Dr B R Ambedkar National Institute of Technology, Jalandhar

B.Tech. 7th Semester(CSE)

CSPC-401- System Programming and Compiler Design

End Semester Examination, November 2024

1. (a) Define and differentiate tokens, patterns and lexeme.

Solution:

A **token** is a sequence of characters categorized as a meaningful unit in a programming language, such as keywords, operators, identifiers, and literals. For example, int, +, and identifier are tokens.

A **pattern** is the rule that defines the structure of a token and is often specified using regular expressions. For example, the pattern [a-zA-Z][a-zA-Z0-9]* can be used to identify an identifier token.

A **lexeme** is the actual sequence of characters in the source code that matches a pattern and is classified as a token. For instance, in the statement int x = 10;, the lexeme int matches the keyword token, while x matches the identifier token. Thus, tokens categorize the input, patterns define the rules, and lexemes are the actual pieces of code being categorized.

(b) State whether the following statement is True or False. Give reason.

"The class of grammar that can be parsed using LR methods is proper subset of the class of grammar that can be parsed by LL method."

Solution: The statement is **false**.

The class of grammars that can be parsed using LR methods is broader than that of LL methods. LR parsers, such as SLR, LALR, and CLR, can handle all context-free grammars, including those with left recursion, which LL parsers cannot parse. LL parsers require the grammar to be left-factored and free of left recursion, making them less powerful. Therefore, LR parsers are capable of handling a wider range of grammars compared to LL parsers.

(c) Write a short note on Common sub expression and dead code elimination.

Solution:

Common subexpression elimination is an optimization technique that identifies expressions evaluated multiple times with the same operands and avoids redundant computation by reusing the previously computed result. For example, in the statement int a = (x + y) * (x + y);, the subexpression (x + y) is computed twice. Optimization involves calculating it once, storing it in a temporary variable, and reusing the result, making the code more efficient.

On the other hand, **dead code elimination** removes code that has no effect on the program's output, such as unused variables or unreachable code. For instance, if a variable is declared and assigned a value but never used, that code can be safely eliminated. Both techniques improve the efficiency and performance of the program by reducing redundant operations and unnecessary code.

(d) **Defined and differentiate S-Attribute and L-Attribute with suitable example.**

Solution:

An **S-attributed grammar** is a type of attribute grammar where all attributes are synthesized. Synthesized attributes are computed based on the values of child nodes in the syntax tree. For example, in an arithmetic expression like

$E \rightarrow E1 + T$, the value of E (E.val) is computed as E1.val + T.val, where E1 and T are its children.

In contrast, an **L-attributed grammar** allows both synthesized and inherited attributes. Inherited attributes are passed from parent or sibling nodes, while synthesized attributes are derived from child nodes. For example, in type checking, a production like $T \rightarrow$ int L may pass the type of T (inherited attribute) to L as L.type = T.type. The key difference is that S-attributed grammars rely only on child nodes, while L-attributed grammars can depend on parent, sibling, and child nodes for attribute computation.

(e) **Consider the following augmented grammar, which is to be parsed with a SLR parser. The set of terminals is {a, b, c, d,#, @ }**

S'→S

S → SS/Aa/bAc/Bc/bBa A

→ d#

B → @

Let I_0 = closure ({S' → • S}) Find the total number of items in the set GOTO(I_0, S)?

Solution: Count the items:

1. S' → S •
2. S → S • S
3. S → • S S
4. S → • A a
5. S → • b A c
6. S → • B c
7. S → • b B a
8. A → • d #
9. B → • @

There are **9 items** in GOTO(I_0, S).

Final Answer: The total number of items in the set GOTO(I_0, S) is **9**.

