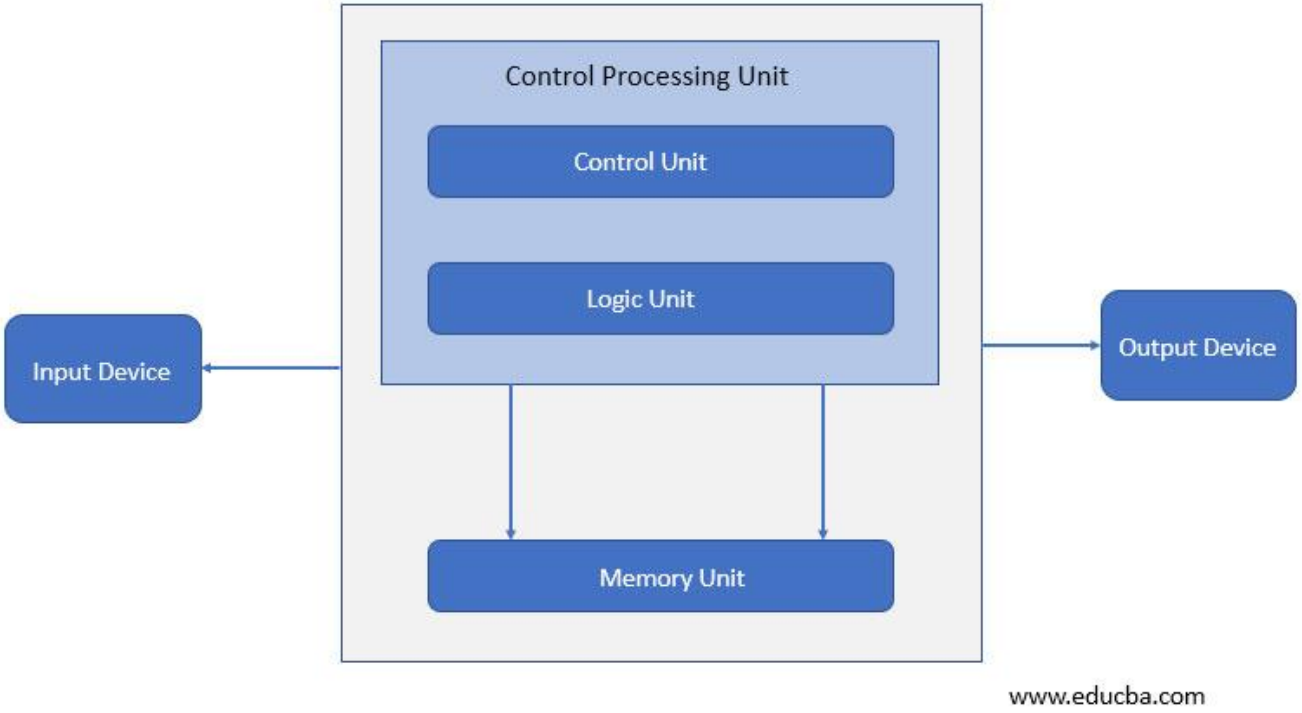
  - I = ကျွန်တော် = 나 = x = person
  - Apple = ပန်းသီး = 사과 = y = fruit
  - Like = ကြိုက် = 좋아한다 = Affinity = Like

Computational Models

- အခြေခံအားဖြင့် Computational Model ၃ မျိုးရှိပါသည်။
  - Operational Computation Model
  - Denotational Computation Model
  - Logical Computation Model
- နောက်ထပ် ထပ်တိုးသော Model များလည်းရှိပါသည်။ ဥပမာ၊ Quantum Computation Model နှင့် Parallel Computation Model။ ဒါတွေကိုတော့ ဒီမှာ မပြောတော့ပါ။

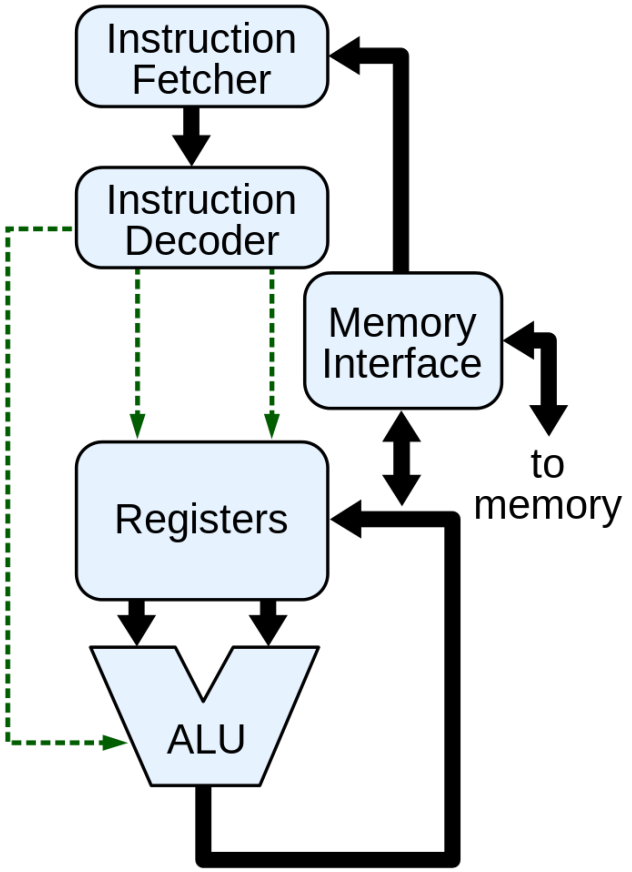
Operational Computation Model

- Operational Computation Model သည် Real Physical Computer ရဲ့ Operations များကို အခြေခံထားသော Computational Model ဖြစ်သည်။



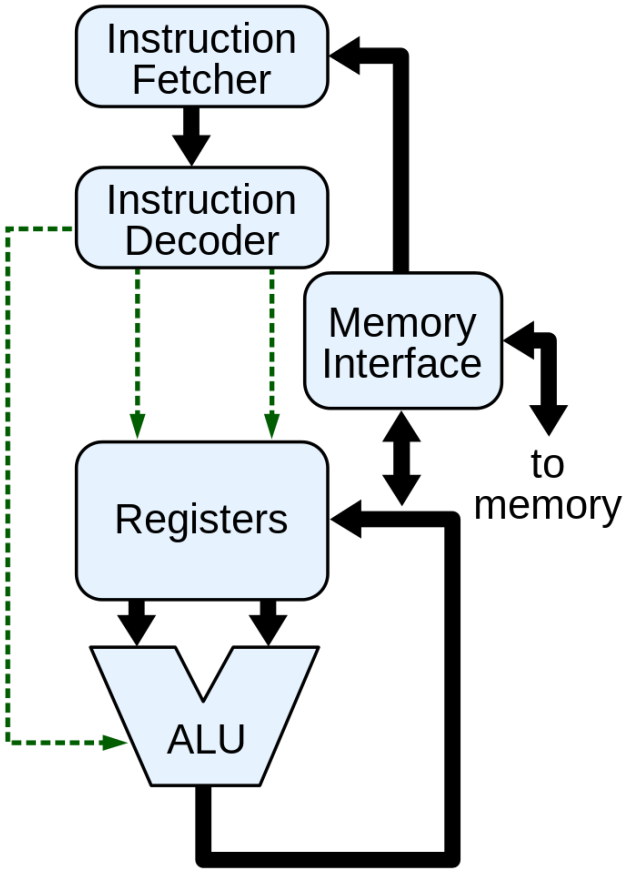
Fetch-Decode-Execute Cycle

- Real Physical Computer က ဘာလုပ်လဲ ဆိုတော့ Fetch-Decode-Execute Cycle ကိုလုပ်ပါသည်။
- Data နဲ့ Instruction (Code) များကို Memory ပေါ် Load လုပ်ပါသည်။
- Memory မှ Code များကို Fetch လုပ်ပြီး Decode (Interpret) လုပ်ပါသည်။
- ပြီရင် Instruction အတိုင်း Execute လုပ်ပြီး Program မပြီးမချင်း လုပ်ဆောင်ပါသည်။



Sample Instructions

```
A = LOAD(X)
1001001011001010 (Load Instruction)
1011011011011011 (Memory Address)
0000000000000110 (Register A)

B = LOAD (Y)
1001001011001010 (Load Instruction)
1011010011010011 (Memory Address)
0000000000000111 (Register B)

C = ADD(A,B)
1111011011001010 (Add Instruction)
0000000000000110 (Register A)
0000000000000111 (Register B)
0000000000000101 (Register C)
```



