Discrete Optimization

Constraint Programming: Part VI

Goal of the Lecture

- Illustrating modeling techniques in constraint programming
  - redundant constraints

Redundant Constraints

- Motivation
  - **semantically redundant**
    - do not exclude any solution
  - **computationally significant**
    - reduce the search space

Redundant Constraints

- Motivation
  - **semantically redundant**
    - do not exclude any solution
  - **computationally significant**
    - reduce the search space
- How do I find redundant constraints?
  - they express properties of the solutions not captured by the model

Redundant Constraints

- Motivation
  - **semantically redundant**
    - do not exclude any solution
  - **computationally significant**
    - reduce the search space
- How do I find redundant constraints?
  - they express properties of the solutions not captured by the model
- Critical aspect of constraint programming!

Magic Series

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

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

- A series $S = (S_0,\ldots,S_n)$ 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 $S = (S_0,...,S_n)$ 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

```
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

```
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);
}
```

- Redundant constraints
  - can you find a property of the solution?

Magic Series

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

- The decision variables denote a number of occurrences
- The number of occurrences is bounded

Magic Series

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

	0	1	2	3	4
Occurrences	?	?	?	?	17

- The decision variables denote a number of occurrences
- The number of occurrences is bounded

Magic Series

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

	0	1	2	3	4
Occurrences	?	?	?	?	17

- The decision variables denote a number of occurrences

Magic Series

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

	0	1	2	3	4
Occurrences	?	?	?	?	17

- The decision variables denote a number of occurrences
- The number of occurrences is bounded

Magic Series

```
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);
   sum(i in D) series[i] = n;
}
```

Magic Series

```
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);
   sum(i in D) series[i] = n;
}
```

- Redundant constraints
  - The decision variables denote a number of occurrences
  - The number of occurrences is bounded

Magic Series

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

- What does “series[2] = 3” mean?

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

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

- What does “series[2] = 3” mean?
  - that there are three “2” in the array “series”

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

- What does “series[2] = 3” mean?
  - that there are three “2” in the array “series”

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

- What does “series[2] = 3” mean?
  - that there are three “2” in the array “series”

- sum(i in D) series[i] =
  sum(i in D) i * series[i]

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

- What does “series[2] = 3” mean?
  - that there are three “2” in the array “series”

- sum(i in D) series[i] =
  sum(i in D) i * series[i]

- Which constraint is stronger?

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

- What does “series[2] = 3” mean?
  - that there are three “2” in the array “series”

- sum(i in D) series[i] =
  sum(i in D) i * series[i]

- Which constraint is stronger?
  - sum(i in D) series[i] = n

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

- What does “series[2] = 3” mean?
  - that there are three “2” in the array “series”

- sum(i in D) series[i] =
  sum(i in D) i * series[i]

- Which constraint is stronger?
  - sum(i in D) series[i] = n
  - sum(i in D) i * series[i] = n

	0	1	2	3	4
Occurrences	?	?	?	?	?

Magic Series

```
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);
   sum(i in D) series[i] = n;
   sum(i in D) i * series[i] = n;
}
```

Magic Series

```
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);
series[1] + 2 series[2] + 3 series[3] + 4 series[4] = 5
```

- The redundant constraint implies
  - series[4] < 2
  - series[3] < 2
  - series[2] < 3
  - series[1] < 6

Magic Series

```
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] = (series[0]=3)+(series[1]=3);
series[4] = (series[0]=4)+(series[1]=4);
series[1] + 2 series[2] + 3 series[3] + 4 series[4] = 5
```

- The redundant constraint implies
  - series[4] < 2
  - series[3] < 2
  - series[2] < 3
  - series[1] < 6

Magic Series

- Assume that series[0] = 2