2. **(a) Draw the derivation tree for the sentence "An elephant is a big animal" by using the following natural language grammar.**

<Sentence> → <Noun phrase> <Verb phrase>
<Noun phrase> → <Article> <Noun> <Verb phrase> → <Verb> <Noun phrase>
<Article> → an / a
<Noun> → <Adjective> <Noun>
<Verb> → is / was
<Noun> → Cat / elephant / Tiger / Elephant / animal
<Adjective> → small / big / dangerous

Solution:

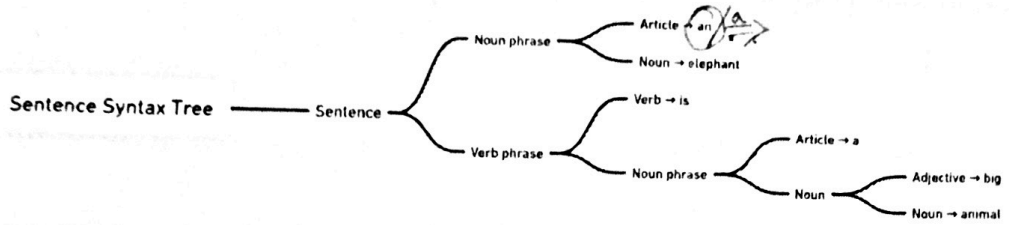


Figure 1: Derivation Tree

(b) Remove the left-factoring from the following grammar.
S → f g G / f G / h i l / h i j G / h i k G

Solution:

S → f X | h i Y
X → g G | G
Y → l | j G | k G

(c) Convert the following ambiguous grammar into equivalent unambiguous grammar E → E+E / E*E / E/E / E-E / E↑E / id

↑ is having lowest priority and left to right associative.
*, + is having next lower priority and right to left associative.
/ is having next lower priority and right to left associative.
- is having highest priority and left to right associative.

Solution:
E → E ↑ W | W
W → X | X * W | X + W
X → A | A / X
A → A - Z | Z
Z → id

(d) Construct the AST, DAG, postfix notation and three-address code for the following expression: x=((x+y) - ((x-y) * (x-y))) + ((x-y) * (x-y))

Solution:

