Date: / / /

* left factoring

$S \to \underbrace{ietses}_{\alpha\beta_1} \Big| \underbrace{iets}_{\alpha\beta_2} \ldots \Rightarrow$

$S \to \alpha A'$

$A' \to \beta_1 / \beta_2 \ldots$

$S \to iets\,S'$

$S' \to es / \epsilon$.

where α is the longest common string.

24/9/18

Parsing

Top-down parsing
(Derive i/p string from Stard symbol.)
- Easy to create
- LL(1)
  - L: Left to right scanning
  - L: Left most derivation
  - 1: i/p symbol

Bottom up parsing
(Backward derivation from i/p string to start symbol)
- Difficult
- LR
  - L: Left to right scanning
  - R: Right most derivative

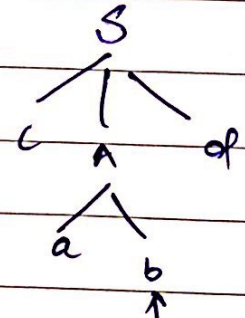
Top - Down Parsers

(1) Recursive descent parser. → Backtracking → Infinite loop for Left recursion.

i/p string w = cad.

S → cAd

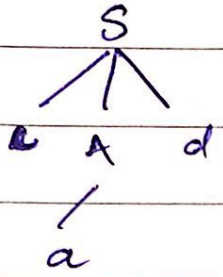
A → ab/a.



Not matching

Date: / / /

⇒ Backtracking



G ⟹ ① Remove LR. ② Left factoring } → Now the grammar will be without backtracking

② Predictive parser

(Recursive descent parser w/o backtracking).

$E \rightarrow TE'$

$E' \rightarrow +TE'/\epsilon$

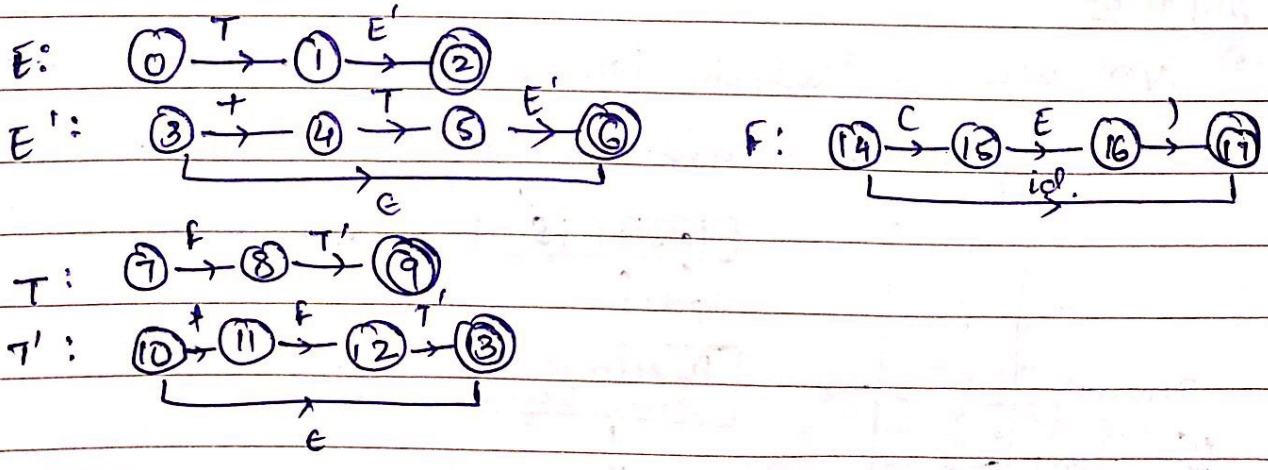
$T \rightarrow FT'$

$T' \rightarrow *FT'/\epsilon$

$F \rightarrow (E)/id.$

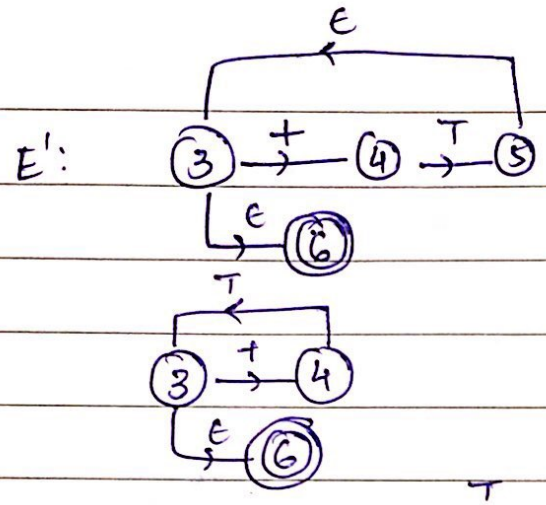
(Every non-terminal will have a transition diagram with single start & end/accepting state)

