Machine - Independent optimizations.

Global code optimization - Improvements taken into account across basic blocks.

- based on data-flow-analysis.

The principal sources of optimization.

Semantics - preserving Transformations.
(function-preserving).

A compiler can improve a program without changing the function it computes.

1. common-subexpression elimination < Local / Global
2. copy propagation
3. dead code elimination
   code motion
4. constant folding.

Induction variables and Reduction in strength.

common - subexpr elimination.

eg:- for local common subexpr elimination.

Before

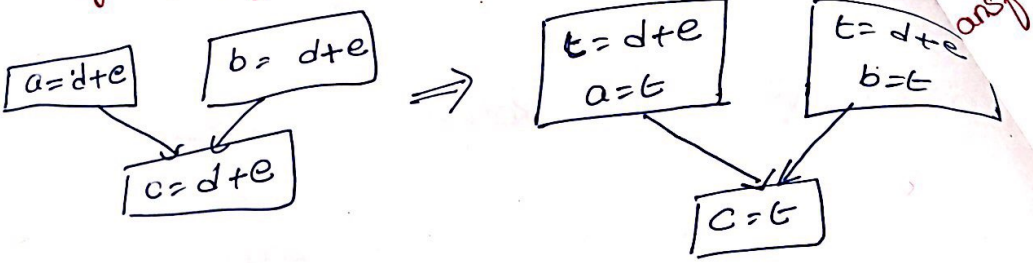
```
t6 = 4*i
x = a[t6]
t7 = 4*i
t8 = 4*j
t9 = a[t8]
a[t7] = t9
t10 = 4*j
a[t10] = x
goto B2
```

After

```
t6 = 4*i
x = a[t6]
t8 = 4*j
t9 = a[t8]
a[t6] = t9
a[t8] = x
goto B2.
```

copy propagation.

eg1:



eg2:

$x = t_3$
$a[t_2] = t_5$
$a[t_4] = x$
goto B_2

$\Rightarrow$

$x = t_3$
$a[t_2] = t_5$
$a[t_4] = t_3$

Dead code elimination.

```
if (debug) print
        :
debug = FALSE
```

Induction variables and Reduction in strength

code motion

which moves the code outside the loop.

```
while (i <= Max-1)
{
    sum = sum + a[i];
}
```

⇓

```
n = max-1
while (i<=n)
{   sum = sum + a[i];
}
```

transfer functions.

$$out[S] = f_S(in[S])$$
$$IN[S] = f_S(out[S])$$

IN[S] & out[S] - data flow values before and after each statement S.

Reaching definitions:

We call a definition of a variable a, if b/w two points along the path, there is an assignment to a.

~~data flow~~ equations:

