Programming Languages

Bibhas Adhikari

IIT Kharagpur

September 18, 2020

Purely interpreted implementation

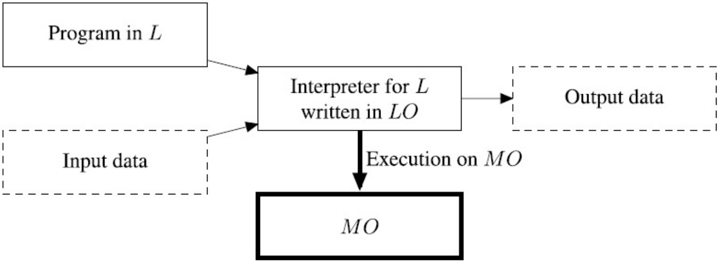

Figure: purely interpreted implementation

- The interpreter for $\mathcal{M}_{\mathcal{L}}$ is implemented using a set of instructions in $\mathcal{L}_o$
- Denote the interpreter as $\mathcal{I}_{\mathcal{L}}^{\mathcal{L}_o}$

Definition 1: Interpreter

An interpreter for language $\mathcal{L}$, written in language $\mathcal{L}_o$, is a program which implements a partial function

$$\mathcal{I}_{\mathcal{L}}^{\mathcal{L}_o} : (\mathcal{P}rog^{\mathcal{L}} \times \mathcal{D}) \to \mathcal{D}$$

such that

$$\mathcal{I}_{\mathcal{L}}^{\mathcal{L}_o}(\mathcal{P}^{\mathcal{L}}, Input) = \mathcal{P}^{\mathcal{L}}(Input)$$

- It is a “decoding” procedure, not a translation

Purely compiled implementation

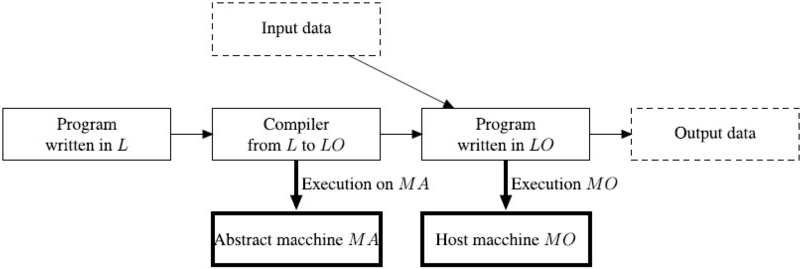

Figure: purely compiled implementation

Purely compiled implementation

- The implementation of $\mathcal{L}$ takes place by explicitly translating programs written in $\mathcal{L}$ to programs written in $\mathcal{L}_o$
- The translation is performed by a special program called compiler, denoted by $\mathcal{C}_{\mathcal{L},\mathcal{L}_o}$
- $\mathcal{L}$ is called the source language, and $\mathcal{L}_o$ is called the object language

Definition

A compiler from $\mathcal{L}$ to $\mathcal{L}_o$ is a program which implements a function:

$$\mathcal{C}_{\mathcal{L},\mathcal{L}_o} : \mathcal{P}rog^{\mathcal{L}} \to \mathcal{P}rog^{\mathcal{L}_o}$$

such that, given a program $\mathcal{P}^{\mathcal{L}}$ if

$$\mathcal{C}_{\mathcal{L},\mathcal{L}_o}(\mathcal{P}^{\mathcal{L}}) = \mathcal{P}_c^{\mathcal{L}_o} \text{ (compiled program)}$$ -Compilation phase

then for any data $Input \in \mathcal{D}$,

$$\mathcal{P}^{\mathcal{L}}(Input) = \mathcal{P}_c^{\mathcal{L}_o}(Input).$$ - Execution process

