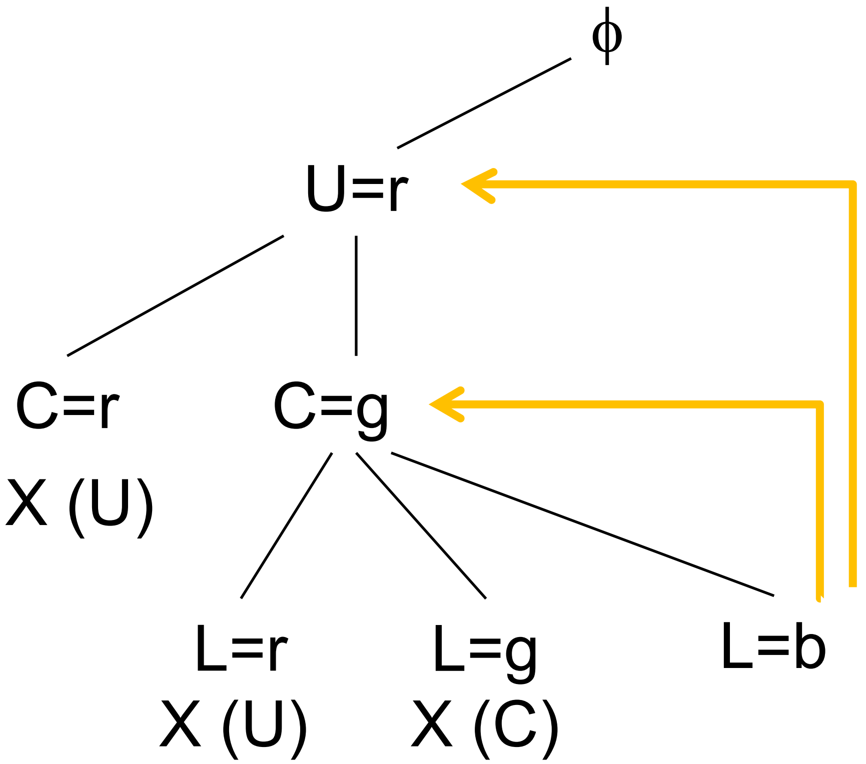
Looking Forward

variable order: U-C-L-P-M; value order: r-g-b



ϕ

U=r

C=r

X (U)

C=g

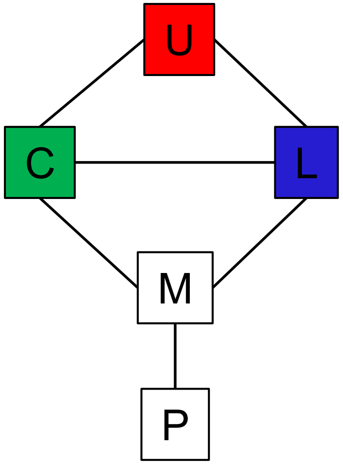
L=r

X (U)

L=g

X (C)

L=b



U C L P M

r g b

Backtracking

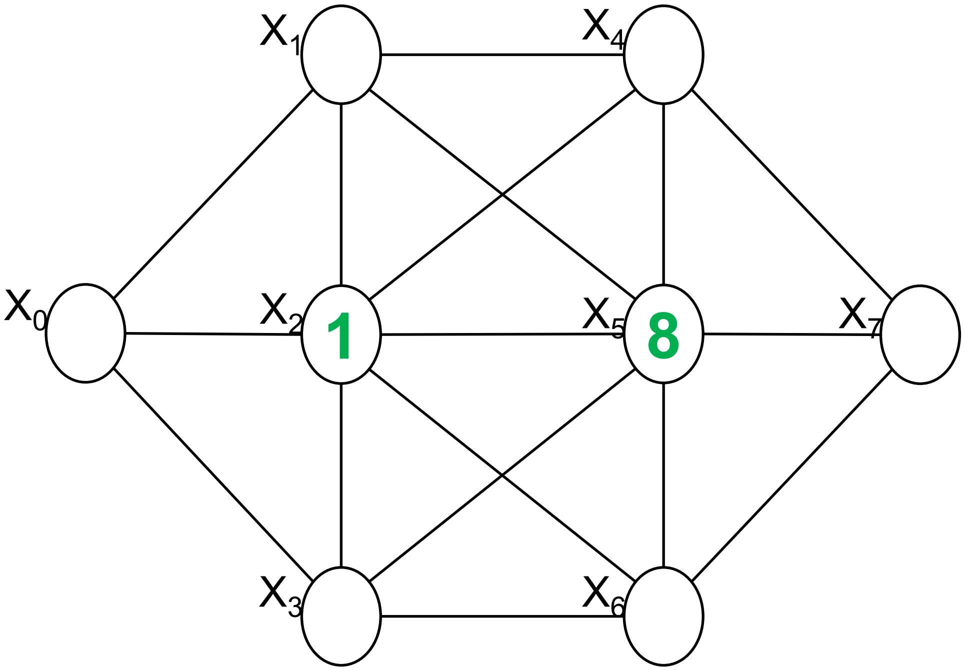
As we move down the tree
check back against higher decisions
to make sure it is worth continuing

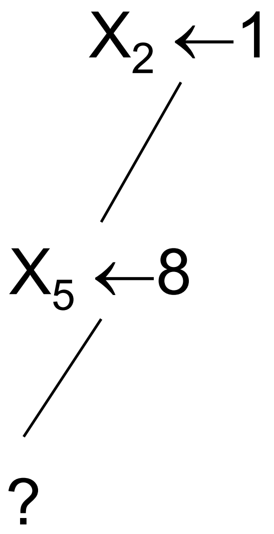
```
backtrack(V, D, C)
input: V (array of n int) D (2d array of n*d int) C (constraints)
output: (bool, V), where bool = true if a solution is found
var := 0  //start with the first variable
val := 0  //start with the first value
done := false  //not finished yet
while not done
  if var == n  return (true, values)    //if reached end, stop
  else if var == -1 return (false, null)//if back to top, stop
  else if val == -1                     //if no values left
     var := var-1                       //step back to previous variable
     val := nextval(var,values[var])  //continue to next value
  else if check(var,val,values,C) == false
     val := nextval(var,val)  //get next value, or -1 if none
  else
```

```
check(var,val,values,C)
output: true if no conflict for var<-val when checking back

// For each constraint in C whose scope is {var,X}, where X is
// before var, check var<-val against X's assignment in values

//i.e. while searching the tree, each time we try a value for
//a variable, check back against previous assignments to ensure
//no constraint is violated
```





That was looking forward in the tree, into domains of variables we haven't assigned.

Is there an algorithm?

At this point, we said:

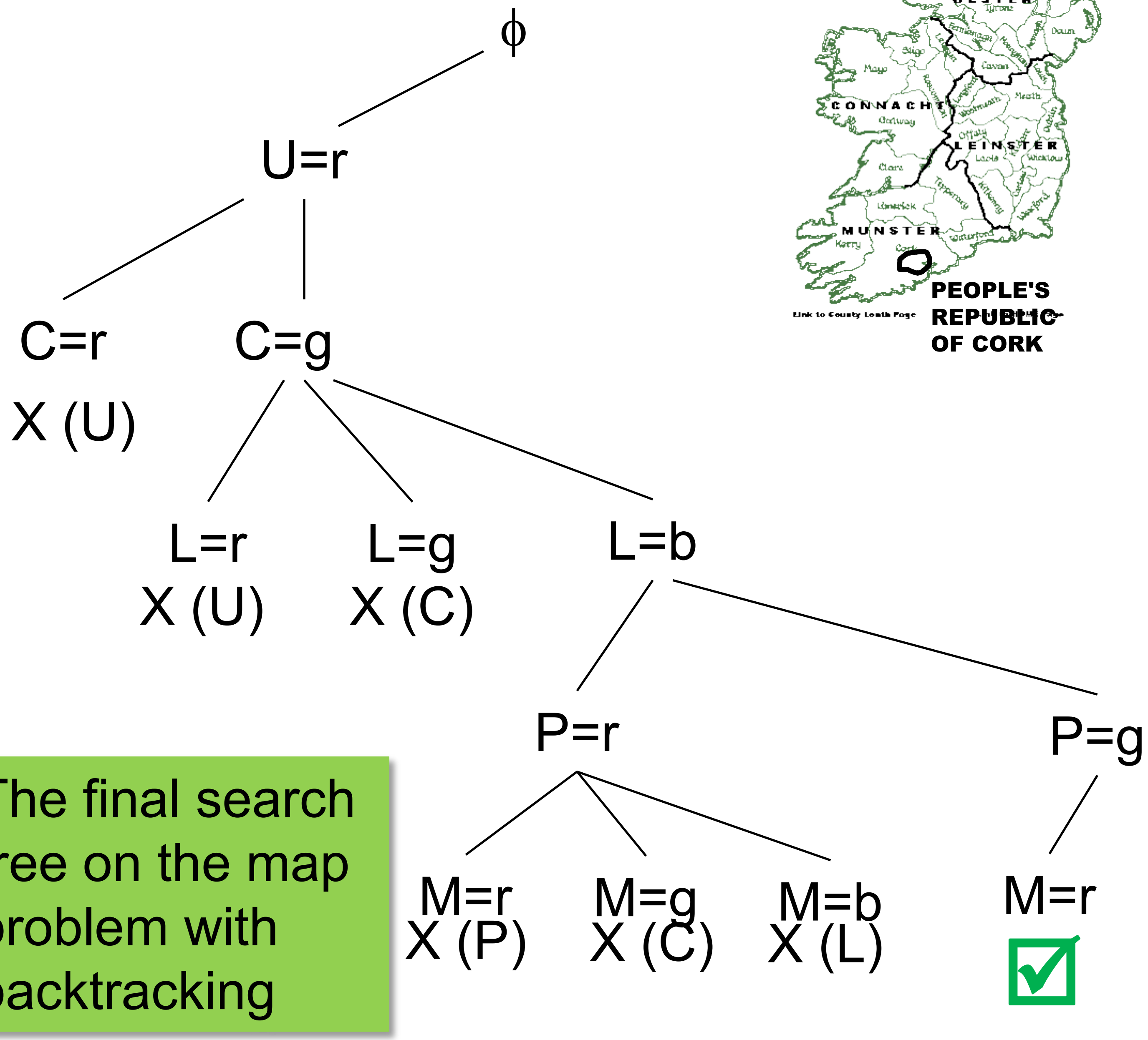
X_1 cannot be 1 or 2 or 7 or 8

...

