

# BCP Algorithm (2/8)

- Let's illustrate this with an example:

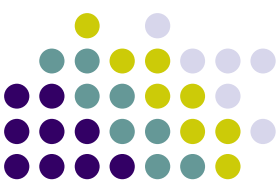
( 2 3 1 4 5)

( 1 2 -3)

( 1 -2)

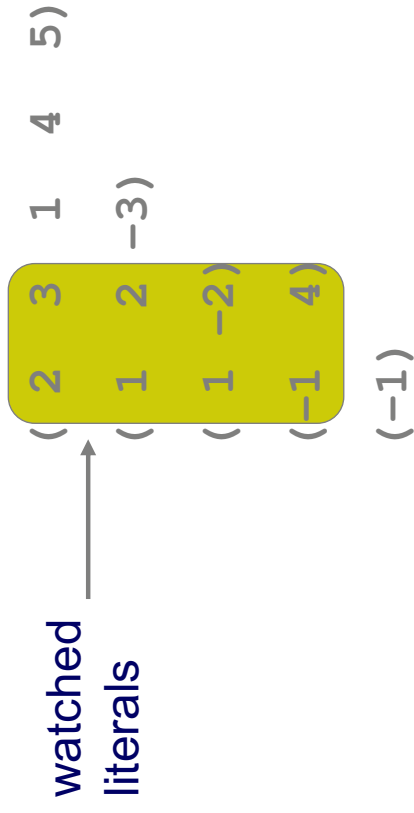
(-1 4)

(-1)



# BCP Algorithm (2.1/8)

- Let's illustrate this with an example:

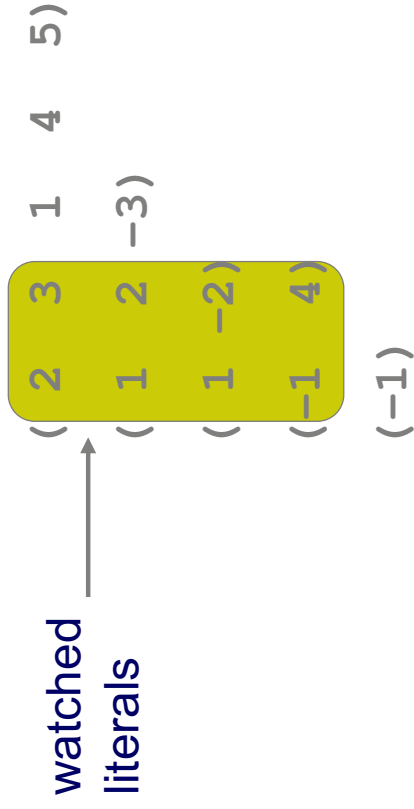


- Conceptually, we identify the first two literals in each clause as the watched ones

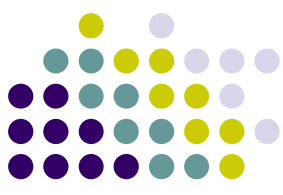


# BCP Algorithm (2.2/8)

- Let's illustrate this with an example:

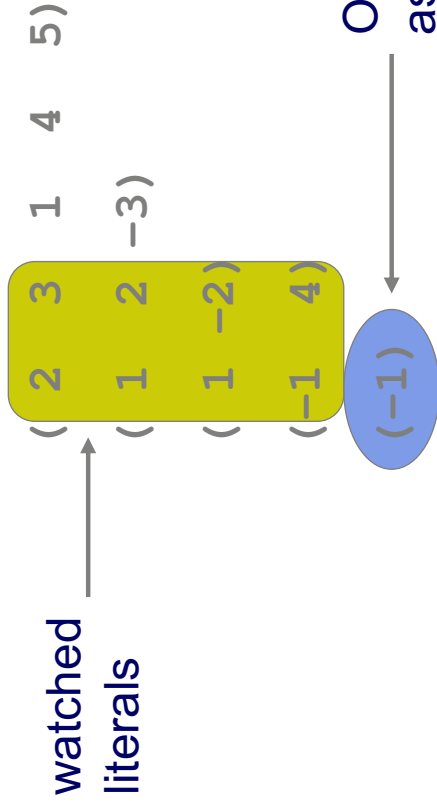


- Conceptually, we identify the first two literals in each clause as the watched ones
- Changing which literals are watched is represented by reordering the literals in the clause (which comes into play later)



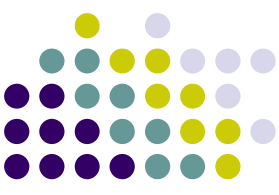
# BCP Algorithm (2.3/8)

- Let's illustrate this with an example:



- Conceptually, we identify the first two literals in each clause as the watched ones
- Changing which literals are watched is represented by reordering the literals in the clause (which comes into play later)
- Clauses of size one are a special case

Lintao Zhang



# BCP Algorithm (3/8)

- We begin by processing the assignment  $v_1 = F$  (which is implied by the size one clause)

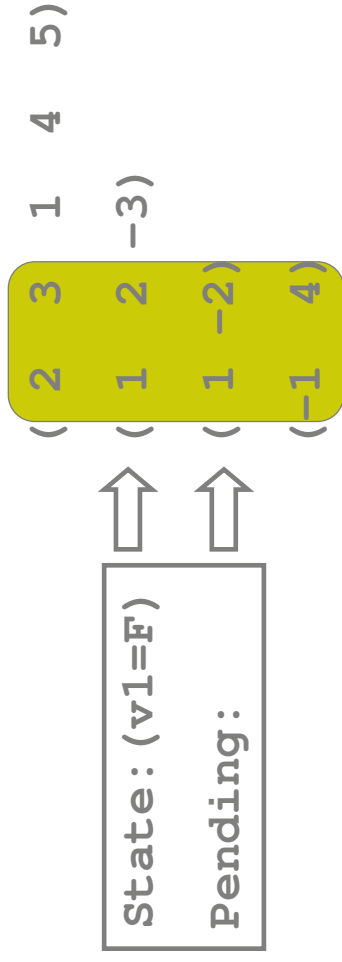
State:  $(v_1=F)$   
Pending:

$(2 \ 3 \ 1 \ 4 \ 5)$   
 $(1 \ 2 \ -3)$   
 $(1 \ -2)$   
 $(-1 \ 4)$



# BCP Algorithm (3.1/8)

- We begin by processing the assignment  $v1 = F$  (which is implied by the size one clause)

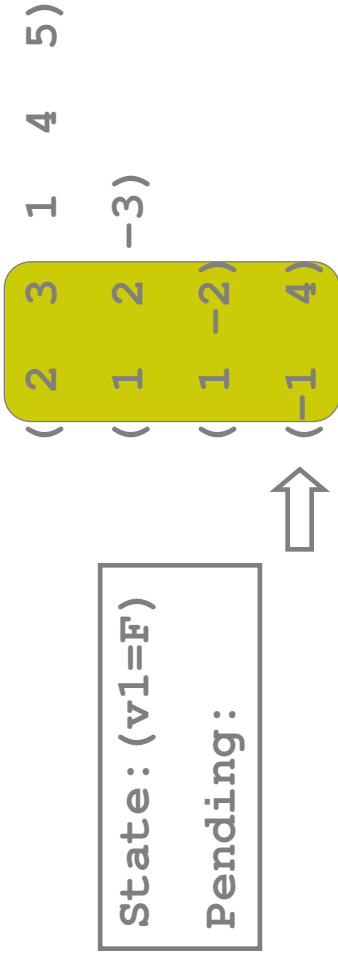


- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to F.

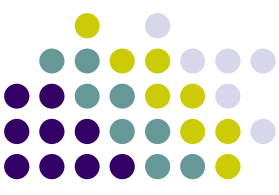


# BCP Algorithm (3.2/8)

- We begin by processing the assignment  $v_1 = F$  (which is implied by the size one clause)

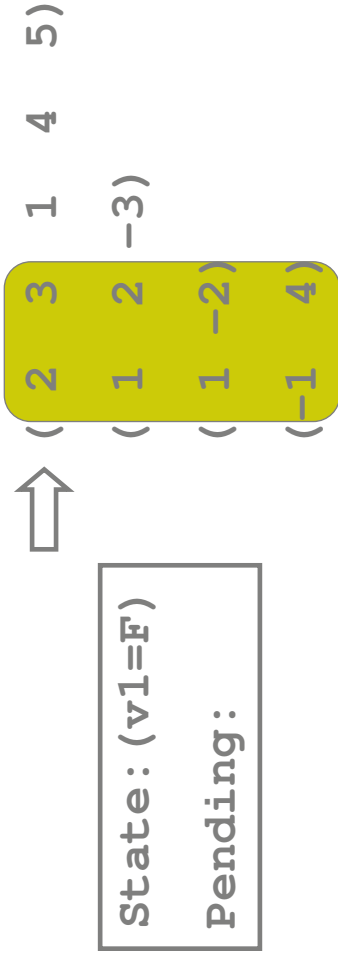


- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to F.
- We need not process clauses where a watched literal has been set to T, because the clause is now satisfied and so can not become implied.