Semantics Gap

- ကျွန်တော်တို့ အခု Real Physical Computer က ဘာလုပ်လဲဆိုတာ နဲနဲသဘောပေါက်သွားပြီ။
- သို့သော် ပြဿနာက အခုမှစသည်။
- ကျွန်တော်တို့ ဖြေရှင်းချင်တာသည် Real World Problem များဖြစ်သည်။ ဥပမာ၊ Library System၊ Inventory Management System စသည်ဖြင့်။
- Library မှာရှိတာက စာအုပ်တွေ၊ စာရေးဆရာတွေ၊ စာအုပ်ငှားသူတွေ၊ စာအုပ်အမျိုးအစား ဝတ္ထု၊ စုံထောက်၊ အချစ်၊ လျှို့ဝှက်သည်းဖို စသည်။
- Real Physical Computer ကနားလည်တာက 1001001011001010 1011011011011011 0000000000000110.
- ဒီနှစ်ခုကို ဘယ်လို Associate လုပ်ကြမလဲ။

Narrowing the Gap

- ဒီပြဿနာကို ရှင်းဖို့ ကျွန်တော်တို့ Operational Computation Model အတွက် Real World ကို ပိုပြီး ဖော်ပြလာနိုင်ဖို့ လိုလာပါသည်။ ဒါကို လုပ်ဖို့က Higher Level Abstraction Layer များ ထပ်ထည့်ခြင်းပါ။

- ဒီလိုလုပ်ရာမှာ ၃ နည်းရှိပါသည်။

- Compiler (Translator)
  - Higher Level Abstract Language များကို Machine Level Language သို့ ပြောင်းပေးပါသည်။

- Interpreter
  - Abstract Virtual Computing Layer (Runtime Environment) ကို ပြုလုပ်ပြီး Higher Level Language များကို တိုက်ရိုက် Execute လုပ်ပါသည်။

- Hybrid
  - Runtime Environment (Machine dependent) များကို ပြုလုပ်ပြီး Different Machines များကို Target ထားပါသည်။ ဥပမာ၊ Java Runtime, .Net Runtime

  - Intermediate Compiler က Higher Level Language (Machine Independent) များကို သက်ဆိုင်ရာ Runtime Environment များ၏ Virtual Machine Level Language (Intermediate Code) သို့ ပြောင်းပေးပါသည်။

  - Intermediate Code များကို Runtime Environment မှာ တိုက်ရိုက် Execute လုပ်ပါသည်။

Machine ပေါ်မှာ တိုက်ရိုက် Execute လုပ်တာကို Native Code ဟုခေါ်ပြီး Runtime Environment ပေါ်မှာ Execute တာကို Managed Code ဟုခေါ်ကြသည်။

History of Higher Level Languages

- Programming Languages များ အထူးသဖြင့် Imperative Languages (Operational Computation Model) များသည် ကွန်ပြူတာသမိုင်း တလျှောက် အဆင့်ဆင့် ပြောင်းလဲလာပါသည်။
- အစဦးဆုံး ကွန်ပြူတာများ အတွက် Machine Instruction များကို Punch Card ပေါ်မှာ တိုက်ရိုက်ရေးရပါသည်။ ထို့ကြောင့် သိပ်ရှုပ်ထွေးသော System များကို မရှင်းနိုင်ပါ။
- ထို့နောက် Assembly Language ဆိုပြီး Machine Instruction များကို လွယ်ကူအောင် ပြုလုပ်ထားသော Language များပေါ်လာပါသည်။ သို့သော် Semantics အားဖြင့် သိပ်မရှုပ်ထွေး။ (1001101 အစား LOAD ဆိုပြီး ဖြစ်လာရုံမျှသာ။)
- ထို့နောက် First Generation Language များပေါ်လာသည်။
- ပြီးတော့ Second Generation Language များပေါ်လာသည်။ ပြီးတော့ Third Generation ၊ Fourth Generation။
- Language Level မြင့်လာလေလေ၊ Real World ကို ပိုပြီး Describe လုပ်ပြီး Process လုပ်ရတာ လွယ်လေလေဖြစ်သည်။

Imperative Languages

- Imperative Languages များသည် Operational Computation Model များ အပေါ် အခြေခံထားသည်။
- Imperative Languages များတွင် Data နှင့် Instruction သည် အခရာဖြစ်သည်။ ဘာသာဗေဒအားဖြင့် Noun နှင့် Verb များနှင့် အလားတူသည်။
- ထို့အတူ Imperative Languages များတွင် Program (Processing) Flow ရှိသည်။ စာများကို Left To Right or Right To Left, Top to Bottom ဖတ်ရပုံနှင့် အလားတူသည်။
- Imperative Languages များ၏ Program (Processing) Flow မှာ
  - Sequence
  - Selection
  - Iteration
- Imperative မဟုတ်သော Languages များသည် Sequence, Selection, Iteration ကို လိုက်နာချင်မှ လိုက်နာမည်ဖြစ်သည်။ သို့သော် Imperative Languages များ အားလုံး Sequence, Selection, Iteration ကို လိုက်နာကြသည်။

Operational Formal Languages

- Imperative Languages များ၏ Semantics နှင့် Syntactic ပြဿနာများကို ရှင်းဖို့ Operational Formal Language များပေါ်လာပါသည်။

- Formal Language များမှာ အခြား Language များနည်းတူ Linguistic Layer ၂ ခု ရှိပါသည်။
  - Semantics Layer
  - Syntactic Layer

- Formal Semantics များက Semantics Layer ကို ဖြေရှင်းပေးပါသည်။ Formal Semantics များသည် Real World ကို Formal Computation Model နဲ့ Associate လုပ်ပြီး အဓိပ္ပါယ် Concept ကို ဖော်ဆောင်ပါသည်။

- Formal Grammar နှင့် Syntax များက Syntactic Layer ကိုဖြေရှင်းပေးပါသည်။ Formal Grammar များက Symbols, Notations, Grammar Rules တွေသုံးပြီး Model တစ်ခုကို Describe လုပ်၍ Process လုပ်ပါသည်။

Formal Semantics

- အပြင်လောကကြီး (Real World) ကို ဘယ်လို ဖော်ပြကြမလဲ။ ဒါသည် Semantics Layer ၏ အဓိက မေးခွန်းဖြစ်သည်။
- ဥပမာ၊ Library System ဆိုပါတော့။ Library မှာရှိတာက စာအုပ်တွေ၊ စာရေးဆရာတွေ၊ စာအုပ်ငှားသူတွေ၊ စာအုပ်အမျိုးအစား ဝတ္ထု၊ စုံထောက်၊ အချစ်၊ လျို့ဝှက်သည်းဖို စသည်တို့ ဖြစ်သည်။
- လူသုံးဘာသာစကားမှာတော့ ဒါကို Nouns နှင့် Verbs များဖြင့် ဖော်ပြကြသည်။ သို့သော် Operational Computation Model မှာရှိတာက Data နှင့် Instructions များဖြစ်သည်။
- ကျွန်တော်တို့ အကြမ်းဖျဉ်းပြောနိုင်တာက Nouns များ (စာကြည့်တိုက်အသင်းဝင်၊ စာအုပ်၊ စာရေးဆရာ) ကို Data ဖြင့် ဖော်ပြနိုင်ပြီး Verbs (ငှားခြင်း၊ အပ်ခြင်း၊ ရှာခြင်း) များကို Instructions ဖြင့်ဖော်ပြလို့ ဖြစ်နိုင်သည်။ သို့သော် ဘယ်လို ဖော်ပြမလဲ။
- ထားပါတော့။ Member Loan Book, Member Search Book, Member Search Author.
- ဒါကို More Formally ပြောချင်တယ်ဆိုရင်တော့ Loan(Member, Book), Search(Member, Book), Search(Member, Author) ဖြစ်လာသည်ပေါ့။ Loan, Search များသည် Instructions များဖြစ်ပြီး Member, Book, Author များသည် Data ဖြစ်လာသည်။
- သို့သော် ဒါဖြင့် လုံလောက်ပြီလား ဆိုတော့၊ မလုံလောက်သေးပါ။

More Semantics

- အပြင်လောကကြီး (Real World) ကို ပိုပြီး ပြည့်ပြည့်စုံစုံ ဖော်ပြဖို့ အောက်ပါ အချက်များလိုအပ်ပါသည်။
  - Abstraction (မိန်းကလေးတစ်ယောက် ရှိသည်)
  - Generalization (လူသည် သေမျိုးဖြစ်သည်)
  - Specialization (မောင်မောင်သည် လူတစ်ယောက်ဖြစ်သည်)
  - Substitution (မောင်မောင်သည် တနေ့ သေမည်ဖြစ်သည်)
  - Recursion (မလုပ်လို့ မဖြစ်တဲ့ အလုပ်တွေကို လုပ်လို့ဖြစ်အောင် လုပ်ကိုလုပ်ရမည်)
  - Condition (မိုးရွာလျှင် ရေချိုးမည်)

- ဒါတွေကို ဖော်ပြဖို့ သာမန် Data နှင့် Instructions သက်သက်များဖြင့် မလုံလောက်တော့ပါ။

- ခေတ်အဆက်ဆက် Programming Languages များကို Semantically Rich ဖြစ်ဖို့ ကြိုးစားလာခဲ့ကြသည်။

- နောက်ဆုံး မျိုးဆက်သည် Object Oriented Concept ဖြစ်သည်။

- ကျွန်တော်တို့ Programming Languages များရဲ့ Semantics များကို ကြည့်ကြပါစို့။

Procedural Semantics

- Procedural Languages များသည် Object Oriented Concept များမပေါ်ခင်က ခေတ်စားခဲ့သည်။ အခုတိုင် အသုံးပြုဆဲဖြစ်သည်။ ဥပမာ၊ C Language.

