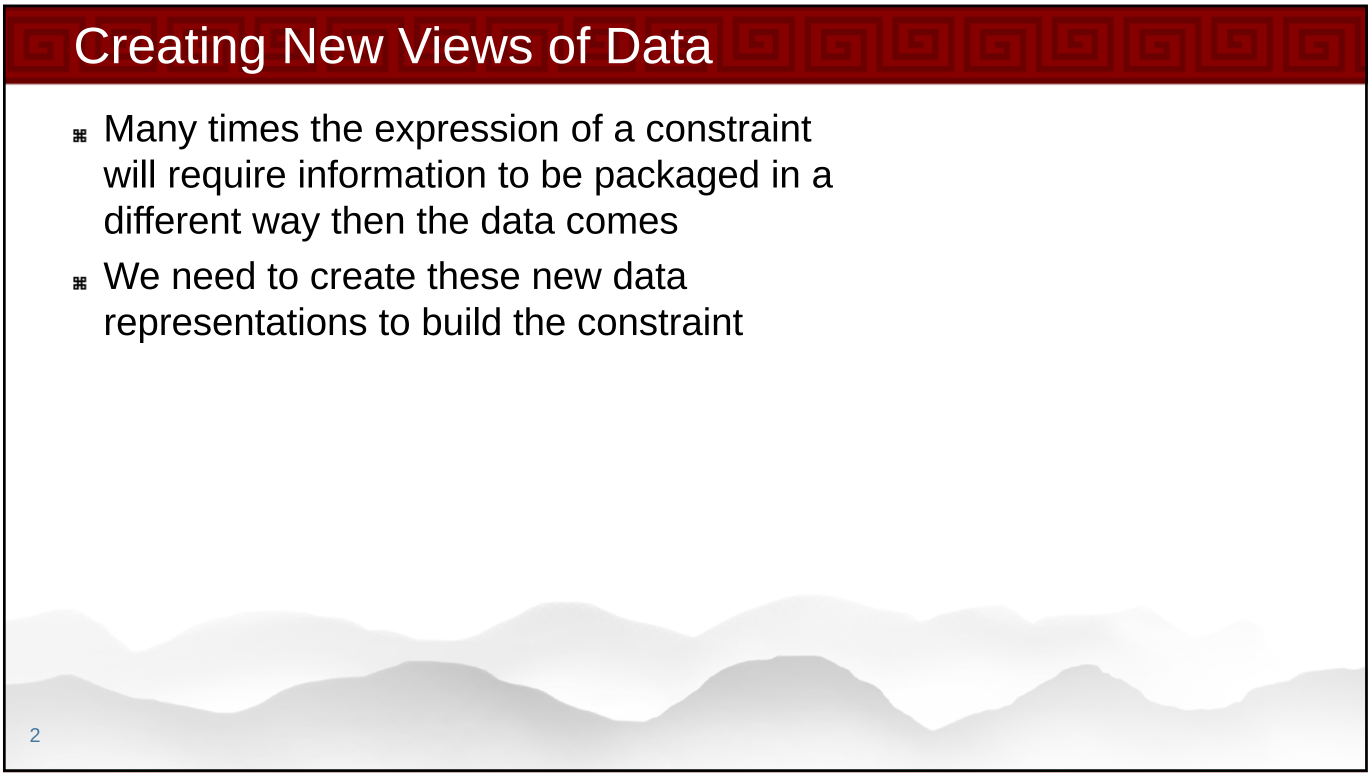
Transforming Data

Jimmy Lee & Peter Stuckey



Creating New Views of Data

- Many times the expression of a constraint will require information to be packaged in a different way then the data comes
- We need to create these new data representations to build the constraint

StableRoommates Problem

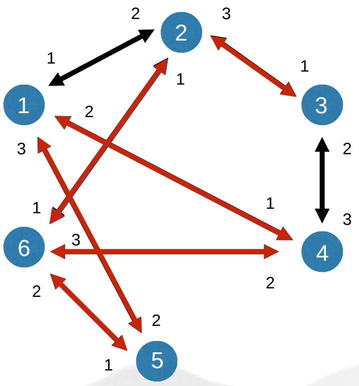
- Given *n* agents we need to pair them up. Each agent has a list of preferred agents and can only be paired with one of them. The pairing must be stable:
  - if agent *i* is paired with agent *j* but prefers agent *k*, then agent *k* must be paired with an agent *l* they prefer to *i*
  - otherwise *i* and *k* will take a room together leaving their roommates.