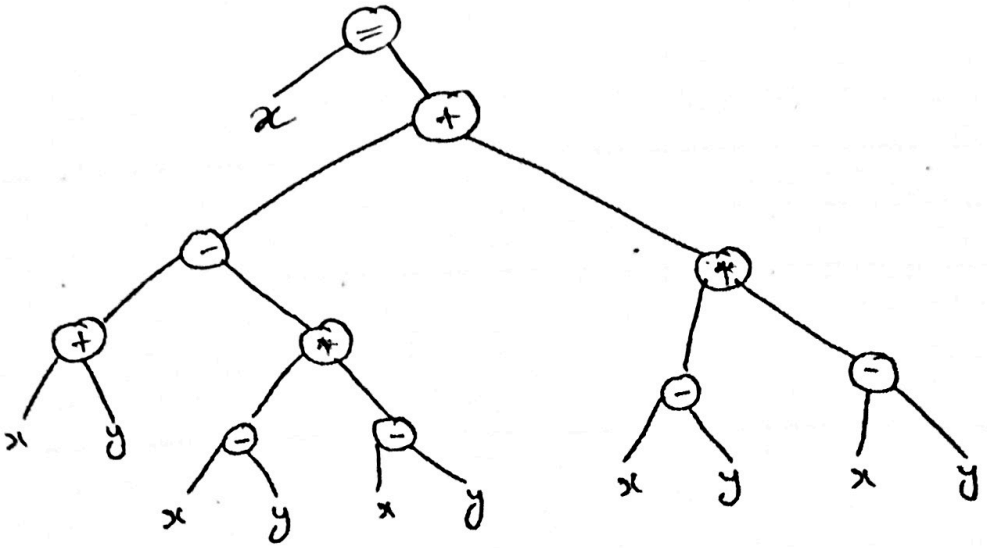


Figure 2: AST Diagram

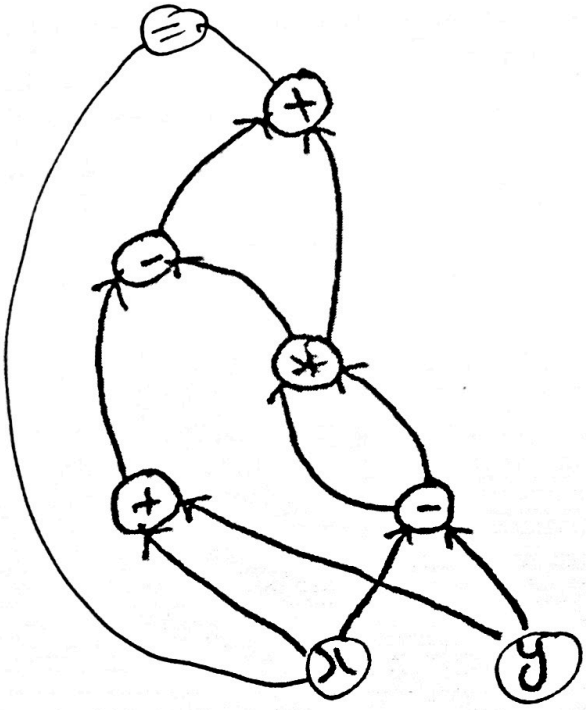


Figure 3: DAG Diagram

Postfix ~~x y + x y - x y * x y - x y - * +~~ (unoptimized) =

x xy+ xy - xy -* - xy - xy -* + =

Three-address code (unoptimized):

T1 = x + y
T2 = x * y
T3 = T2 + T2
T4 = T1 - T3
T5 = T4+T3
x = T5

3. **(a) Consider the following three separate grammars G1, G2 and G3:**

A → b A c / ϵ (G1)
A → b A b / b (G2)
A → b A b / c (G3)
Symbol A is the start symbol and single non-terminal, and b and c are terminals.
For all three grammars:
(i) Calculate the First and Follow sets of A.

Solution:

For G1:
- FIRST(A)={b,ϵ}
- FOLLOW(A)={c,$}

For G2:
- FIRST(A)={b}
- FOLLOW(A)={b,$}

For G3:
- FIRST(A)={b,c}
- FOLLOW(A)={b,$}

(ii) After extending the grammar with a new start symbol and production A′ →A, draw the LR(0)-DFA.

Solutions:

DFA States:

- I0:{A'→· A,A→· bAc,A→· ϵ}
- I1:{A'→A□}I1 [Got by shifting on A from I0]
- I2:{A→b□Ac, A→□bAc, A→□ϵ}[Got by shifting on b from I0]
- I3:{A→bA□c} [Got by shifting on A from I2]
- I4:{A→bAc□} [Got by shifting on c from I3]

Transitions:

From I0:

On A: Goto I1

On b: Goto I2

From I2:

On A: Goto I3

From I3:

On c: Goto I4

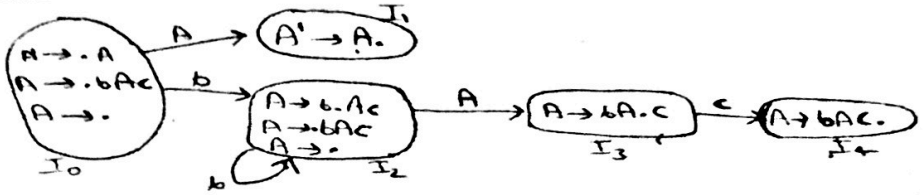
This gives the LR(0) DFA for G1. Similar constructions can be done for G2 and G3.