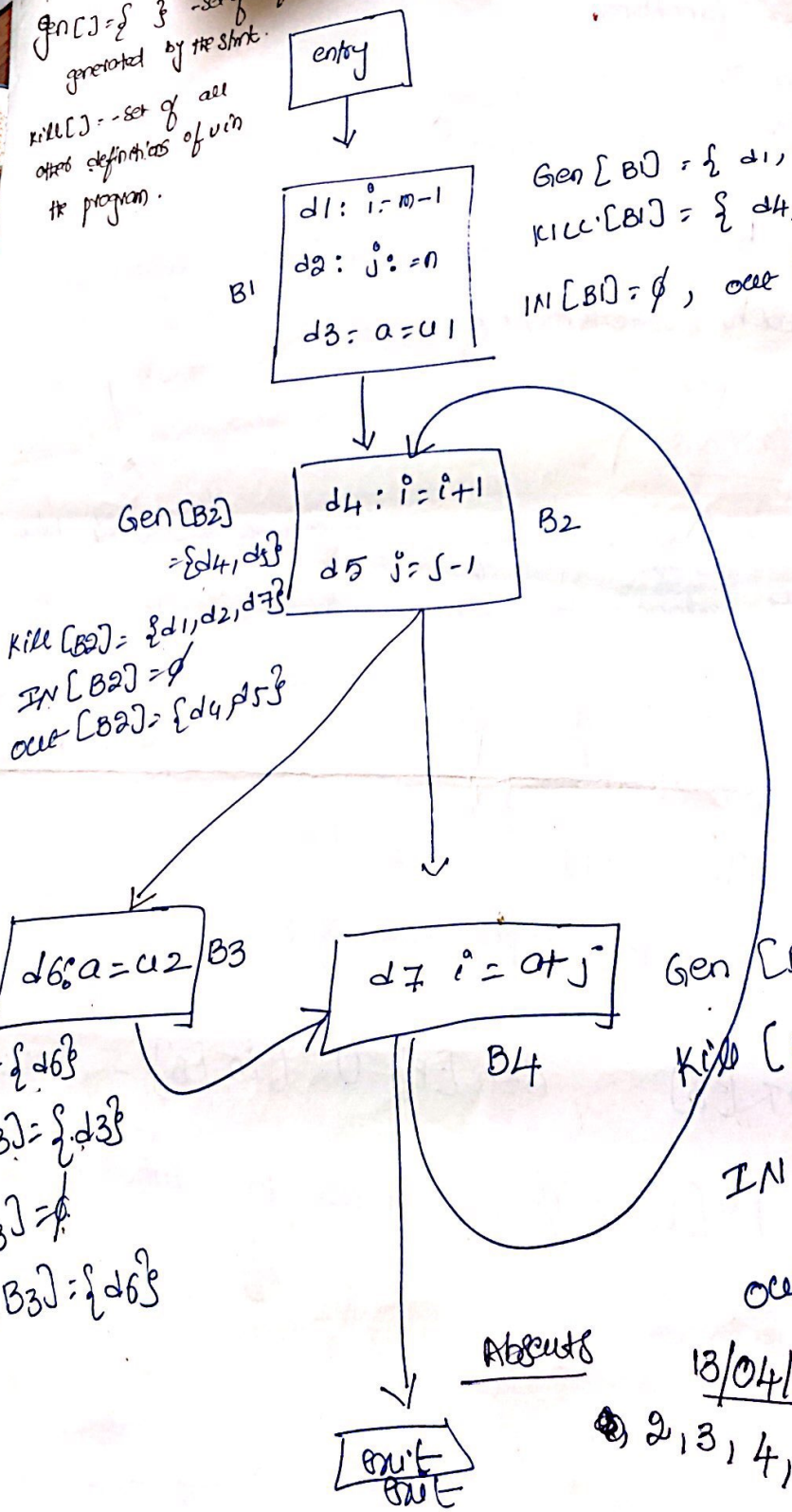
$$IN[B] = \bigcup_{P \text{ is a predecessor of } B} OUT[P]$$

$$OUT[B] = Gen[B] \cup (in[B] - KILL[B])$$

$IN[B] = \phi$, for all B (initialization only)

Gen[ ] = { } - set of definitions generated by the stmt.

Kill[ ] - set of all other definitions of v in the program.