StableRoommates Example

- Given 6 people and rankings
  - 1 prefers 2,4,5 in order
  - 2 prefers 6,1,3
  - 3 prefers 2,4
  - 4 prefers 1,6,3
  - 5 prefers 6,1
  - 6 prefers 2,5,4
- Generate a stable pairing!

StableRoommates

- Initial matching
  - Unstable Pair
  - Better Matching
  - Unstable Pair
  - Better Matching

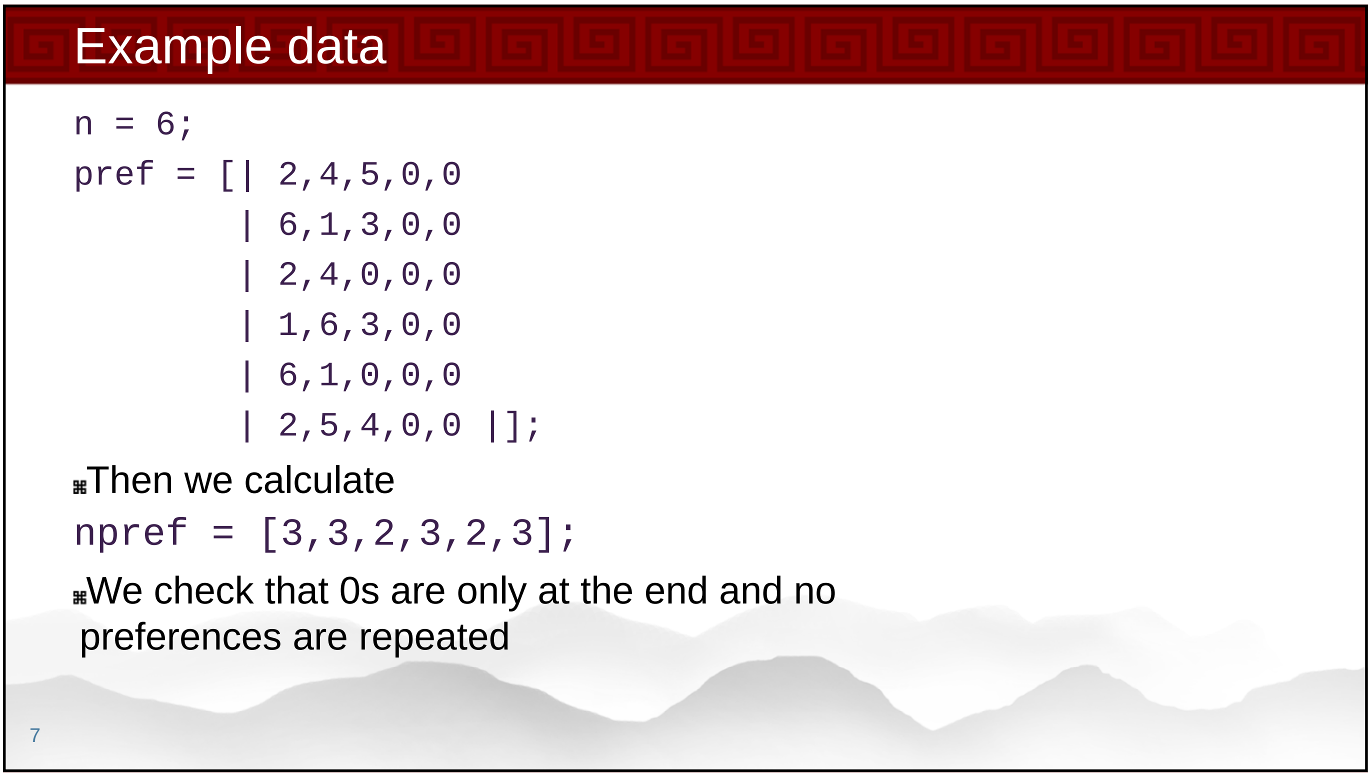


StableRoommates Data

```
int: n; % number of agents
set of int: AGENT = 1..n;
```