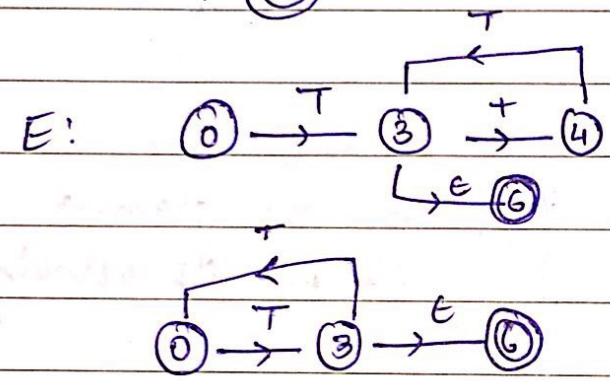
(Will only work if there is no non-determinism in transition diagrams).

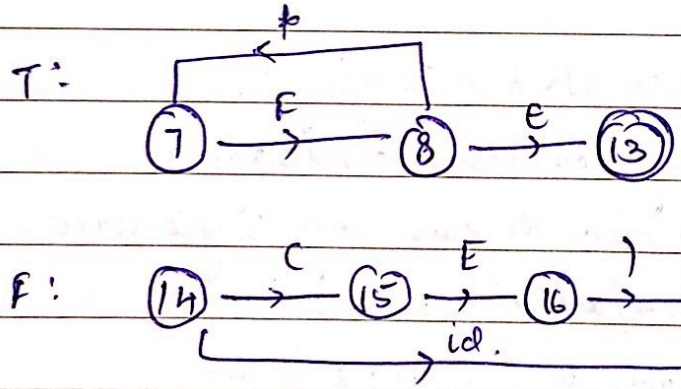


MATRIKAS

Date: / / /







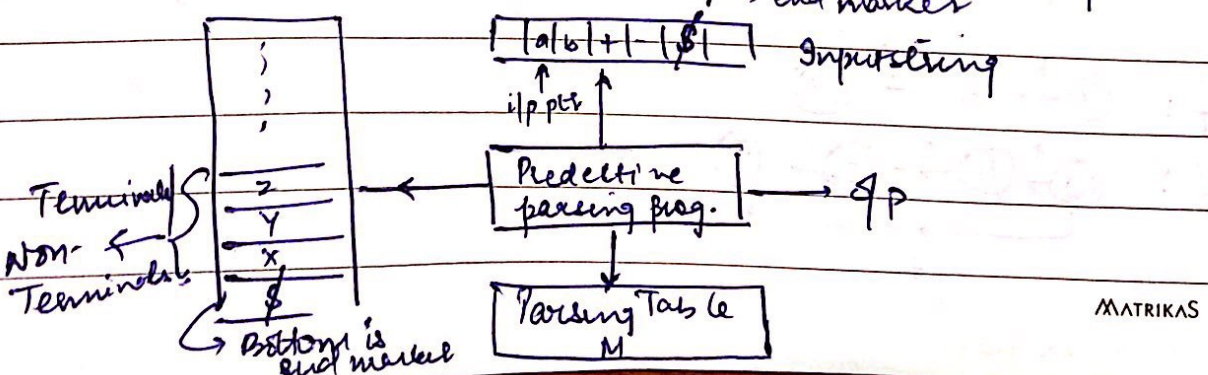
Final 3 transition diagrams that need to be converted to procedures

id + id * id.

25/9/18

③ Non-recursive Predictive Parser.

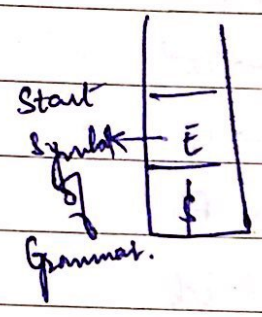
– Use of stack. to avoid ~~to~~ calling recursive procedures.



MATRIKAS

Date: / / /

id + id * id $



comparing i/p pointer to top of stack.
at end if $ - $ → Parsing possible
terminal x = x → Pop advance i/p pointer.
terminal x ≠ y → Not parsable.
terminal x & Non-terminal y → Use parsing table to push production according to terminal x after popping Y (in backward order of production).

Parsing Table.

Non-terminals \ Terminals	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ϵ	E' → ϵ
T	T → FT'			T → FT'		
T'		T' → ϵ	T' → *FT'		T' → ϵ	T' → ϵ
F	F → id			F → (E)		

all blank entries ⇒ error.

- LL(1) grammars ke liye predictive parser banaate hain.

LL(1) ⇔ predictive parse

- L to R scanning
- Leftmost Derivative
- Look ahead buffers in i/p string

MATRIKAS

Date: / / /

id + id * id $

(list the production used).