(b) What do
and

G_1 | First | Follow

$A \to bAc \mid \epsilon$ | $\{b, \epsilon\}$ | $\{\$, c\}$

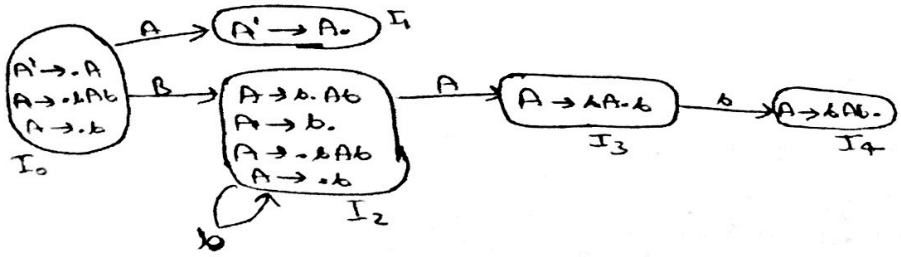
LR(0) DFA



G_2 | First | Follow

$A \to bAb \mid b$ | $\{b\}$ | $\{\$, b\}$

LR(0) DFA



G_3 | First | Follow

$A \to bab \mid c$ | $\{b, c\}$ | $\{\$, b\}$

LR(0) DFA

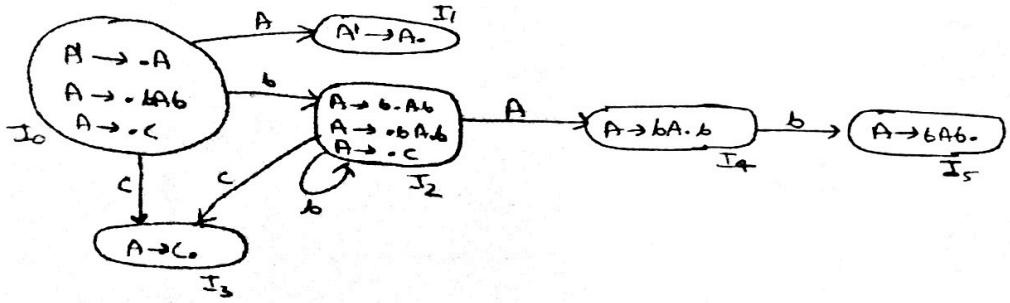


Figure 4: LR(0) DFA

(b) **What do you understand by left recursion and left factoring? Eliminate the left factoring and left recursion, if required, then construct LL(1) parse table for the following Grammar.**

S → qABC A
→ a / bbD B →
a / ϵ C → b / ϵ
D → c / ϵ

Solution:

Final LL(1) Parse Table (Non-terminals as rows, terminals as columns):

	q	a	b	c	$
S	S→qABC				
A		A→a	A→bbD		
B		B→a	B→ϵ		B→ϵ
C			C→b		C→ϵ
D		D→ϵ	D→ϵ	D→c	D→ϵ

Note: For D → ε, we have entries in M[D, a], M[D, b], M[D, $].

4. (a) Write a Syntax Directed Translation for following two grammars G1 and G2, to convert binary into equivalent decimal number i.e. the binary number 1001 equivalent to decimal number 9.

G_1	G_2
N → L	S→S0
L→LB	S→S1
L → B	S → ϵ
B → 0	
B → 1	

Solution:

Syntax Directed Translation for G1:

Production: N→LN

Action: N.val=L.val

Production: L→LB
Action: L.val=2×L1.val+B.val
(Here L1 is the left L in the production.)

Production: L→B

Action: L.val=B.val

Production: B→0B

Action B.val=0

Production B→1B

Action B.val=1

Syntax Directed Translation for G2:

Production S→S0S

Action S.val=2×S1.val+0=2×S1.val

Production S→S1S

Action S.val=2×S1.val+1

Production S→ϵS

Action S.val=0

(b) Consider the Given Grammar:

$$S \rightarrow ABCT \quad \{x = 5 \times x + 1\}$$
$$T \rightarrow a \quad \{x = 4 \times x + 1\}$$
$$C \rightarrow bb \quad \{x = 3 \times x - 3\}$$
$$B \rightarrow Be \quad \{x = 2 \times x + 1\}$$
$$B \rightarrow e \quad \{x = x + 1\}$$
$$A \rightarrow gBd \quad \{x = 5 \times x - 1\}$$

All semantic action updates the same global variable x. Assume x is initialized before parsing to zero. What is the final value of x in top-down parse of following i/p string "gedeebba" ?