- Preference lists: 0 means no agent

```
set of int: AGENT0 = 0..n;
array[AGENT,1..n-1] of AGENT0: pref;
array[AGENT] of int: npref =
  [ sum(i in 1..n-1)(bool2int(agent[a,i] >
0))
  | a in AGENT ];
forall(a in AGENT)(assert(
  forall(i,j in 1..npref[a] where i < j)
        (pref[a,i] != pref[a,j]) /\
  forall(i in 1..npref[a])(pref[a,j] > 0),
  “Agent \(a)’s pref data is wrong”));
```



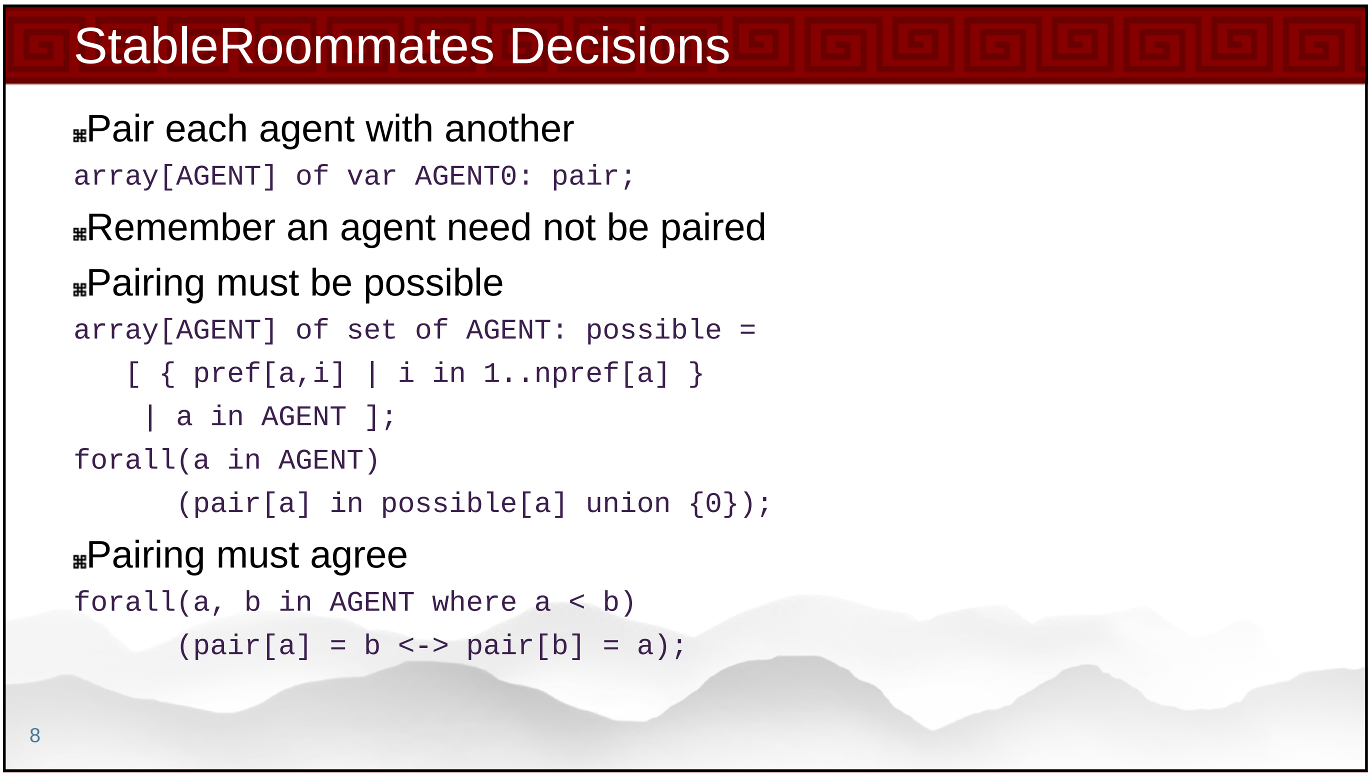
Example data

```
n = 6;
pref = [| 2,4,5,0,0
        | 6,1,3,0,0
        | 2,4,0,0,0
        | 1,6,3,0,0
        | 6,1,0,0,0
        | 2,5,4,0,0 |];
```