- Semantics Problems များကို ဖြေရှင်းဖို့ Procedural Languages များသည် အောက်ပါ အချက်များကို Language Semantics အဖြစ် သတ်မှတ်ခဲ့သည်။

- Data Abstraction and Structures
  - Abstraction, Generalization တို့ကို ဖြေရှင်းပါသည်။

- Data Scope and Accessibility
  - Specialization and Substitution တို့ကို ဖြေရှင်းပါသည်။

- Method (Functional and Procedural) Modularity
  - Recursion တို့ကို ဖြေရှင်းပါသည်။

Procedural Semantics တွင် Data နှင့် Instructions များသည် သက်သက်စီ ဖြစ်သည်။

ADT (Abstract Data Type)

- Library Member သည် လူတစ်ယောက်ဖြစ်သည်။ သို့သော် သူ့ရဲ့ အကြောင်းအရာ အားလုံးကို ဖော်ပြစရာမလို။
- ထို့ကြောင့် Library Member ရဲ့ သက်ဆိုင်ရာ အချက်အလက်များကို ဖော်ပြရင် လုံလောက်သည်။ ဒါသည် Abstraction ဖြစ်သည်။
- ဒါ့အပြင် Library Member ရဲ့ ADT သည် Library Member အားလုံးကို ကိုယ်စားပြုသော Data Structure ဖြစ်လာသည်။ ။ ဒါသည် Generalization ဖြစ်သည်။

```
Member =
{
      Name : Text(30),
      Age : Integer,
      DOB : Date,
      Status : Integer,
      Gender : Text(1)
};
```

Scope and Accessibility

- Robert သည် Library Member တစ်ယောက်ဖြစ်သည် ဆိုပါတော့။

```
Member Robert;
```

- ဒါသည် Specialization ဖြစ်သည်။ Instantiation လို့လဲ ပြောကြသည်။

```
Member Tom;

Tom := Robert;
```

- ဒါသည် Substitution ဖြစ်သည်။

```
Member Tom;
Book Bible;

Procedure Loan(Member x, Book b)
{
          //Do Loaning
}

Loan(Tom, Bible);
```

ဒါ့အပြင် Procedural Language များတွင် Global နှင့် Local Scope ဆိုပြီးရှိပါသည်။

Tom သည် Global ဖြစ်ပြီး x သည် Local ဖြစ်သည်။

Instantiation နှင့် Substitution ကို လုပ်ဆောင်သွားသည်။

Method Modularity

- Modularity က Recursion ကို ဖြေရှင်းဖို့ဖြစ်သည်။
- စာအုပ်များကို စာမျက်နှာများဖြင့် တည်ဆောက်ထားပြီး စာမျက်နှာများကို စာပိုဒ်များဖြင့် လည်းကောင်း၊ စာပိုဒ်များကို စာကြောင်းများဖြင့် လည်းကောင်း၊ စာကြောင်းများကို စာလုံးများဖြင့် လည်းကောင်း၊ စာလုံးများကို အက္ခရာများဖြင့် လည်းကောင်း တည်ဆောက်ထားသည်။
- ထို့အတူ စာအုပ်ငှားခြင်းမှာ စာအုပ်စာရင်းမှတ်ခြင်း၊ စာကြည့်တိုက်ကတ်ပြားကို မှတ်ခြင်း စသည်တို့ပါဝင်သည်။
- Function နှင့် Procedure များကို ခြုံ၍ Method ဟုခေါ်သည်။
- ရှုပ်ထွေးသော Method များကို ပိုသေး၊ ပိုရှင်းသော Method လေးများကို စုပေါင်းပြီး Modularize လုပ်ခြင်းအားဖြင့် Recursion ကို Support လုပ်ပါသည်။

```
Procedure Loan(Member x, Book b)
{
          Boolean ok = Checkout(b);
          Record(x);
}

Function Checkout (Book b)
{
          //Do Checkout
          Return result;
}

Procedure Record(Member x)
{
          //Do Record
}
```

Object Oriented Semantics

- ၁၉၆၀ နှောင်းပိုင်းနဲ့ ၁၉၇၀ အစောပိုင်းလောက်မှာ Object Oriented Concept များ ခေတ်စားလာခဲ့သည်။
- Object Oriented Semantics များသည် Real World ကိုပိုပြီး ပြည့်ပြည့်စုံစုံ ဖော်ပြလာနိုင်သည်။
- Encapsulation
  - Data နှင့် Method များကို ပေါင်းစည်းထားသည်။ မော်တော်ကားတစ်ခုကို ဖော်ပြမည် ဆိုရင် ကားအမျိုးအစား၊ အရောင်၊ အရွယ် (Properties) အပြင် မောင်းနှင်ခြင်း (Actions) စသည့် တို့ကို တပါတည်း ပေါင်းစပ်ထားသည်။ Encapsulation သည် အမှန်တော့ Simulation (e.g. Game)များကို ဖော်ပြရာမှာ ပိုအဆင်ပြေလာသည်။
- Inheritance
  - Parent နှင့် Child Relationship များကို ဖော်ပြလာနိုင်သည်။ Inheritance သည် Subset (Child)/Superset (Parent) သဘောတရားကို အခြေခံသည်။ သက်ရှိများ > နို့တိုက်သတ္တဝါများ > ပရိုင်းမိတ်များ > လူများ > ကျောင်းသားများ။
- Composition
  - အမျိုးမတူသော အရာတို့ပေါင်းစပ်ထားခြင်းကို ဖော်ပြလာနိုင်သည်။ ဥပမာ၊ ကားကို မှန်များ၊ သတ္တုများ၊ ရာဘာများဖြင့် ပေါင်းစပ်တည်ဆောက်ထားခြင်းဖြစ်သည်။ Inheritance နှင့် Composition ကွာသည်က Inheritance မှာ Parent ၏ Properties နှင့် Actions များကို Child က ယူသုံးနိုင်သည်။ Composition က လက်ထပ်ခြင်းနှင့်တူသည်။ Spouse ပိုင်ဆိုင်သည်များကို သက်ဆိုင်သည်။ ပြီးတော့ ၂ ယောက်ပေါင်းပြီး မရှိသော အရာများကို ဖန်တီးနိုင်သည်။ ကလေးမွေးခြင်း စသည်။
- Accessibility (Information Hiding)
  - လျို့ဝှက်အပ်သည်များကို လျို့ဝှက်ရာ၏။ မလိုအပ်ဘဲ မပြသင့်သည်များကို မပြ။ Parentပဲဖြစ်ဖြစ်၊ Spouseပဲဖြစ်ဖြစ် မလိုအပ်ဘဲ မပြသင့်သည်များကို မပြ။
- Polymorphism
  - Encapsulation, Inheritance, Composition, Accessibility တို့ကို သုံးပြီး အခြေအနေပေါ်မူတည်ပြီး ပြောင်းလဲနိုင်ခြင်းကို လုပ်ဆောင်လာနိုင်သည်။ Object Oriented Semantics များ Popular ဖြစ်လာခြင်းသည် Polymorphism ကြောင့်ဖြစ်သည်။