Solution: The final value of x after parsing the given input string gedeebba is **606**.

5. **Consider the given code segment and answer the following questions:**

```
if (((a + b) < (c + d)) || ((e==f) && (g > h-k))) then
p= b*(-c) + b*(-d);
else
q=(-b)*(-b);
r= (-h)*(-k);
```

(a) Write the Three-Address Code (TAC)

Solution:

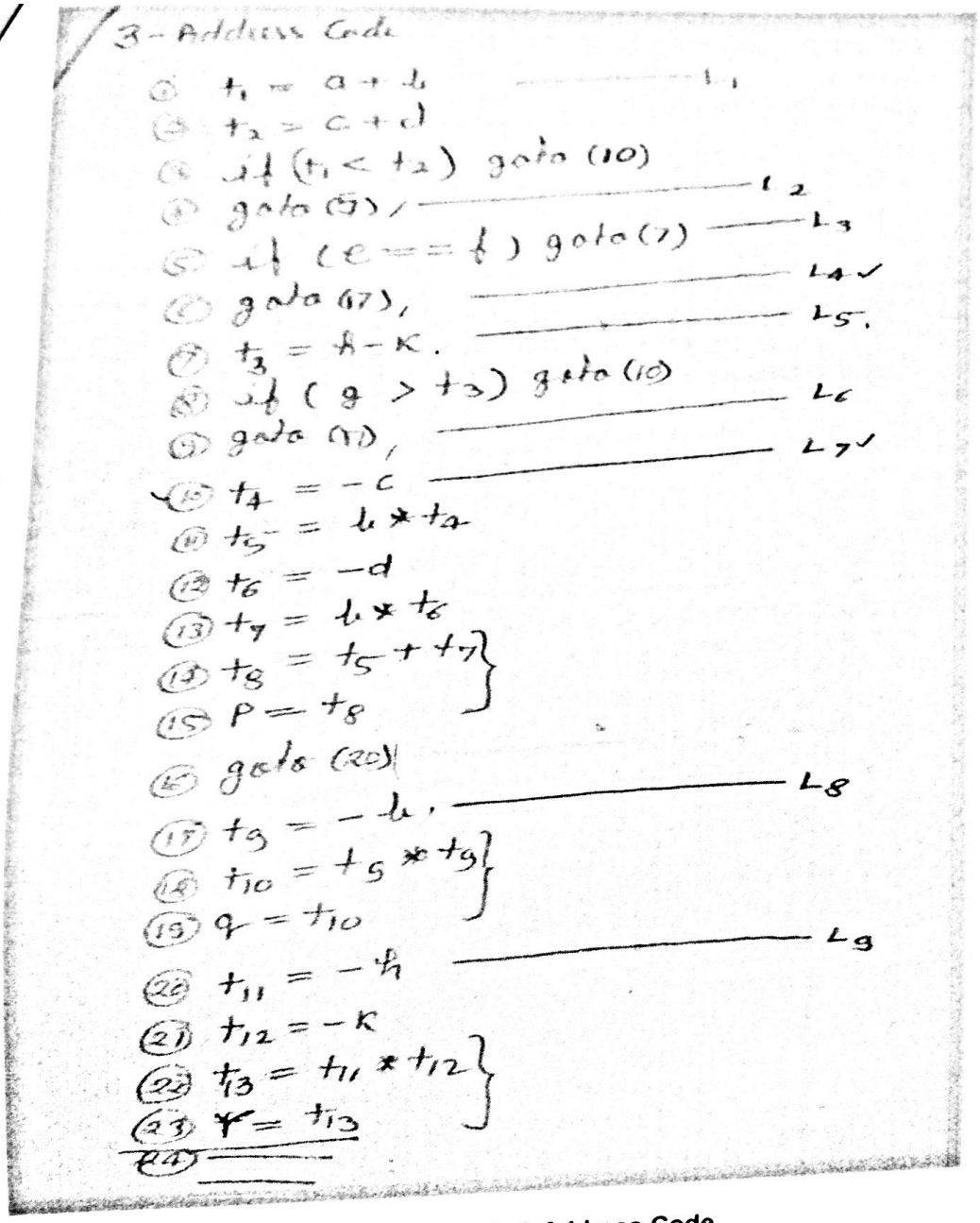


Figure 5: 3-Address Code

(b) Write procedure to select leaders for the construction of Basic Blocks

Solution: eaders are identified by the following rules:

1. The first instruction of the program is a leader.
2. The target of any conditional or unconditional jump is a leader.
3. Any instruction that immediately follows a conditional or unconditional jump is also a leader.

Applying these rules: As shown in Figure 5.

(c) Construct the Control Flow Graph (CFG)

Solution:

