Introduction to Intermediate Code Generator.

19 April 2025
Saturday

DAG = Directed Accyclic Graph

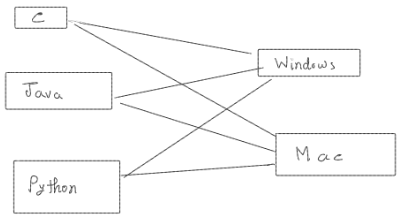
Analysis Phase → frontend → syntax → machine independent front end

↳ Analysis phase sesh hoy 3 address code পাই {
↳ means আমি এটা যেকোনো জায়গায় run করাতে পারবো.
↳ for example: byte code
↳ frontend : Analysis phase.

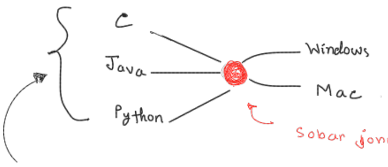
Synthesis ⇒ backend → assembly code, machine code

↳ synthesis phase sesh hole machine code পাই

\# Backend (Synthesis phase) is machine dependent; কোন machine এ করছি তার compiler এর উপর নির্ভর করে,



3টা language এর জন্য 6টা compiler লাগছে,



Sobar jonno byte code generate করলাম.

ei bytecode টা নিয়ে windows & mac both এ run করাতে পারবো

↳ ∴ 5 compiler needed now.

m x n Compiler :

- first এ সবার জন্য একটা compiler লাগতো ; কিন্তু এখন তা লাগে না
- byte code generate করে then আমার OS (Windows / MAC | Linux) এর উপর base করে আমার result ; as a result আমার total number of compiler কমে যায়.

Representations of Intermediate Code Generations :

- Intermediate Code generator can be represented in 2 ways;
  - Three Adress Code } → maximum 3 টা code address or less.
  - Directed Acyclic Graph.

\# Assembler এর কাছে three address code যায়.

Syntax Tree → Reg Ex to DFA direct method

↳ has a condition that internal node এ operator থাকবে এবং leaf node এ letter থাকবে.

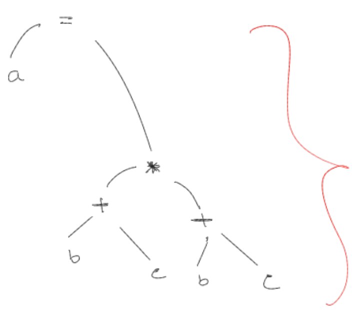
\# Parse Tree → Generates CFG.

■ Draw DAG for the following equation

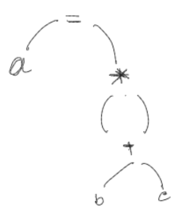
$a = (b + c) * (b + c)$

→ internal nodes ⇒ operators {=, +, *} → (,) থাকবে কি না ?

leaf = {a, b, c}

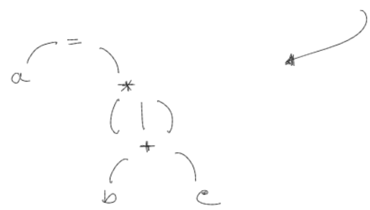


Syntax Tree.



DAG ⇒ definition

→ যদি (b+c) three times repeat hoto then,

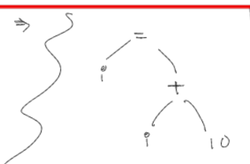


DAG

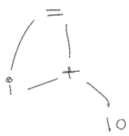
- Directed Accyclic Graph.
- One kind of Syntax Tree ; but there won't be any repetation.
- In the given example (b+c) was repeated twice.

Draw DAG for the following input string

$\boxed{i = i + 10}$



Syntax Tree



generated lexemes, id = id + num

0	id	i	
1	num	10	
2	+	(0)	(1)
3	=	(0)	(2)

array of records

$t1 = i$
$t2 = 10$
$t3 = t1 + t2$
$i = t3$

*R^ • first check if the data is already in array or hashtable, as DAG এ কোনো repetation হয় না।
since nai, we put i in 0 index (hash function থাকলে hash index অনুযায়ী → index এ দিয়ে বসবে।

• i = i + 10; আগে add করবে then assign করবে।
i এর index 0th index এ
10 এর index 1st index এ,

same for 10

• now assign i = ...

Simple Three Address Code:

$i = i + 10$

$\boxed{\begin{array}{l} t_1 = i + 10 \\ i = t_1 \end{array}}$ three address representation

$a = (b+c) * (b+c)$

$\boxed{\begin{array}{l} t_1 = (b+c) \\ t_2 = (b+c) \\ t_3 = t_1 * t_2 \\ a = t_3 \end{array}}$ Three Address Code

DAG disadvantage:
- line by line execute kora lage (no jumping possible)

Small Code → DAG better

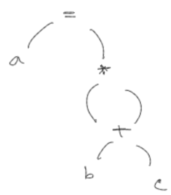
Complex Code ⇒ Three Adress Code better

Practice:

Draw the array of record representation for the following input string

$a = (b+c) * (b+c)$

⇒ 0. t1 = a
1. t2 = b
2. t3 = c
3. t4 = t2 + t3
4. t5 = t2 + t3) repetative
5. t6 = t4 * t5
6. a = t6

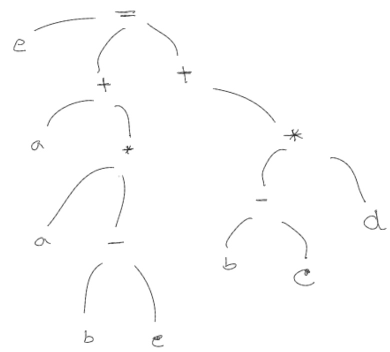


0	id	a	
1	id	b	
2	id	c	
3	+	(1)	(2)
4	*	(3)	(3)
5	=	(0)	(4)

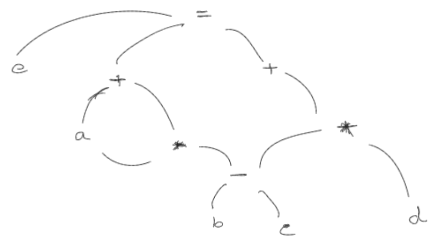
Draw both DAG and array of records for the input string.

$e = a + a * (b-c) + (b-c) * d$

$t_1 = b - c$
$t_2 = b - c$
$t_3 = a * t_1$
$t_4 = t_2 * d$
$t_5 = a + t_3$
$t_6 = t_5 + t_3$
$e = t_6$



Syntax Tree



DAG

0	id	e	
1	id	a	
2	id	b	
3	id	c	
4	id	d	
5	-	(2)	(3)
6	*	(1)	(5)
7	+	(1)	(6)
8	*	(5)	(4)
9	+	(7)	(8)
10	=	(0)	(9)