X_0 is the only variable that can still take value 7

...

variable order: U-C-L-P-M; value order: r-g-b



ϕ

U=r

C=r

X (U)

C=g

L=r

X (U)

L=g

X (C)

L=b

P=r

P=g

M=r

X (P)

M=g

X (C)

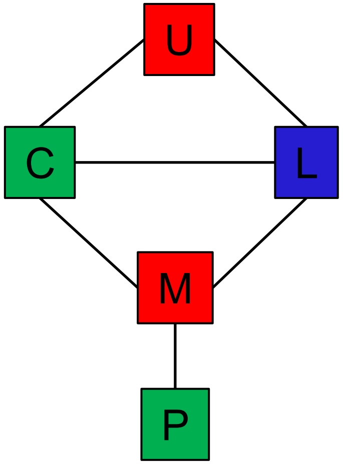
M=b

X (L)

M=r

☑

IRELAND - PROVINCE MAP
ULSTER
CONNACHT
LEINSTER
MUNSTER
Derry
Antrim
Donegal
Tyrone
Fermanagh
Down
Sligo
Mayo
Cavan
Meath
Galway
Offaly
Dublin
Wicklow
Laois
Clare
Tipperary
Kilkenny
Limerick
Kerry
Cork
Waterford
Wexford
Link to County Louth Page
PEOPLE'S
REPUBLIC
OF CORK

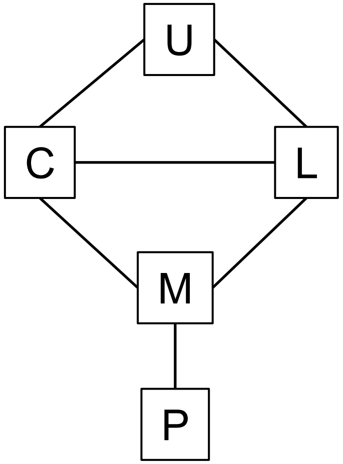


U	C	L	P	M
r	g	b	g	r

The final search tree on the map problem with backtracking

variable order: U-C-L-P-M; value order: r-g-b

ϕ

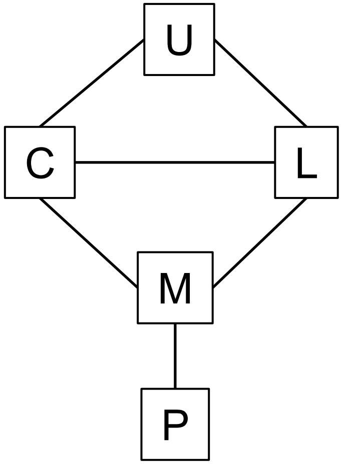


U C L P M

variable order: U-C-L-P-M; value order: r-g-b

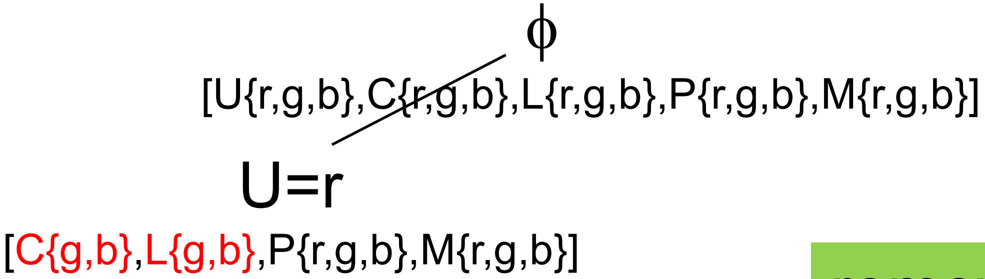
ϕ

[U{r,g,b},C{r,g,b},L{r,g,b},P{r,g,b},M{r,g,b}]



U	C	L	P	M
r	r	r	r	r
g	g	g	g	g
b	b	b	b	b

variable order: U-C-L-P-M; value order: r-g-b



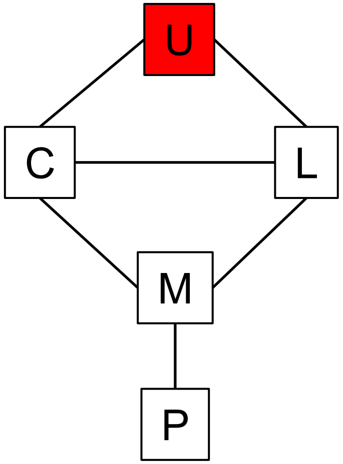
φ

[U{r,g,b},C{r,g,b},L{r,g,b},P{r,g,b},M{r,g,b}]

U=r

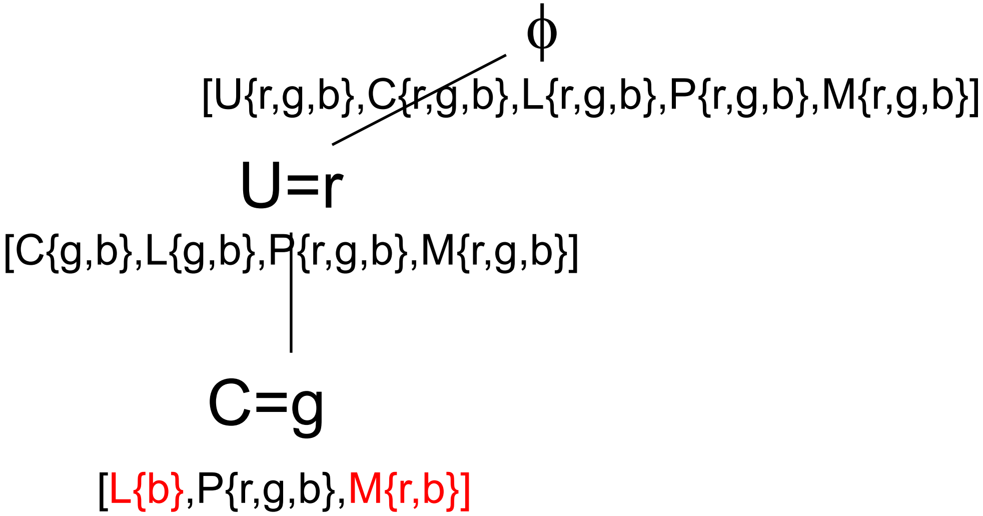
[C{g,b},L{g,b},P{r,g,b},M{r,g,b}]

remove r as values for C and L



U	C	L	P	M
r				
	.	.	r	r
	g	g	g	g
	b	b	b	b

variable order: U-C-L-P-M; value order: r-g-b



ϕ

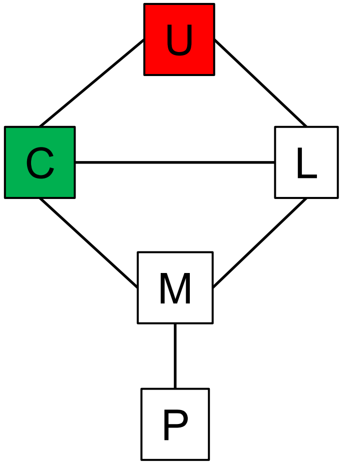
[U{r,g,b},C{r,g,b},L{r,g,b},P{r,g,b},M{r,g,b}]

U=r

[C{g,b},L{g,b},P{r,g,b},M{r,g,b}]

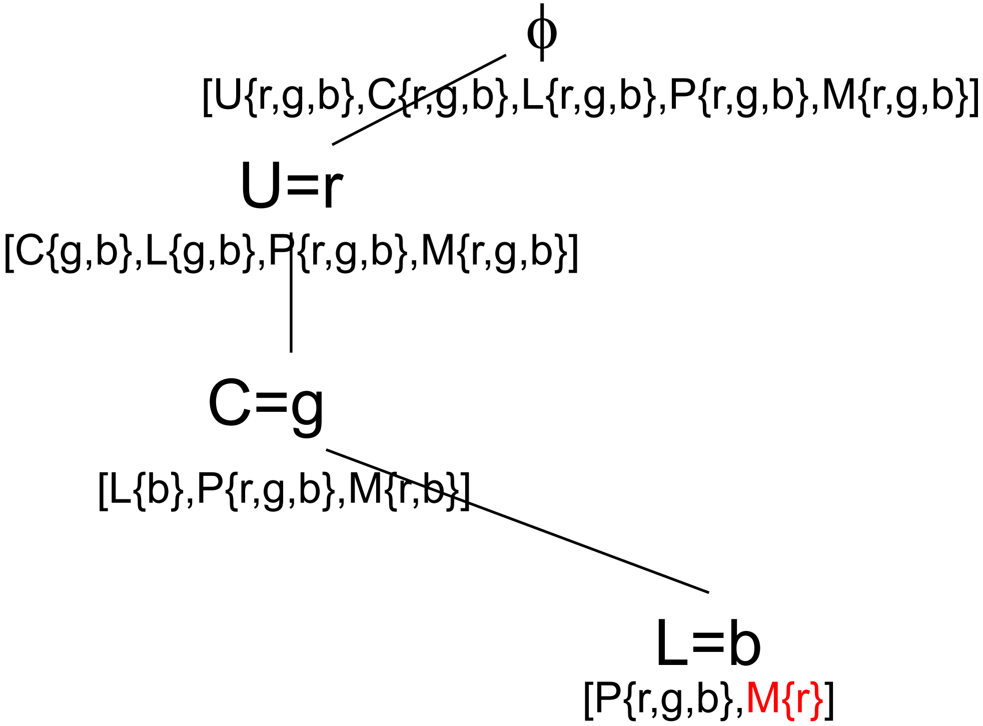
C=g

[L{b},P{r,g,b},M{r,b}]