- Then we calculate

```
npref = [3,3,2,3,2,3];
```

- We check that 0s are only at the end and no preferences are repeated



StableRoommates Decisions

- Pair each agent with another

```
array[AGENT] of var AGENT0: pair;
```

- Remember an agent need not be paired
- Pairing must be possible

```
array[AGENT] of set of AGENT: possible =
   [ { pref[a,i] | i in 1..npref[a] }
    | a in AGENT ];
forall(a in AGENT)
      (pair[a] in possible[a] union {0});
```

- Pairing must agree

```
forall(a, b in AGENT where a < b)
      (pair[a] = b <-> pair[b] = a);
```

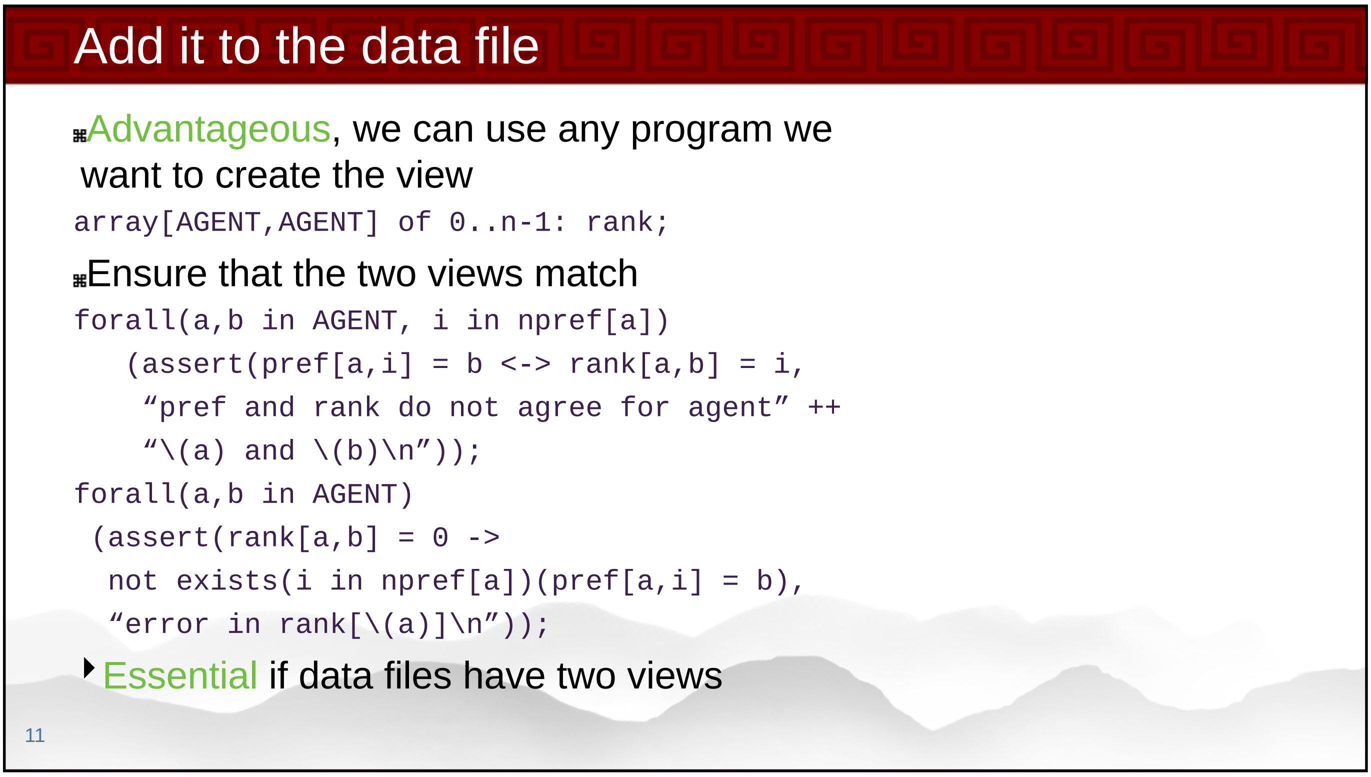
StableRoommates Constraints

- How can we express stability
- We need to know how each person ranks each other?
  - rank[a,b] = i if pref[a,i] = b
- But what if a does not rank b
  - rank[a,b] = 0 if pref[a,i] != b for all i
- How do we create the rank array?
- Three approaches
  - Add it to the data file
  - Use constraints
  - Use complex comprehensions

