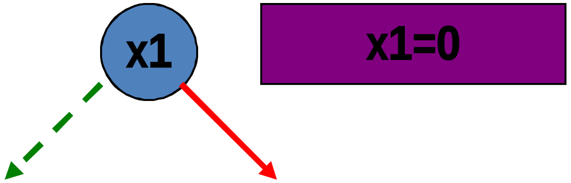
Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

Conflict Driven Learning and Non-chronological Backtracking

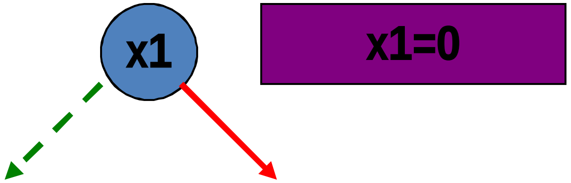
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



● x1=0

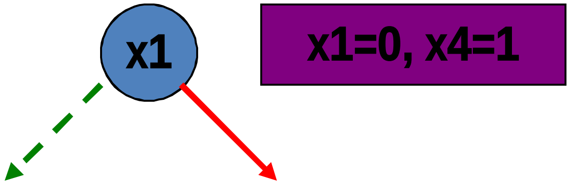
Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

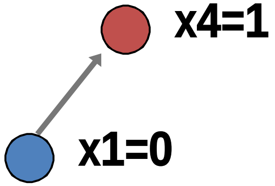


● x1=0

Conflict Driven Learning and Non-chronological Backtracking



x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

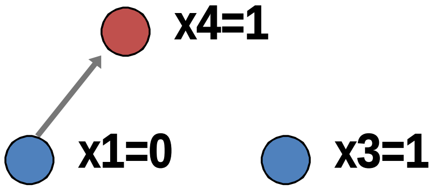
x2 + x11

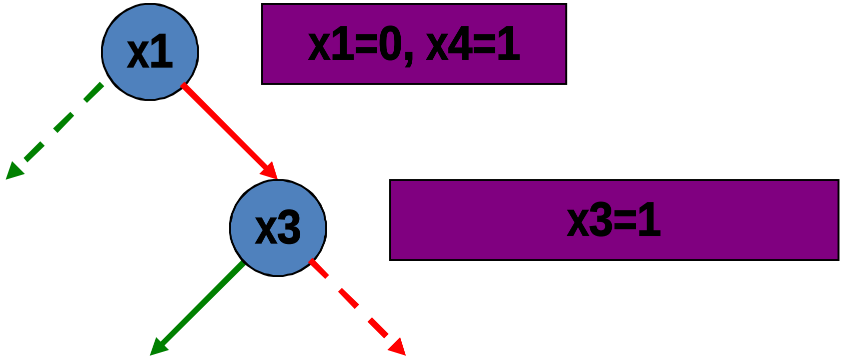
x7’ + x3’ + x9

x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’



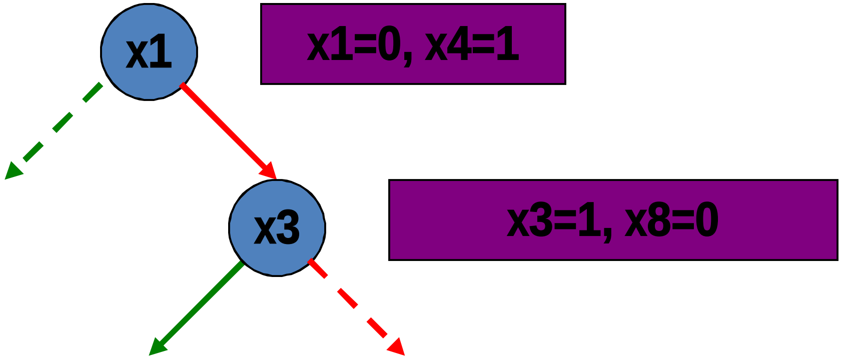


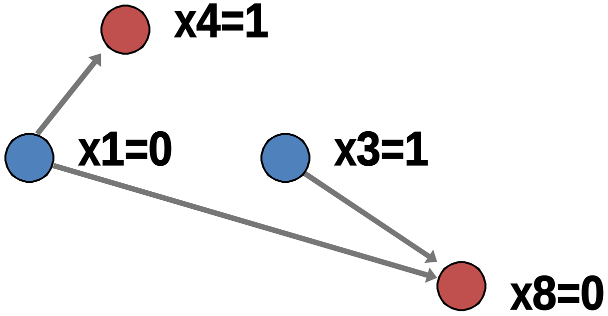
x1=0, x4=1

x3=1

Conflict Driven Learning and Non-chronological Backtracking

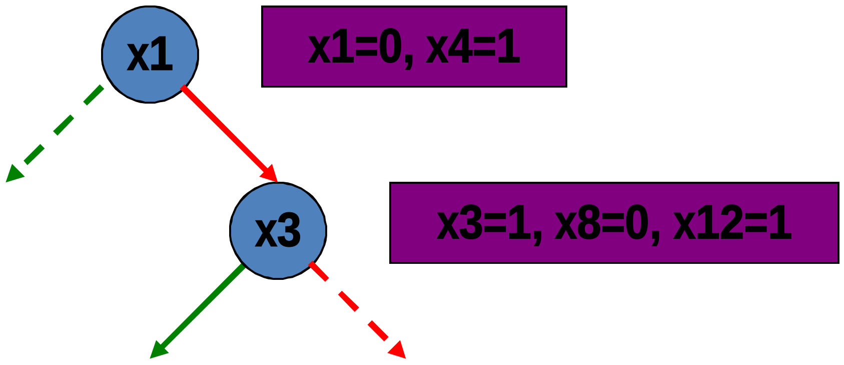
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

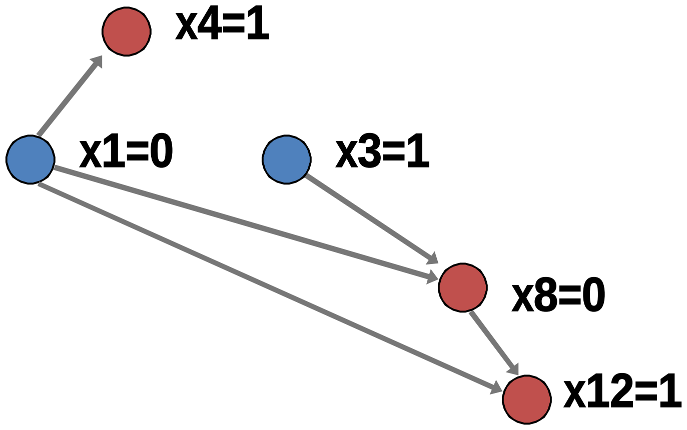

x1
x1=0, x4=1
x3
x3=1, x8=0


x4=1
x1=0
x3=1
x8=0

Conflict Driven Learning and Non-chronological Backtracking

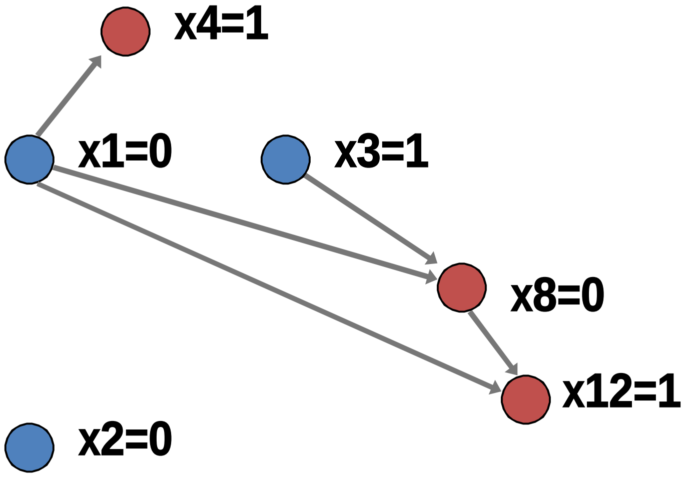
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