U	C	L	P	M
r	g			
		.	r	r
		.	g	.
		b	b	b

variable order: U-C-L-P-M; value order: r-g-b



ϕ

[U{r,g,b},C{r,g,b},L{r,g,b},P{r,g,b},M{r,g,b}]

U=r

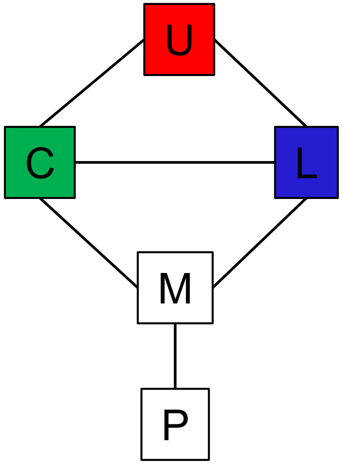
[C{g,b},L{g,b},P{r,g,b},M{r,g,b}]

C=g

[L{b},P{r,g,b},M{r,b}]

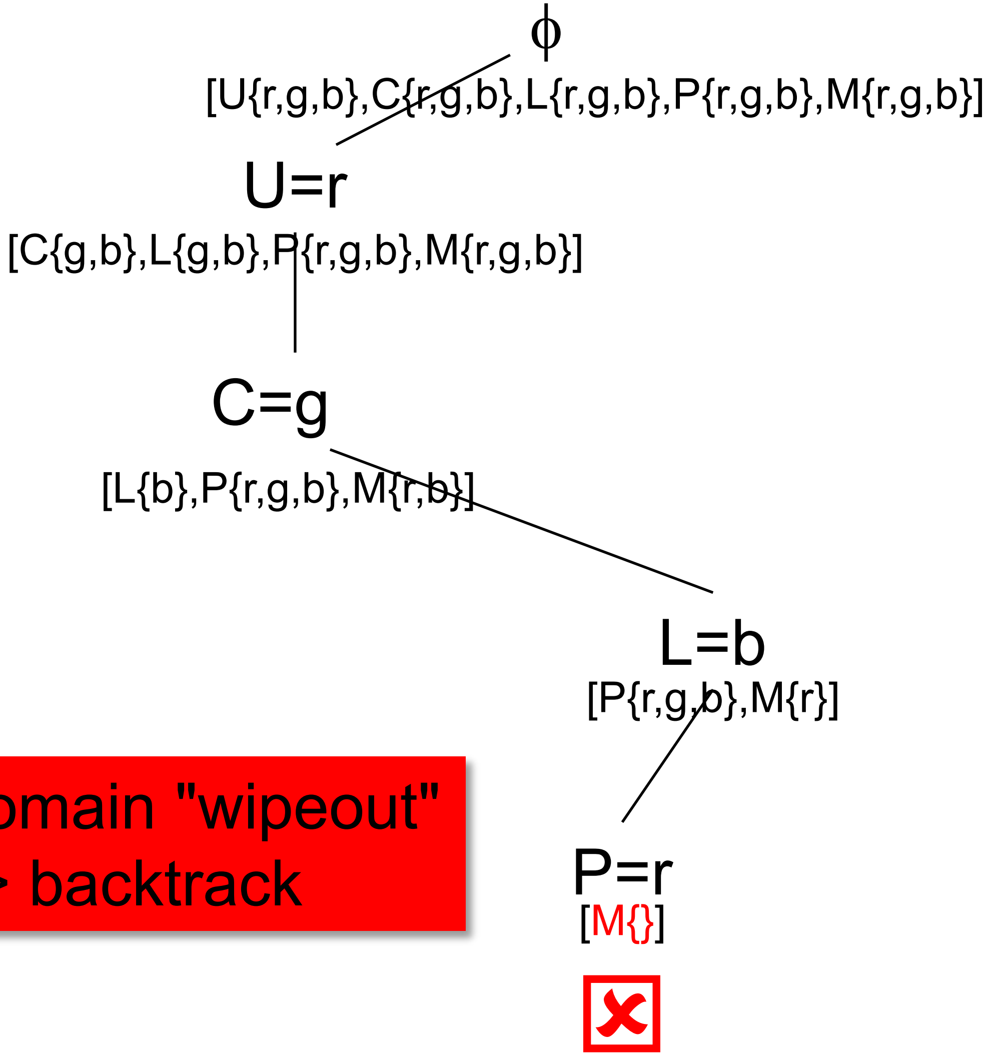
L=b

[P{r,g,b},M{r}]



U	C	L	P	M
r	g	b		
			r	r
			g	.
			b	.

variable order: U-C-L-P-M; value order: r-g-b



ϕ

[U{r,g,b},C{r,g,b},L{r,g,b},P{r,g,b},M{r,g,b}]

U=r

[C{g,b},L{g,b},P{r,g,b},M{r,g,b}]

C=g

[L{b},P{r,g,b},M{r,b}]

L=b

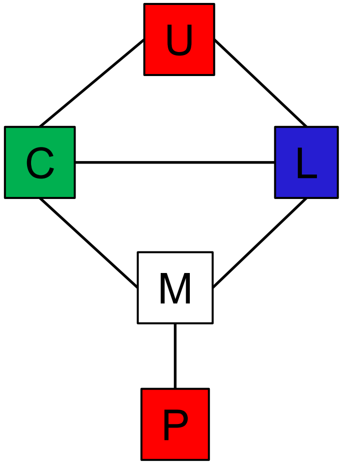
[P{r,g,b},M{r}]

P=r

[M{}]

☒

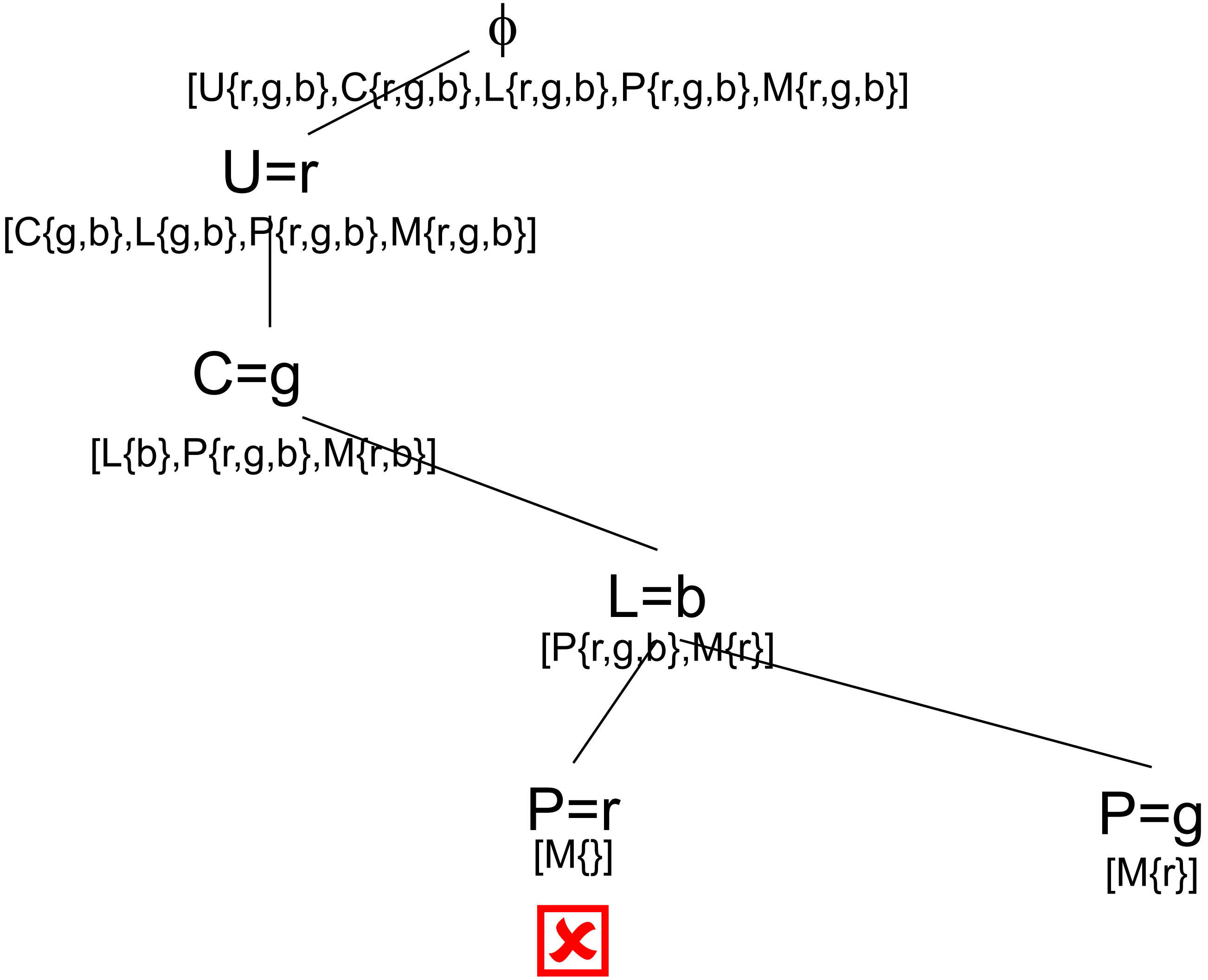
Domain "wipeout"
=> backtrack



U	C	L	P	M
r	g	b	r	

.
.
.

variable order: U-C-L-P-M; value order: r-g-b



ϕ

[U{r,g,b},C{r,g,b},L{r,g,b},P{r,g,b},M{r,g,b}]

U=r

[C{g,b},L{g,b},P{r,g,b},M{r,g,b}]

C=g

[L{b},P{r,g,b},M{r,b}]

L=b

[P{r,g,b},M{r}]