```
2         =    (series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0);
series[1] =    (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);
series[2] = 1 +(series[1]=2)+(series[2]=2);
series[3] =    (series[1]=3);
series[4] =    (series[1]=4);
series[1] + 2 series[2] + 3 series[3] + 4 series[4] = 5
```

- It follows that series[2] > 0
  - series[1] + 3 series[3] + 4 series[4] <= 3

Magic Series

- Assume that series[0] = 2

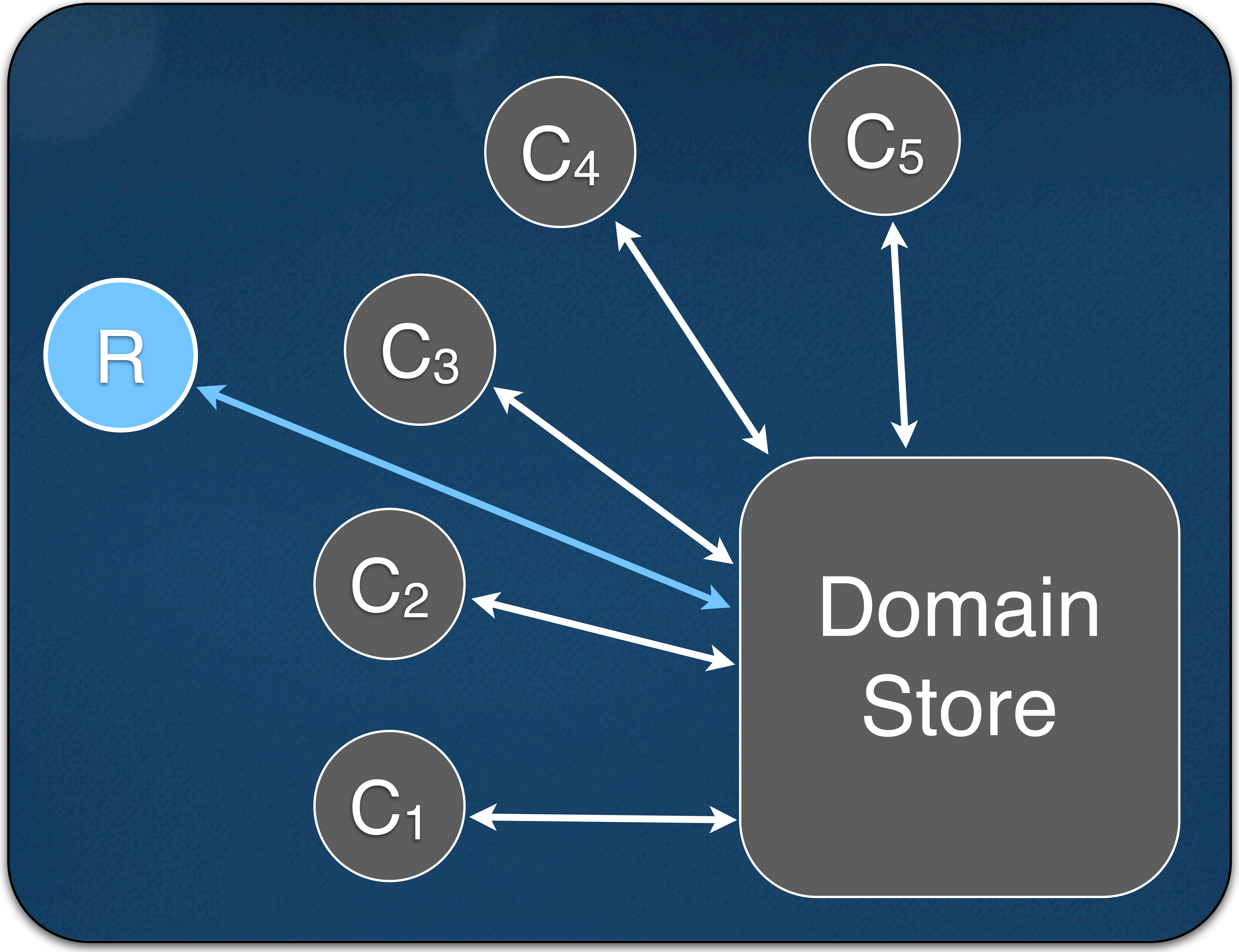
```
2          =    (series[1]=0)+(series[2]=0)+(series[3]=0)+ 1;
series[1]  =    (series[1]=1)+(series[2]=1)+(series[3]=1);
series[2]  = 1 +(series[1]=2)+(series[2]=2);
series[3]  =    (series[1]=3);
0          =    (series[1]=4);
series[1] + 2 series[2] + 3 series[3] = 5
```

- It follows that series[2] > 0
  - series[1] + 3 series[3] + 4 series[4] <= 3
  - series[4] <= 0
  - series[3] <= 1

Redundant Constraints

- First role
  - express properties of the solutions
  - boost the propagation of other constraints

Redundant Constraint



Constraint Store

Redundant Constraints

- First role
  - express properties of the solutions
  - boost the propagation of other constraints
- Second role
  - provide a more global view
  - combine existing constraints
  - improve communication

Market Split Problems

```
range C = ...;
range V = ...;
int w[C,V] = ...;
int rhs[C];
var{int} x[V] in 0..1;
solve {
   forall(c in C)
      sum(v in V) w[c,v] x[v] = rhs[c];
}
```

Market Split Problems