Gen [B1] = { d1, d2, d3 }

KILL [B1] = { d4, d5, d6, d7 }

IN [B1] = ∅, OUT [B1] = {d1, d2, d3}

Gen [B2] = {d4, d5}

Kill [B2] = {d1, d2, d7}

IN [B2] = ∅

OUT [B2] = {d4, d5}

Gen [B3] = {d6}

Kill [B3] = {d3}

in [B3] = ∅

out [B3] = {d6}

Gen [B4] = {d7}

Kill [B4] = {d1, d4}

IN [B4] = ∅

OUT [B4] = {d7}

Absents

13/04/15 1.35 to 2.25

2, 3, 4, 5, 6, 7, 10,

12, 20, 21, 22, 24,

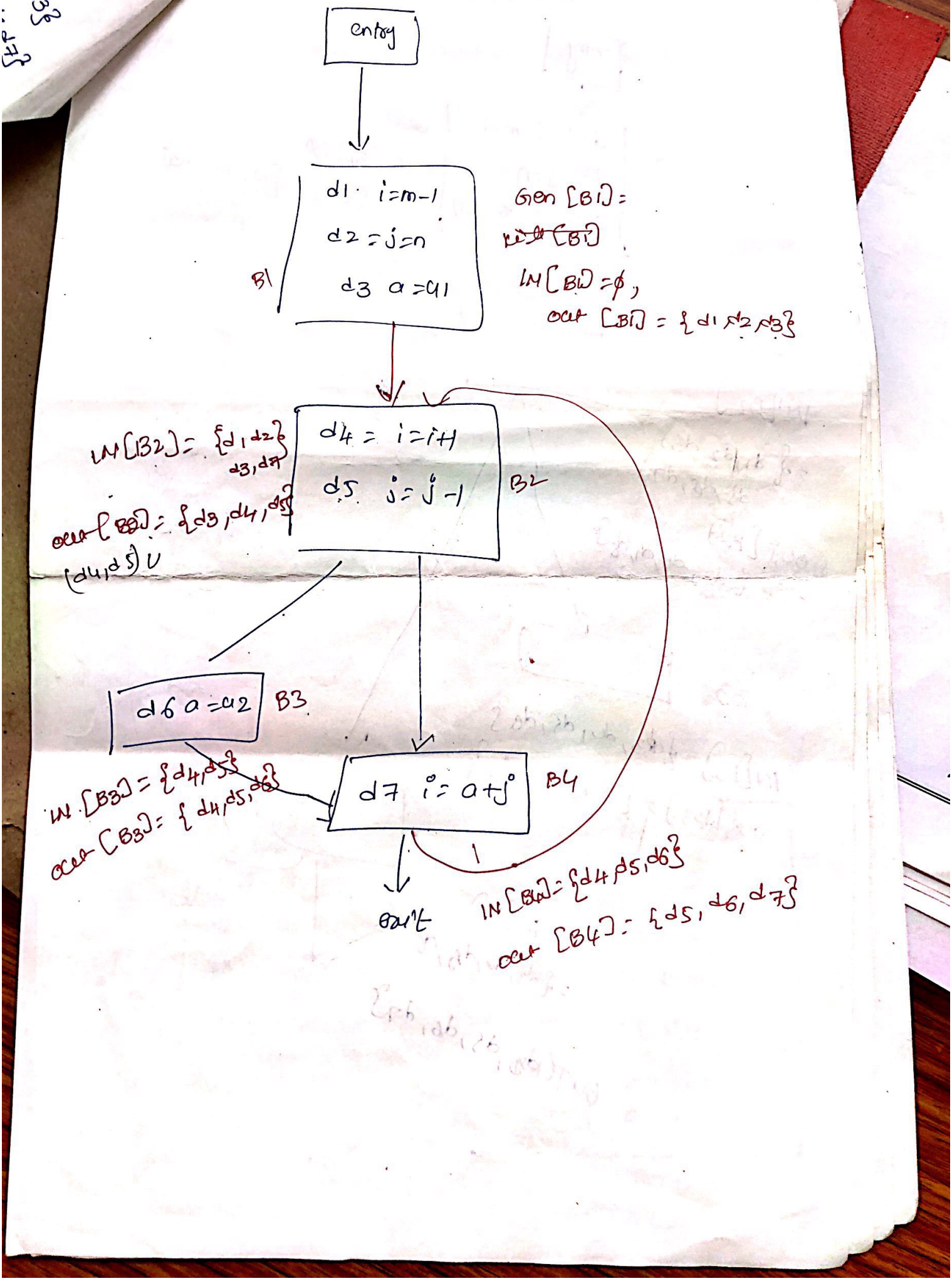
26, 27, 28, 29, 3

31, 36, 37, 38, 39, 45, 46,

48, 49, 55, 58, 59, 60, 61,

302, 304, 307, 309,

33



Gen [B1] =

IN [B1] = ϕ,

out [B1] = {d1, d2, d3}

IN [B2] = {d1, d2, d3, d7}

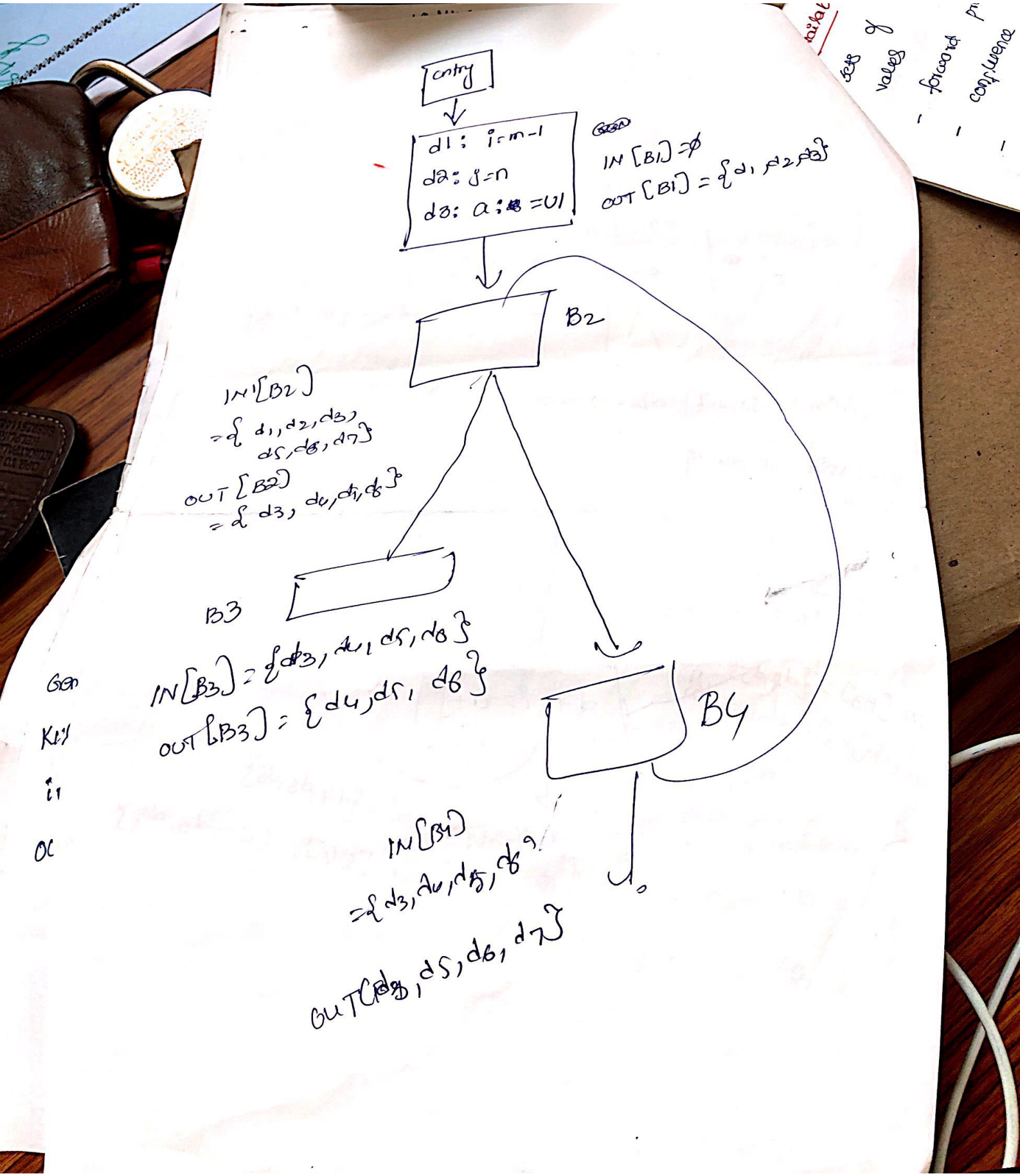
out [B2] = {d3, d4, d5} ∪ (d4, d5)