Add it to the data file

```
array[AGENT,AGENT] of 0..n-1: rank;

rank = [| 0,1,0,2,3,0        % 2,4,5
        | 2,0,3,0,0,1        % 6,1,3
        | 0,1,0,2,0,0        % 2,4
        | 1,0,3,0,0,2        % 1,6,3
        | 2,0,0,0,0,1        % 6,1
        | 0,1,0,3,2,0 |];    % 2,5,4
```



Add it to the data file

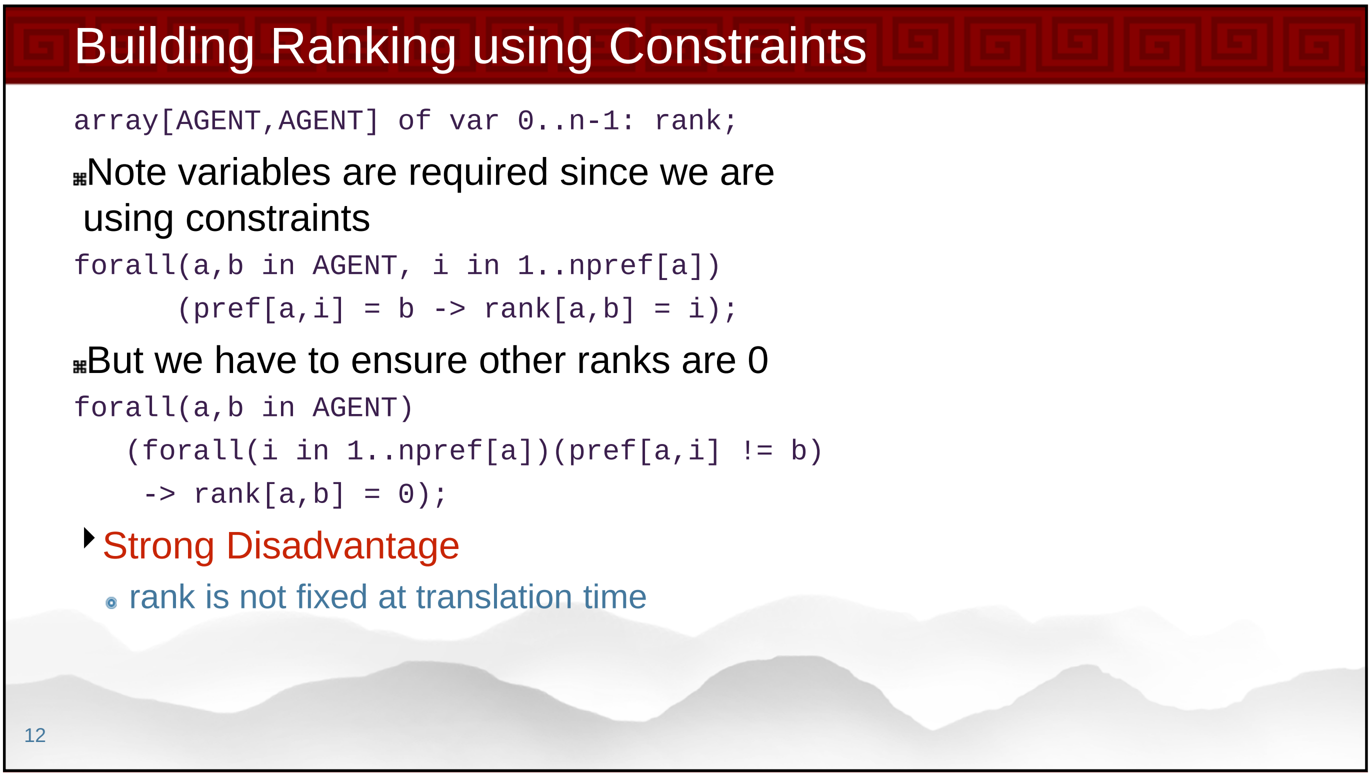
- Advantageous, we can use any program we want to create the view

```
array[AGENT,AGENT] of 0..n-1: rank;
```