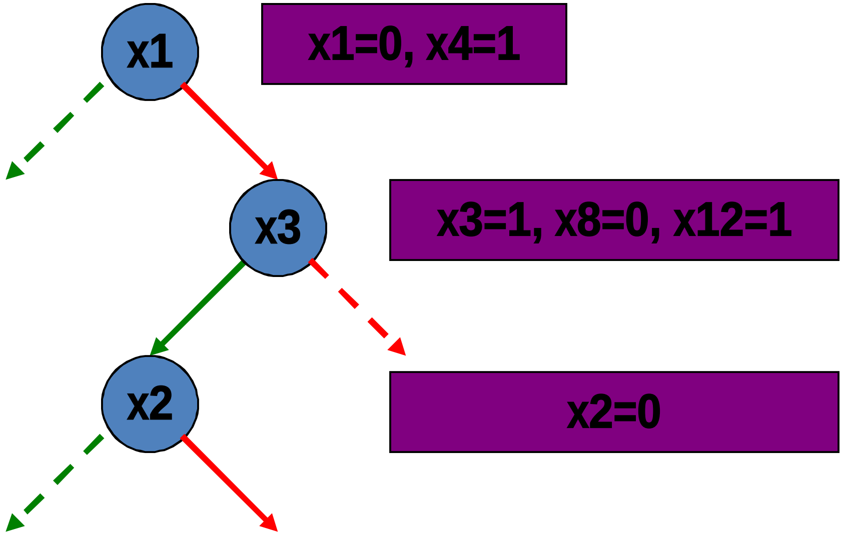




Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’





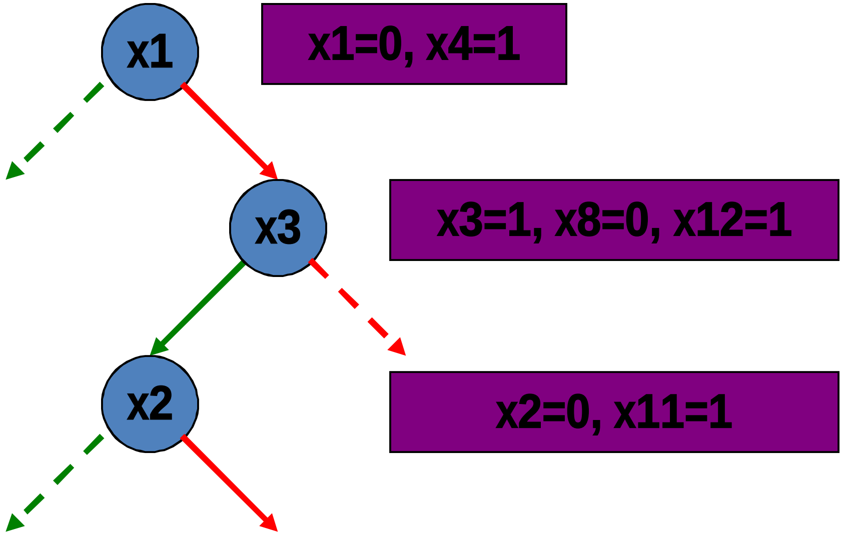
x1=0, x4=1

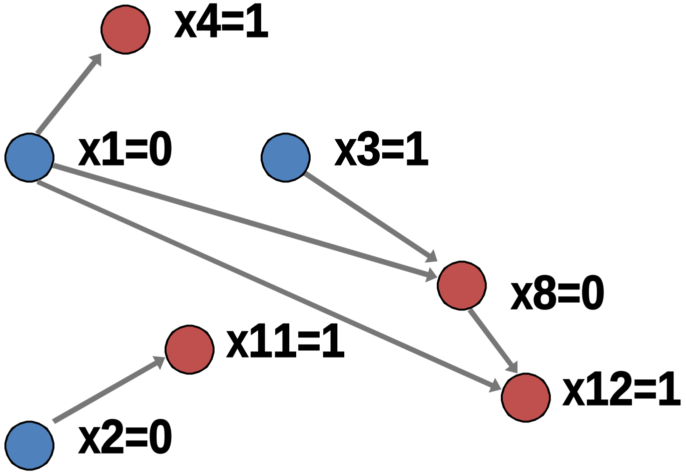
x3=1, x8=0, x12=1

x2=0

Conflict Driven Learning and Non-chronological Backtracking

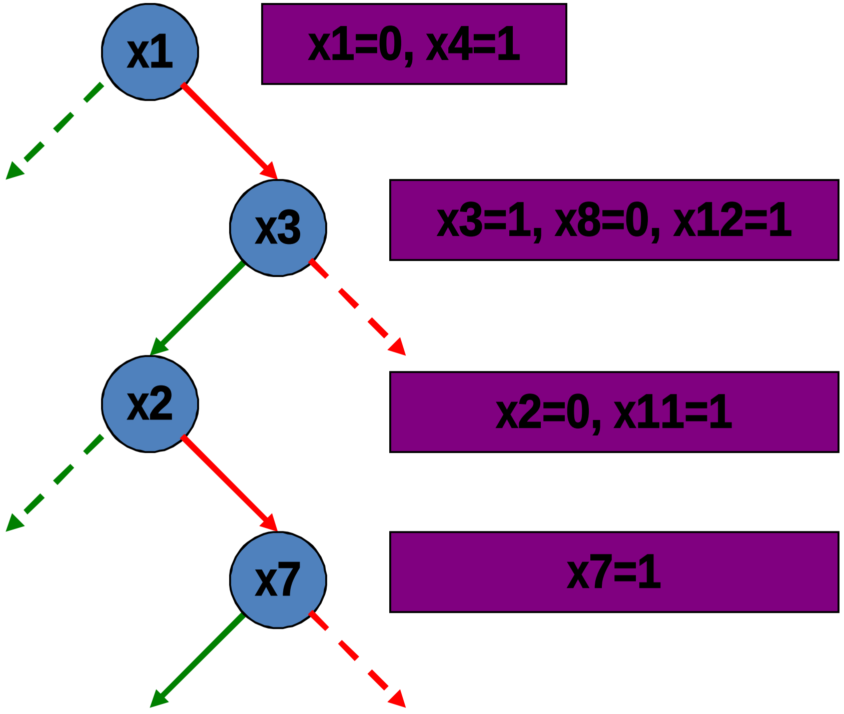
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