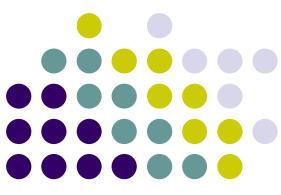
# BCP Algorithm (3.3/8)

- We begin by processing the assignment  $v1 = F$  (which is implied by the size one clause)



- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to F.
- We need not process clauses where a watched literal has been set to T, because the clause is now satisfied and so can not become implied.
- We *certainly* need not process any clauses where neither watched literal changes state (in this example, where  $v1$  is not watched).





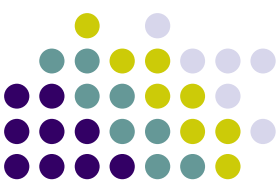
# BCP Algorithm (4/8)

- Now let's actually process the second and third clauses:

```
(( 2 3 1 4 5)
 ( 1 2 -3)
 ( 1 -2)
 (-1 4))
```

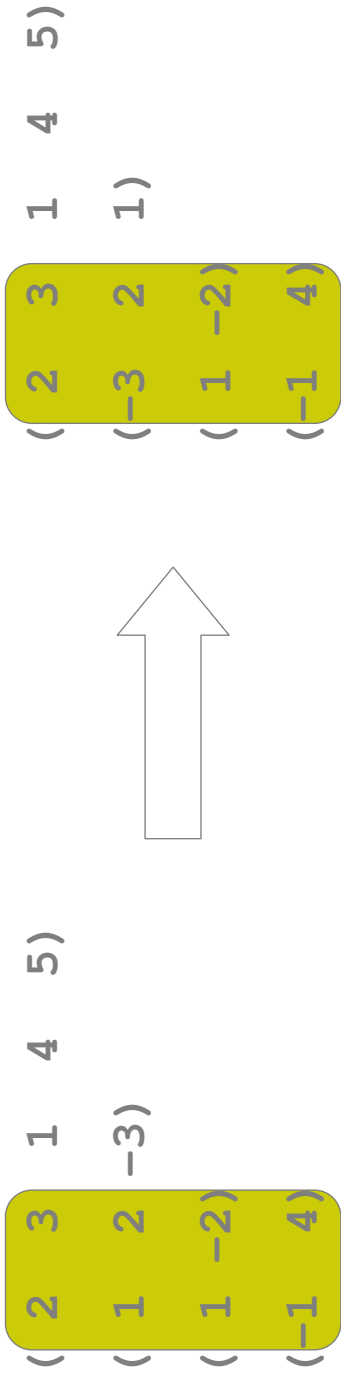
State: (v1=F)

Pending:



# BCP Algorithm (4.1/8)

- Now let's actually process the second and third clauses:



State: (v1=F)

Pending:

State: (v1=F)

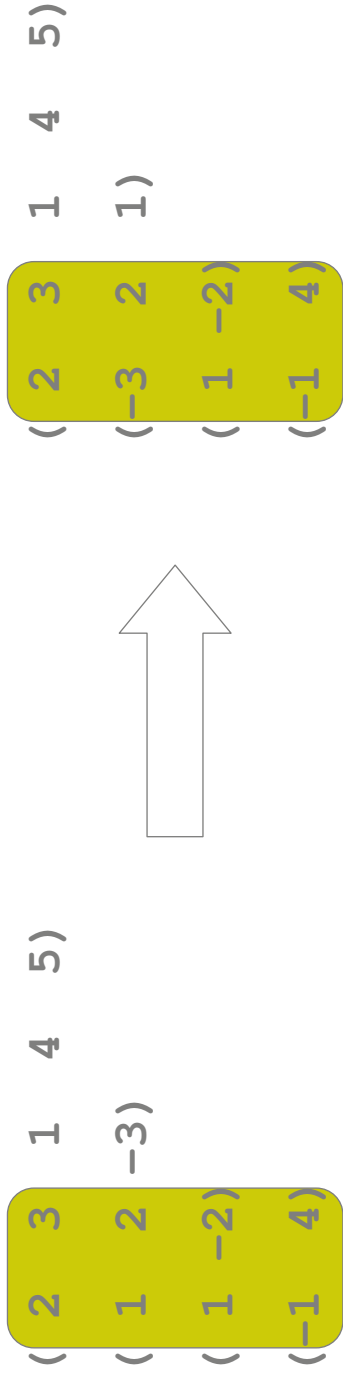
Pending:

- For the second clause, we replace v1 with  $\neg v3$  as a new watched literal. Since  $\neg v3$  is not assigned to F, this maintains our invariants.