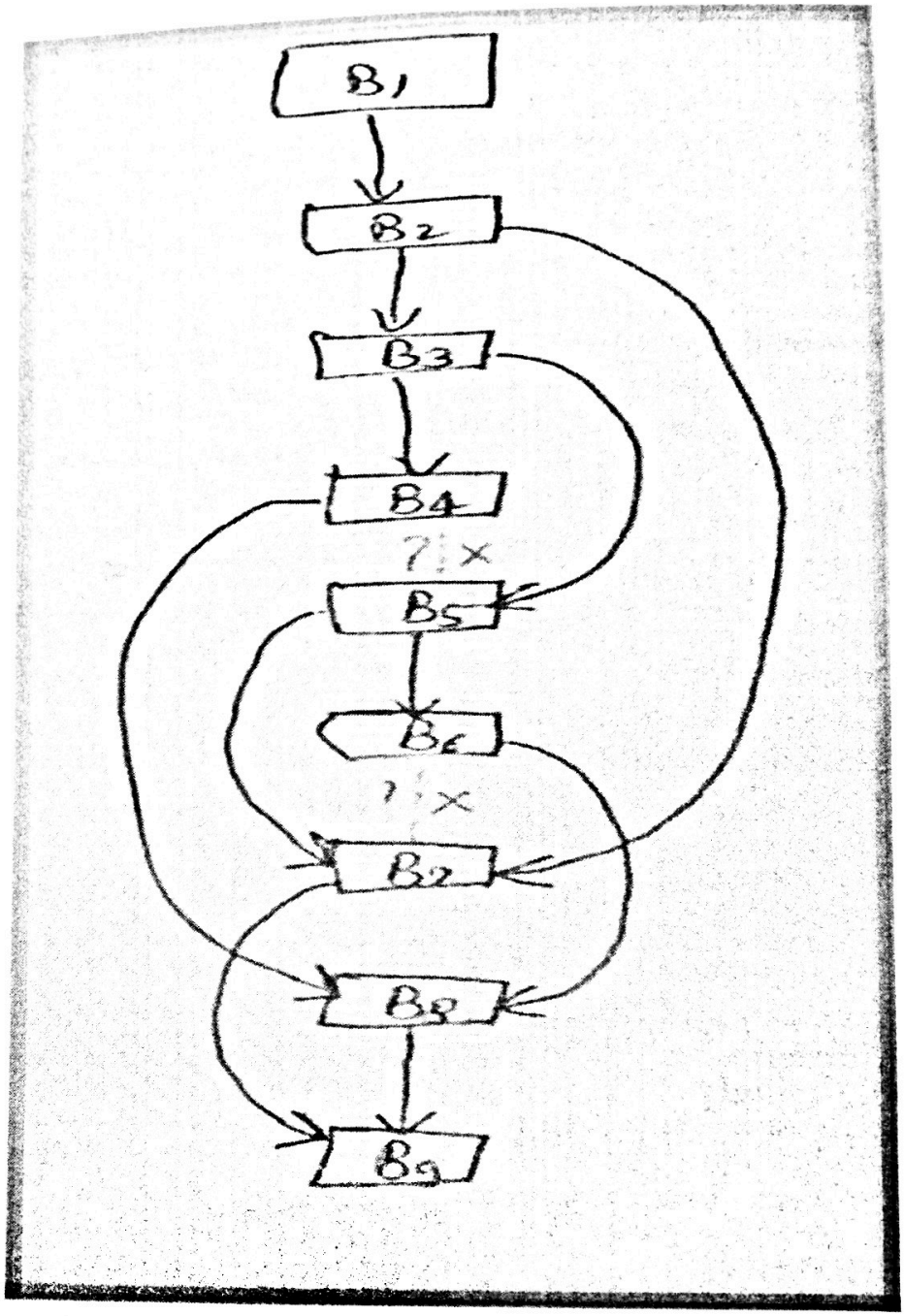


Figure 6: Control Flow Graph (CFG)

(d) Identify