Comparing the two approches

purely interpreted implementation

Disadvantages: Low efficiency

Advantage: Flexibility

purely interpreted implementation

Disadvantages: Flexibility

Advantage: Efficiency

Real scenario: interactive framework

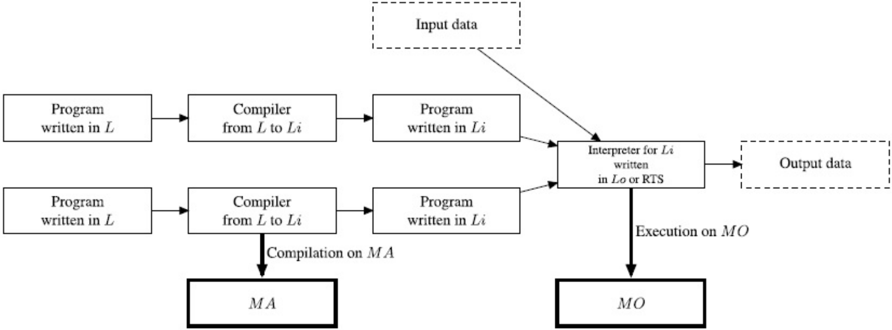

Figure: Interactive machine

Source - Intermediate - Host

1. $\mathcal{M}_{\mathcal{L}} = \mathcal{M}_{i\mathcal{L}_i}$: purely interpreted implementation
2. $\mathcal{M}_{\mathcal{L}} \neq \mathcal{M}_{i\mathcal{L}_i}\mathcal{M}_{0\mathcal{L}_0}$:
    - An implementation of an interpretative type if interpreter of $\mathcal{M}_{i\mathcal{L}_i}$ is different from interpreter of $\mathcal{M}_{0\mathcal{L}_0}$
    - An implementation of a compiled type if interpreter of $\mathcal{M}_{i\mathcal{L}_i}$ is substantially the same as the interpreter for $M_{o\mathcal{L}_o}$
3. $\mathcal{M}_i\mathcal{L}_i = \mathcal{M}_{0\mathcal{L}_0}$: purely compiled inplementation

Hierarchy

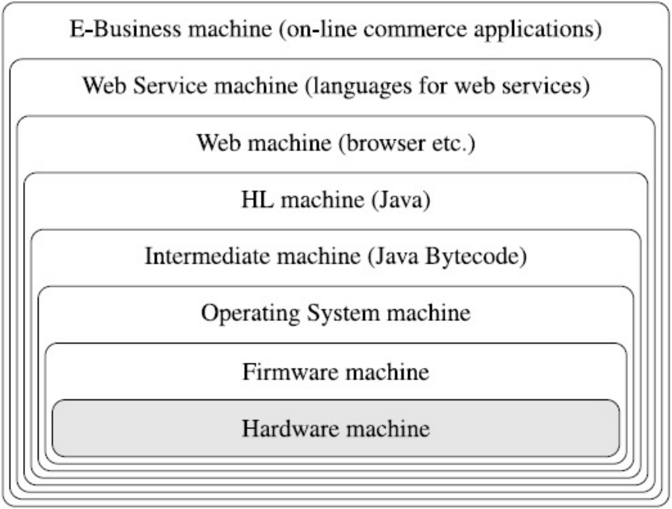

Figure: A hierarchy of abstract machines

Framework of a programming language

- Grammar: which phrases are correct? (syntax)
- Semantics: what does a correct phrase mean?
- Pragmatics: how do we use a meaningful sentence?
- Implementation level

Grammar and Syntax

Grammar

- It establishes the alphabet and lexicon
- It defines sequences of symbols corresponding to well-formed phrases and sentences

Syntax

- Is a set of rules that define what sequences of symbols are considered to be valid expressions (programs) in the language
- A widespread formal notation for syntax is Extended Backus-Naur Form (EBNF).

Example of a grammar and a syntax in C

Grammar

Let $A = \{a, b\}$ be the alphabet. Define the language as all palindromic strings (s) of the symbols a and b. For instance, *asa*, *abba* etc.

Syntax

In C the syntax of an if-statement is given by the rule:

if-statement ::= **if** (expression) statement [**else** statement]

- ::= -"is composed of"
- *if* - syntactic category
- [ ] - optional

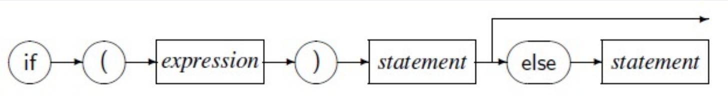

Figure: Syntax diagram

Context-Free Grammars

- A fundamental device for the description of programming languages

Notation and terminologies

- Alphabet: A finite or countable set, denoted by A
- Kleene's star of A : Set of all finite strings over A, denoted by A^*
- The empty string, denoted by ϵ
- A formal language over the alphabet A is a subset of A^* .

Context-Free Grammars

Definition

A context-free grammar is a quadruple (NT, T, R, S) where:

- NT is a finite set of symbols (non-terminal symbols, or variables, or syntactic categories)
- T is a finite set of symbols (terminal symbols)
- R is a finite set of productions (or rules), each of which is composed of an expression of the form:

  $$V \to w$$

  where V (the head of the production) is a single non-terminal symbol and w (the body) is a string composed of zero or more terminal or non-terminal symbols (that is w is a string over $T \cup NT$)
- S is an element of NT (the initial symbol).