# BCP Algorithm (4.2/8)

- Now let's actually process the second and third clauses:



State: (v1=F)

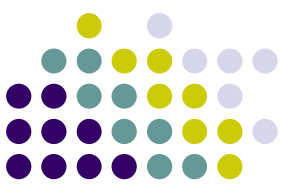
Pending:

State: (v1=F)

Pending: (v2=F)

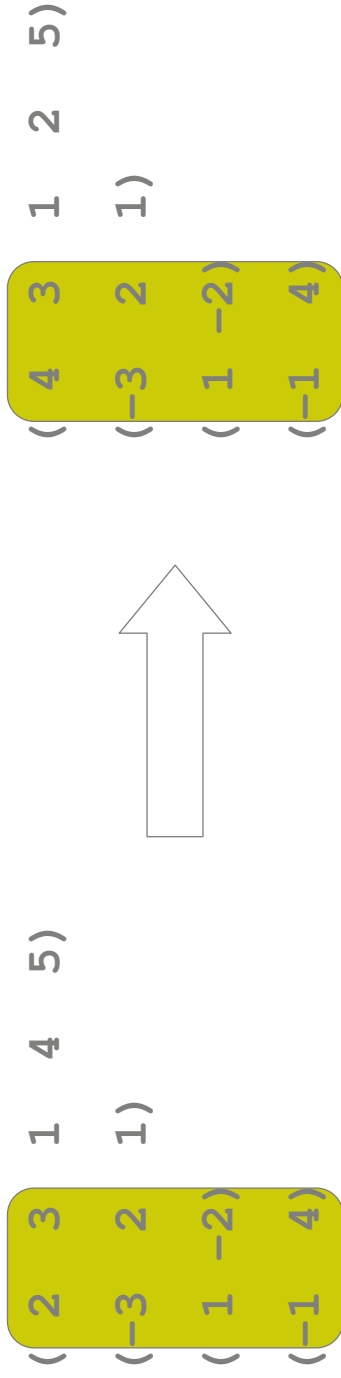
- For the second clause, we replace v1 with  $\neg v3$  as a new watched literal. Since  $\neg v3$  is not assigned to F, this maintains our invariants.
- The third clause is implied. We record the new implication of  $\neg v2$ , and add it to the queue of assignments to process. Since the clause cannot again

become newly implied, our invariants are maintained.



# BCP Algorithm (5/8)

- Next, we process  $\neg v2$ . We only examine the first 2 clauses.



State: (v1=F, v2=F)

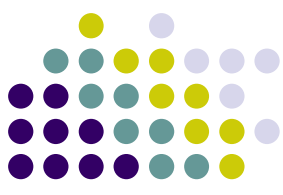
Pending:

State: (v1=F, v2=F)

Pending: (v3=F)

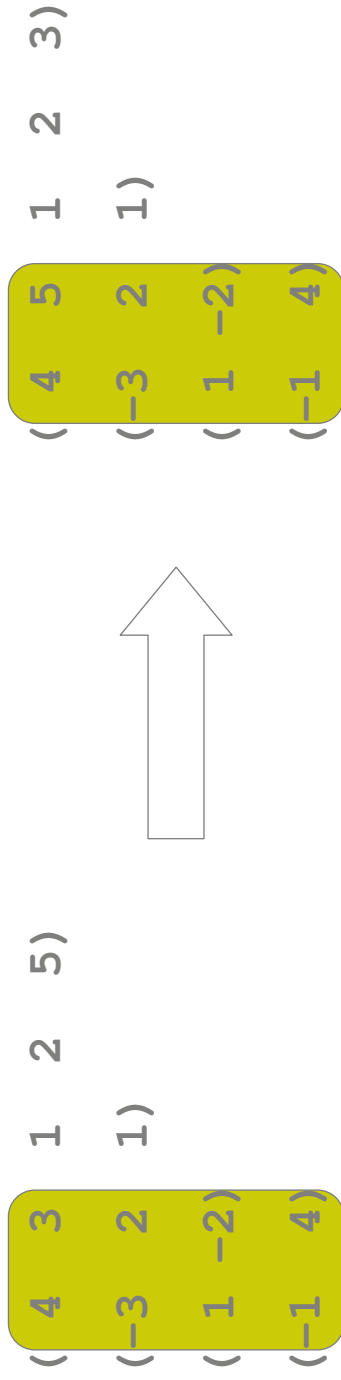
- For the first clause, we replace v2 with v4 as a new watched literal. Since v4 is not assigned to F, this maintains our invariants.
- The second clause is implied. We record the new implication of v3, and add it to the queue of assignments to process. Since the clause cannot again become newly implied, our invariants are maintained.

