

# Discrete Optimization

Constraint Programming: Part III

# Goals of the Lecture

- ▶ Illustrating the rich modeling language of constraint programming
- ▶ Key aspect of constraint programming
  - ability to state complex, idiosyncratic constraints

# Magic Series

- ▶ A series  $S = (S_0, \dots, S_n)$  is magic if  $S_i$  represents the number of occurrences of  $i$  in  $S$

	0	1	2	3	4
Occurrences	?	?	?	?	?

# Magic Series

- ▶ A series  $S = (S_0, \dots, S_n)$  is magic if  $S_i$  represents the number of occurrences of  $i$  in  $S$

	0	1	2	3	4
Occurrences	?	?	?	?	?

- ▶ Can you find a magic series?

# Magic Series

- ▶ A series  $S = (S_0, \dots, S_n)$  is magic if  $S_i$  represents the number of occurrences of  $i$  in  $S$

	0	1	2	3	4
Occurrences	2	1	2	0	0

- ▶ Can you find a magic series?



# Magic Series and Reification

```
int n = 5;  
range D = 0..n-1;  
var{int} series[D] in D;  
solve {  
  forall(k in D)  
    series[k] = sum(i in D) (series[i]=k);  
}
```

# Magic Series and Reification

```
int n = 5;  
range D = 0..n-1;  
var{int} series[D] in D;  
solve {  
  forall(k in D)  
    series[k] = sum(i in D) (series[i]=k);  
}
```

# Magic Series and Reification

```
int n = 5;  
range D = 0..n-1;  
var{int} series[D] in D;  
solve {  
    forall(k in D)  
        series[k] = sum(i in D) (series[i]=k);  
}
```

## ► Reification

- the ability to transform a constraint into a 0/1 variable
- the variable has the value 1 if the constraint is true and 0 otherwise



# Magic Series and Reification

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0);  
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);  
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2);  
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3);  
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4);
```

# Magic Series and Reification

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

	0	1	2	3	4
Occurrences	?	?	?	?	?

# Magic Series and Reification

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;  
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;  
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;  
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;  
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

# Magic Series and Reification

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;  
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;  
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;  
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;  
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

► What if series[0]=1?



# Magic Series and Reification

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;  
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;  
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;  
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;  
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

► What if series[0]=1?

```
1          = 0 + (series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;  
series[1]  = 1 + (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;  
series[2]  = 0 + (series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;  
series[3]  = 0 + (series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;  
series[4]  = 0 + (series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```



# Magic Series and Reification

```
1      = 0 + (series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;  
series[1] = 1 + (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;  
series[2] = 0 + (series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;  
series[3] = 0 + (series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;  
series[4] = 0 + (series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

# Magic Series and Reification

```
1      = 0 + (series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;  
series[1] = 1 + (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;  
series[2] = 0 + (series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;  
series[3] = 0 + (series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;  
series[4] = 0 + (series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

► But now `series[1] > 0`?

# Magic Series and Reification

```
1          = 0 + (series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0) ;
series[1]  = 1 + (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;
series[2]  = 0 + (series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;
series[3]  = 0 + (series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;
series[4]  = 0 + (series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

► But now `series[1] > 0`?

```
1          =          (series[2]=0)+(series[3]=0)+(series[4]=0) ;
series[1]  = 1 + (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1) ;
series[2]  =          (series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2) ;
series[3]  =          (series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3) ;
series[4]  =          (series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4) ;
```

# Reification

## ► What is happening behind the scene?

```
int n = 5;  
range D = 0..n-1;  
var{int} series[D] in D;  
solve {  
  forall(k in D) {  
    var{int} b[D] in 0..1;  
    forall(i in D)  
      booleq(b[i],s[i],k);  
    series[k] = sum(i in D) b[i];  
  }  
}
```



# Reification

- What is happening behind the scene?

```
int n = 5;
range D = 0..n-1;
var{int} series[D] in D;
solve {
  forall(k in D) {
    var{int} b[D] in 0..1;
    forall(i in D)
      booleq(b[i], s[i], k);
    series[k] = sum(i in D) b[i];
  }
}
```

- Constraint  $\text{booleq}(b, x, v)$  holds  
if  $(b=1 \text{ and } x=v)$  or  $(b=0 \text{ and } x \neq v)$ .

$$\text{booleq}(b, x, v) \Leftrightarrow (b=1 \wedge x=v) \vee (b=0 \wedge x \neq v)$$



# Reification

- What is happening behind the scene?