Object Oriented Semantics တွင် Data နှင့် Instructions များသည် ပေါင်းစည်းထားသည်။

Encapsulation

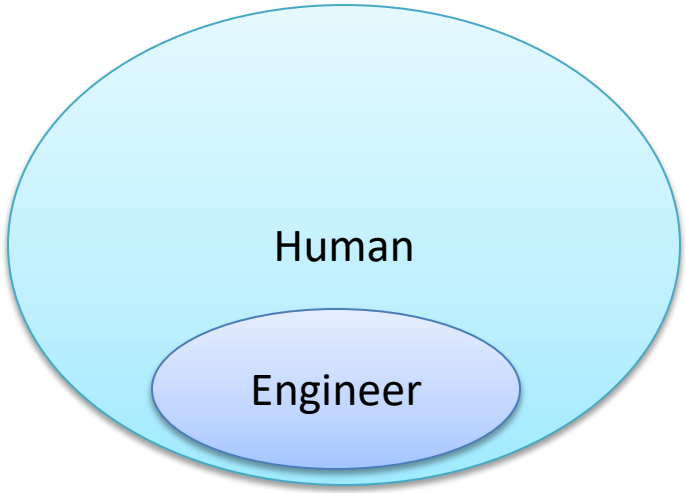
- Encapsulation ၏ အဓိက ရည်မှန်းချက်က Simulation အတွက်ဖြစ်သည်။
- အရာဝတ္ထု (Entity) တစ်ခုကို သူ၏ Properties များသာမက၊ သူ၏ Behaviors များ ကိုပါ တခါတည်း ဖော်ပြသည်။
- ထိုကြောင့် Real World ကို Interactive Model များဖြင့် လွယ်လွယ်ကူကူ ဖော်ပြလာနိုင်သည်။
- Encapsulation ကြောင့် Real World Model များသည် Interactive ပိုဖြစ်လာသည်။
- Encapsulation ကြောင့် Virtual Reality များကို တည်ဆောက်လာနိုင်သည်လို့ ဆိုရင် သိပ်မမှား။

```
Class Student()
{
    Private Integer Id;
    Public Text Name;

     Public Function Enroll(Course c)
     {
          //Do Enrolment
          Return result;
     }

     Public Procedure Withdraw(Course c)
     {
          //Do Withdrawal
          Return result;
     }
}
```

Inheritance

- Inheritance ၏ အဓိက ရည်မှန်းချက်က Subset (Child)/Superset (Parent) အတွက်ဖြစ်သည်။

- အရာဝတ္ထု (Entity) တစ်ခုသည် သူ၏ Superset ရဲ့ Properties များသာမက၊ Behaviors များ ကိုပါ အမွေဆက်ခံသည်။ လူလို့ ပြောရင် အသက်ရှိကြောင်း၊ အစာစားကြောင်း ပါပြောပြီးသားဖြစ်သည်။




```
Class Human()
{
      Private Integer Id;
      Public Text Name;
      Public Integer Gender;

      Public Function Eat (Food f)
      {
      }

      Public Procedure Move(Direction d)
      {
      }
}

Class Engineer()  : Human ()
{
      Private Integer Id;
      Public Skill[] Skills;

      Public Function Build(Building b)
      {
      }

      Public Procedure Invent(Invention i)
      {
      }
}
```

Composition

- Composition ၏ အဓိက ရည်မှန်းချက်က Synthesis အတွက်ဖြစ်သည်။

- အရာဝတ္ထု (Entity) တစ်ခုကို အခြားဘယ် အရာများဖြင့် ပေါင်းစပ်ဖွဲ့စည်း တည်ဆောက်ထားခြင်းကို ဖော်ပြသည်။

```
Class Vehicle()
{
      Private Integer Id;
      Public Mechanism Power;
      Public Maneuver Drive;
      Public Domain Terrain;

      Public Function Move(Direction d, Domain t)
      {
            If (Power >= Empty)
            {
                  If (t == Land)
                      Drive.Move(d);
                  Else If (t == Sea)
                      Drive.Down(d)
                  Else If (t == Air)
                      Drive.Up(d)
            }
      }

}
```

Accessibility

- Accessibility ၏ အဓိက ရည်မှန်းချက်က Only Need to Know Basis အတွက်ဖြစ်သည်။
- မလိုအပ်ဘဲ မပြသင့်သည်များကို မပြ။ ဒီအတွက် ၂ ပိုင်းပါသည်။ တစ်ခုက Data Security အတွက်ဖြစ်သည်။ နောက်တစ်ခုက Complexity Reduction အတွက်ဖြစ်သည်။
- အရာဝတ္ထုတစ်ခုကို အကုန်လုံးမြင်နေရ, ကြားနေရရင် မကောင်း။ Security and Privacy မရှိတော့။
- အရာဝတ္ထုတစ်ခု၏ အတွင်းပိုင်း အလုပ်လုပ်ပုံ အားလုံးကို သိဖို့လဲ မလိုအပ်။ ကားမောင်းဖို့ ကားအင်ဂျင် ဘယ်လို အလုပ်လုပ်လဲ သိစရာမလို။

```
Class Student()
{
      Private Integer Id; //Hide
      Public Text Name; //Show

      Public Function Enroll(Course c)
      {
            //Do Enrolment
            Return result;
      }

      Public Procedure Withdraw(Course c)
      {
            //Do Withdrawal
            Return result;
      }
}
```

Polymorphism

- Object Oriented Semantics ၏အသက်သည် Polymorphism ဖြစ်သည်။
- အခြေအနေပေါ်မူတည်ပြီး အရာဝတ္ထု၏ Properties နှင့် Behaviors များပြောင်းလဲသွားသည်။

```
Var B = New Bird();

If ( Duck)
{
          B = New Duck();
          B.Tweet();  //It will quack
}
Else
{
          B = New Chicken();
          B.Tweet(); //It will chirp
}
```

```
Class Bird()
{
     Private Integer Id;
     Public Text Name;
     Public Function Tweet()
     {
          //Do make sound
      }
}
Class Duck() : Bird()
{
     Private Function Quack()
     {
     }
     Public  Override Function Tweet()
     {
          Quack();
     }
}
Class Chicken() : Bird()
{
     Private Function Chirp()
     {
     }
     Public  Override Function Tweet()
     {
          Chirp();
     }
}
```

Conceptual Semantics

- Language တစ်ခုရဲ့ Semantics သည် Conceptual Level ဖြစ်သည်။
- Operational Semantics များသည် Conceptual Level Semantics များဖြစ်သည်။
- ဆိုလိုသည်က Conceptual Level Semantics များသည် Meaning များ နှင့် Associations များပေါ်ပဲ မူတည်သည်။ Symbols နှင့် Structures များပေါ်မူမတည်။ ဒီအချက်က အင်မတန် အရေးကြီးသည်။

Language Dependent Syntax

ကျွန်တော် ဗိုက်ဆာသည်။

I am hungry.

Independent Conceptual Semantics

Semantics Thinking

- ဆရာကြီး ဦးရာဇတ်ပြောဖူးတာ ရှိသည်။ အင်္ဂလိပ်စာ တော်ချင်ရင် အင်္ဂလိပ်လိုတွေးပါတဲ့။
- ဒီအချက်ကို ကျွန်တော်ကိုယ်တိုင်လဲ ငယ်ငယ်က သဘောမပေါက်။ ကြီးလာမှ နည်းနည်းသိလာသည်။ ဆရာကြီးက Conceptual Semantics ကို ဆိုလိုတာဖြစ်သည်။ ဘာသာစကားတစ်ခု၏ အသက်သည် Conceptual Semantics ဖြစ်သည်။
- မြန်မာတစ်ယောက် တွေးပုံနှင့် အင်္ဂလိပ်တစ်ယောက် တွေးပုံမတူပါ။ ဒါကြောင့် အင်္ဂလိပ်စာကို မြန်မာလို တွေးပြီးပြောရင် သိပ်အလုပ်မဖြစ်။ အင်္ဂလိပ်လို တိုက်ရိုက်တွေးနိုင်မှ ဖြစ်သည်။
- ထို့အတူ Programming Languages များ၏ အသက်သည် သူတို့၏ သက်ဆိုင်ရာ Semantics များဖြစ်သည်။
- Procedural Programming ကို နားလည်တတ်ကျွမ်းသည်၊ Object Oriented Programming ကို နားလည်တတ်ကျွမ်းဆိုတာသည် C, C++, Java တို့ကို သိတာကို ဆိုလိုတာ မဟုတ်ပါ။
- Procedural Semantics ၊ Object Oriented Semantics များကို သုံးပြီး Real World ကို ဖော်ပြနိုင်၊ ပြဿနာများကို ဆက်စပ်တွေးခေါ်ပြီး ဖြေရှင်းနိုင်ခြင်းကို ဆိုလိုတာဖြစ်သည်။
- ဒါဆိုရင် C, C++, Java တို့ကို မသိ၊ မသုံးဘဲ Programming ရေးလို့ ရလားလို့ မေးရင် ရသည်လို့ ဖြေရမှာပါ။ ဥပမာ၊ Software Architect, Project Manager, Solution Architect Level ရောက်သွားတဲ့ သူတွေက Code တွေသိပ်မရေးတော့ပါ။ သို့သော် သူတို့က Millions of Line of Code ရှိတဲ့ Software တွေကို Build လုပ်နိုင်ပါသည်။ ဘယ်လိုလုပ်လဲ? Thinking ကြောင့်ဖြစ်သည်။
- The Essence of any Language is thinking.