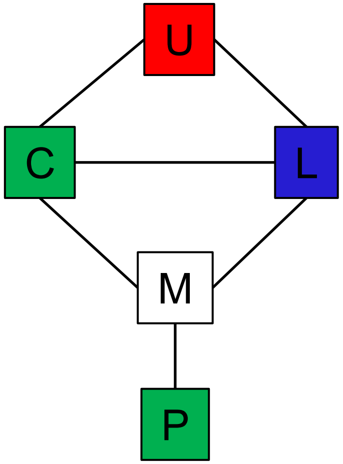
P=r

[M{}]

✗

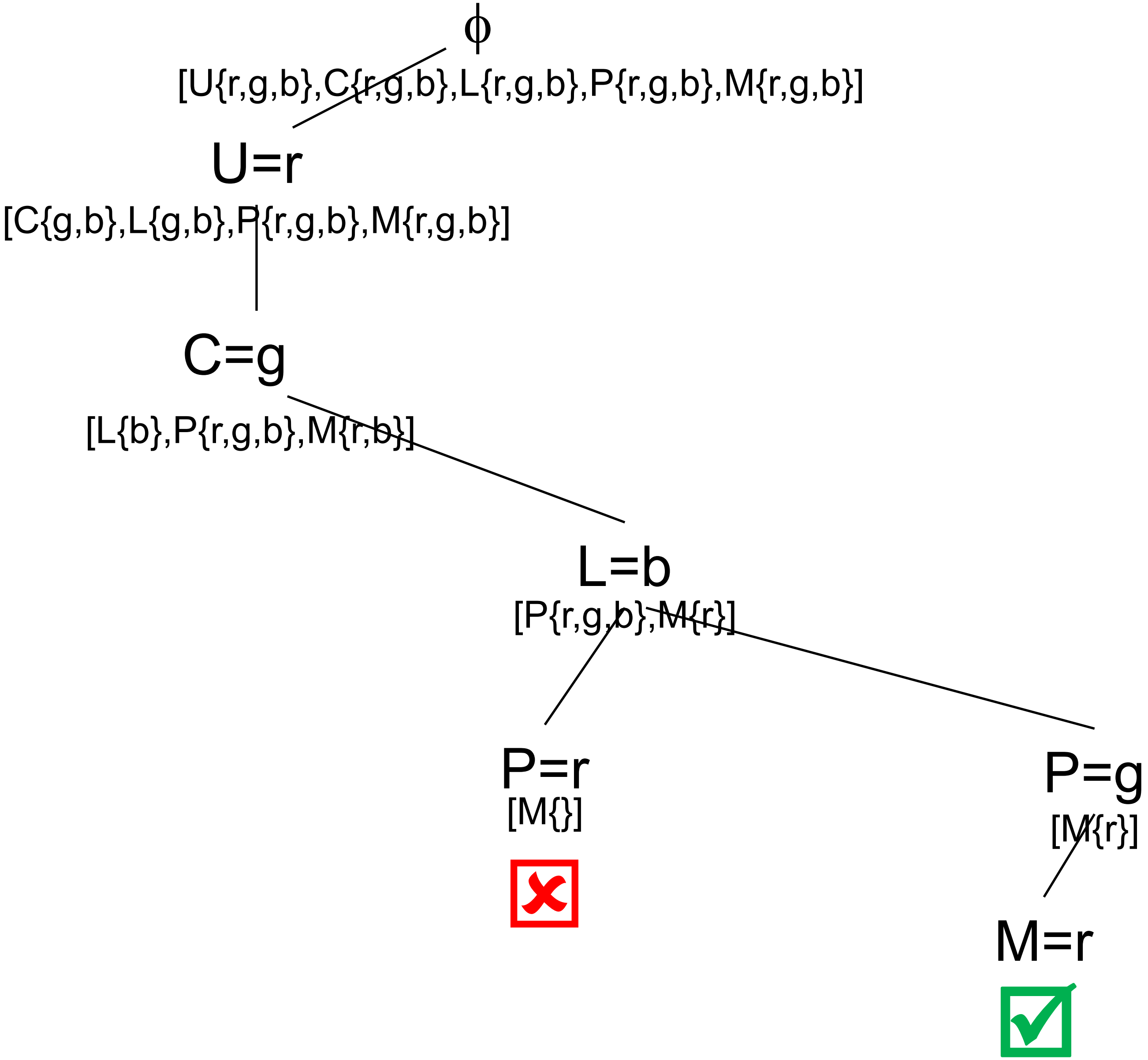
P=g

[M{r}]



U	C	L	P	M
r	g	b	g	
				r
				.
				.

variable order: U-C-L-P-M; value order: r-g-b



ϕ

[U{r,g,b},C{r,g,b},L{r,g,b},P{r,g,b},M{r,g,b}]

U=r

[C{g,b},L{g,b},P{r,g,b},M{r,g,b}]

C=g

[L{b},P{r,g,b},M{r,b}]

L=b

[P{r,g,b},M{r}]

P=r

[M{}]

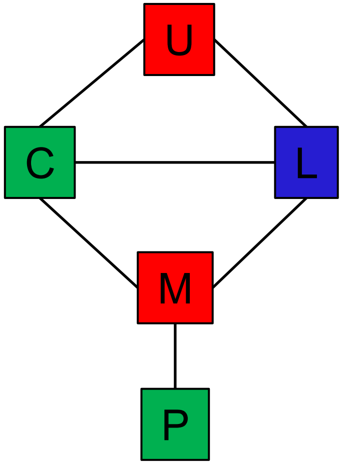
☒

P=g

[M{r}]

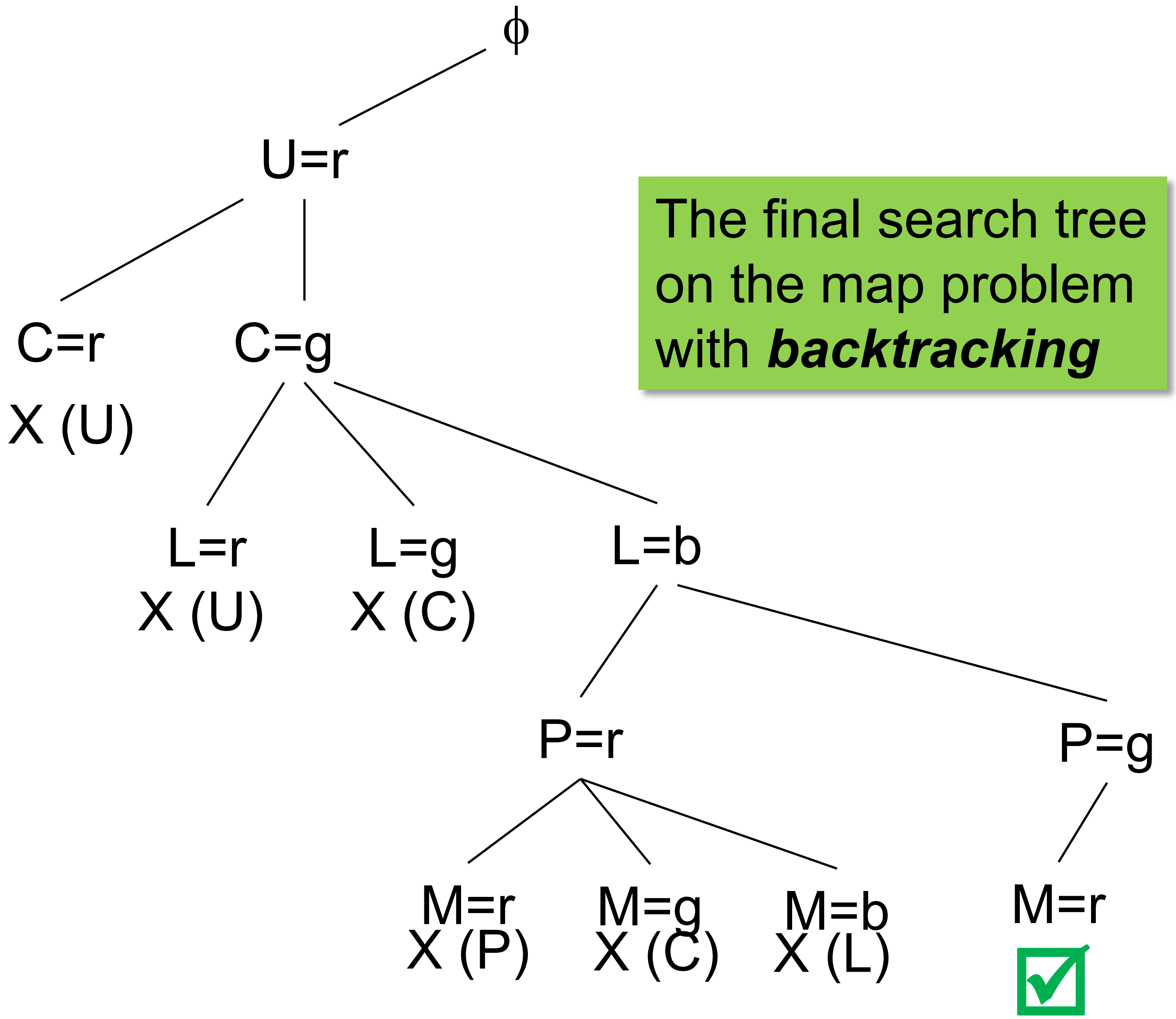
M=r

☑



U	C	L	P	M
r	g	b	g	r

variable order: U-C-L-P-M; value order: r-g-b



ϕ

U=r

C=r
X (U)

C=g

L=r
X (U)

L=g
X (C)

L=b

P=r

M=r
X (P)

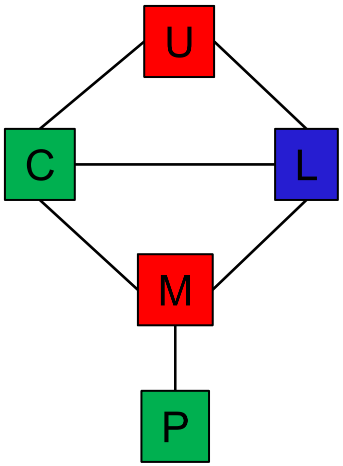
M=g
X (C)

M=b
X (L)

P=g

M=r
☑

The final search tree on the map problem with ***backtracking***



U	C	L	P	M
r	g	b	g	r

Forward checking

In backtracking, replace

```
  check(var, val, values, C)
```

with

```
  fcheck(var, val, domains, C)
```

where `fcheck` reduces all domains of unassigned variables that are constrained with *var*, based on assigning *var←val.* If any domain is emptied, return *false*; else return *true.*

Note: the complexity of backtracking is based on the size of the domains, so reducing the forward domains will shrink the tree.