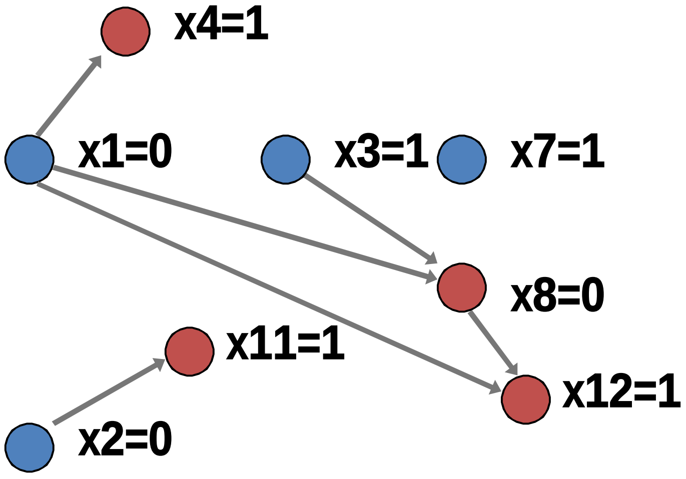


Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



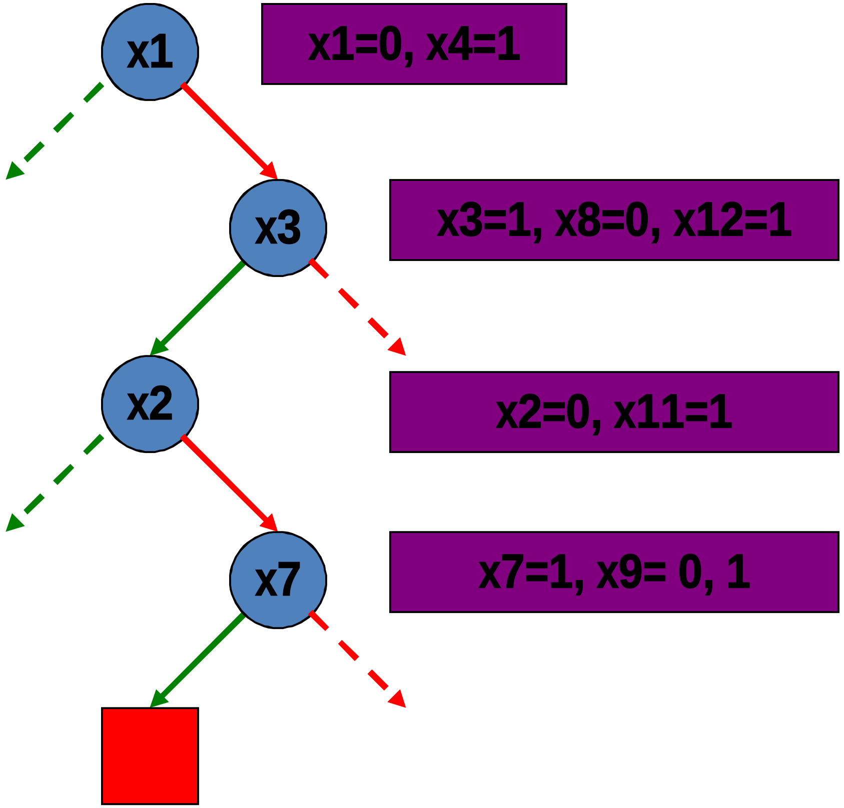
x1=0, x4=1
x3=1, x8=0, x12=1
x2=0, x11=1
x7=1

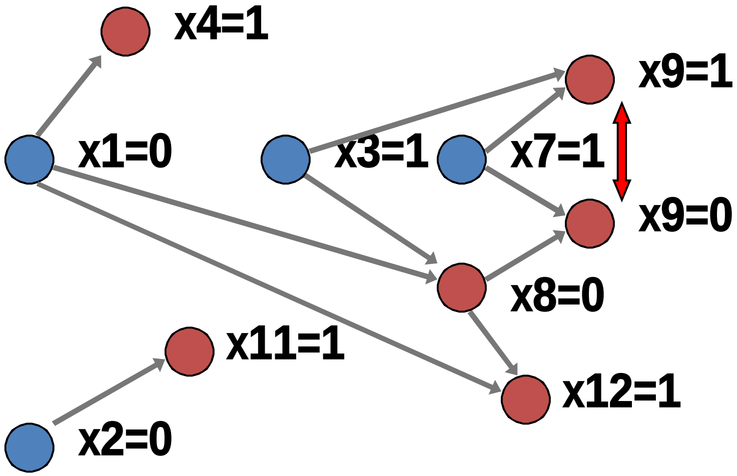


x4=1
x1=0
x3=1
x7=1
x8=0
x11=1
x12=1
x2=0

Conflict Driven Learning and Non-chronological Backtracking

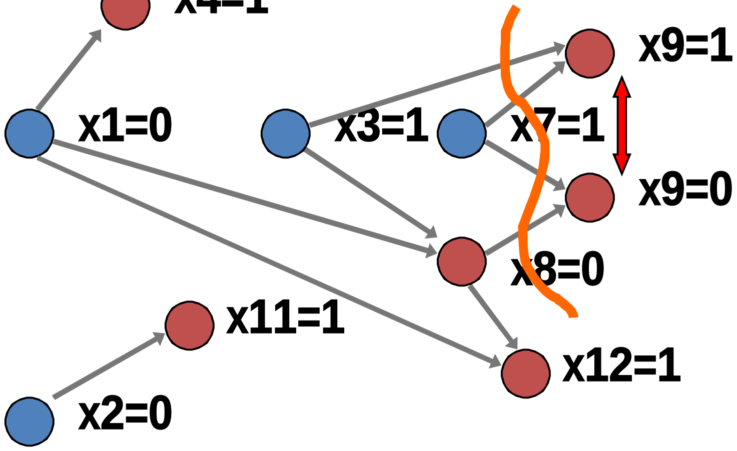
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

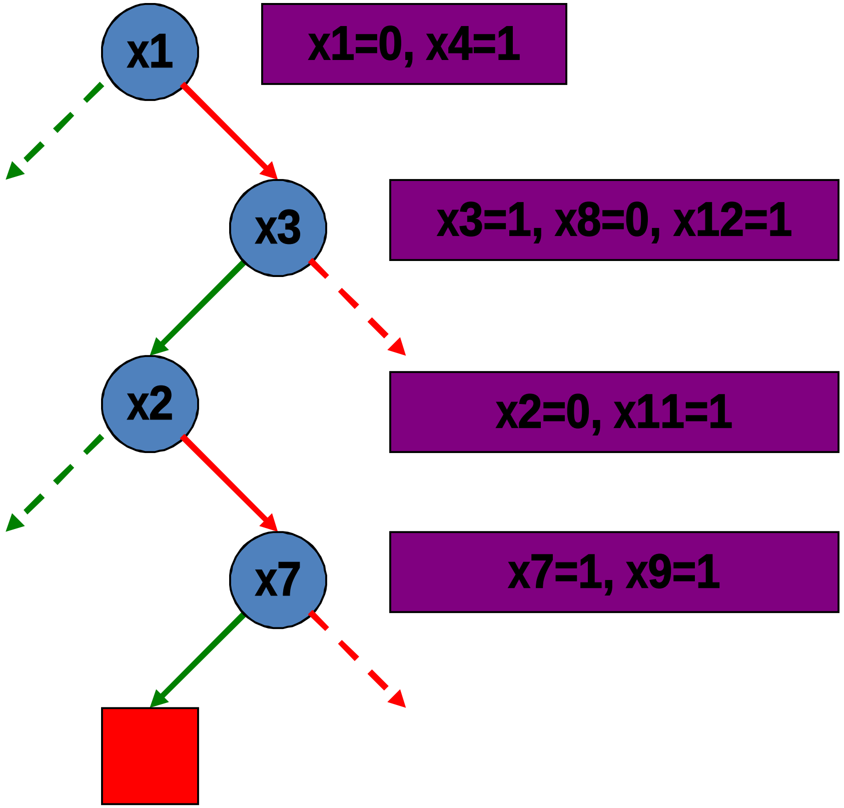




Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’