Visual Languages (Models)

- Language တွေတစ်ခုမှ မသုံးဘဲ Thinking လုပ်လို့ရလား ဆိုရင် ရပါသည်လို့ ဖြေရမည်။
- Engineering မှာ Language များထက် Visual Drawings or Diagrams များကို ပိုသုံးပါသည်။
- Procedural Semantics များအတွက် Data Flow Diagram, Flow Chart စသည့် Visual Diagrams များကို သုံးပါသည်။
- https://en.wikipedia.org/wiki/Data-flow_diagram
- Object Oriented Semantics များအတွက် UML (Unified Modeling Language) Diagram များကို သုံးပါသည်။
- https://en.wikipedia.org/wiki/Unified_Modeling_Language
- Procedural Semantics များရော၊ Object Oriented Semantics များပါ Programming Language Independent ဖြစ်ပါသည်။

Systems Thinking

- Semantics ၏ အဓိက တာဝန်သည် Real World ကို ပိုပြီးပြည့်စုံအောင် ဖော်ပြပြီး Concept တည်ဆောက်ဖို့ ဖြစ်သည်။
- ပြီးတော့ ဒီ Concept ကို အခြေခံပြီးတော့ ပြဿနာများကို ဖြေရှင်းကြသည်။
- သို့သော် ပြဿနာရှိသည်က Real World ဆိုတာကြီးဖြစ်သည်။ Real World ဆိုတာ လောကကြီးတစ်ခုလုံးကို ဆိုလိုတာလား။ ကျွန်တော်က Real World ကို ခဏခဏ သုံးခဲ့ပါသည်။ အခုမေးစရာရှိလာပါပြီ။ ဘာကြီးတုံး။ Real World ဆိုတာ?
- အမှန်တော့ Real World ဆိုတာသည် ကျွန်တော်တို့ စိတ်ဝင်စားတဲ့ Domain တစ်ခုကို ဆိုလိုတာဖြစ်သည်။ လောကကြီးတစ်ခုလုံး မဟုတ်ပါ။ ဥပမာ၊ ကျွန်တော်က မြန်မာပြည်ကို စိတ်ဝင်စားရင် မြန်မာပြည်သည် Real World ဖြစ်သည်။ တစ်နည်း၊ Real World ဆိုတာ လောကကြီးတစ်ခုလုံးမှ ကျွန်တော်စိတ်ဝင်စားတဲ့ အစိတ်အပိုင်း တစ်ခုဖြစ်သည်။
- ပြင်ပလောကကြီး၏ အစိတ်အပိုင်း တစ်ခုကို သက်ဆိုင်ရာ Model တည်ဆောက်လိုက်ခြင်းသည် System တစ်ခုဖြစ်သွားသည်။
- တစ်နည်း Real World ၏ Equivalent Model တစ်ခုသည် System တစ်ခုဖြစ်သည်။ System တစ်ခုမှာ
  - Environment
  - System Boundary and Interface
  - System Inputs
  - System Processes and Components
  - System Outputs

Systems Engineering

- Systems Engineering ဆိုတာသည် System တစ်ခုကို ဘယ်လိုတည်ဆောက် မလဲဆိုတာကို လေ့လာခြင်းဖြစ်သည်။

- အမှန်တော့ အရာအားလုံးသည် System တစ်ခုဖြစ်သည်။ Atom လေးတစ်ခုမှ Galaxy ကြီးတစ်ခု အထိ။ System တစ်ခုမှာ ဖော်ပြပါ အချက် ၅ ချက်ပါဝင်နေသည်။

- Systems Engineering မှာ အောက်ပါအဆင့်များ ပါဝင်သည်။
  - Systems Analysis
  - Systems Design
  - Systems Development
  - System Implementation
  - Systems Evaluation

- Systems Engineering သည် System တစ်ခုကို Perspective ၃ ခုမှ လေ့လာပြီးတည်ဆောက်ပါသည်။

  - Structural (Space) ဘယ်လို ဖွဲ့စည်းတည်ဆောက်ထားလဲ။
  - Dynamical or Flow(Time) အချိန်နှင့် အမျှ ဘယ်လိုပြောင်းလဲသွားတယ်။
  - Behavioral (Actions/ Responses) ဘယ်လိုတွေဖြစ်ရင် ဘာတွေလုပ်မလဲ။

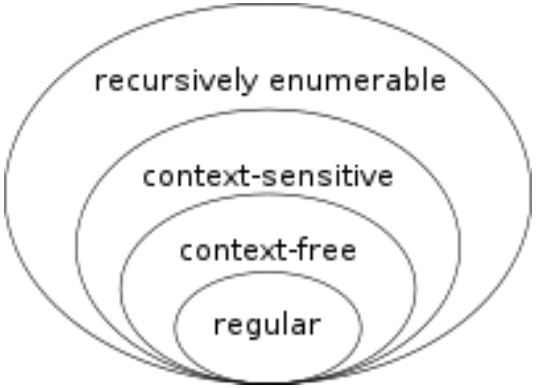
- UML မှာဆိုရင်
  - Structural (Class Diagram, Component Diagrams)
  - Dynamical or Flow (Sequence Diagram)
  - Behavioral (Use Case Diagram, Activity Diagram)

Semantics Summary

- Semantics ဟူသည် Meaning နှင့် Association များကို အခြေခံ၍ Understanding (Concept) များကို ဖော်ပြသည်။
- Semantics ၏ အဓိက တာဝန်သည် Real World ကို Concept များဖြင့် ဖော်ပြဖို့ဖြစ်သည်။ Concept များသည် Language Syntax Independent ဖြစ်သည်။
- Systems Thinking အရ Real World ကို Concept များဖြင့် ဖော်ပြသည့်အခါ System Model တစ်ခု ရလာသည်။
- System Model တစ်ခုကို Structural, Dynamical and Behavioral Perspectives များဖြင့် လေ့လာပြီး လက်တွေ့တည်ဆောက်ကြည့်သည်။ ဒါသည်ပင် Systems Engineering ဖြစ်လာသည်။
- Software Engineering သည် Systems Engineering တစ်ခု ဖြစ်ပြီး ထိုနည်းဖြင့် ရှုပ်ထွေးသော Software Systems များကို တည်ဆောက်ကြသည်။
- ဤနည်းဖြင့် Real World ကို ကွန်ပြူတာက တွက်ချက်နိုင်သော Computational Semantics (Procedural Semantics or Object Oriented Semantics) အဖြစ် ဖော်ပြကြသည်။
- ကျွန်တော်တို့ Computational Semantics များကို Computational Language Syntax အဖြစ် ဘယ်လိုဖော်ပြမလဲ ဆက်လက်ကြည့်ပါစို့။

Formal Grammar

- Noam Chomsky ကို Formal Grammar ရဲ့ ဘိုးအေကြီး အဖြစ် တင်စားကြပါသည်။
- Formal Grammar ကို Hierarchical Layer ၅ မျိုးခွဲခြားထားပါသည်။
- Regular Grammar : Regular Expression တွေမှာ သုံးပါသည်။
- Context Free Grammar: Programming Languages တွေအတွက် သုံးပါသည်။ Grammar သည် Context ပေါ် မူမတည်။
- Context Sensitive Grammar : Natural Languages တွေအတွက် သုံးပါသည်။ Grammar သည် Context ပေါ် မူတည်သည်။ ဒီဟာကို ဘယ်စျေးမှာ ဘယ်စျေးနဲ့ ရနိုင်မလဲ။ ဘယ်စျေး ၂ ခုမတူ။
- Recursively Enumerable Grammar: Universal Language (တကယ်ရှိ၊မရှိ မသိ) အတွက်ဖြစ်သည်။ (Maybe, we can talk with Aliens.)



Terms

- Formal Grammar မှာ အခြေခံအကျဆုံး Grammatical Unit များကို Terms များဟု ခေါ်သည်။
- Terms များကို Noun (Operands, Identifiers) သို့မဟုတ် Verb (Operations, Functions) များဖြင့် ဖော်ပြနိုင်သည်။
- x, y, z သည် Variable Terms များဖြစ်သည်။ 1, True, “A” သည် Literal Terms များဖြစ်သည်။ +, -, >=, Sin(), AND, OR တို့သည် Operand Terms များဖြစ်သည်။
- Terms များဖြင့် ပေါင်းစပ်ထားပြီး Noun (States) နှင့် Verb (Actions) တစ်ခု သို့မဟုတ် တစ်ခုထက် ပိုသော Grammatical Unit ကို Expression ဟု ခေါ်သည်။ Expression များသည် Formal Grammar ၏ Phrases များဟုပြောလျှင် မမှားပါ။
- Expression များကို
  - Regular Expression
  - Mathematical Expression
  - Functional Expression
  - Logical Expression
  - Relational Expression