Lintao Zhang



# BCP Algorithm (6/8)

- Next, we process  $\neg v3$ . We only examine the first clause.



State:  $(v1=F, v2=F, v3=F)$

Pending:

State:  $(v1=F, v2=F, v3=F)$

Pending:

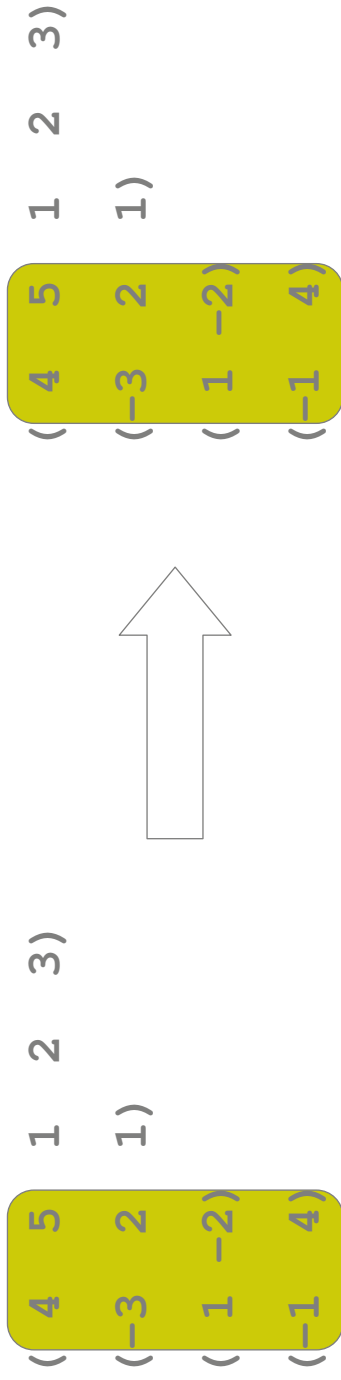
- For the first clause, we replace  $v3$  with  $v5$  as a new watched literal. Since  $v5$  is not assigned to  $F$ , this maintains our invariants.
- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. Both  $v4$  and  $v5$  are unassigned. Let's say we decide to assign  $v4=T$  and proceed.

Lintao Zhang



# BCP Algorithm (7/8)

- Next, we process v4. We do nothing at all.



State: (v1=F, v2=F, v3=F,  
v4=T)

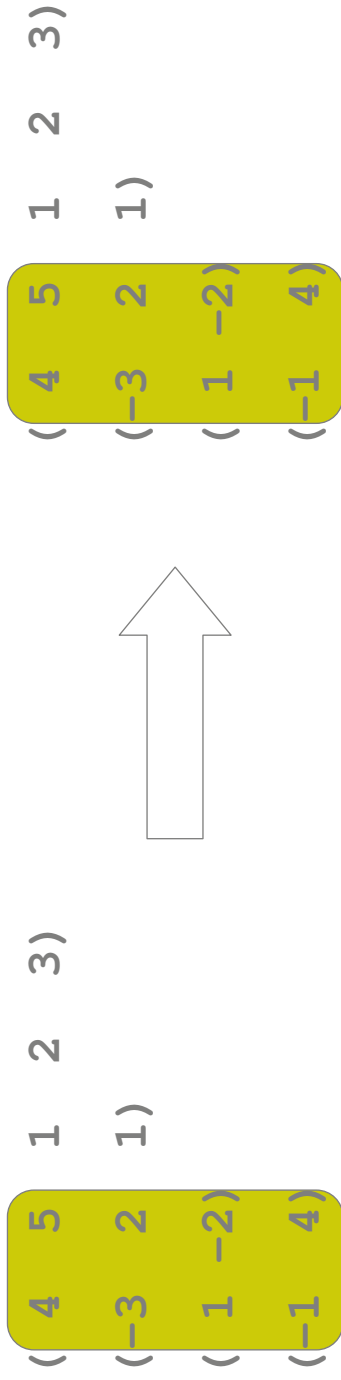
State: (v1=F, v2=F, v3=F,  
v4=T)

- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. Only v5 is unassigned. Let's say we decide to assign v5=F and proceed.



# BCP Algorithm (8/8)

- Next, we process  $v_5 = F$ . We examine the first clause.



State:  $(v_1 = F, v_2 = F, v_3 = F, v_4 = T, v_5 = F)$

State:  $(v_1 = F, v_2 = F, v_3 = F, v_4 = T, v_5 = F)$

- The first clause is implied. However, the implication is  $v_4 = T$ , which is a duplicate (since  $v_4 = T$  already) so we ignore it.
- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. No variables are unassigned, so the problem is sat, and we are done.

Lintao Zhang