Context-Free Grammars

Example

In the example of the language defined on the alphabet $A = \{a, b\}$ mentioned before, inductive definition of palindromic strings can be expressed in grammatical form as:

$$
\begin{aligned}
P &\rightarrow \\
P &\rightarrow a \\
P &\rightarrow b \\
P &\rightarrow aPa \\
P &\rightarrow bPb
\end{aligned}
$$

where P is a palindromic string, and $\rightarrow$ stands for "can be"

Context-Free Grammars

Example

$G = (\{E, I\}, \{a, b, +, *, -, (,)\}, R, E)$, where R is the following set of productions:

1. E $\rightarrow I$,	7. I $\rightarrow a$
2. E$\rightarrow E + E$,	8. I $\rightarrow b$
3. E$\rightarrow E * E$	9. I $\rightarrow Ia$
4. E$\rightarrow EE$,	10. I$\rightarrow Ib$
5. E$\rightarrow -E$,	
6. E$\rightarrow (E)$	

Context-Free Grammars : Backus Naur normal form (BNF)

- First time context-free-language was used in the definition of Algol60
- BNF, named after two authoritative members of the Algol committee John Backus and Peter Naur

Definition: Derivation

For a fixed grammar, $G = (NT, T, R, S)$, and assigned two strings, v and w over $NT \cup T$, we say that w is immediately derived from v (or v is rewritten in a single step into w) if w is obtained from v by substituting the body of a production of R whose head is V for a non-terminal symbol, V , in v. In this case, we will write $v \Rightarrow w$.
We say that w is derived from v (or v is rewritten to w) and we write $v \Rightarrow^* w$, if there exists a finite (possibly empty) sequence of immediate derivations $v \Rightarrow w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w$.

Context-Free Grammars

Example: derivation

$$
\begin{aligned}
E &\Rightarrow_3 E * E \\
&\Rightarrow_1 I * E \\
&\Rightarrow_{10} I\mathbf{b} * E \\
&\Rightarrow_7 \mathbf{ab} * E \\
&\Rightarrow_6 \mathbf{ab} * (E) \\
&\Rightarrow_2 \mathbf{ab} * (E + E) \\
&\Rightarrow_1 \mathbf{ab} * (I + E) \\
&\Rightarrow_7 \mathbf{ab} * (\mathbf{a} + E) \\
&\Rightarrow_1 \mathbf{ab} * (\mathbf{a} + I) \\
&\Rightarrow_8 \mathbf{ab} * (\mathbf{a} + \mathbf{b})
\end{aligned}
$$

Figure: Derivation of $ab * (a + b)$

Context-Free Grammars

Definition: Generated Language

The language generated by a grammar $G = (NT, T, R, S)$ is the set

$$\mathbb{L}(G) = \{w \in T^* | S \Rightarrow^* w\}$$

.

Observation: This is a language over T^*.

Derivation tree

The derivation of a string is not unique.

Basics of Graph theory

Graph

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges. Formally, a graph is a pair of sets (V, E), where V is the set of vertices and E is the set of edges, connecting the pairs of vertices.
$V = \{a, b, c, d, e\}$ $E = \{(a, b), (a, c), (b, d), (c, d), (d, e)\}$

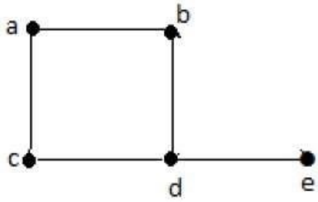

Figure: Example of a graph

Basics of Graph theory

Degree of a node of a graph

It is the number of vertices adjacent to a vertex V.

Directed graph

When each edge of a graph is given a sense of direction.

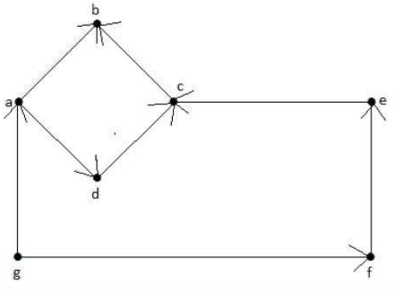

Figure: Example of a directed graph

Basics of Graph theory

Degree of a node in a directed graph

- Each vertex has an indegree and an outdegree
- Indegree of vertex V is the number of edges which are coming into the vertex V
- Outdegree of vertex V is the number of edges which are going out from the vertex V

- A vertex with degree one is called a pendent vertex
- A vertex with degree zero is called an isolated vertex
- Two vertices are said to be adjacent, if there is an edge between the two vertices
- If a pair of vertices is connected by more than one edge, then those edges are called parallel edges
- A graph having parallel edges is known as a Multigraph. A graph with no parallel edges is called simple