```
int n = 5;
range D = 0..n-1;
var{int} series[D] in D;
solve {
  forall(k in D) {
    var{int} b[D] in 0..1;
    forall(i in D)
      booleq(b[i], s[i], k);
    series[k] = sum(i in D) b[i];
  }
}
```

- Constraint  $\text{booleq}(b, x, v)$  holds  
if  $(b=1 \text{ and } x=v)$  or  $(b=0 \text{ and } x \neq v)$ .

$$\text{booleq}(b, x, v) \Leftrightarrow (b=1 \wedge x=v) \vee (b=0 \wedge x \neq v)$$

# Stable Marriages

- ▶ A marriage between Hugh and Angelina is stable provided that
  - If Angelina prefers another man, say George, over Hugh, then George must prefer his spouse over Angelina
  - If Hugh prefers another woman, say Julia, over Angelina, then Julia must prefer her spouse over Hugh
- ▶ These stability rules make the marriage stable!

# Stable Marriages

- What are the decision variables?

# Stable Marriages

## ► Data and decision variables

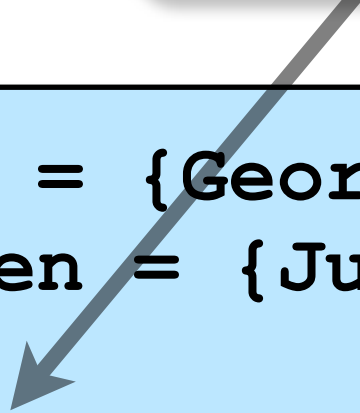
```
enum Men = {George, Hugh, Will, Clive};  
enum Women = {Julia, Halle, Angelina, Keira};  
  
int wrank[Men, Women];  
int mrank[Women, Men];  
...  
  
var{Women} wife[Men];  
var{Men} husband[Women];
```

# Stable Marriages

## ► Data and decision variables

**wrank**[Hugh, Julia] is the ranking of Julia in Hugh's preferences

```
enum Men = {George, Hugh, Will, Clive};  
enum Women = {Julia, Halle, Angelina, Keira};  
  
int wrank[Men, Women];  
int mrank[Women, Men];  
...  
  
var{Women} wife[Men];  
var{Men} husband[Women];
```

A grey arrow points from the text box above to the 'wranks' variable in the code block below.



# Stable Marriages

## ► Data and decision variables

**wrank [Hugh, Julia]** is the ranking of Julia in Hugh's preferences

```
enum Men = {George, Hugh, Will, Clive};  
enum Women = {Julia, Halle, Angelina, Keira};  
  
int wrank[Men, Women];  
int mrank[Women, Men];  
...  
  
var{Women} wife[Men];  
var{Men} husband[Women];
```

**mrnk [Julia, Hugh]** is the ranking of Hugh in Julia's preference

# Stable Marriages

```
solve {  
  forall(m in Men)  
    husband[wife[m]] = m;  
  forall(w in Women)  
    wife[husband[w]] = w;  
  
  ...  
}
```

# Stable Marriages

```
solve {  
  forall(m in Men)  
    husband[wife[m]] = m;  
  forall(w in Women)  
    wife[husband[w]] = w;  
  
  forall(m in Men, w in Women)  
    wrank[m,w] < wrank[w,wife[m]] => mrank[w,husband[w]] < mrank[w,m];  
  forall(w in Women, m in Men)  
    mrank[w,m] < mrank[w,husband[m]] => wrank[m,wife[m]] < mrank[m,w];  
}
```

# Stable Marriages

```
solve {  
  forall(m in Men)  
    husband[wife[m]] = m;  
  forall(w in Women)  
    wife[husband[w]] = w;  
  
  forall(m in Men, w in Women)  
    [ wrank[m,w] < wrank[w,wife[m]] ] => mrank[w,husband[w]] < mrank[w,m];  
  forall(w in Women, m in Men)  
    mrank[w,m] < mrank[w,husband[m]] => wrank[m,wife[m]] < mrank[m,w];  
}
```

**m prefers w over his wife**

# Stable Marriages