But how should we implement it?

Maintain a separate copy of the domains, and remove (and replace) values in those domains as we move up and down the tree.

To make sure that we replace values properly, at each node in the search tree, maintain a data structure stating which values of which variables were removed because of this choice.

When we backtrack over a choice, restore the values.

Rather than maintain the domain as a set of values, perhaps maintain each domain as a 2D array

- first row is the values
- second row is a boolean saying the value is still alive

```
V0: a b
    T T
V1: a b
    T T
V2: a b
    T T
V3: a b
    T T
V4: a b
    T T
V5: a b
    T T
V6: a b
    T T
V7: a b
    T T
V8: a b
    T T
V9: a b
    T T
V10: a b
     T T
V11: a b
     T T
V12: a b
     T T
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12
```

V0≠V3

V3=V9

V6=V12

V9=V12

all domains={a,b}

```
V0: a b
    T T
V1: a b
    T T
V2: a b
    T T
V3: a b
    T T
V4: a b
    T T
V5: a b
    T T
V6: a b
    T T
V7: a b
    T T
V8: a b
    T T
V9: a b
    T T
V10: a b
     T T
V11: a b
     T T
V12: a b
     T T
```

V0←a

0	1	2	3	4	5	6	7	8	9	10	11	12
a												

V0≠V3
V3=V9
V6=V12
V9=V12

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     T  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      T  T
```

V0←a

![V3:a]

0	1	2	3	4	5	6	7	8	9	10	11	12
a												

V0≠V3

V3=V9

V6=V12

V9=V12

V0:	a	b
	T	F
V1:	a	b
	T	T
V2:	a	b
	T	T
V3:	a	b
	F	T
V4:	a	b
	T	T
V5:	a	b
	T	T
V6:	a	b
	T	T
V7:	a	b
	T	T
V8:	a	b
	T	T
V9:	a	b
	T	T
V10:	a	b
	T	T
V11:	a	b
	T	T
V12:	a	b
	T	T

V0←a

![V3:a]

V1←a

0	1	2	3	4	5	6	7	8	9	10	11	12
a	a											

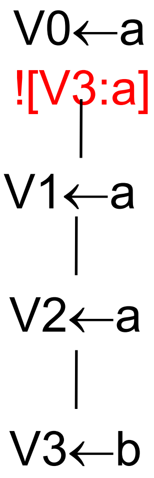
$V0 \neq V3$

$V3 = V9$

$V6 = V12$

$V9 = V12$

```
V0:  a b
     T F
V1:  a b
     T T
V2:  a b
     T T
V3:  a b
     F T
V4:  a b
     T T
V5:  a b
     T T
V6:  a b
     T T
V7:  a b
     T T
V8:  a b
     T T
V9:  a b
     T T
V10:  a b
      T T
V11:  a b
      T T
V12:  a b
      T T
```



V0←a

![V3:a]

V1←a

V2←a

V3←b

0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b									

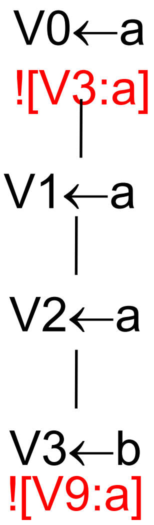
$V0 \neq V3$

$V3 = V9$

$V6 = V12$

$V9 = V12$

V0: a b
T F
V1: a b
T T
V2: a b
T T
V3: a b
F T
V4: a b
T T
V5: a b
T T
V6: a b
T T
V7: a b
T T
V8: a b
T T
V9: a b
F T
V10: a b
T T
V11: a b
T T
V12: a b
T T



V0←a
![V3:a]

V1←a

V2←a

V3←b
![V9:a]

0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b									

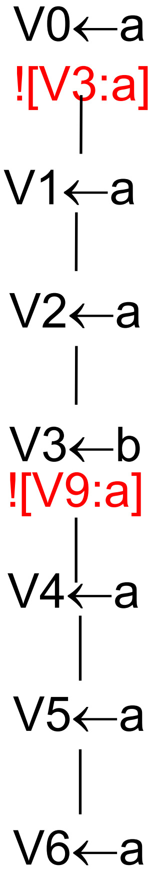
V0≠V3

V3=V9

V6=V12

V9=V12

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     F  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      T  T
```



0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	a						

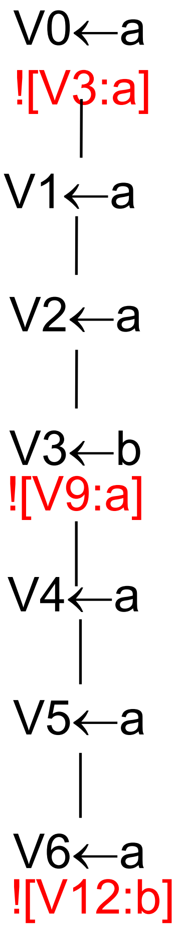
$V0 \neq V3$

$V3 = V9$

$V6 = V12$

$V9 = V12$

V0:	a	b
	T	F
V1:	a	b
	T	T
V2:	a	b
	T	T
V3:	a	b
	F	T
V4:	a	b
	T	T
V5:	a	b
	T	T
V6:	a	b
	T	T
V7:	a	b
	T	T
V8:	a	b
	T	T
V9:	a	b
	F	T
V10:	a	b
	T	T
V11:	a	b
	T	T
V12:	a	b
	T	**F**



0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	a						

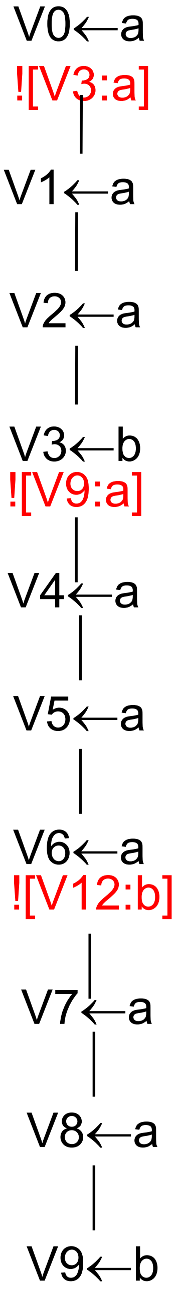
$V0 \neq V3$

$V3 = V9$

$V6 = V12$

$V9 = V12$

```
V0: a b
    T F
V1: a b
    T T
V2: a b
    T T
V3: a b
    F T
V4: a b
    T T
V5: a b
    T T
V6: a b
    T T
V7: a b
    T T
V8: a b
    T T
V9: a b
    F T
V10: a b
     T T
V11: a b
     T T
V12: a b
     T F
```



V0←a

![V3:a]

V1←a

V2←a

V3←b

![V9:a]

V4←a

V5←a

V6←a

![V12:b]

V7←a

V8←a

V9←b

0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	a	a	a	b			

V0≠V3

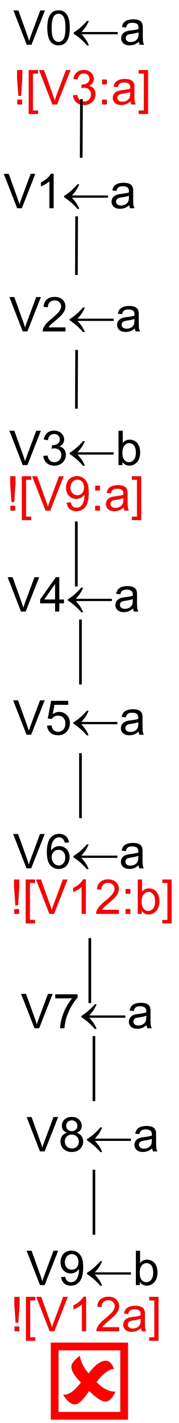
V3=V9

V6=V12

V9=V12

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     F  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      F  F
```

WIPEOUT



0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	a	a	a	b			

V0≠V3

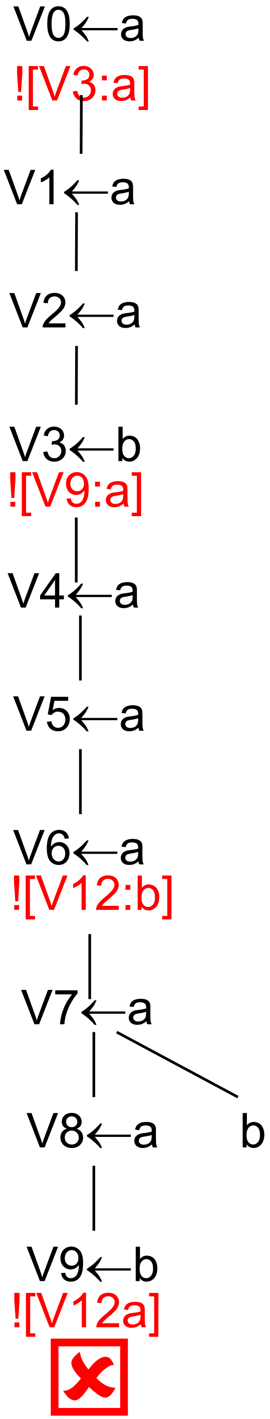
V3=V9

V6=V12

V9=V12