Stack	I/P string	O/P
$E	id + id * id $	~~E → TE'~~
$E'T	id + id * id $	E → TE'
$E'T'F	id + id * id $	T → FT'
$E'T'id	id + id * id $	F → id.
$E'T'	+ id * id $	
$E'	+ id * id $	T' → ϵ.
$E'T+	+ id * id $	E' → +TE'
$E'T	id * id $	
$E'T'F	id * id $	T → FT'
$E'T'id	id * id $	F → id.
$E'T'	* id $	
$E'T'F*	* id $	T' → *FT'
$E'T'F	id $	
$E'T'id	id $	F → id
$E'T'	$	
$E'	$	T' → ϵ
$	$	E' → ϵ.

$ = $

⇒ Parsed.

MATRIKAS

Date: 26/9/18.1

Parsing table calculation.

$E \rightarrow TE'$

$E' \rightarrow +TE' / \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' / \epsilon$

$F \rightarrow (E) / id.$

FIRST

- Terminals & Non-terminals
- FIRST(A) = {All terminals that begin string derived from A}.
- First of Terminal is terminal itself.

→ $X \rightarrow Y_1 Y_2 Y_3 \ldots Y_i$

① FIRST(X) → FIRST(Y_1)

$Y_1 \in T$

$Y_1 \in NT$.

② if $Y_1 \overset{*}{\Rightarrow} \epsilon$

FIRST(X) → FIRST(Y_1) ∪ FIRST(Y_2)

and checks so on.

② if $Y_1, Y_2 \ldots$ all derive ϵ FIRST(X) ⇒ will include FIRST(Y_1) ... (Y_i) union and ϵ also.

FOLLOW

- ~~Tet~~ Non-Terminals
- FOLLOW(B) ($A \rightarrow \alpha B \beta$)

① FOLLOW(B) = {All terminals that follow B. ⇒ FIRST(β)}.

② If FIRST(β) contains ϵ or we can say $\beta \overset{*}{\Rightarrow} \epsilon$. then FOLLOW(B) = FIRST(β) ∪ FOLLOW(A).

③ Start symbol follow contains $

FOLLOW(S) = {$ ...

MATRIKAS

Date: / / /

FIRST

FIRST(E) = FIRST(T) = FIRST(F)
= { (, id }

FIRST(E') = { +, ϵ }

FIRST(T) = ~~{ *, ϵ }~~ . ✓

FIRST(T') = ~~{~~ { *, ϵ }.

FIRST(F) = ✓

① For each terminal a in FIRST(A) add A → α to M[A, a]

② If ϵ is in FIRST(α) add A → α to M[A, b] for each terminal b in FOLLOW(A).

FOLLOW.

FOLLOW(E) = { $,) }.

FOLLOW(E') = FOLLOW(E)
= { $,) }.

FOLLOW(T) = FIRST(E')
∪ FOLLOW(E') ∪
FIRST(E) ∪ FOLLOW(E).
= { +, $,) }.

FOLLOW(T') = FOLLOW(T)
= { +, $,) }.

FOLLOW(F) = FIRST(T') ∪
FOLLOW(T') ∪ FIRST(T')
∪ FOLLOW(T)
= { *, +, $,) }.

	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ϵ	E' → ϵ.
T	T → FT'			T → FT'		
T'		T' → ϵ	T' → *FT		T' → ϵ	T' → ϵ.
F	F → id			F → (E)		

MATRIKAS

Date: 28/9/18

Q $S \rightarrow iEtSS' / a$
$S' \rightarrow eS / \epsilon$
$E \rightarrow b$.

Construct a parsing table and tell if LL(1) grammar or not.

FIRST

FIRST(S) = {i, a}.
FIRST(S') = {e, ε}.
FIRST(E) = {b}.

FOLLOW

FOLLOW(S) = {$} ∪ FIRST(S') ∪ FOLLOW(S') ∪ FOLLOW(S') = {$, e}.
FOLLOW(S') = FOLLOW(S) = {$, e}.
FOLLOW(E) = {t}.

	i	t	e	a	b	$
S	S→iEtSS'			S→a.		
S'			S'→ε S'→eS			S'→ε.
E					E→b.	

It is not LL(1) grammar as we have 2 options when top of stack is S' and i/p ptr is at e.
→ Conflict; therefore we cannot make a predictive parser.

MATRIKAS

Date: / / /

Q $S \to (L) \mid a$ $(a, (a, a))$.

$L \to L, S \mid S$

α β

$S \to (L) \mid a$

$L \to SL'$

$L' \to , SL' \mid \epsilon$.

FIRST

FIRST(S) = {(, a}.

FIRST(L) = FIRST(S)

= {(, a}.

FIRST(L') = {, , ε}.

FOLLOW.