```
range C = ...;
range V = ...;
int w[C,V] = ...;
int rhs[C];
var{int} x[V] in 0..1;
solve {
   forall(c in C)
      sum(v in V) w[c,v] x[v] = rhs[c];
}
```

- Observe that
  - the constraints only communicate through the domains

Market Split Problems

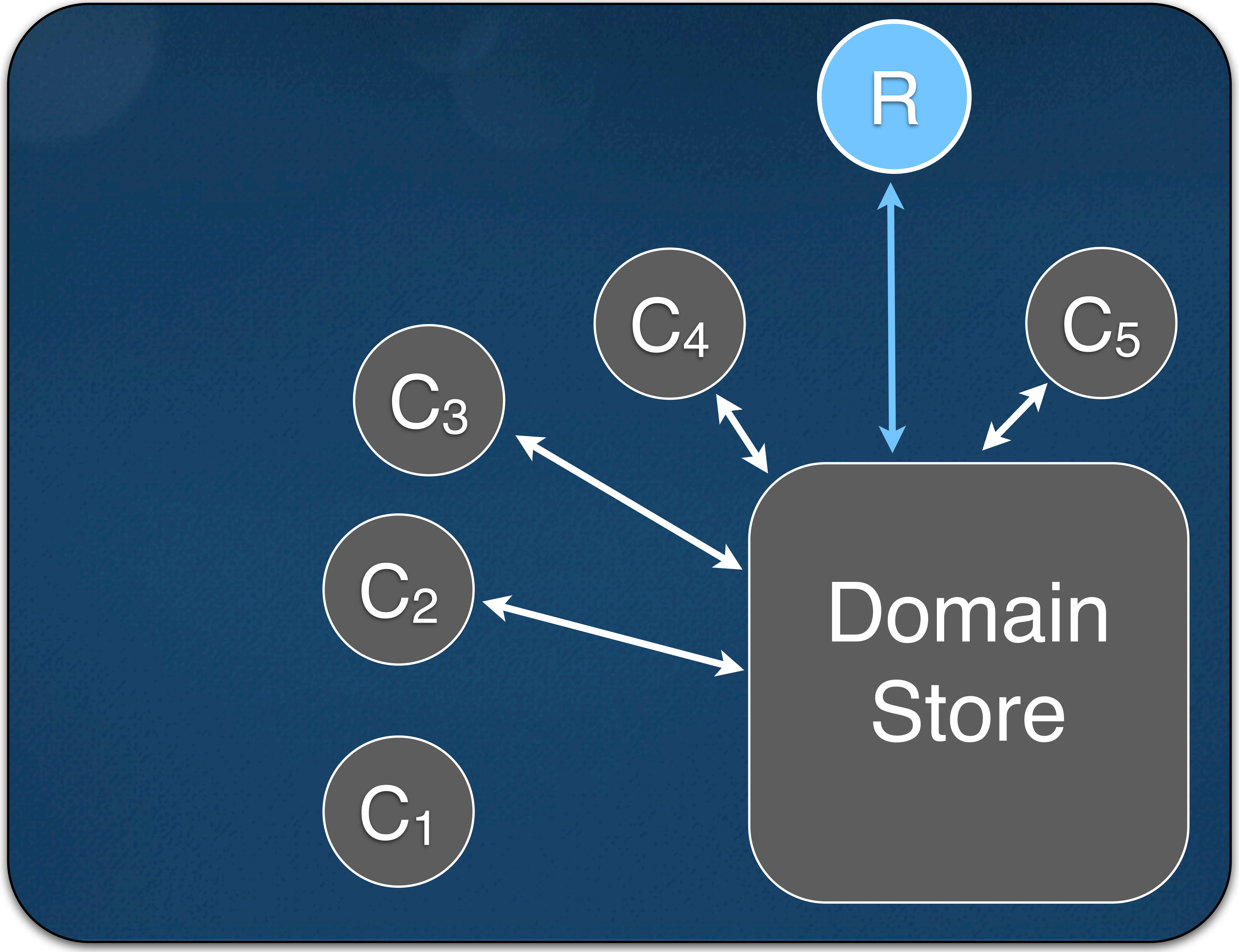
```
range C = ...;
range V = ...;
int w[C,V] = ...;
int rhs[C];
var{int} x[V] in 0..1;
solve {
   forall(c in C)
      sum(v in V) w[c,v] x[v] = rhs[c];
   sum(v in V) (sum(c in C) alpha^c * w[c,v]) * x[v] = sum(c in C) alpha^c * rhs[c];
}
```

Market Split Problems