IN [B3] = {d4, d5}

out [B3] = {d4, d5, d6}

IN [B4] = {d4, d5, d6}

out [B4] = {d5, d6, d7}



Available Expression computation.

- sets of expressions constitute the domain of data flow values
- forward problem.
- confluence operator is $\wedge$
- An expression $x+y$ is available at a point p, if every path from the initial node to p evaluates $x+y$, and after the last such evaluation, prior to reaching p, there are no subsequent assignments to x or y.
- A block kills $x+y$, if it assigns to x or y and does not subsequently recompute $x+y$.
- A block generates $x+y$, if it definitely evaluates $x+y$, and does not subsequently redefine x or y.

eg:-

B

d1 :	a = f+1
d2 :	b = a+7
d3 :	c = b+d
d4 :	a = d+c

In other blocks

d5 :	b = a+4
d6 :	f = e+c
d7 :	e = b+d
d8 :	d = a+b
d9 :	a = c+f
d10 :	c = e+a

subset of set of all expressions form the domain of data flow values.

set of all expressions = { f+1, a+7, b+d, d+c, a+4, e+c, a+b, c+f, e+a }

EGEN [B] = { f+1, b+d, d+c }

EKILL [B] = { a+4, a+b, e+a, e+c, c+f, a+7 }

EGEN [B] : In this f+1 - f is not assigned latter

b+d - b or d are not assigned latter

d+c - d or c not assigned any value after the expr. in this block

a+7 → here a is re-defined latter and after that after that a+7 is not re computed

So a+7 not generated.

→ what are the generated expressions those are available at end of the block.

EKILL (B):

a = f+1 → all the expr involves 'a' should be killed

f+1 assigns :

a a+7 should not be killed :- it is after the definition of "a". prior. before the definition of "a" or in another blocks should be killed.

i.e a+4, a+b, e+a.

~~a+7~~

a = d+c kill a+7 ∵ a+7 appears before definition a = d+c.

Data - Flow Equations.

$$IN[B] = \bigcap_{P \text{ is a predecessor of } B} OUT[P], \quad B \text{ not initial}$$

$$OUT[B] = e\text{-}gen[B] \cup (IN[B] - e\text{-}kill[B])$$

$IN[B1] = \phi$ (permanently and no change)

$IN[B] = U$, for all $B \neq B1$ (initialization only)

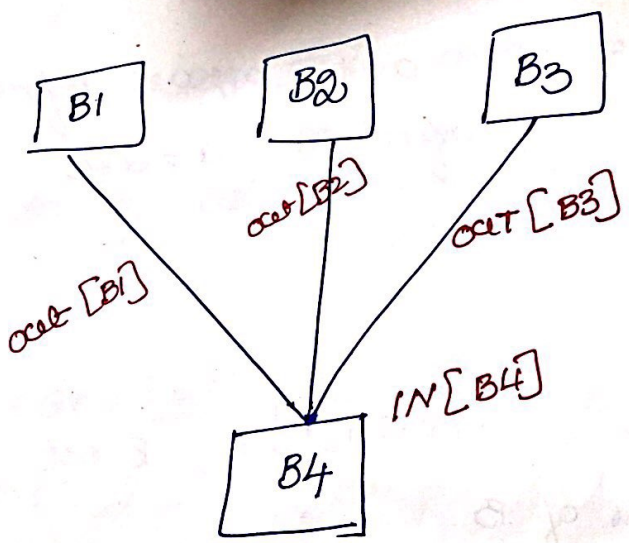
e-gen, e-kill are not changed.