- Ensure that the two views match

```
forall(a,b in AGENT, i in npref[a])
   (assert(pref[a,i] = b <-> rank[a,b] = i,
    “pref and rank do not agree for agent” ++
    “\(a) and \(b)\n”));
forall(a,b in AGENT)
 (assert(rank[a,b] = 0 ->
  not exists(i in npref[a])(pref[a,i] = b),
  “error in rank[\(a)]\n”));
```

  - Essential if data files have two views



Building Ranking using Constraints

```
array[AGENT,AGENT] of var 0..n-1: rank;
```

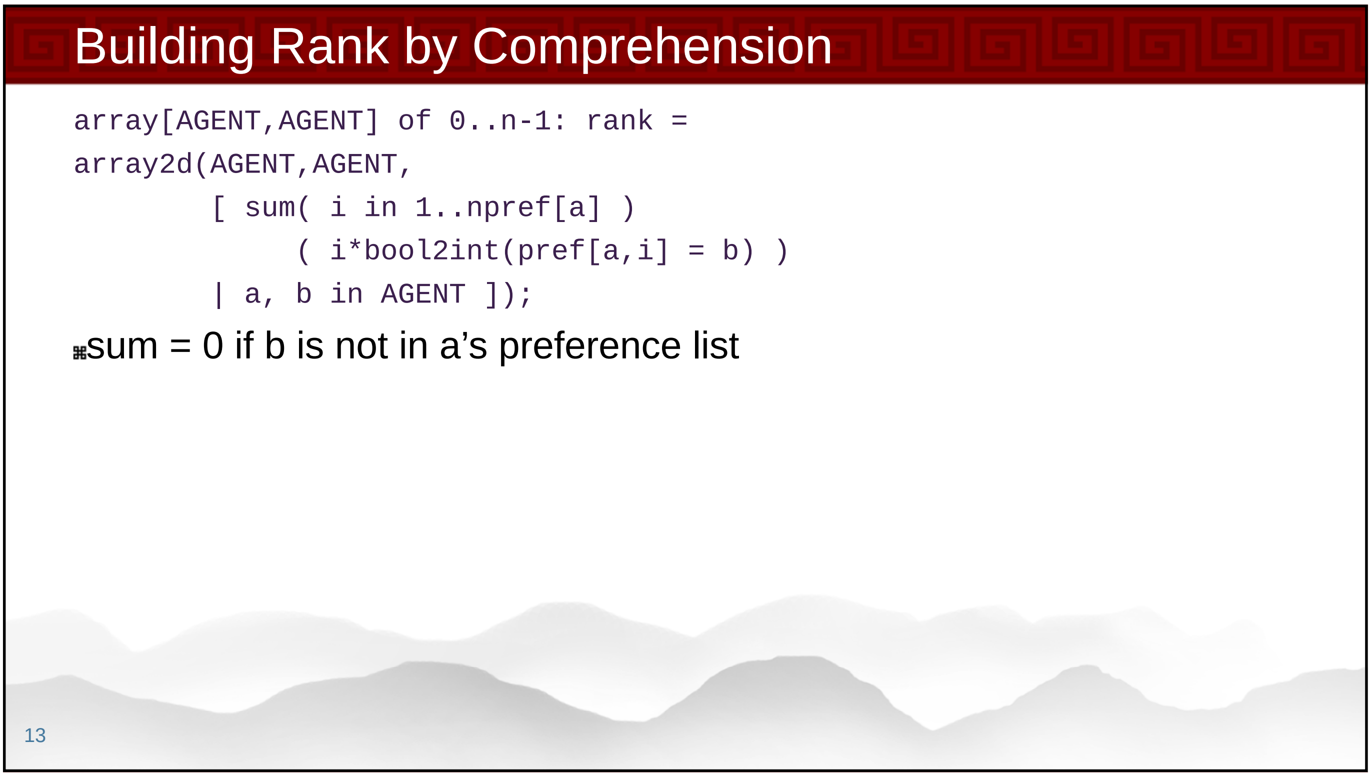
- Note variables are required since we are using constraints

```
forall(a,b in AGENT, i in 1..npref[a])
      (pref[a,i] = b -> rank[a,b] = i);
```