FOLLOW(S) = FIRST(L') ∪ FOLLOW(L)

∪ FOLLOW(L') ∪ {$}.

= {$, , ,)}.

FOLLOW(L) = {)}.

FOLLOW(L') = FOLLOW(L) = {)}.

	(	)	,	a	$
S	S→(L)			S→a	
L	L→SL'			L→SL'	
L'		L'→ε	L'→,SL'		

MATRIKAS

Date: / / /

Stack	ip string	O/p
$ S	(a, (a,a)) $	
$) L (	(a, (a,a)) $	S → (L)
$) L	a, (a,a)) $	
$) L'S	a, (a,a)) $	L → SL'
$) L'a	a, (a,a)) $	S → a
$) L'	, (a,a)) $	
$) L'S,	, (a,a)) $	L' → , SL'
$) L'S	(a,a)) $	
$) L') L (	(a,a)) $	S → (L)
$) L') L	a,a)) $	
$) L') L'a	a,a)) $	L → SL'
$) L') L'a	a,a)) $	S → a
$) L') L'	,a)) $	
$) L') L'S,	,a)) $	L' → , SL'
$) L') L'S	a)) $	
$) L') L'a	a)) $	S → a
$) L') L'	)) $	
$) L')	)) $	L' → ε
$) L'	) $	
$)	) $	L' → ε
$	$	

$ = $

⇒ Hence Parsed.

MATRIKAS

1/10/18 Missed class. Date: / / /

5/10/18 Operator Precedence Grammar.

* $a <\cdot b \rightarrow$ b higher precedence

$a \cdot> b \rightarrow$ b lower precedence

$a \doteq b \rightarrow$ equal precendence.

* No two non-terminals will come together

Operator predecence table.

	id	+	*	$
id	—	$\cdot>$	$\cdot>$	$\cdot>$
+	$<\cdot$	$\cdot>$	$<\cdot$	$\cdot>$
*	$<\cdot$	$\cdot>$	$\cdot>$	$\cdot>$
$	$<\cdot$	$<\cdot$	$<\cdot$	—

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id.$

$ id + id * id $

→ $ $<\cdot$ id $\cdot>$ + $<\cdot$ id $\cdot>$ * $<\cdot$ id $\cdot>$ $

find first closing bracket and then backtrack to find corresponding closing and this section is an handle

→ $ $\cdot$ E + E * E $\cdot$ $

→ $ $<\cdot$ + $<\cdot$ * $\cdot>$ $

→ $ $<\cdot$ + E $

→ $ $<\cdot$ + $\cdot>$ $

→ $ E $

→ $ $.