x1=0, x4=1

x3=1, x8=0, x12=1

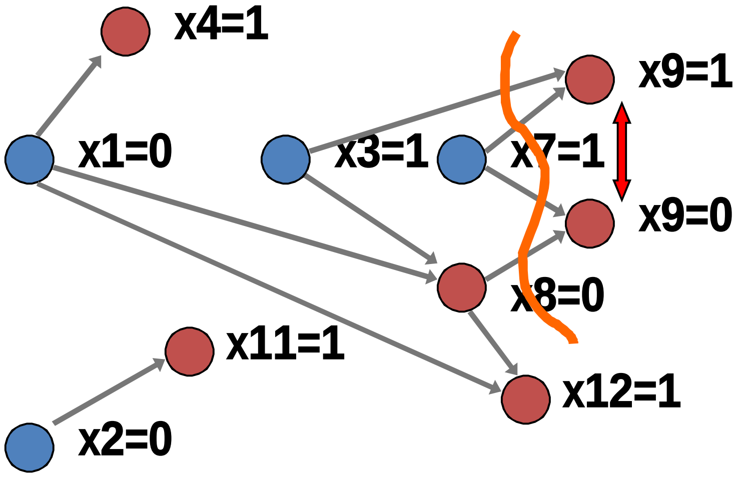
x2=0, x11=1

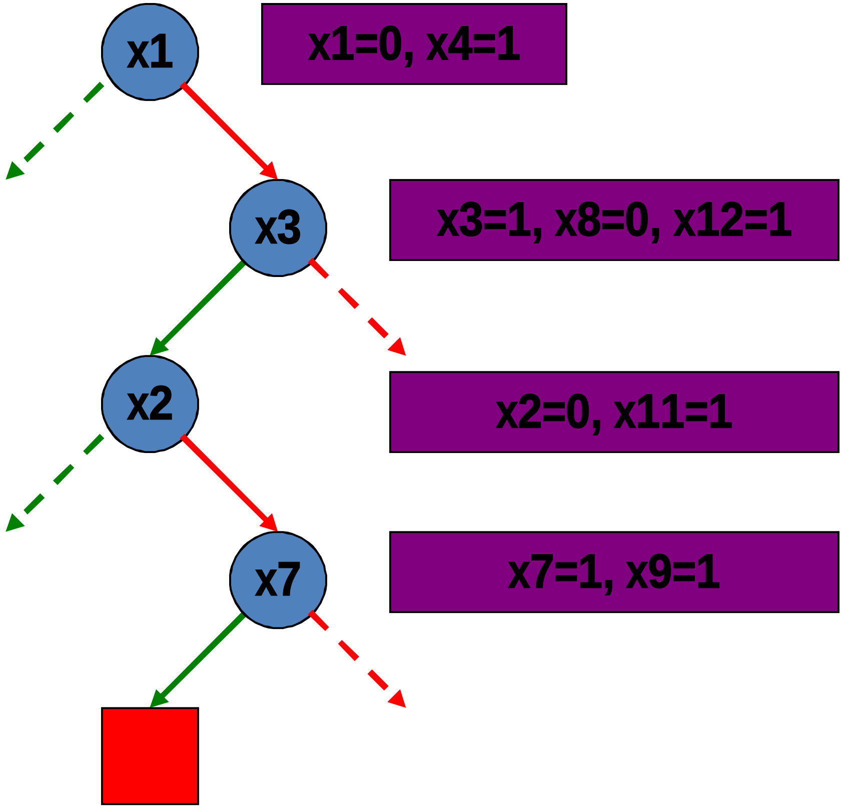
x7=1, x9=1

x3=1∧x7=1∧x8=0 → conflict

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’





x1=0, x4=1

x3=1, x8=0, x12=1

x2=0, x11=1

x7=1, x9=1

x3=1∧x7=1∧x8=0 → conflict

Add conflict clause: x3’+x7’+x8

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

x2 + x11

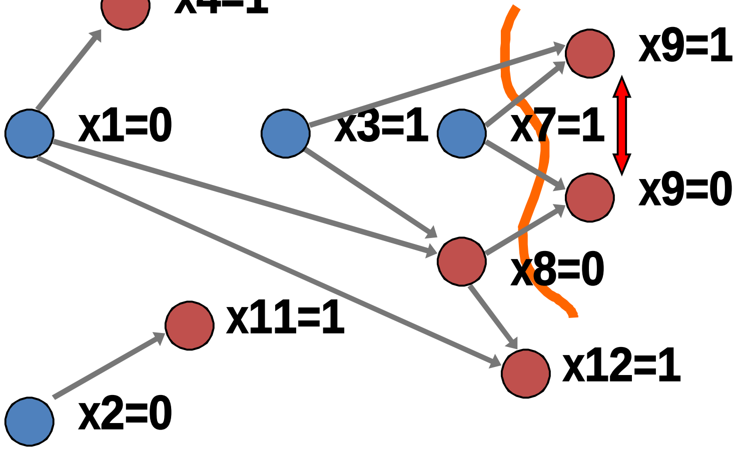
x7’ + x3’ + x9

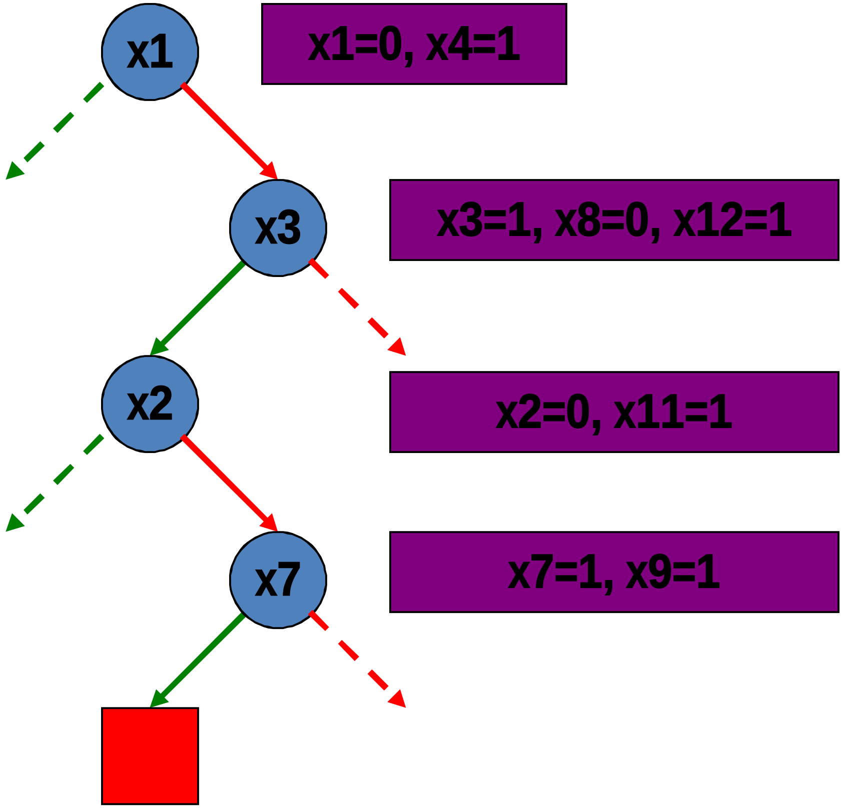
x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’