Regular Expressions

- Expression များတွင် အခြေခံ အကျဆုံးက Regular Expression များဖြစ်သည်။ Regular Expression များသည် Regular Grammar ကို လိုက်နာသည်။
- Regular Grammar တစ်ခုတွင် Start Symbol (S), Non-Terminal Symbols (N), Terminal Symbol (a), Empty or Null Symbol (ŋ) နှင့် Production Rules (P) များပါဝင်သည်။
- Regular Grammar များကို Finite States များဖြင့် ဖော်ပြနိုင်ပြီး FSA (Finite State Automata) များဖြင့် Solve (Evaluate) လုပ်နိုင်ပါသည်။
- Regular Grammar နှင့် Regular Expression များကို ပိုပြီး အသေးစိတ်သိလိုပါက

https://people.montefiore.uliege.be/pw/cours/psfiles/calc-chap3.pdf

General Expressions

- Mathematical Operands နှင့် Operations များပါဝင်သော Expressions များကို Mathematical Expressions များဟု ခေါ်သည်။
- ထို့အတူ Logical Expressions၊ Relational Expressions၊ Functional Expressions များကို သတ်မှတ်နိုင်သည်။
- Complete Expression တစ်ခုကို Statement ဟုခေါ်သည်။ Statement သည် Sentence တစ်ခုနှင့် အလားတူသည်။
- Programming Language များကို Statement များဖြင့် တည်ဆောက်ထားခြင်း ဖြစ်သည်။

Operational Precedence

```
x := a * b + c / d * sin(x) – 4 * c;
```

- အထက်ပါ Expression ကိုEvaluate လုပ်ဖို့ Order of Operation လိုပါသည်။ Order of Operation က ဘယ် Expressionကို အရင် Evaluate လုပ်မလဲကို Define လုပ်ပါသည်။

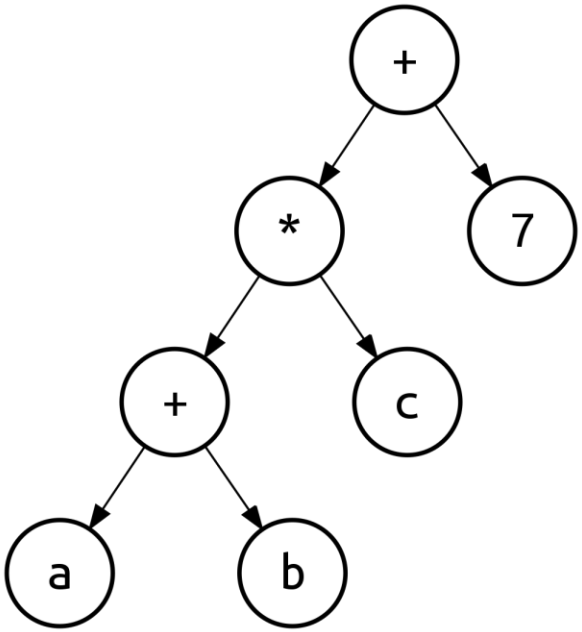
- Order of Operation ကို သတ်မှတ်ခြင်းကို Operational Precedence ဟုခေါ်သည်။

Operations	Precedence
Parenthesis	7
Unary, Function	6
Exponent	5
Multiplication, Division	4
Addition, Subtraction	3
Relational, Logical	2
Assignment	1

Expression Tree

```
(a + b) *c + 7
```

- အထက်ပါ Expression ကို Expression Tree ဖြင့်ဖော်ပြနိုင်သည်။
- Expression Tree ဖြင့် ဖော်ပြခြင်းသည် Order of Operation သာမက Dependency of Expression ကိုပါ ဖော်ပြပြီသားဖြစ်သွားဖြစ်သည်။



Expressional Notation

```
(a + b) *c + 7
```

- Expression Tree ကို Traversal လုပ်ခြင်းအားဖြင့် Expressional Notation များကို ရရှိပါသည်။
- Pre-order Traversal လုပ်ခြင်းအားဖြင့် Prefix Expressional Notation ကိုရရှိပါသည်။

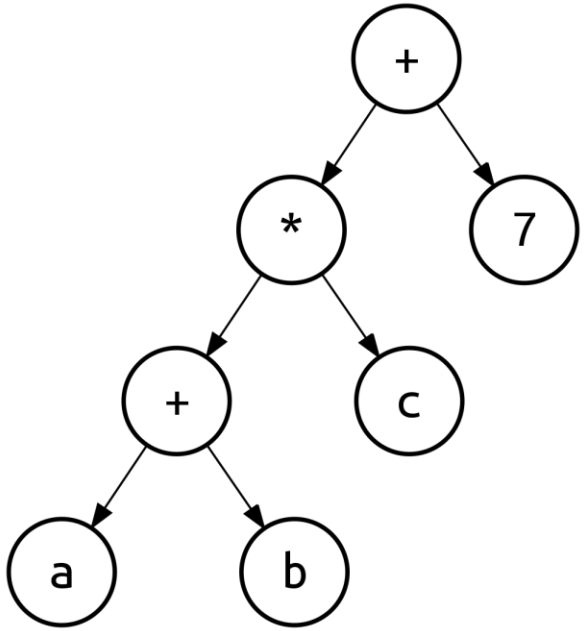
  Prefix : +a b * c + 7

- In-order Traversal လုပ်ခြင်းအားဖြင့် Infix Expressional Notation ကိုရရှိပါသည်။

  Infix: (a + b) * c + 7

- Post-order Traversal လုပ်ခြင်းအားဖြင့် Postfix Expressional Notation ကိုရရှိပါသည်။

  Postfix: a b + c * 7 +



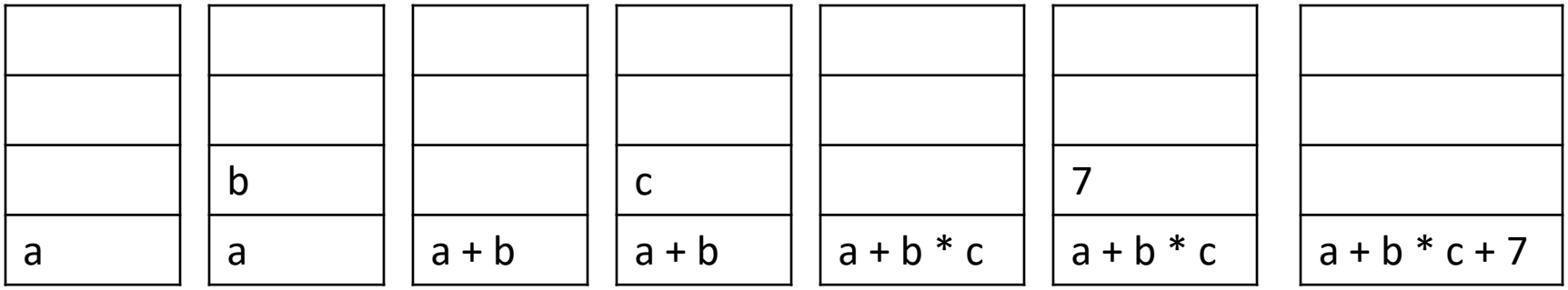
Expressional Evaluation

```
(a + b) *c + 7
```

- Regular Expression များကို FSA (Finite State Automata)များဖြင့် Evaluate လုပ်လို့ ရသော်လည်း General Expression များကို Evaluate လုပ်ဖို့ PDA (Push-Down Automata) လို့ ခေါ်သော Stack Machine များလိုအပ်ပါသည်။
- Expression များကို Evaluate လုပ်ဖို့ Infix Notation မှ Postfix Notation သို့ ပထမ ပြောင်းရပါသည်။

```
a b + c * 7 +
```

- ဒုတိယ PDA ကိုသုံးပြီး Evaluate လုပ်ပါသည်။ Postfix Expression ကို Left to Right ဖတ်ပါ။ Operand တွေ့ရင် Stack ပေါ်တင်ပါ။ Unary Operator တွေ့ရင် Operand ၁ ခု သို့မဟုတ် Binary Operator တွေ့ရင် ၂ခု ထုတ်ပြီး Evaluate လုပ်ပါ။ Result ကို Stack ပေါ်တင်ပါ။



Regular Grammar (RG)

- Regular Grammar တစ်ခုတွင် Start Symbol (S), Non-Terminal Symbols (N), Terminal Symbol (T), Empty or Null Symbol (ŋ) နှင့် Production Rules (P) များပါဝင်သည်။
- Regular Grammar = G(N, T, ŋ , P), where N = { E }, T = { a, b }, S ∈ N
- Sample Production Rules (P)
- E -> Ea
- E -> Eb
- E -> aEb
- E -> ŋ
- Regular Grammar များကို FSA (Finite State Automata) များဖြင့် Solve လုပ်လို့ရပြီး Pattern Matching များကို ဖြေရှင်းရာတွင် အသုံးပြုသည်။

Context Free Grammar (CFG)