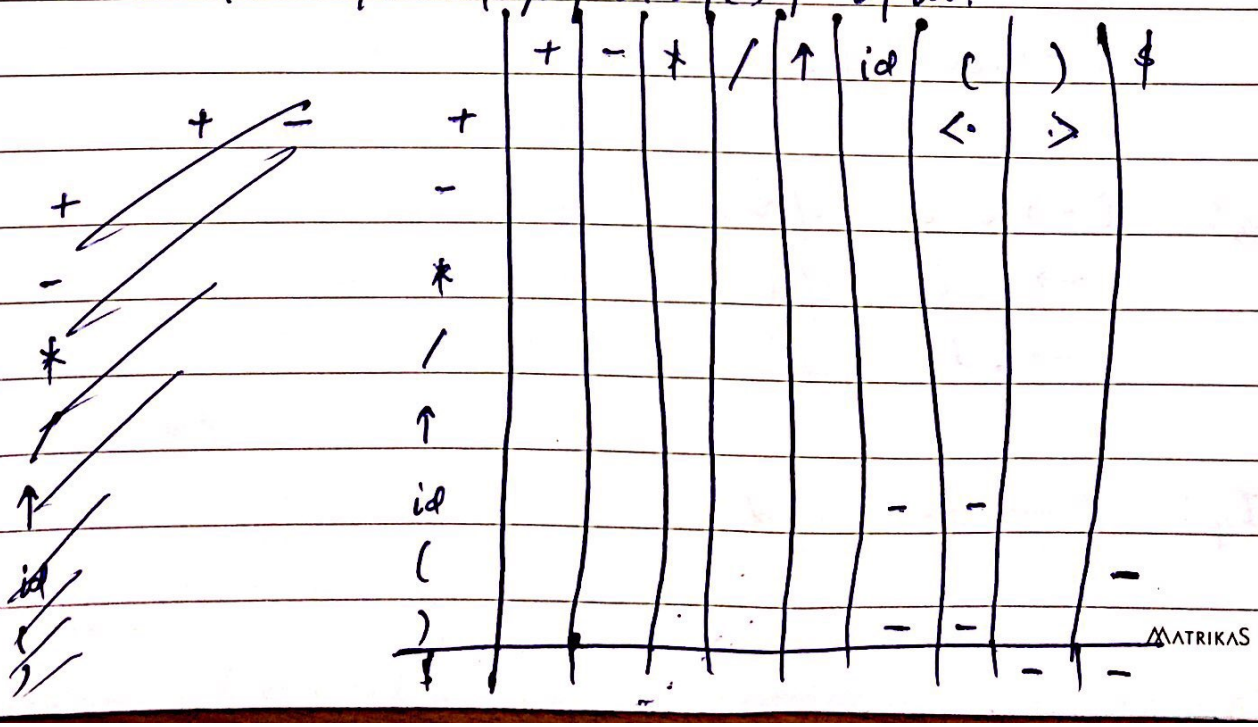
MATRIKAS

Date: / / /

Stack	i/p string	Action
$	id + id * id $	shift
$ <· id	+ id * id $	reduce, E → id.
$	+ id * id $	shift
$ <· +	id * id $	shift
$ <· + <· id	* id $	reduce, E → id.
$ <· +	* id $	shift.
$ <· + <· *	id $	shift.
$ <· + <· * <· id	$	reduce, E → id
$ <· + <· *	$	reduce, E → E * E.
$ <· +	$	reduce E → E + E
$	$	

id * (id ↑ id) − id / id.

E → E+E | E*E | E−E | E/E | E↑E | (E) | −E | id.



9/10/18 Missed Class

Date: / / /

15/10/18

Canonical LR parser.

$S' \to S$ r_0

$S \to CC$ r_1

$C \to cC$ r_2

$C \to d$ r_3

- LR(1) set of items.
- No. of states are higher.

I_0 : $S' \to .S$, $\$$

$S \to .CC$, $\$$

$C \to .cC$, c/d

$C \to .d$, c/d

$A \to \alpha . B\beta$, a

$B \to .Y$, $first(\beta)$

if β is ϵ

then ~~first~~ a $first(a)$

∴ in general.

$B \to .Y$, $first(\beta a)$

I_1 : $S' \to S.$, $\$$

I_2 : $S \to C.C$, $\$$

$C \to .cC$, $\$$

$C \to .d$, $\$$.

I_3 : $C \to c.C$, c/d.

$C \to .cC$, c/d

$C \to .d$, c/d.

I_4 : $C \to d.$, c/d.

MATRIKAS

Date: / / /

I_5: $S \to CC\cdot, \$$

I_6: $C \to c\cdot C, \$$
$C \to \cdot cC, \$$
$C \to \cdot d, \$$

I_7: $C \to d\cdot, \$$

I_8: $C \to cC\cdot, c/d$.

I_9: $C \to cC\cdot, \$$

	c	d	$	S	C
0	s3	s4		1	2
1			accept		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

MATRIKAS

Date: / / /

LALR Parser

- Find 3 states where set of items are same even when lookahead isn't same.

I_3 I_6
I_4 I_7 } → Combine.
I_8 I_9

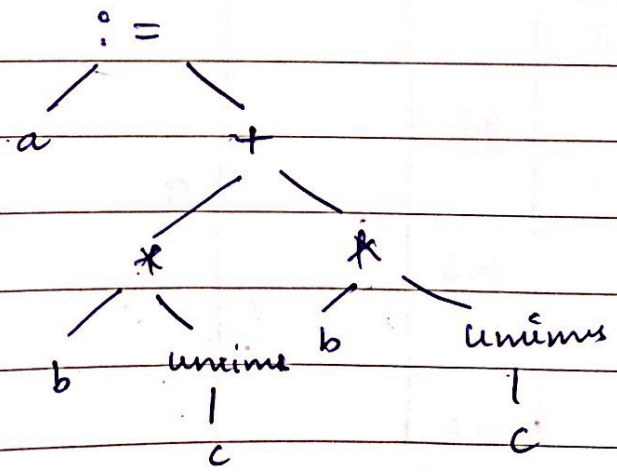
- Finally the states are reduced.

INTERMEDIATE CODE GENERATION.

3 ways of representation
→ Syntax tree
→ Postfix ~~exe~~ notation.
→ Three-address code.

a := b * -c + b * -c.

(1)



MATRIKAS

Date: / / /

(2) a b c uminus * b c uminus + :=

(3) $t_1 := -c$

$t_2 := b * t_1$

$t_3 := -c$

$t_4 := b * t_3$

$t_5 := t_2 + t_4$

$a := t_5$

Implementation in compiler → quadruples
(in form of records being saved). triples
Indirect triples.
↳ refer triple records.

quadruples

	op	arg 1	arg 2	result.
(0)	uminus	c		t_1
(1)	*	b	t_1	t_2
(2)	uminus	c		t_3
(3)	*	b	t_3	t_4
(4)	+	t_2	t_4	t_5
(5)	:=	t_5		a

triples.