x3’+x7’+x8





x1=0, x4=1

x3=1, x8=0, x12=1

x2=0, x11=1

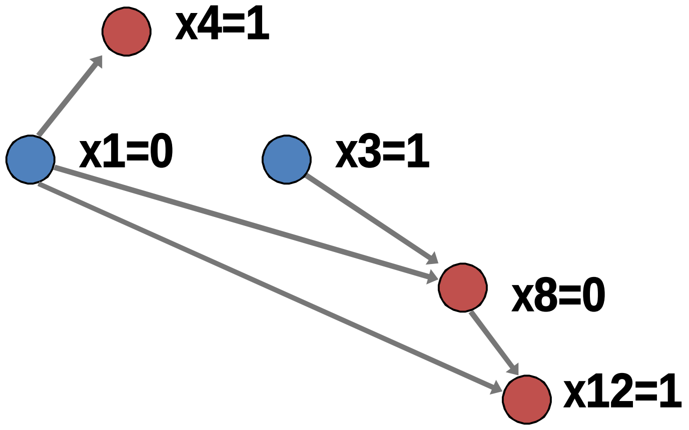
x7=1, x9=1

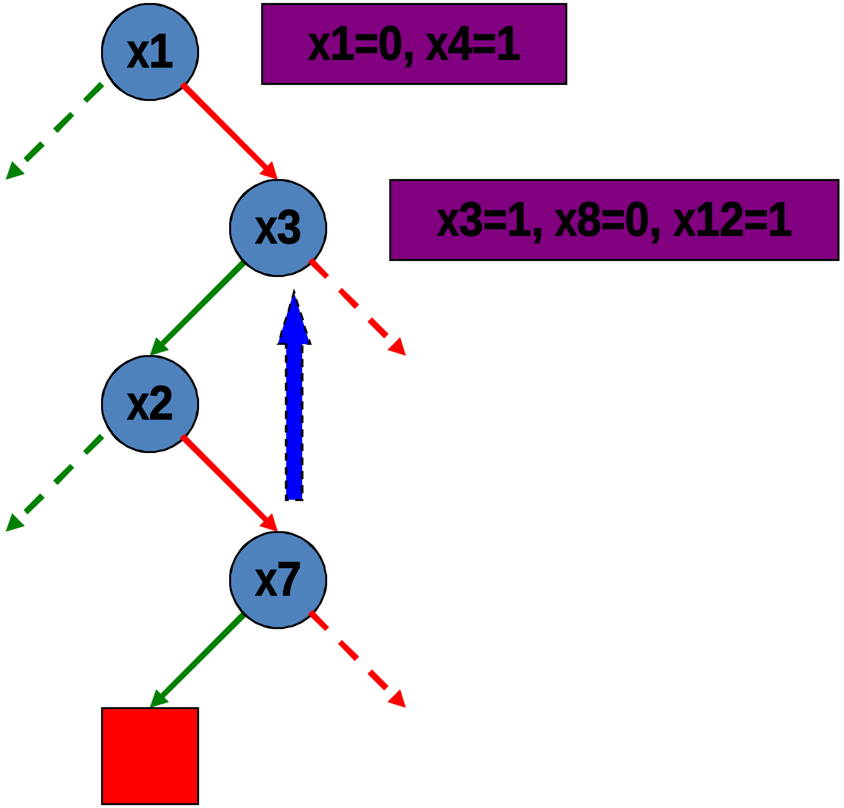
x3=1∧x7=1∧x8=0 → conflict

Add conflict clause: x3’+x7’+x8

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’
x3’ + x8 + x7’



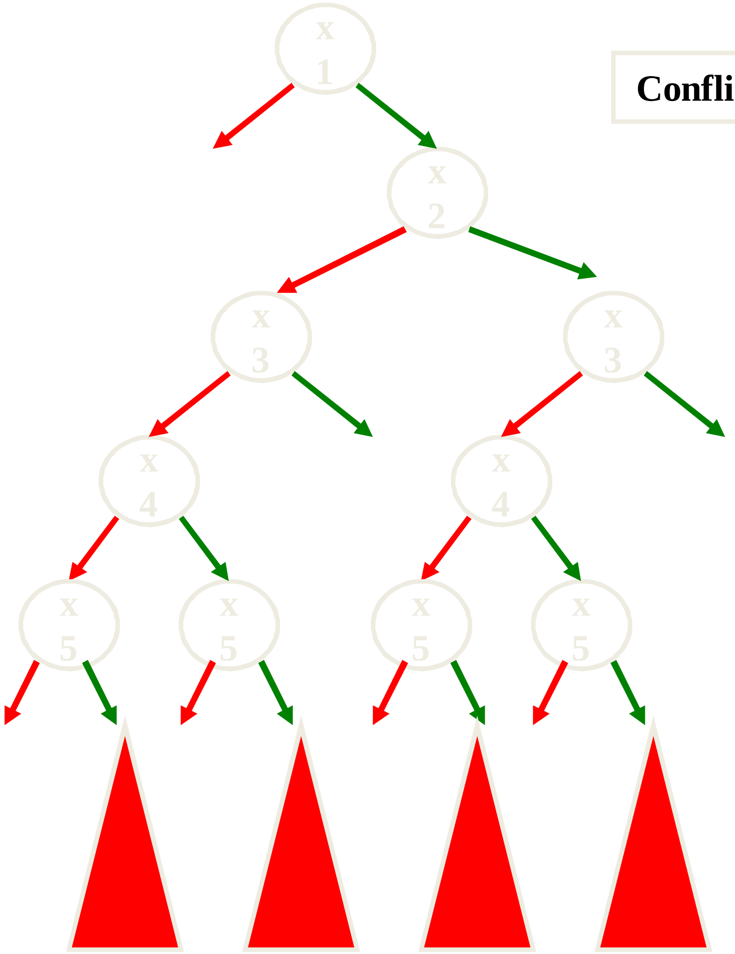


x1=0, x4=1

x3=1, x8=0, x12=1

Backtrack to the decision level of x3=1
With implication x7 = 0

What’s the big deal?



Conflict clause: x1’+x3+x5’

Significantly prune the search space – learned clause is useful forever!

Useful in generating future conflict clauses.