```
solve {  
  forall(m in Men)  
    husband[wife[m]] = m;  
  forall(w in Women)  
    wife[husband[w]] = w;  
  
  forall(m in Men, w in Women)  
    wrank[m,w] < wrank[w,wife[m]] => mrank[w,husband[w]] < mrank[w,m];  
  forall(w in Women, m in Men)  
    mrank[w,m] < mrank[w,husband[m]] => wrank[m,wife[m]] < mrank[m,w];  
}
```



# Stable Marriages

```
solve {  
  forall(m in Men)  
    husband[wife[m]] = m;  
  forall(w in Women)  
    wife[husband[w]] = w;  
  
  forall(m in Men, w in Women)  
    wrank[m,w] < wrank[w,wife[m]] => mrnk[w,husband[w]] < mrnk[w,m];  
  forall(w in Women, m in Men)  
    mrnk[w,m] < mrnk[w,husband[m]] => wrank[m,wife[m]] < mrnk[m,w];  
}
```

**w prefers her husband over m**



# Stable Marriages

```
enum Men = {George,Hugh,Will,Clive};
enum Women = {Julia,Halle,Angelina,Keira};
int wrank[Men,Women];
int mrank[Women,Men];
...
var{Women} wife[Men];
var{Men} husband[Women];
solve {
    forall(m in Men)
        husband[wife[m]] = m;
    forall(w in Women)
        wife[husband[w]] = w;