(Need to change all records if any restructuring happens).

	op	arg 1	arg 2
(0)	uminus	- c	
(1)	*	b	(0)
(2)	uminus	- c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	:=	a	(4)

MATRIKAS

Date: 17/10/18

Intermediate Code Generation & Code Optimization.

```
begin X
prod = 0;
i = 1;
do begin
    prod = prod + a[i] * b[i];
    i = i+1;
end
while i <= 20
End.
```

Only convert assignment ~~and~~ statements, loops. and ~~code~~ conditional, ~~also~~ statements, arithmetic statements.

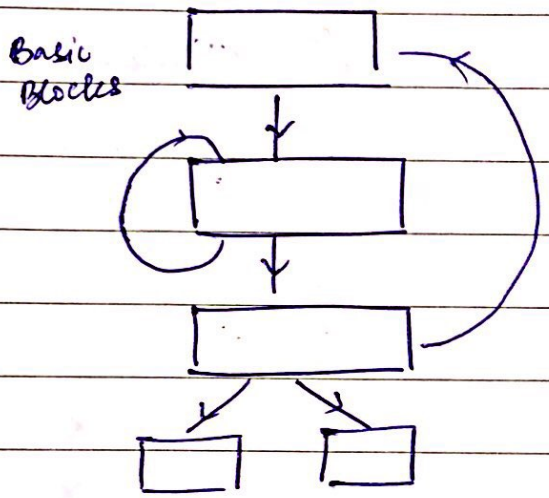
1. prod = 0
2. i = 1
3. $t_1 = 4 * i$
4. $t_2 = a[t_1]$
5. $t_3 = 4 * i$
6. $t_4 = b[t_3]$
7. $t_5 = t_2 * t_4$
8. $t_6 = prod + t_5$
9. $prod = t_6$
10. $t_7 = i + 1$
11. $i = t_7$
12. if ($i <= 20$) goto (3). [⇒ 3 is a leader]

MATRIKAS

Date: / / /

3 address code ⇒ Flow graph.

Flow Graph: →



1st statement in the basic block is a "leader" and other statements will follow until a new leader is encountered.

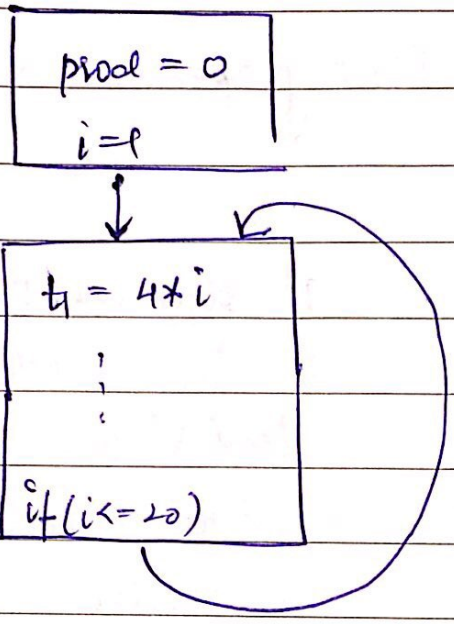
① 1st statement of a 3-address code is a leader.

② Any conditional/unconditional goto's target is a leader.

③ ~~Every~~ Immediate statement following any goto is a leader.

MATRIKAS

Date: / / /



B1
(Basic Block 1)

B2
(Basic Block 2)

```
void quicksort(m, n)
int m, n;
{
    int i, j;
    int v, x;
    if (n<=m) return;
    i = m-1; j = n; v = a[n];
    while (1) {
        do i = i+1; while (a[i]< v);
        do j = j-1; while (a[j] >v);
        if (i>=j) break;
            x = a[i]; a[i] = a[j]; a[j] = x;
    }
    x = a[i]; a[i] = a[n]; a[n] = x;
    quicksort (m,j);
    quicksort (i+1, n);
}
```