- Context Free Grammar ကို Regular Grammar ရဲ့ Super-Set လို့ ယူဆရင်မမှား။ Regular Grammar များကို Context Free Grammar များဖြင့် ဖော်ပြလို့ ရသည်။ တစ်ချိန်တည်းမှာပင် Context Free Grammar သည် Context Sensitive Grammar ၏ Subset ဖြစ်သည်။ Natural Language များကို Context Sensitive Grammar ဖြင့်သာ ဖော်ပြနိုင်သည်။

- Context Free Grammar = G(N, T, ŋ , P), where N = { E }, T = { a, b }, S ∈ N

- Sample Production Rules (P)

- A -> w , where A ∈ N, w ∈ (N U T), S ∈ N

- Context Free Grammar များကို PDA (Push Down Automata) များဖြင့် Solve လုပ်လို့ရပြီး Parsing (Parser) များကို ဖြေရှင်းရာတွင် အသုံးပြုသည်။

- Context Free Grammar များကို ပိုပြီး အသေးစိတ်သိလိုပါက

http://web.stanford.edu/class/archive/cs/cs103/cs103.1184/lectures/19/Small19.pdf

Backus-Naur Form (BNF)

- Backus-Naur Form ဆိုတာ CFG ဖြင့် Programming Language Syntax များကို ဖော်ပြဖို့ Special Notation တစ်ခုဖြစ်သည်။

- BNF for Postal Address

```
<postal-address> ::= <name-part> <street-address> <zip-part>

<name-part> ::= <personal-part> <last-name> <opt-suffix-part> <EOL> |
<personal-part> <name-part>

<personal-part> ::= <initial> "." | <first-name>

<street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>

<zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

<opt-suffix-part> ::= "Sr." | "Jr." | <roman-numeral> | ""

<opt-apt-num> ::= <apt-num> | ""
```

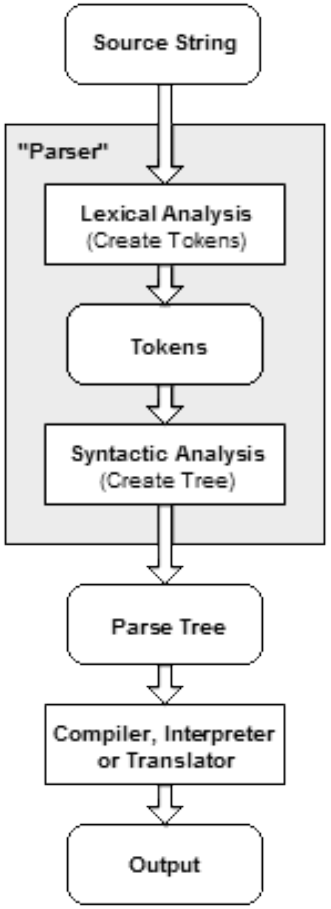
Parsing

- Machines (Computer) များအနေဖြင့် Language များကို ဖတ်ပြီးနားလည်ဖို့ ပထမဆုံး လုပ်ရမည်မှာ String of Symbols (သင်္ကေတများ) ကို ဖတ်ပြီး သက်ဆိုင်ရာ Grammar များဖြင့် Structure နှင့် Rule များကို တည်ဆောက်ဖို့ လိုအပ်ပါသည်။ ဒါကို Language Parsing လုပ်သည် ဟုခေါ်သည်။

- Programming Languages များကိုလည်း Language Parsing လုပ်ဖို့လိုအပ်ပြီး ဒီလိုလုပ်တဲ့ Software Algorithm ကို Parser ဟုခေါ်ပါသည်။

- Programming Languages များသည် CFG များဖြစ်သည့်အတွက် Programming Language Parser များသည် Push-Down Automata (PDA) များဖြစ်ကြပါသည်။

- Natural Language များအတွက် Natural Language Parser များသက်သက်ရှိပါသည်။

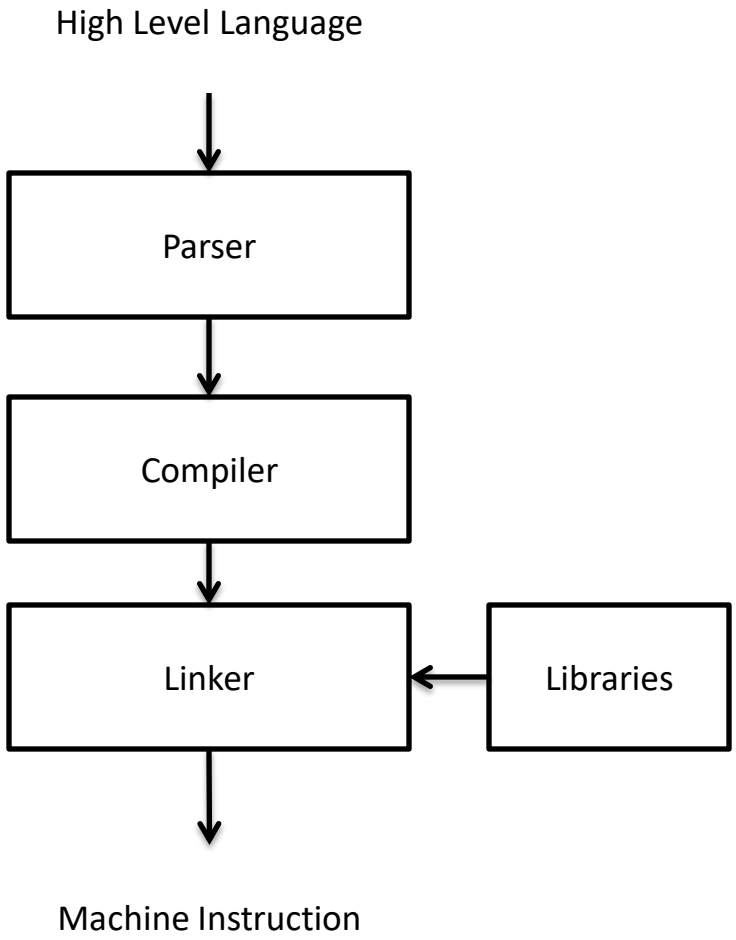


Parser

- Parser များသည် String of Symbols များကို Input အဖြစ်လက်ခံပြီး Parse Tree သို့မဟုတ် Syntax Tree များကို Output ထုတ်ပေးပါသည်။
- Parser မှ ရလာသော Parse Tree များကို Semantic Analysis လုပ်ဆောင်ပြီး သက်ဆိုင်ရာ Compilation (Translation) သို့မဟုတ် Interpretation များကို ဆက်လက်လုပ်ဆောင်ပါသည်။
- Semantic Analysis ဆိုသည်မှာ အရှေ့ပိုင်းပြောခဲ့တဲ့ Procedural Semantics နှင့် Object Oriented Semantics များကို ဆိုလိုပါသည်။
- သက်ဆိုင်ရာ Semantics များကို အခြေခံပြီး နောက်ဆုံး Computational Model ကိုတည်ဆောက်ခြင်းဖြင့် Program များကို Run လုပ်လို့ရပါသည်။
- Computational Model လုပ်ဆောင်ရာတွင် Compilation (Translation)၊ Interpretation နှင့် Run-Time Environment ကို အခြေခံတဲ့ Hybrid ဆိုပြီး ၃ မျိုးရှိပါသည်။