```
V0:  a b
     T F
V1:  a b
     T T
V2:  a b
     T T
V3:  a b
     F T
V4:  a b
     T T
V5:  a b
     T T
V6:  a b
     T T
V7:  a b
     T T
V8:  a b
     T T
V9:  a b
     F T
V10:  a b
      T T
V11:  a b
      T T
V12:  a b
      T F
```

REPLACE



0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	a	a	b				

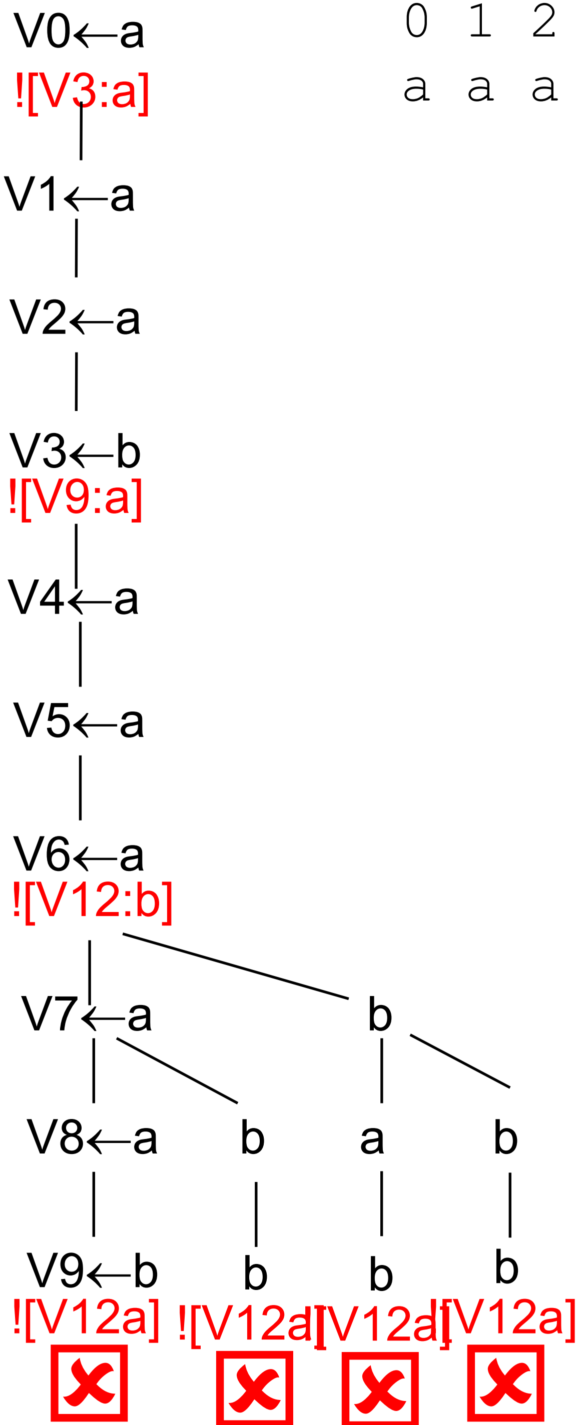
V0≠V3

V3=V9

V6=V12

V9=V12

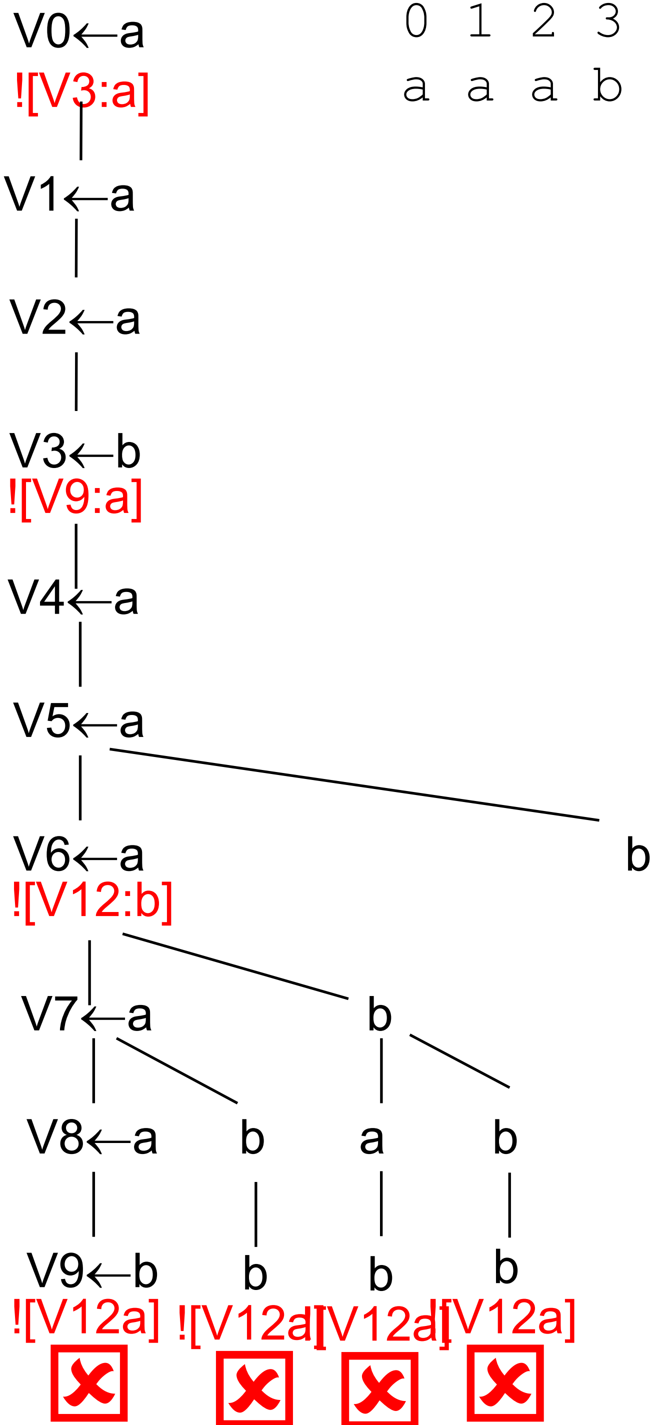
```
V0:  a b
     T F
V1:  a b
     T T
V2:  a b
     T T
V3:  a b
     F T
V4:  a b
     T T
V5:  a b
     T T
V6:  a b
     T T
V7:  a b
     T T
V8:  a b
     T T
V9:  a b
     F T
V10:  a b
      T T
V11:  a b
      T T
V12:  a b
      T F
```



0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	a	b	b	b			

V0≠V3
V3=V9
V6=V12
V9=V12

V0:	a	b
	T	F
V1:	a	b
	T	T
V2:	a	b
	T	T
V3:	a	b
	F	T
V4:	a	b
	T	T
V5:	a	b
	T	T
V6:	a	b
	T	T
V7:	a	b
	T	T
V8:	a	b
	T	T
V9:	a	b
	F	T
V10:	a	b
	T	T
V11:	a	b
	T	T
V12:	a	b
	T	**T**



V0←a

![V3:a]

V1←a

V2←a

V3←b

![V9:a]

V4←a

V5←a b

V6←a b

![V12:b]

V7←a b

V8←a b a b

V9←b b b b

![V12a] ![V12a[V12a[V12a]

☒ ☒ ☒ ☒

0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	b						

$V0 \neq V3$

$V3 = V9$

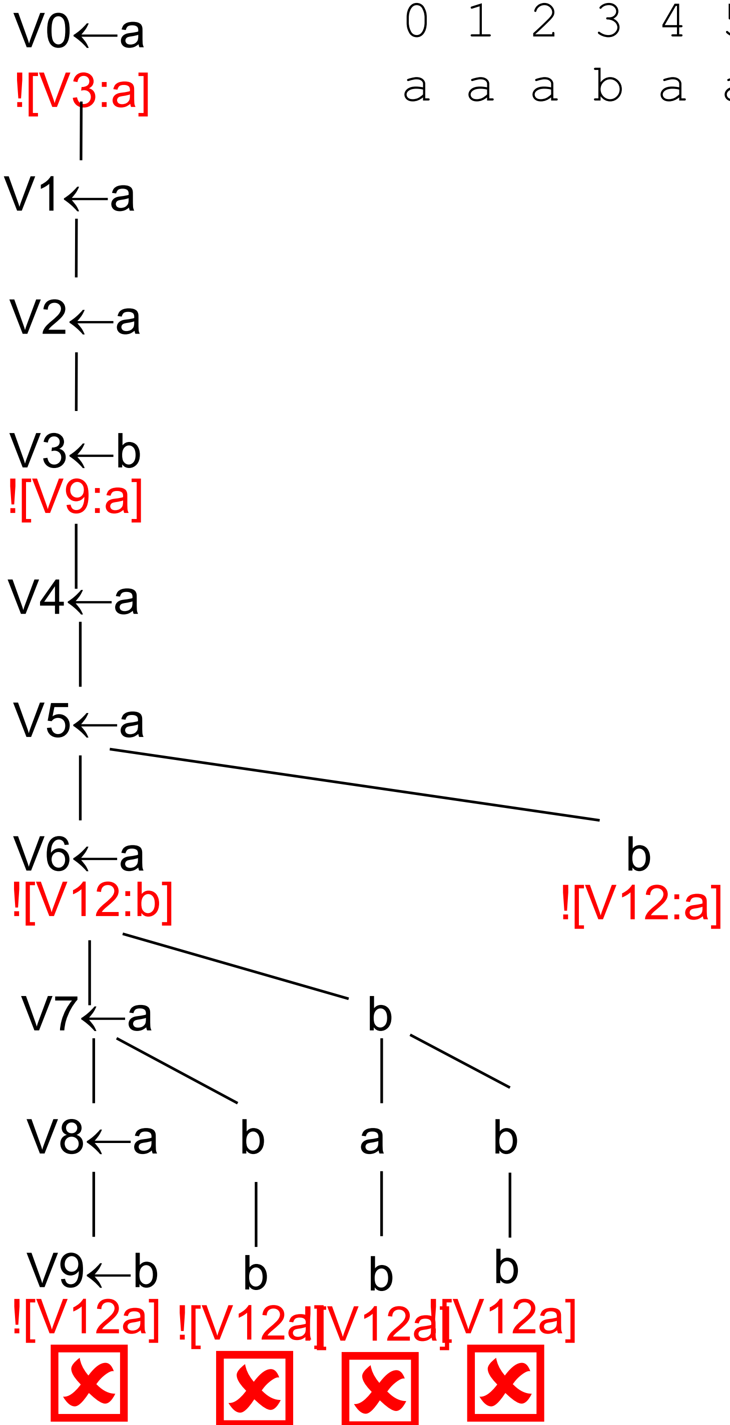
$V6 = V12$

$V9 = V12$

REPLACE