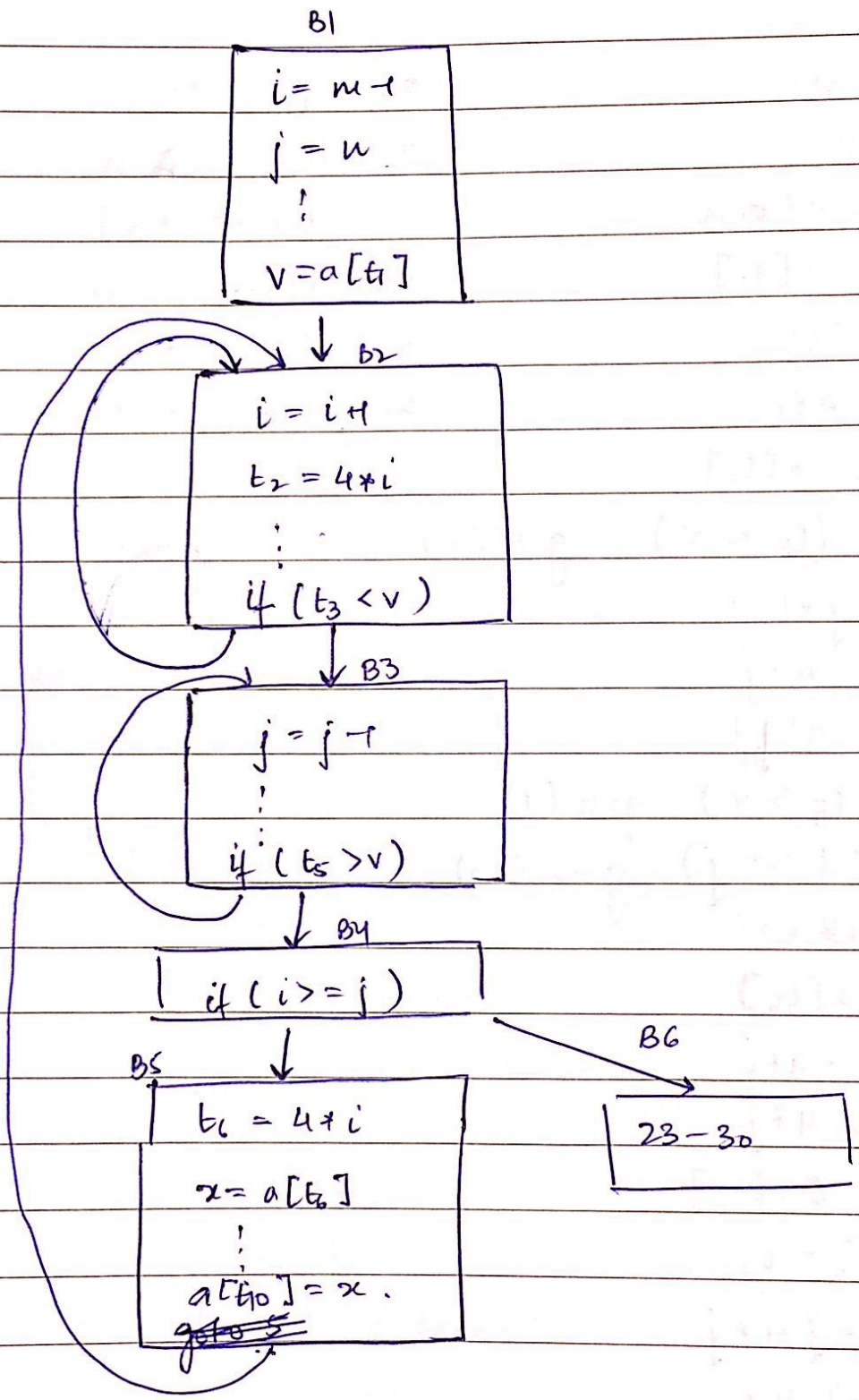
MATRIKAS

Date: / / /

1. $i = m-1$
2. $j = n$
3. $t_1 = 4 * n$
4. $v = a[t_1]$
5. $i = i+1$
6. $t_2 = 4*i$
7. $t_3 = a[t_2]$
8. if $(t_3 < v)$ goto (5)
9. $j = j-1$
10. $t_4 = 4*j$
11. $t_5 = a[t_4]$
12. if $(t_5 > v)$ goto (9)
13. if $(i >= j)$ goto (23)
14. $t_6 = 4*i$
15. $x = a[t_6]$
16. $t_7 = 4*i$
17. $t_8 = 4*j$
18. $t_9 = a[t_8]$
19. $a[t_7] = t_9$
20. $t_{10} = 4*j$
21. $a[t_{10}] = x$
22. goto (5)
23. $t_{11} = 4*i$
24. $x = a[t_{11}]$
25. $t_{12} = 4*i$
26. $t_{13} = 4*n$
27. $t_{14} = a[t_{13}]$
28. $a[t_{12}] = t_{14}$
29. $t_{15} = 4*n$
30. $a[t_{15}] = x$