the basic blocks

Solution:

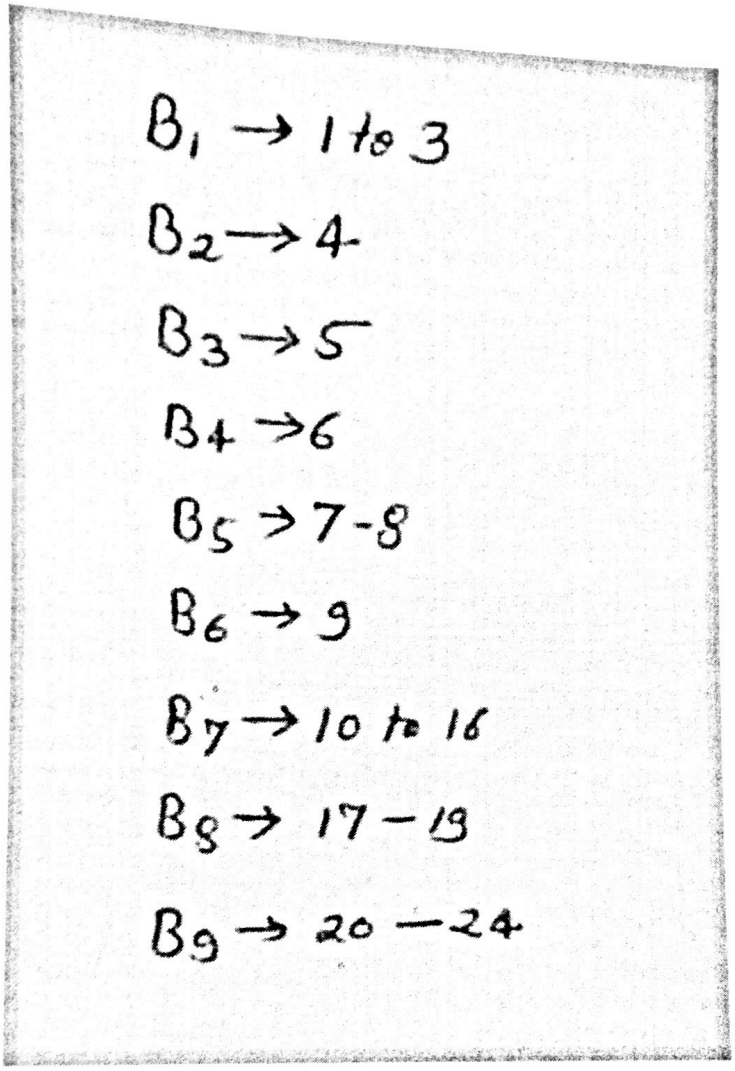


Figure 7: Basic Blocks