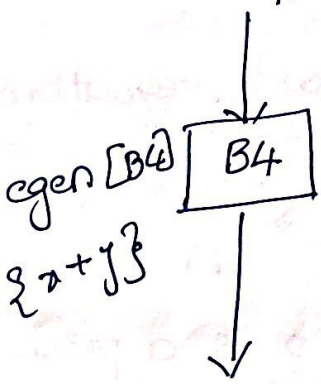
B1 – the initial (or) entry block and is special because nothing is available when the program begins execution

IN[B1] – always ϕ.

U – universal set of all expressions

Initializing IN[B] to ϕ for all $B \neq B1$, is restrictive.





IN [B4] = out [B1] ∩ out [B2] ∩ out [B3]

out [B4] = egen [B4] ∪ (in [B4] - ekill [B4])

out [B4] { x+y, P+q}

[B4] 2a+b

Live variable Analysis.

(3)

The variable x is live at the point p, if the value of x at p could be used along some path in the flow graph., starting at p, otherwise x is dead at p.

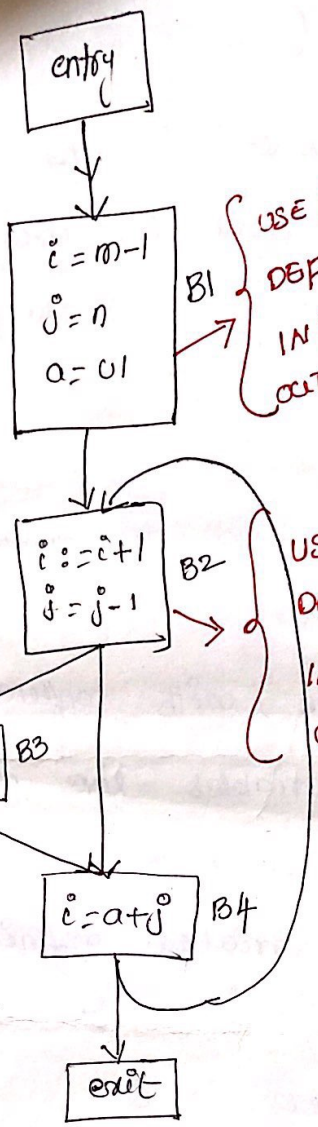
- sets of variables constitute the domain of data-flow values.
- Backward flow problem, with confluence operator ∪
- IN[B] - set of variables live at the beginning of B.
- OUT[B] - set of variables live just after B.
- (Kill) DEF[B] - set of variables definitely assigned values in B, prior to any use of that variable in B.
- USE[B] - set of variables whose values may be used in B prior to any definition of the variable. (Gen)

$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \cup (OUT[B] - DEF[B])$$

$IN[B] = \emptyset$, for all B (initialization only)

21, 33, 40, 51



USE [B1] = { m, n, u1 }
DEF [B1] = { i, j, a }
IN [B1] = { m, n, u1 } — use [B1]
OUT [B1] = { } - Initialization

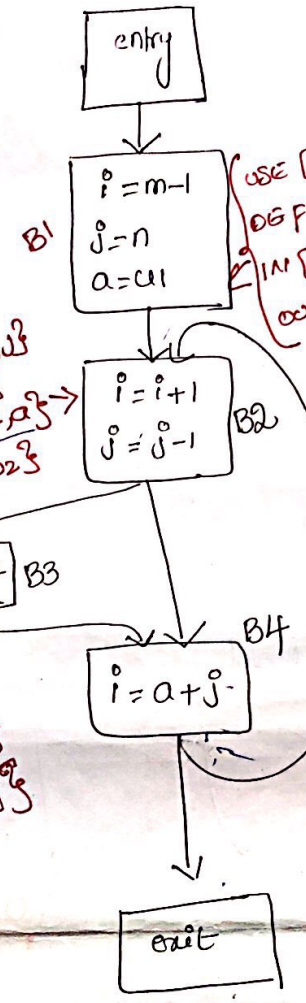
USE [B2] = { i, j }
DEF [B2] = { }
IN [B2] = { i, j }
OUT [B2] = { }