MATRIKAS

Date: / / /

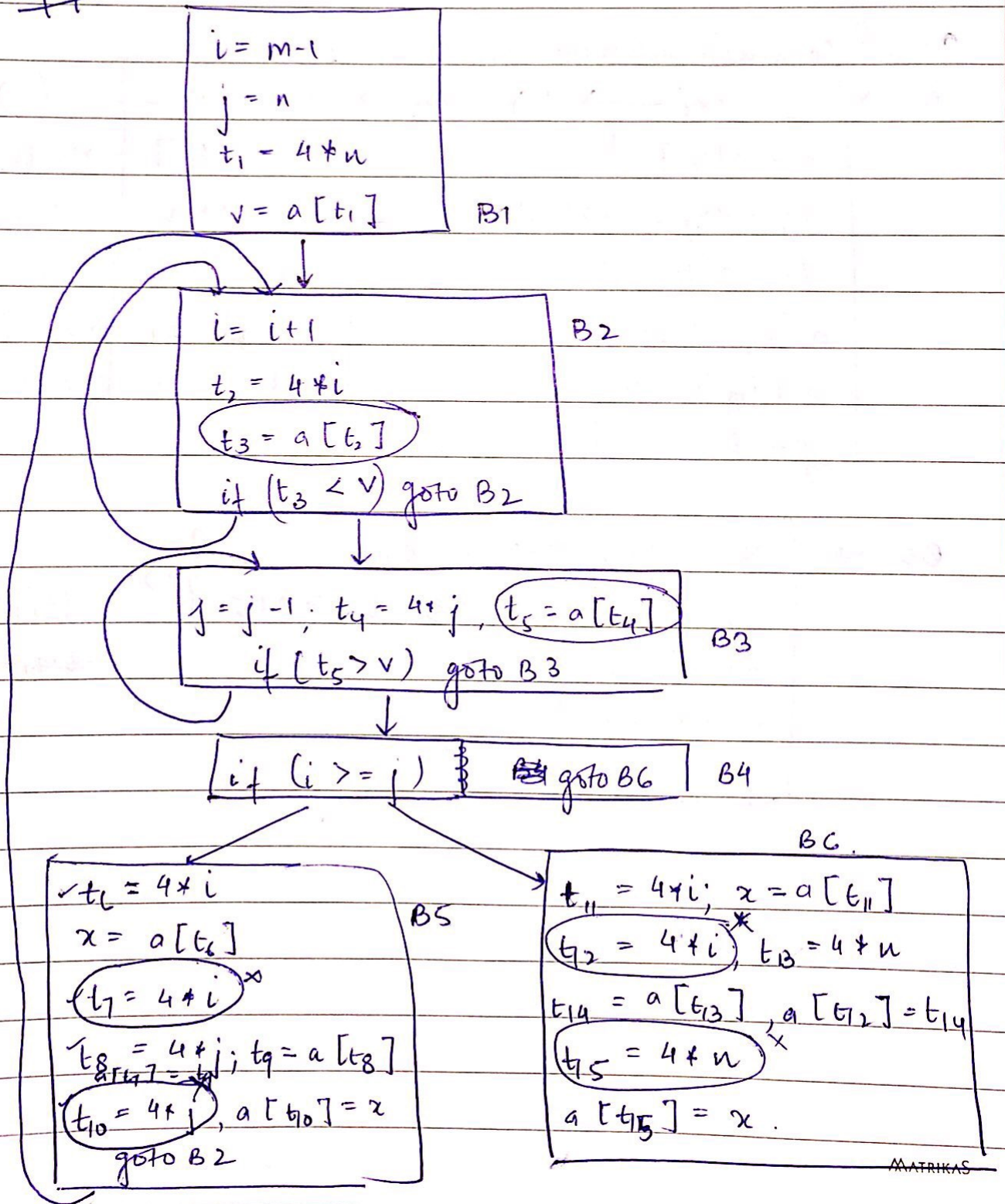


MATRIKAS

Date : / / /

- Local optimization within a block.
- Global optimization is among all blocks.

23/10/18



Date: / / /

Code Optimization Techniques.

1. Common Subexpression elimination.
2. Dead code elimination.
3. Copy Propagation.

1. Common subexpression elimination.

B5 ⇒

$t_6 = 4*i$	$\times(t_2)$
$x = a[t_6]$	
$t_8 = 4*j$	$\times(t_4)$
$t_9 = a[t_8]$	
$a[t_6] = t_9$	
$a[t_8] = x$	
goto B2	

B6 ⇒

$t_{11} = 4*i$	$\times(t_2)$
$x = a[t_{11}]$	$\rightarrow x = (t_3)$
$t_{13} = 4*n$	$\times t_4$
$t_{14} = a[t_{13}]$	$\rightarrow t_{14} = a[t_4]$
$a[t_{12}] = t_{14}$	$\rightarrow a[t_2] = t_{14}$
$a[t_{13}] = x$	$\rightarrow a[t_4] = x.$

B5 ⇒

$x = a[t_2]$	$\rightarrow x = t_3$
$t_9 = a[t_4]$	$\rightarrow$ ~~t_5~~ $t_9 = t_5$
$a[t_2] = t_9$	
$a[t_4] = x$	
goto B2	

$\Rightarrow$ $a[t_2] = t_5$

~~$a[t_4] = x$~~.

MATRIKAS