```
V0:  a b
     T F
V1:  a b
     T T
V2:  a b
     T T
V3:  a b
     F T
V4:  a b
     T T
V5:  a b
     T T
V6:  a b
     T T
V7:  a b
     T T
V8:  a b
     T T
V9:  a b
     F T
V10:  a b
      T T
V11:  a b
      T T
V12:  a b
      F T
```

0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	b						



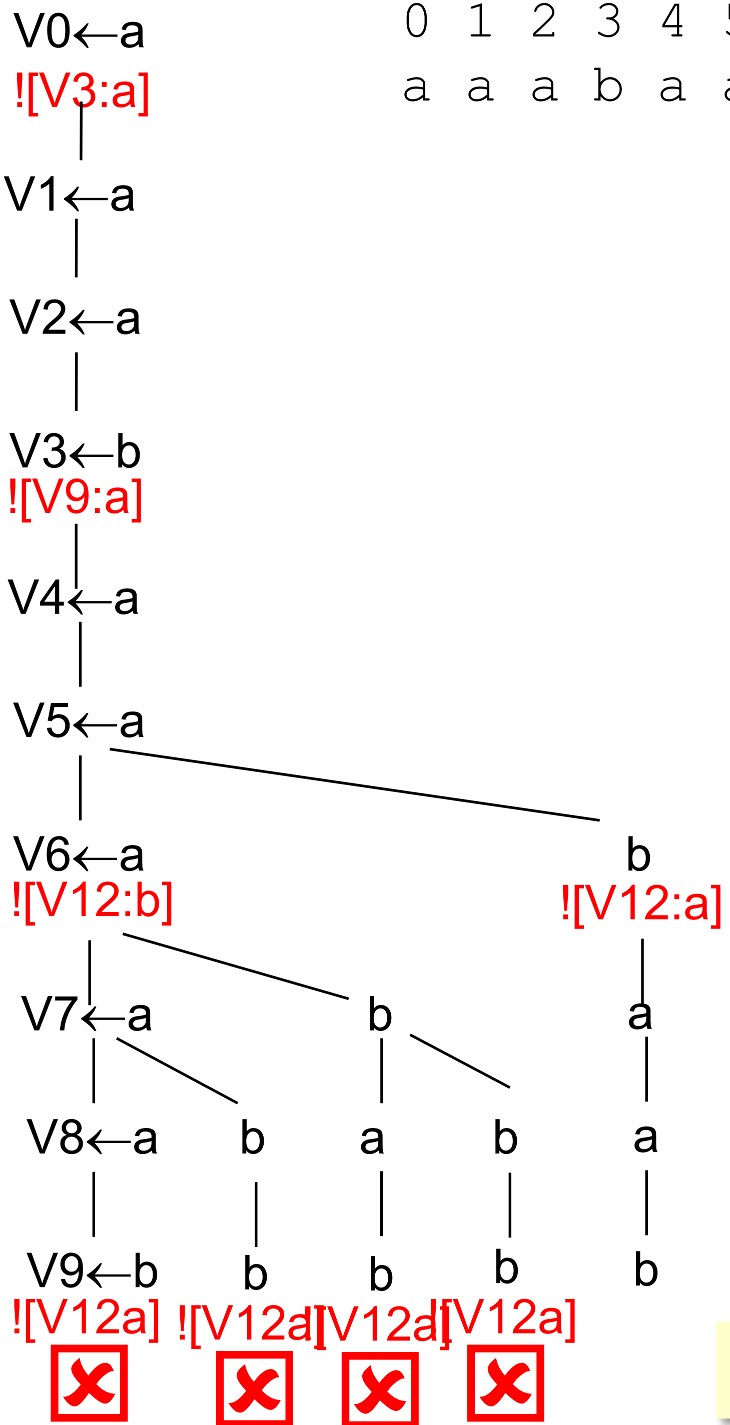
V0≠V3

V3=V9

V6=V12

V9=V12

V0:	a	b
	T	F
V1:	a	b
	T	T
V2:	a	b
	T	T
V3:	a	b
	F	T
V4:	a	b
	T	T
V5:	a	b
	T	T
V6:	a	b
	T	T
V7:	a	b
	T	T
V8:	a	b
	T	T
V9:	a	b
	F	T
V10:	a	b
	T	T
V11:	a	b
	T	T
V12:	a	b
	F	T



0	1	2	3	4	5	6	7	8	9	10	11	12
a	a	a	b	a	a	a	a	a	b			

$V0 \neq V3$

$V3 = V9$

$V6 = V12$

$V9 = V12$

and finish from here

```
fc(V, D, C)
input: V (array of n int) D (2d array of n*d int) C (constraints)
output: (bool, V), where bool = true if a solution is found
dom := copied augmented D
var := 0  //start with the first variable
val := 0  //start with the first value
stack := {}
done := false  //not finished yet
while not done
  if var == n  return (true, values)    //if reached end, stop
  else if var == -1 return (false, null)//if back to top, stop
  else if val == -1           //if no values left
     var := var-1             //step back to previous variable
     restore(pop(stack),&dom); //replace dels from var<-val
     val := nextval(var,values[var])  //continue to next value
  else if fcheck(var,val,&dom, &stack, C) == false
     restore(pop(stack),&dom); //replace dels from var<-val
     val := nextval(var,val)  //get next value, or -1 if none
  else
     values[var] := val       //assign current value
     var := var + 1           //move to next var
     val := 0                 //start with first value
```

```
fcheck(var, val,*dom, *stack, C)
output: true if no wipeout for var<-val when checking forwards

// For each constraint in C whose scope is {var,X}, where X is
// after var, check var<-val against X's domain, removing
// incompatible values from *dom, and before returning pushing
// those deletions as one entry onto the top of *stack

//i.e. while searching the tree, each time we try a value for
//a variable, check forward against unassigned variables to
//remove bad values
```

Forward checking still thrashes. E.g., repeated search over choices for V7 and V8, even though no solution is possible in that area of the tree.

Can we improve it?

- record conflicts and combine with CBJ?
    - Yes (but we won't in CS4093 – see Prosser 1993)

- when a domain drops to a single value, why wait?
    - bring it forward and assign immediately

V0:	a	b
	T	F
V1:	a	b
	T	T
V2:	a	b
	T	T
V3:	a	b
	F	T
V4:	a	b
	T	T
V5:	a	b
	T	T
V6:	a	b
	T	T
V7:	a	b
	T	T
V8:	a	b
	T	T
V9:	a	b
	T	T
V10:	a	b
	T	T
V11:	a	b
	T	T
V12:	a	b
	T	T

V0←a

![V3:a]

0	1	2	3	4	5	6	7	8	9	10	11	12
a												

At this point, V3 only has one value. Do it now.

Maintain the search tree in a stack?

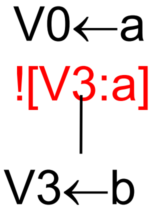
V0≠V3

V3=V9

V6=V12

V9=V12

```
V0: a b
    T F
V1: a b
    T T
V2: a b
    T T
V3: a b
    F T
V4: a b
    T T
V5: a b
    T T
V6: a b
    T T
V7: a b
    T T
V8: a b
    T T
V9: a b
    T T
V10: a b
     T T
V11: a b
     T T
V12: a b
     T T
```



V0←a

![V3:a]

V3←b

```
0 1 2 3 4 5 6 7 8 9 10 11 12
a     b


0 3 ->
```

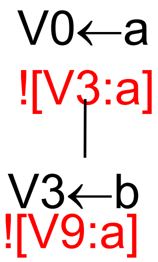
V0≠V3

V3=V9

V6=V12

V9=V12

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     F  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      T  T
```



V0←a

![V3:a]

V3←b

![V9:a]

```
0  1  2  3  4  5  6  7  8  9  10  11  12
a        b


0  3  ->
```

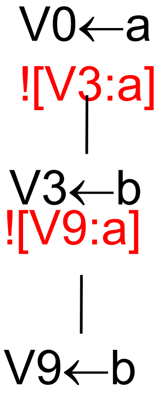
$V0 \neq V3$

$V3 = V9$

$V6 = V12$

$V9 = V12$

```
V0: a b
    T F
V1: a b
    T T
V2: a b
    T T
V3: a b
    F T
V4: a b
    T T
V5: a b
    T T
V6: a b
    T T
V7: a b
    T T
V8: a b
    T T
V9: a b
    F T
V10: a b
     T T
V11: a b
     T T
V12: a b
     T T
```



```
0 1 2 3 4 5 6 7 8 9 10 11 12
a     b           b
```

```
0 3 9 ->
```

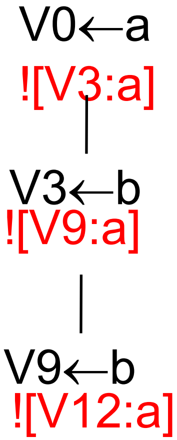
V0≠V3

V3=V9

V6=V12

V9=V12

```
V0: a b
    T F
V1: a b
    T T
V2: a b
    T T
V3: a b
    F T
V4: a b
    T T
V5: a b
    T T
V6: a b
    T T
V7: a b
    T T
V8: a b
    T T
V9: a b
    F T
V10: a b
     T T
V11: a b
     T T
V12: a b
     F T
```


V0←a

![V3:a]

V3←b

![V9:a]

V9←b

![V12:a]