USE [B3] = { u2 }
DEF [B3] = { a }
IN [B3] = { u2 }
OUT [B3] = { }

USE [B4] = { a, j }
DEF [B4] = { i }
IN [B4] = { a, j }
OUT [B4] = { }

initialization
out [B5] = ∅.

def
usage first and after words def.
then don't insert
define variables



USE [B1] = {m, n, u1}
DEF [B1] = {i, j, a}
IN [B1] = {m, n, u1}
OUT [B1] = {i, j}

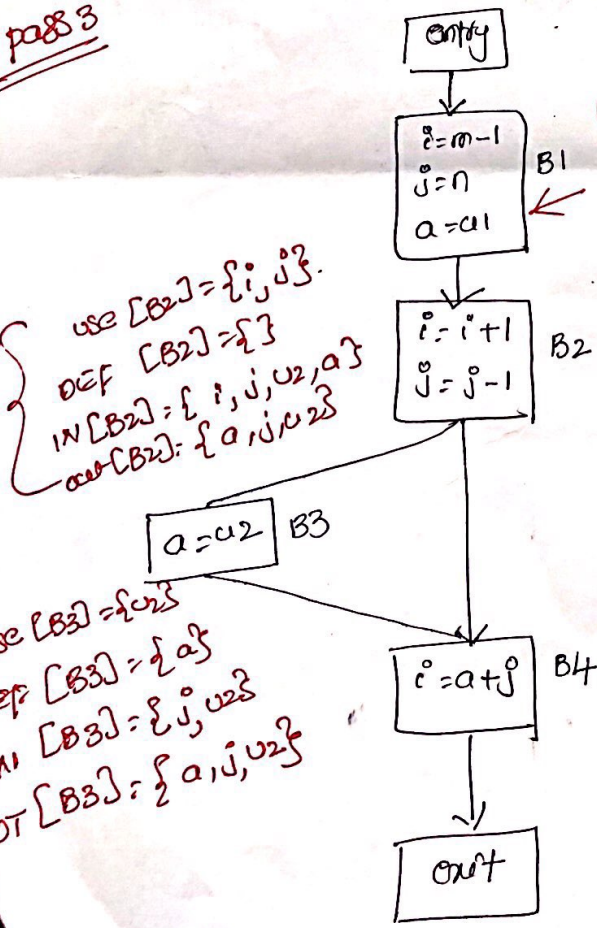
USE [B2] = {i, j}
DEF [B2] = {}
IN [B2] = {i, j, u2, a}
OUT [B2] = {a, j, u2}

USE [B3] = {u2}
DEF [B3] = {a}
IN [B3] = {j, u2}
OUT [B3] = {a, j}

USE [B4] = {a, j}
DEF [B4] = {i}
IN [B4] = {a, j, u2}
OUT [B4] = {i, j, u2, a}

for B4 next blocks are exit & B2

pass 3



USE [B1] = {m, n, u1}
DEF [B1] = {i, j, a}
IN [B1] = {m, n, u1, u2}
OUT [B1] = {i, j, u2, a}

USE [B2] = {i, j}
DEF [B2] = {}
IN [B2] = {i, j, u2, a}
OUT [B2] = {a, j, u2}

USE [B3] = {u2}
DEF [B3] = {a}
IN [B3] = {j, u2}
OUT [B3] = {a, j, u2}

USE [B4] = {a, j}
DEF [B4] = {i}
IN [B4] = {a, j, u2}
OUT [B4] = {a, i, j, u2}