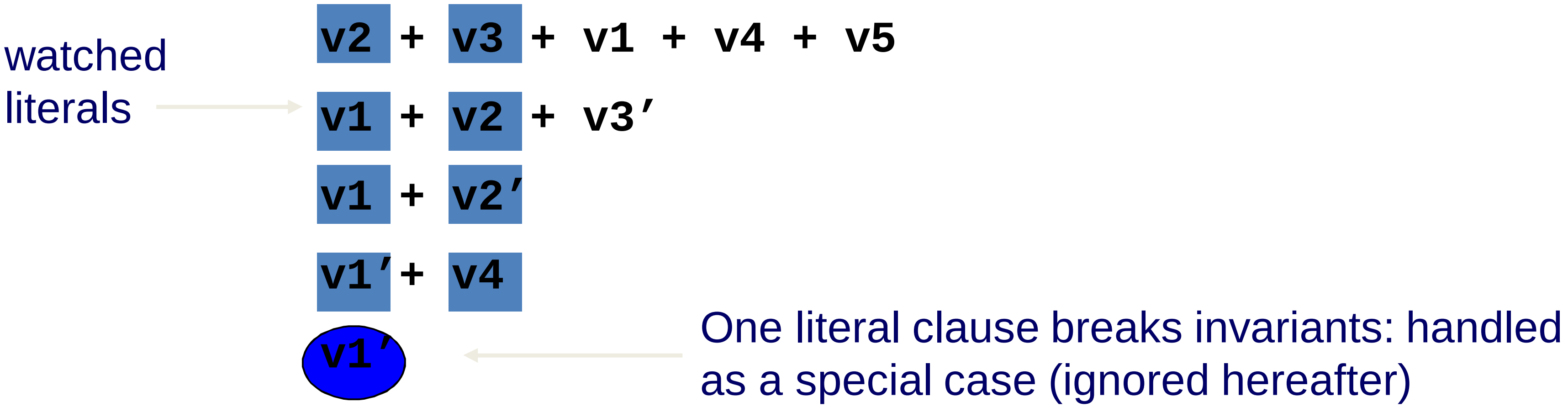
BCP Algorithm (2/8)

- Let’s illustrate this with an example:

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3’
v1 + v2’
v1’+ v4
v1’
```

BCP Algorithm (2.1/8)

- Let's illustrate this with an example:



watched literals

v2 + v3 + v1 + v4 + v5

v1 + v2 + v3'

v1 + v2'

v1'+ v4

v1'

One literal clause breaks invariants: handled as a special case (ignored hereafter)

- Initially, we identify any two literals in each clause as the watched ones
- Clauses of size one are a special case

BCP Algorithm (3/8)

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

State:(v1=F)

Pending:

v2 + v3 + v1 + v4 + v5

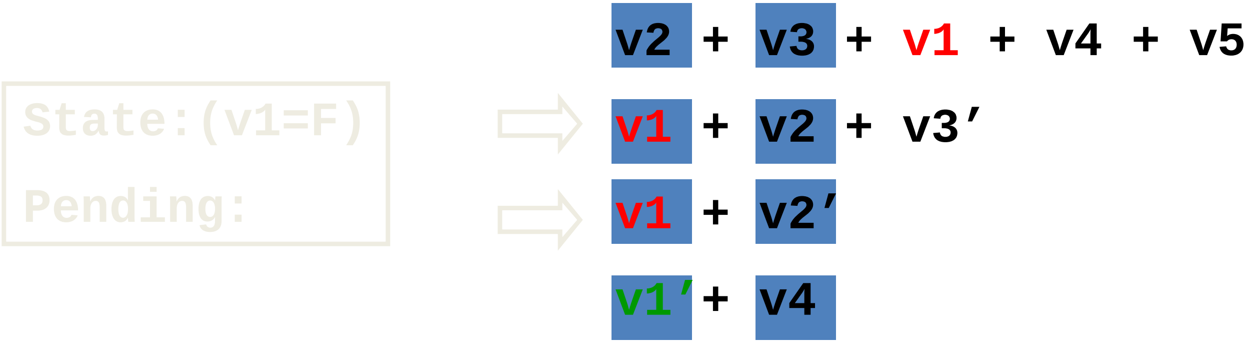
v1 + v2 + v3’

v1 + v2’

v1’+ v4

BCP Algorithm (3.1/8)

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



State:(v1=F)

Pending:

v2 + v3 + v1 + v4 + v5

v1 + v2 + v3’

v1 + v2’

v1’+ v4

- 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 v1 = F (which is implied by the size one clause)

State:(v1=F)

Pending:

v2 + v3 + v1 + v4 + v5

v1 + v2 + v3’

v1 + v2’

v1’+ v4

- 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 unit.

BCP Algorithm (3.3/8)

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

```
State:(v1=F)        v2 + v3 + v1 + v4 + v5

Pending:            v1 + v2 + v3'

                    v1 + v2'

                    v1'+ v4
```

- 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 unit.
- 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)

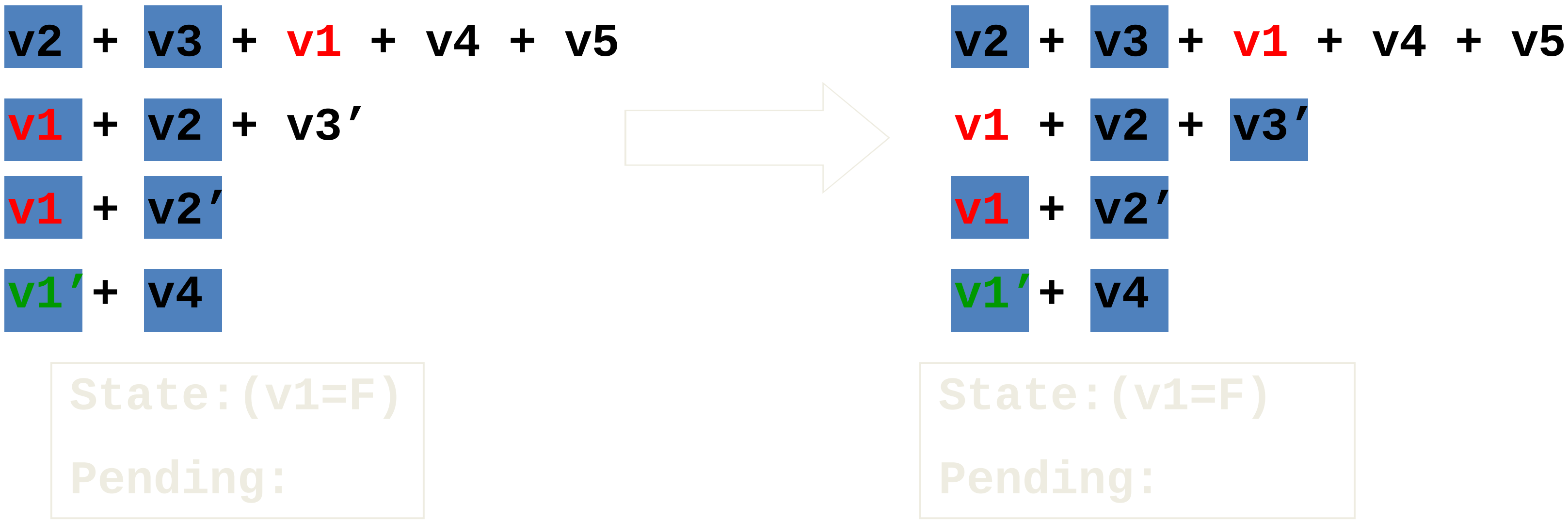
- Now let’s actually process the second and third clauses:

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3’
v1 + v2’
v1’+ v4
```

```
State:(v1=F)
Pending:
```

BCP Algorithm (4.1/8)

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



```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3’
v1 + v2’
v1’+ v4

State:(v1=F)
Pending:
```