```
range C = ...;
range V = ...;
int w[C,V] = ...;
int rhs[C];
var{int} x[V] in 0..1;
solve {
   forall(c in C)
      sum(v in V) w[c,v] x[v] = rhs[c];
   sum(v in V) (sum(c in C) alpha^c * w[c,v]) * x[v] = sum(c in C) alpha^c * rhs[c];
}
```

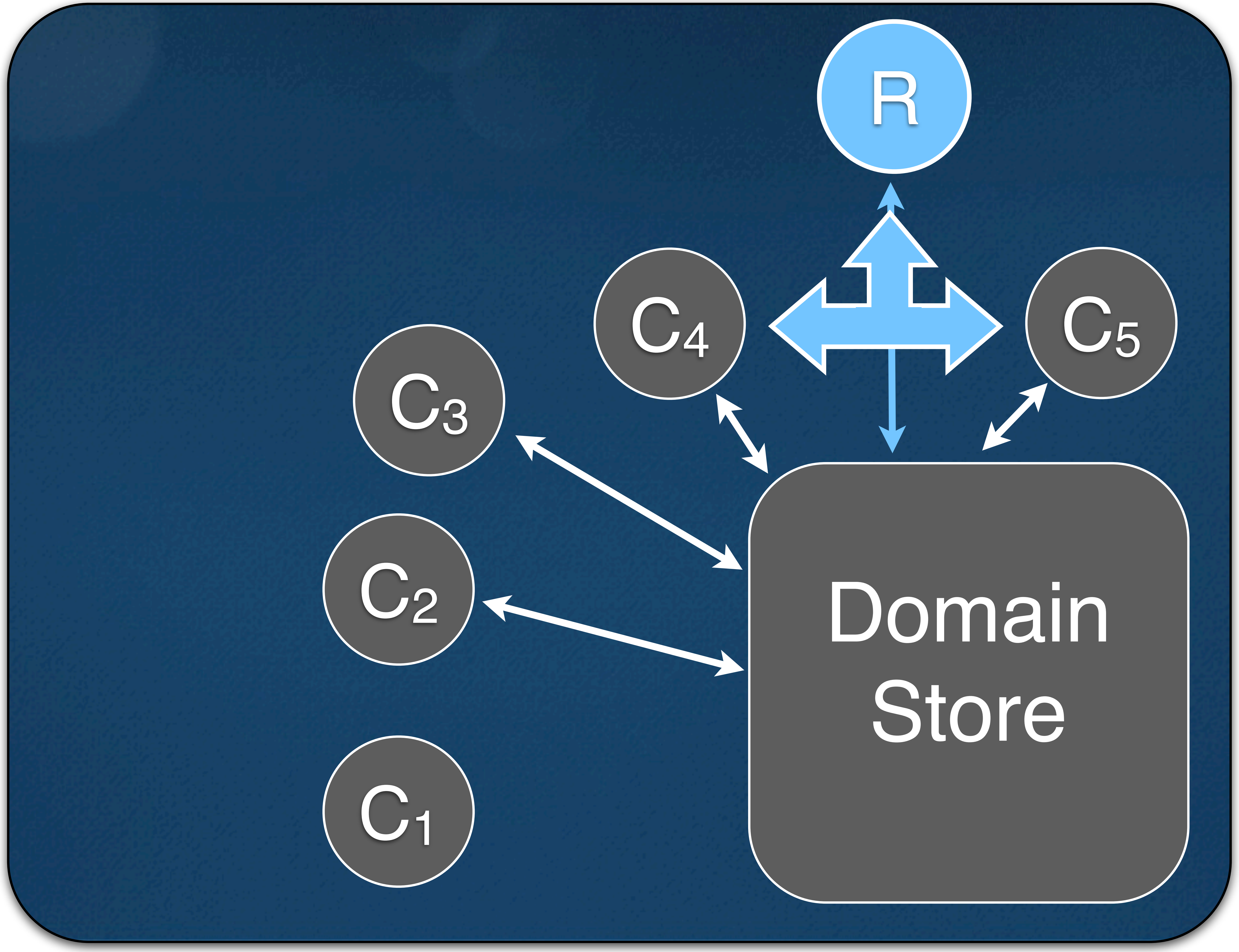
- Surrogate constraints
  - combination of existing constraints

Surrogate Constraint



Constraint Store

Surrogate Constraint



Constraint Store