0	1	2	3	4	5	6	7	8	9	10	11	12
a			b						b			

0 3 9 ->

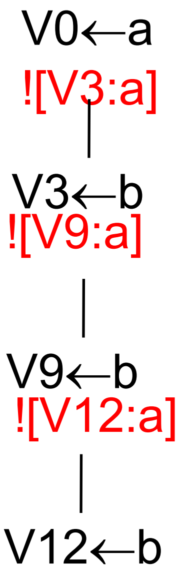
V0≠V3

V3=V9

V6=V12

V9=V12

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     F  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      F  T
```

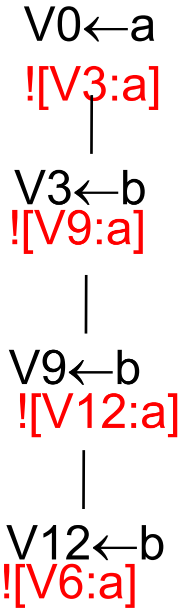


0	1	2	3	4	5	6	7	8	9	10	11	12
a			b						b			b

0 3 9 12 ->

V0≠V3
V3=V9
V6=V12
V9=V12

```
V0: a b
    T F
V1: a b
    T T
V2: a b
    T T
V3: a b
    F T
V4: a b
    T T
V5: a b
    T T
V6: a b
    F T
V7: a b
    T T
V8: a b
    T T
V9: a b
    F T
V10: a b
     T T
V11: a b
     T T
V12: a b
     F T
```



V0←a

![V3:a]

V3←b

![V9:a]

V9←b

![V12:a]

V12←b

![V6:a]

0	1	2	3	4	5	6	7	8	9	10	11	12
a			b						b			b

0 3 9 12 ->

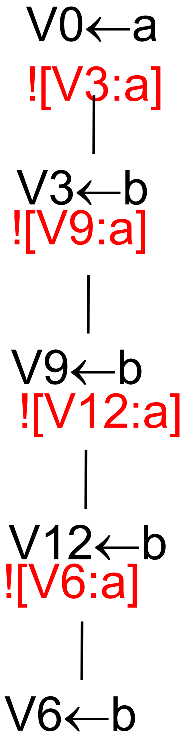
V0≠V3

V3=V9

V6=V12

V9=V12

```
V0:  a b
     T F
V1:  a b
     T T
V2:  a b
     T T
V3:  a b
     F T
V4:  a b
     T T
V5:  a b
     T T
V6:  a b
     F T
V7:  a b
     T T
V8:  a b
     T T
V9:  a b
     F T
V10:  a b
      T T
V11:  a b
      T T
V12:  a b
      F T
```



V0←a

![V3:a]

V3←b

![V9:a]

V9←b

![V12:a]

V12←b

![V6:a]

V6←b

0	1	2	3	4	5	6	7	8	9	10	11	12
a			b			b			b			b

0 3 9 12 6 ->

... and finish from here

$V0 \neq V3$

$V3 = V9$

$V6 = V12$

$V9 = V12$

Forward checking still thrashes. E.g., repeated search over choices for V7 and V8, even though no solution is possible in that area of the tree.

Can we improve it?

- record conflicts and combine with CBJ?
  - Yes (but we won't in CS4093 – see Prosser 1993)

- when a domain drops to a single value, why wait?
  - bring it forward and assign immediately

- make the variable order dynamic, and always choose the variable with the smallest domain?

Exercise

How would you modify the FC algorithm to allow the variable ordering to change during search?

Note: in our original version, we numbered the variables, and simply stepped though the sequence.

(Hint: see the traces above)

Forward checking still thrashes. E.g., repeated search over choices for V7 and V8, even though no solution is possible in that area of the tree.

Can we improve it?

- record conflicts and combine with CBJ?
    - Yes (but we won't in CS4093 – see Prosser 1993)
- when a domain drops to a single value, why wait?
    - bring it forward and assign immediately
- make the variable order dynamic, and always choose the variable with the smallest domain?
- whenever a domain is reduced, why not apply AC?

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     T  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      T  T
```

V0←a

![V3:a]

0

a

0 ->

V0≠V3

V3=V9

V6=V12

V9=V12

AC3 Q: C93

V0: a b
T F
V1: a b
T T
V2: a b
T T
V3: a b
F T
V4: a b
T T
V5: a b
T T
V6: a b
T T
V7: a b
T T
V8: a b
T T
V9: a b
F T
V10: a b
T T
V11: a b
T T
V12: a b
T T

V0←a

![V3:a,V9:a]

0

a

0 ->

V0≠V3

V3=V9

V6=V12

V9=V12

AC3 Q: C93

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     F  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      T  T
```

V0←a

![V3:a,V9:a]

```
0
a


0  ->
```

V0≠V3

V3=V9

V6=V12

V9=V12

AC3 Q: ~~C93~~, C12-9

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     F  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      F  T
```

V0←a

![V3:a,V9:a,
V12:a]

0

a

0 ->

V0≠V3
V3=V9
V6=V12
V9=V12

AC3 Q: ~~C93~~, C12-9

```
V0:  a  b
     T  F
V1:  a  b
     T  T
V2:  a  b
     T  T
V3:  a  b
     F  T
V4:  a  b
     T  T
V5:  a  b
     T  T
V6:  a  b
     T  T
V7:  a  b
     T  T
V8:  a  b
     T  T
V9:  a  b
     F  T
V10:  a  b
      T  T
V11:  a  b
      T  T
V12:  a  b
      F  T
```

V0←a

![V3:a,V9:a,
V12:a]

```
0
a


0  ->
```

V0≠V3
V3=V9
V6=V12
V9=V12

AC3 Q: ~~C93~~, ~~C12-9~~, C6-12

V0:	a	b
	T	F
V1:	a	b
	T	T
V2:	a	b
	T	T
V3:	a	b
	F	T
V4:	a	b
	T	T
V5:	a	b
	T	T
V6:	a	b
	F	T
V7:	a	b
	T	T
V8:	a	b
	T	T
V9:	a	b
	F	T
V10:	a	b
	T	T
V11:	a	b
	T	T
V12:	a	b
	F	T

V0←a

![V3:a,V9:a,
V12:a, V6:a]

0

a

0 ->

V0≠V3

V3=V9

V6=V12

V9=V12

AC3 Q: ~~C93~~, ~~C12-9~~, C6-12

```
V0:  a b
     T F
V1:  a b
     T T
V2:  a b
     T T
V3:  a b
     F T
V4:  a b
     T T
V5:  a b
     T T
V6:  a b
     F T
V7:  a b
     T T
V8:  a b
     T T
V9:  a b
     F T
V10:  a b
      T T
V11:  a b
      T T
V12:  a b
      F T
```

V0←a

![V3:a,V9:a, V12:a, V6:a]

```
0 1 2 3 4 5 6 7 8 9 10 11 12
a     b     b     b        b
```

```
0 3 9 12 6 ->
```

and finish from here ...

V0≠V3

V3=V9

V6=V12

V9=V12

AC3 Q: ~~C93~~, ~~C12-9~~, ~~C6-12~~

MAC: Maintaining Arc Consistency

```
mac(V, D, C)
input: V (array of n int) D (2d array of n*d int) C (constraints)
output: (bool, V), where bool = true if a solution is found
dom := copied augmented D
var := 0  //start with the first variable
val := 0  //start with the first value
stack := {}
done := false  //not finished yet
while not done
  if var == n  return (true, values)    //if reached end, stop
  else if var == -1 return (false, null)//if back to top, stop
  else if val == -1           //if no values left
     var := var-1             //step back to previous variable
     restore(pop(stack),&dom); //replace dels from var<-val
     val := nextval(var,values[var])  //continue to next value
  else if fc-ac(var,val,&dom, &stack, C) == false
     restore(pop(stack),&dom); //replace dels from var<-val
     val := nextval(var,val)    //get next value, or -1 if none
  else
     values[var] := val       //assign current value
     var := var + 1           //move to next var
     val := 0                 //start with first value
```

```
fc-ac(var, val,*dom, *stack, C)
output: true if no wipeout for var<-val when checking forwards

// For each constraint in C whose scope is {var,X}, where X is
// after var, check var<-val against X's domain, removing
// incompatible values from *dom, then establish AC on the
// unassigned domains,  and before returning pushing
// those deletions as one entry onto the top of *stack

//i.e. while searching the tree, each time we try a value for
//a variable, check forward against unassigned variables to
//remove bad values
```

Things to note

- AC algorithms are polynomial (e.g. AC2001 is O(ed^2)), while backtracking is exponential (O(d^n))
  - ed^2 seems like a lot of work at possibly every node in the search tree, but if the problem is big enough it will pay off
  - do more work at each node, and reduce the tree size
- Arc consistency during search is the core of constraint programming
  - on practical problems, it will be run millions of times
  - efficient data structures and algorithms are critical
- Any AC algorithm can be slotted into the MAC algorithm
  - (or indeed, any consistency algorithm, but in practice, going higher than AC is rarely worth it)

Exercise 2

- Solve the Crystal maze problem by hand, using first FC and then MAC (using AC3)
  - use the version with binary inequality constraints, on slide 6 of Lecture 1
  - try the fixed variable ordering X_2, X_5, X_1, X_4, X_3, X_6, X_0, X_7

Exercise 3

- have a look at Marc van Dongen's "How to solve the Zebra problem"

http://www.cs.ucc.ie/~dongen/ZEBRA.pdf

Notes

- Forward checking was first proposed by Haralick and Elliott (1980)
- Results in that paper and others led people to believe maintaining AC was too expensive, until Sabin and Freuder (1994)

R. M. Haralick and G. L. Elliott. "Increasing tree search efficiency for constraint satisfaction problems". *Artificial Intelligence*, 14:263–313, 1980.
D. Sabin and E. Freuder (1994), "Contradicting conventional wisdom in constraint satisfaction", *Proceedings of the European Conference on Artificial Intelligence*, pp125--129, 1994.

Next lecture ...

Using extra knowledge when maintaining arc consistency:

the *alldifferent* global constraint

The map colouring example and the CSPs were adapted from examples by Patrick Prosser