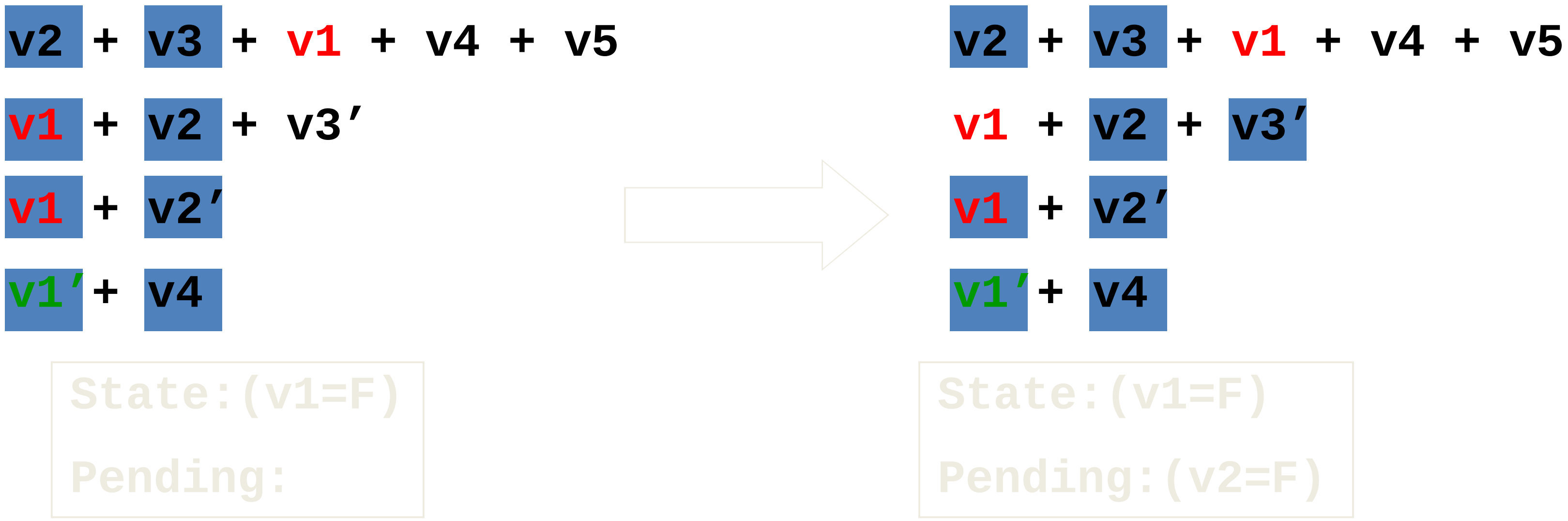
```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3’
v1 + v2’
v1’+ v4

State:(v1=F)
Pending:
```

- For the second clause, we replace v1 with v3’ as a new watched literal. Since 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:



v2 + v3 + v1 + v4 + v5

v1 + v2 + v3’

v1 + v2’

v1’ + v4

State:(v1=F)

Pending:

v2 + v3 + v1 + v4 + v5

v1 + v2 + v3’

v1 + v2’

v1’ + v4

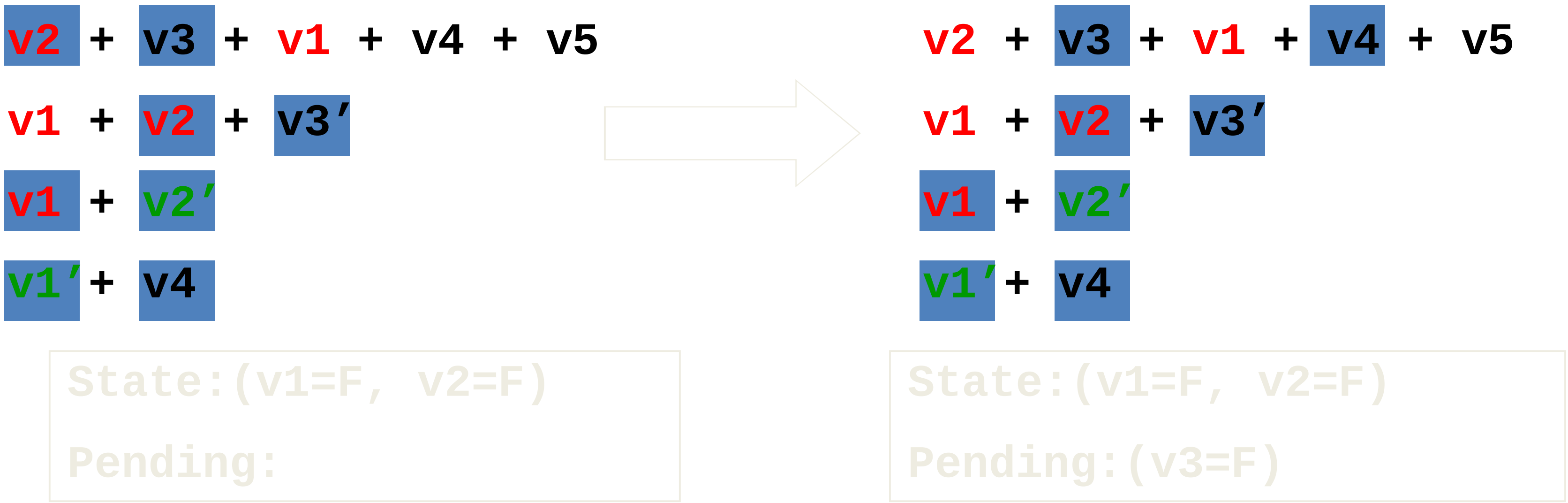
State:(v1=F)

Pending:(v2=F)

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

BCP Algorithm (5/8)

- Next, we process v2’. We only examine the first 2 clauses.



```
v2 + v3 + v1 + v4 + v5          v2 + v3 + v1 + v4 + v5
v1 + v2 + v3’                   v1 + v2 + v3’
v1 + v2’                        v1 + v2’
v1’+ v4                         v1’+ v4
```

State:(v1=F, v2=F)

Pending:

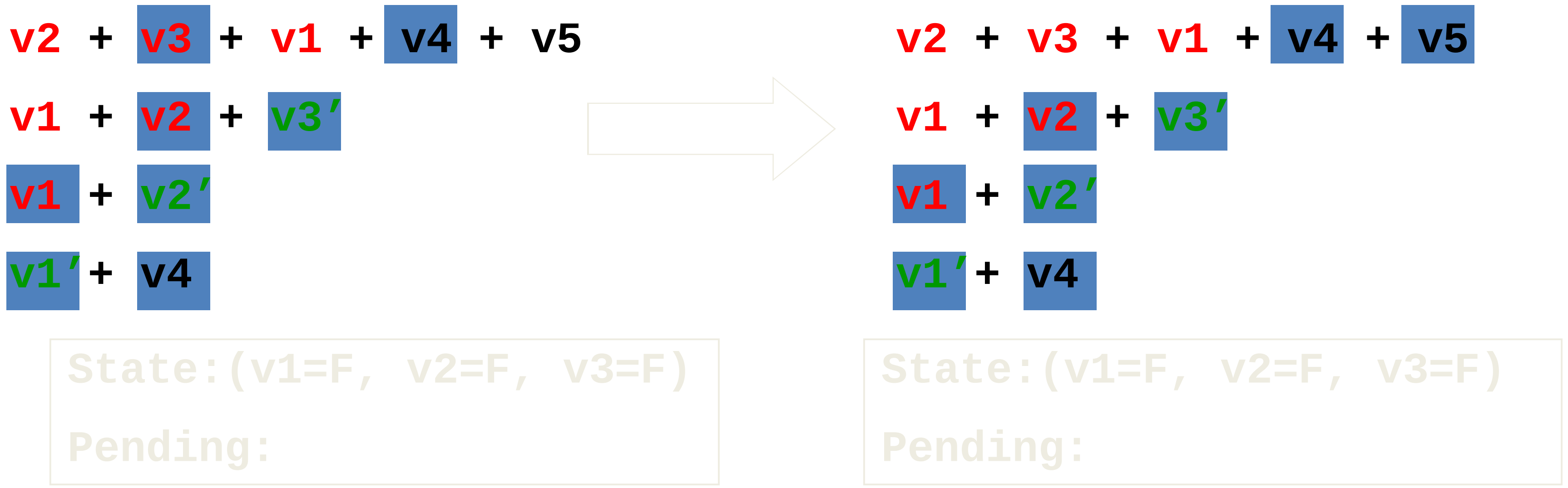
State:(v1=F, v2=F)