    forall(m in Men, w in Women)
        wrank[m,w] < wrank[w,wife[m]] => mrank[w,husband[w]] < mrank[w,m];
    forall(w in Women, m in Men)
        mrank[w,m] < mrank[w,husband[m]] => wrank[m,wife[m]] < mrank[m,w];
}
```

# Stable Marriages

- ▶ Two interesting features
  - Element constraint
    - useful in many applications
  - Logical combination of constraints

# Stable Marriages

- ▶ Two interesting features
  - Element constraint
    - useful in many applications
  - Logical combination of constraints
- ▶ The element constraint
  - ability to index an array/matrix with a variable or an expression containing variables



# Stable Marriages

- ▶ Two interesting features
  - Element constraint
    - useful in many applications
  - Logical combination of constraints
- ▶ The element constraint
  - ability to index an array/matrix with a variable or an expression containing variables
- ▶ Logical combination of constraints
  - can be handled by reification for instance

# The Basic Element Constraint

- ▶  $x, y$ : variables
- ▶  $c$  is an array of integers
- ▶ constraint  $x = c[y]$

# The Basic Element Constraint

- ▶  $x, y$ : variables
- ▶  $c$  is an array of integers
- ▶ constraint  $x = c[y]$

**Y**

$\in \{0, 1, 2, 3, 4, 5\}$

**X**

$\in \{3, 4, 5\}$

Array  $c$

$i$	0	1	2	3	4	5
$c[i]$	3	4	5	5	4	3

# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$\boxed{Y} \in \{0, 1, 2, 3, 4, 5\}$$

$$\boxed{X} \in \{3, 4, 5\}$$

Array  $c$

$i$	0	1	2	3	4	5
$c[i]$	3	4	5	5	4	3

→  $X \neq 3$



# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$\boxed{Y} \in \{0, 1, 2, 3, 4, 5\}$$

$$\boxed{X} \in \{3, 4, 5\}$$

Array  $c$

$i$	0	1	2	3	4	5
$c[i]$	3	4	5	5	4	3

→  $X \neq 3$

# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

**Y**

$$\in \{0, 1, 2, 3, 4, 5\}$$

**X**

$$\in \{3, 4, 5\}$$