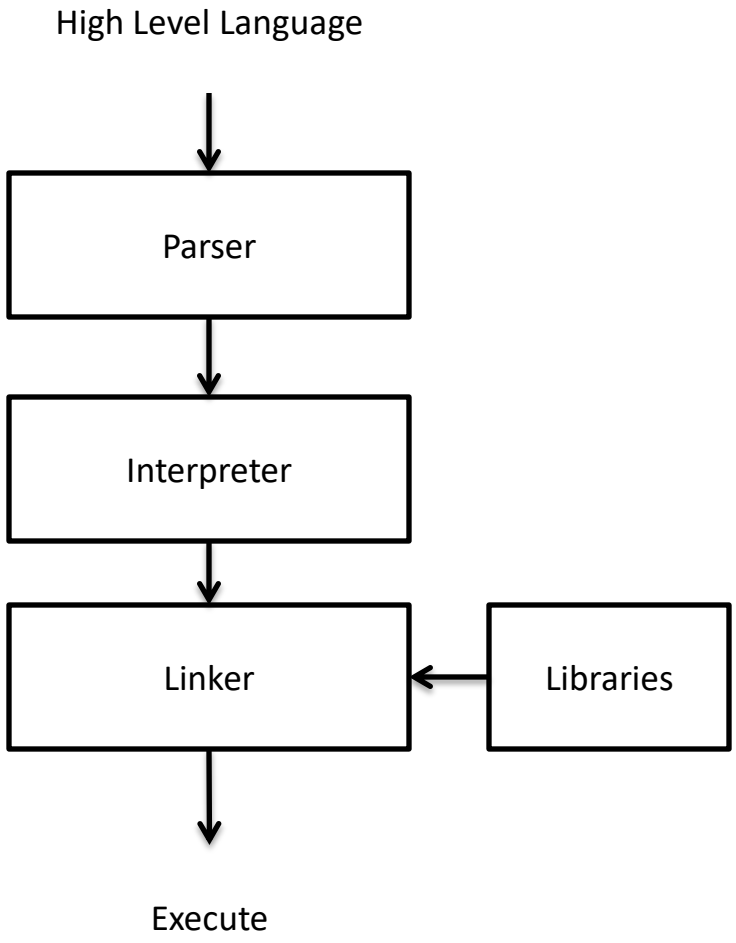
Compilation and Compiler

- အရှေ့ပိုင်းမှာ ပြောခဲ့သလို Computer သည် Computation လုပ်သော Machine တစ်ခုဖြစ်ပြီး Computation Model သည် Binary Instruction များ အပေါ် အခြေခံထားသည်။
- Compiler များသည် High Level Programming Language များကို Translate လုပ်ပေးပါသည်။
- Compiler မှထွက်လာသော Machine Instruction များသည် Machine Dependent ဖြစ်ပါသည်။ 32-bit Computer အတွက် လုပ်ထားရင် 63-bit Computer အတွက် အဆင်မပြေပါ။
- Different Computer Architecture များအတွက် Recompile ပြန်လုပ်ပေးရပါသည်။
- Compiler များ၏ အားနည်းချက်က Portable မဖြစ်။



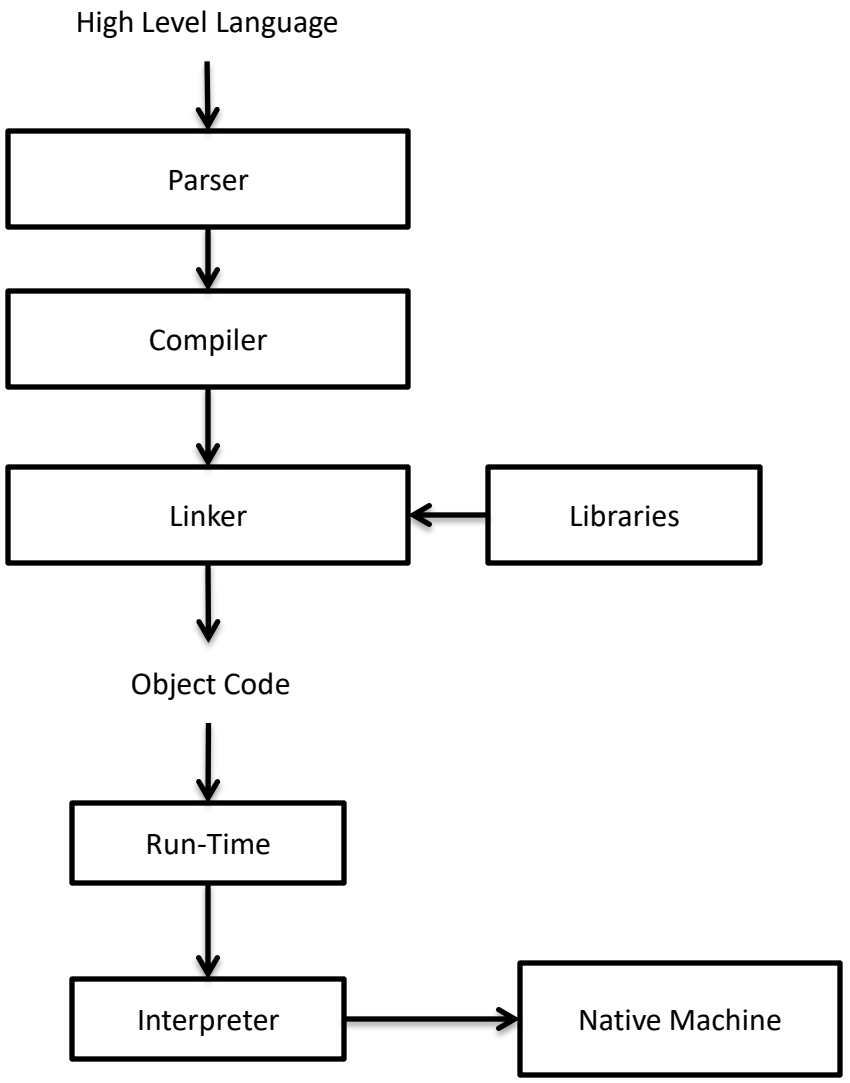
Interpretation and Interpreter

- Interpreter များသည် Compiler များဖြင့် တော်တော်လေး ဆင်တူပါသည်။
- အဓိက ကွာခြားချက်က Interpreter များက Machine Instruction Code ထုတ်မပေးဘဲ တခါတည်း တန်းလုပ်ပါသည်။
- Web Browser များသည် လူသိအများဆုံး Interpreter များဖြစ်ကြသည်။ Web Browser သည် HTML၊ CSS နှင့် Javascript များကို Interpret လုပ်ပြီး Result ကို Browser ပေါ်မှာ တန်းပြပါသည်။ Machine Instruction Code ထုတ်မပေးပါ။
- Interpreter များ၏ အားနည်းချက်က Web Browser လို Container (Virtual Computation Layer) တစ်ခုလိုအပ်သည်။



Run-Time Environment

- Interpreter နှင့် Compiler တို့၏ Best of the 2 World ကို စတင် အကောင်အထည်ဖော်သည်က Java ဖြစ်သည်။
- Java သည် ပထမဆုံး Java Run-Time လို့ ခေါ်တဲ့ Run-Time Environment ကို စတင်အသုံးပြုပါသည်။
- Run-Time Environment သည် Virtual Machine (Container) တစ်ခုဖြစ်သည်။ သူ့မှာ ကိုယ်ပိုင် Machine Instruction များရှိသည်။ Java Run-Time ကို Different Machine များမှာ သက်သက်စီ Install လုပ်ရသည်။ 32-Bit Computer အတွက် 32-Bit Java Run-Time ၊ 64-bit Computer အတွက် 64-bit Java Run-Time။ Run-Time များအတွက် Machine Instruction များ အာလုံးအတူတူ ဖြစ်သည်။
- Java Compiler က Java Language ကို သက်ဆိုင်ရာComputer အတွက် Compile လုပ်စရာ မလိုတော့။ Run-Time အတွက်ပဲ လုပ်စရာလိုသည်။ ဒီလို Compile လုပ်ပြီးသား Code ကို Object Code ဟုခေါ်သည်။
- Object Code ကိုမှ Run-Time က Interpret လုပ်ပြီး Native Machine ပေါ်မှာ တန်း Execute လုပ်သည်။



Syntactic Summary

- Semantics ဟူသည် Meaning နှင့် Association များကို အခြေခံ၍ Understanding (Concept) များကို ဖော်ပြသည်။
- Semantics ၏ အဓိက တာဝန်သည် Real World ကို Concept များဖြင့် ဖော်ပြဖို့ဖြစ်သည်။ Concept များသည် Language Syntax Independent ဖြစ်သည်။
- Syntactic Layer မှာတော့ Concept များကို Symbols နှင့် Structures များဖြင့် ဘယ်လို ဖော်ပြမလဲ၊ ပြီးရင် ဒါကို ဘယ်လို Process လုပ်မလဲကို အဓိကထားသည်။
- Symbols နှင့် Structures များကို Formal Grammar များဖြင့် ဖော်ပြပြီး PDA များကို သုံး၍ Syntax Tree များကို တည်ဆောက်ကြသည်။
- Syntax Tree များကိုမှ Semantics Analysis လုပ်ပြီး သက်ဆိုင်ရာ Compilation သို့မဟုတ် Interpretation ကို လုပ်ဆောင်သည်။
- ဤနည်းဖြင့် Programming Language များနှင့် သက်ဆိုင်ရာ Compiler၊ Interpreter များကို တည်ဆောက်ကြပါသည်။

Project

- အခုအထိတော့ သီအိုရီများသာဖြစ်သည်။ အင်မတန်လည်း ပျင်းစရာကောင်းပါလိမ့်မည်။
- ဒီတော့က လက်တွေ့လေးလည်း လုပ်ကြည့်ရအောင်ပါ။
- ကျွန်တော် `Interpreter` သေးသေးတစ်ခုကို စမ်းရေးထားတာရှိသည်။
- ဒါကို လေ့လာပြီး ကိုယ့်ပိုင် တစ်ခုကို `Compiler` ဖြစ်ဖြစ်၊ `Interpreter` ဖြစ်ဖြစ် စမ်းရေးကြည့်ပါ။
- သိရုံသာမဟုတ် ကိုယ်ပိုင်ဖန်တီးတည်ဆောက်နိုင်သော ပညာရှင်များ ဖြစ်ကြပါစေ။