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 unit. We record the new implication of v3’, and add it to the queue of assignments to process. Since the clause cannot again become unit, our invariants are maintained.

BCP Algorithm (6/8)

- Next, we process v3’. We only examine the first clause.



v2 + v3 + v1 + v4 + v5

v1 + v2 + v3’

v1 + v2’

v1’ + v4

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

Pending:

v2 + v3 + v1 + v4 + v5

v1 + v2 + v3’

v1 + v2’

v1’ + v4

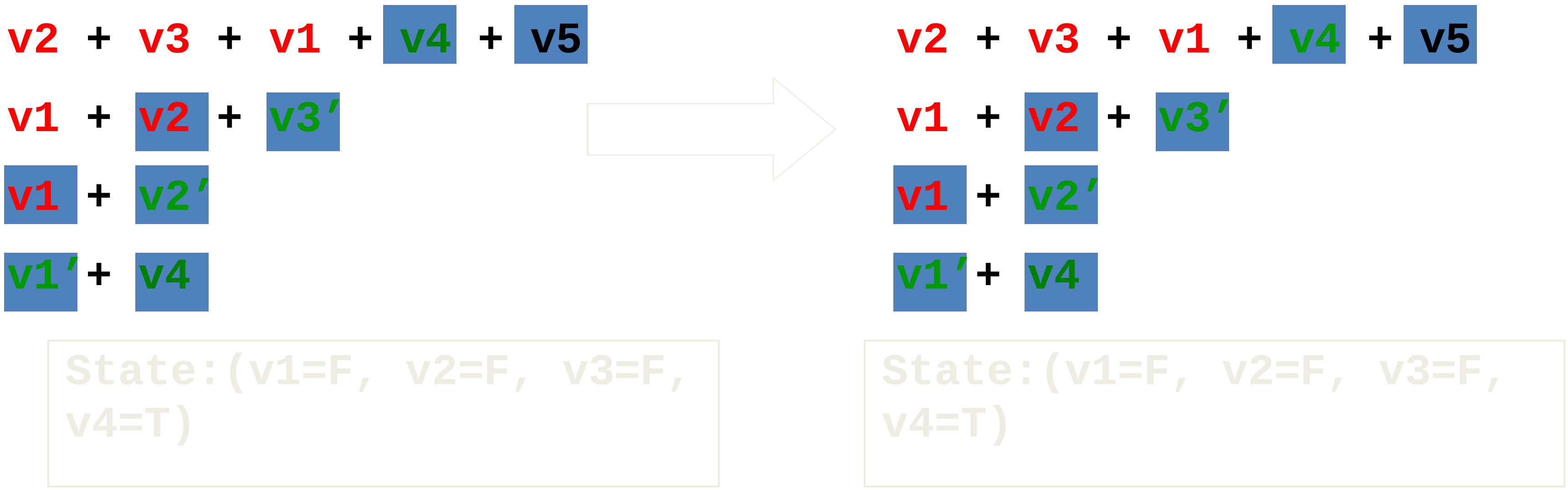
State:(v1=F, v2=F, v3=F)

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.

BCP Algorithm (7/8)

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



v2 + v3 + v1 + v4 + v5

v1 + v2 + v3′

v1 + v2′

v1′ + v4

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

v2 + v3 + v1 + v4 + v5

v1 + v2 + v3′

v1 + v2′

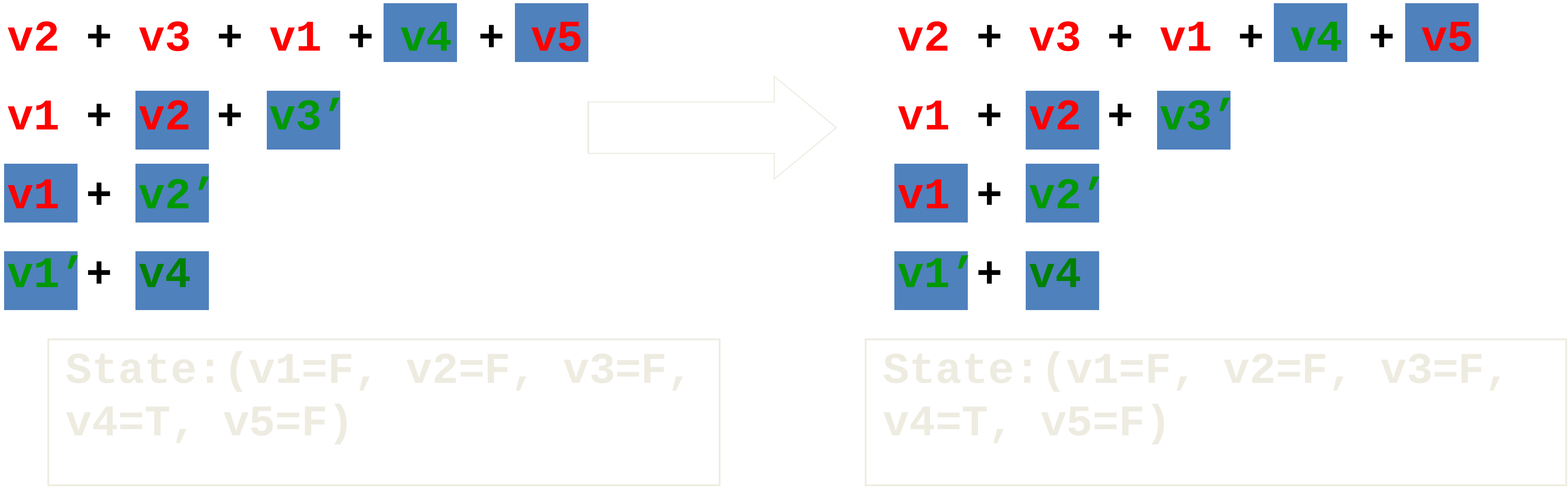
v1′ + v4

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 v5=F. We examine the first clause.



v2 + v3 + v1 + v4 + v5

v1 + v2 + v3′

v1 + v2′

v1′ + v4

State:(v1=F, v2=F, v3=F, v4=T, v5=F)

v2 + v3 + v1 + v4 + v5

v1 + v2 + v3′

v1 + v2′

v1′ + v4

State:(v1=F, v2=F, v3=F, v4=T, v5=F)

- The first clause is already satisfied by v4 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 instance is SAT, and we are done.