Array  $c$

i	0	1	2	3	4	5
c[i]	3	4	5	5	4	3

→  $X \neq 3$

# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	4	5
$c[i]$	<del>3</del>	4	5	5	4	3

→  $X \neq 3$



# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	4	<del>5</del>
$c[i]$	<del>3</del>	4	5	5	4	<del>3</del>

→  $X \neq 3$



# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	4	<del>5</del>
$c[i]$	<del>3</del>	4	5	5	4	<del>3</del>

→  $X \neq 3$

# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	4	<del>5</del>
$c[i]$	<del>3</del>	4	5	5	4	<del>3</del>

$$X \neq 3$$

$$Y \neq 4$$

# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	4	<del>5</del>
$c[i]$	<del>3</del>	4	5	5	4	<del>3</del>

→  $X \neq 3$

→  $Y \neq 4$



# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	<del>4</del>	<del>5</del>
$c[i]$	<del>3</del>	4	5	5	<del>4</del>	<del>3</del>

$$X \neq 3$$

$$Y \neq 4$$



# The Basic Element Constraint

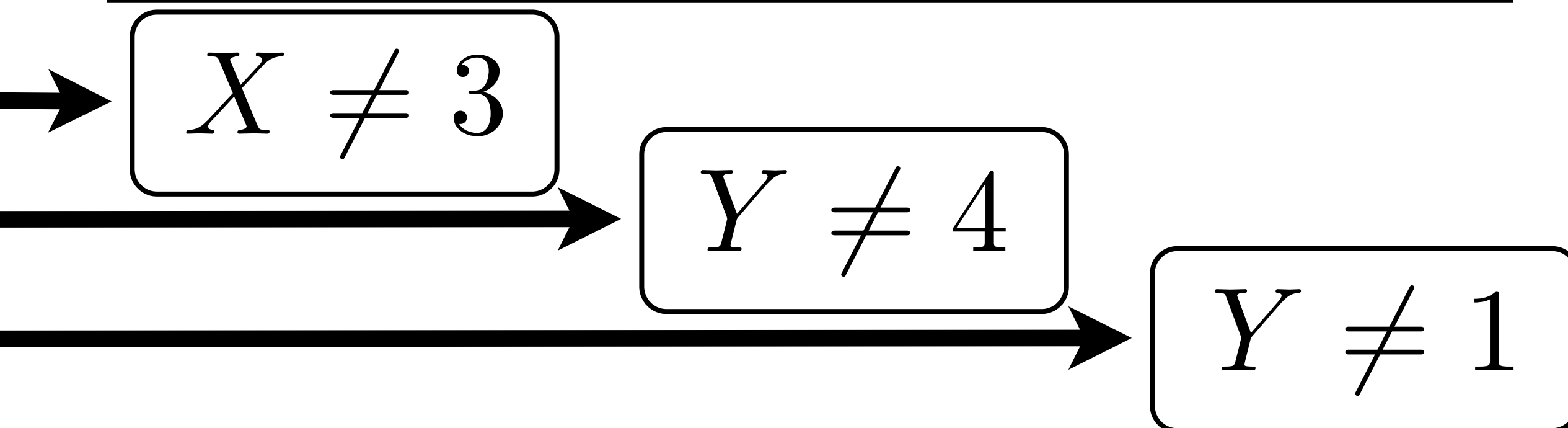
- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	<del>4</del>	<del>5</del>
$c[i]$	<del>3</del>	4	5	5	<del>4</del>	<del>3</del>



# The Basic Element Constraint

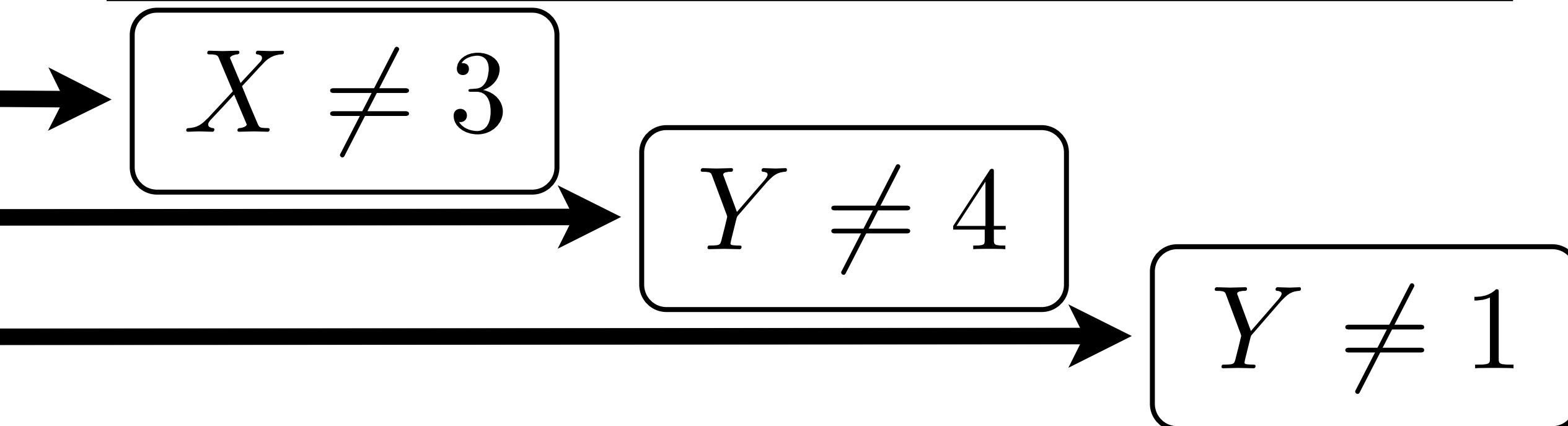
- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	1	2	3	<del>4</del>	<del>5</del>
$c[i]$	<del>3</del>	4	5	5	<del>4</del>	<del>3</del>



# The Basic Element Constraint

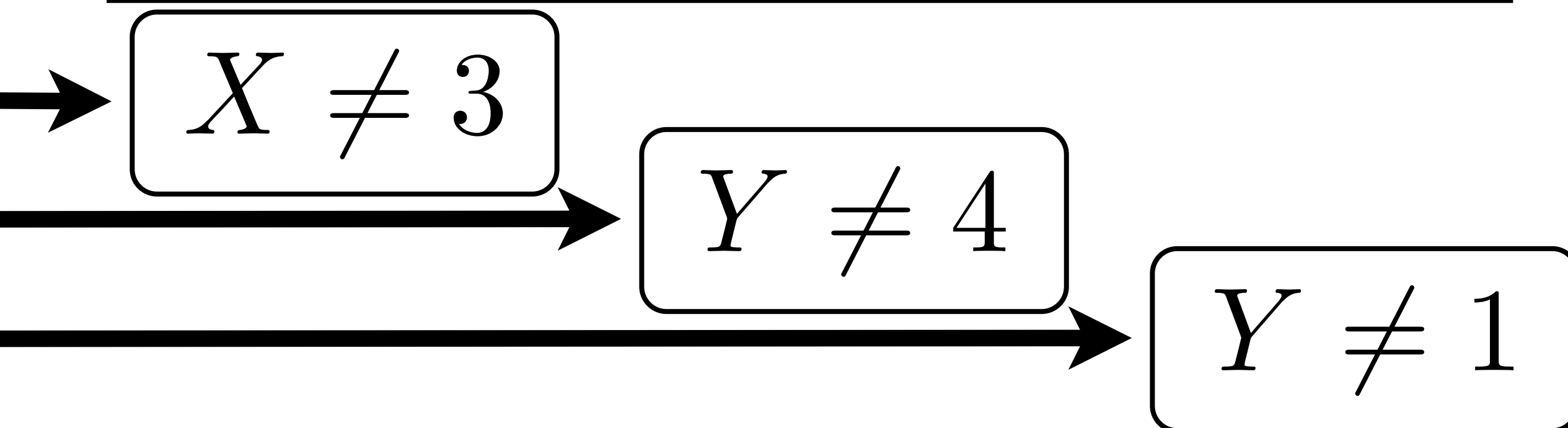
- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	<del>1</del>	2	3	<del>4</del>	<del>5</del>
$c[i]$	<del>3</del>	<del>4</del>	5	5	<del>4</del>	<del>3</del>





# The Basic Element Constraint

- $x, y$ : variables
- $c$  is an array of integers
- constraint  $x = c[y]$

$$Y \in \{0, 1, 2, 3, 4, 5\}$$

$$X \in \{3, 4, 5\}$$

Array  $c$

$i$	<del>0</del>	<del>1</del>	2	3	<del>4</del>	<del>5</del>
$c[i]$	<del>3</del>	<del>4</del>	5	5	<del>4</del>	<del>3</del>