- But we have to ensure other ranks are 0

```
forall(a,b in AGENT)
   (forall(i in 1..npref[a])(pref[a,i] != b)
    -> rank[a,b] = 0);
```

  - Strong Disadvantage
    - rank is not fixed at translation time

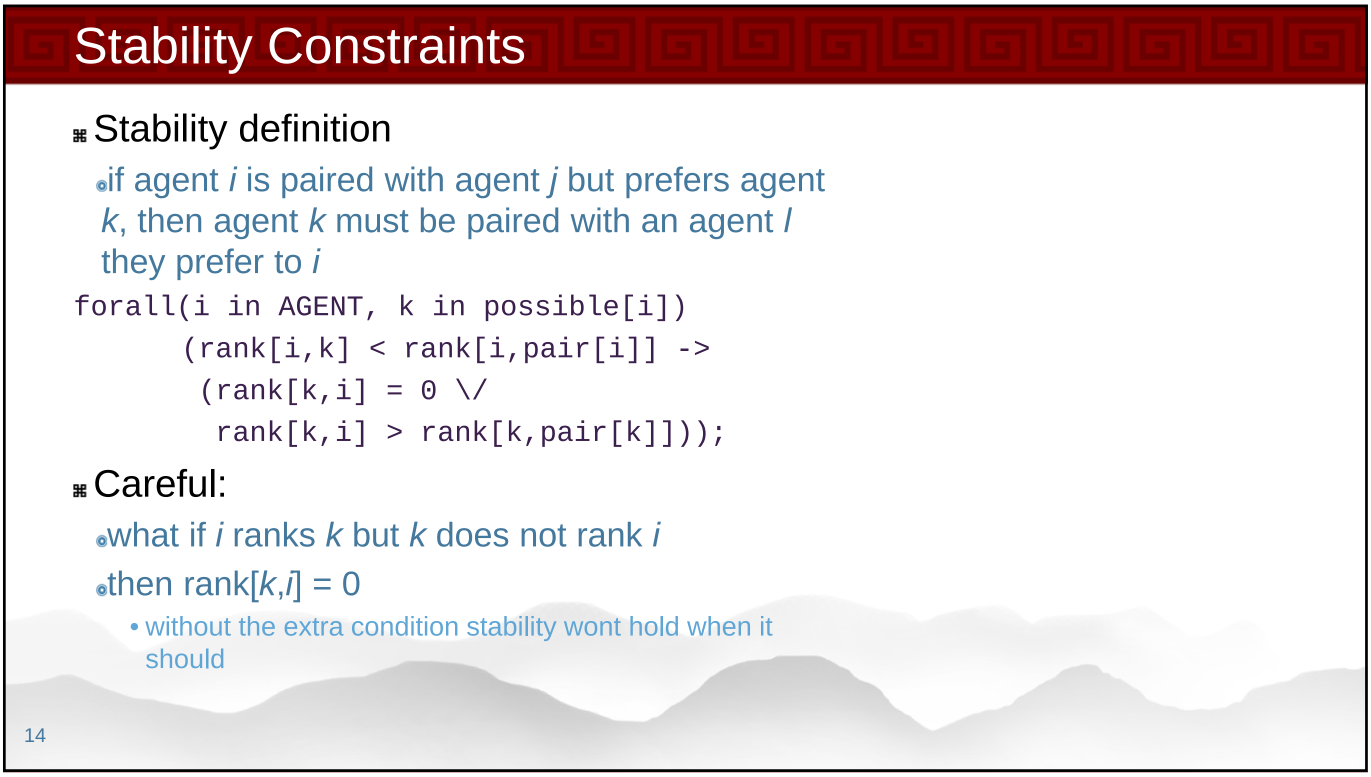


Building Rank by Comprehension

```
array[AGENT,AGENT] of 0..n-1: rank =
array2d(AGENT,AGENT,
        [ sum( i in 1..npref[a] )
             ( i*bool2int(pref[a,i] = b) )
        | a, b in AGENT ]);
```

- sum = 0 if b is not in a’s preference list

13



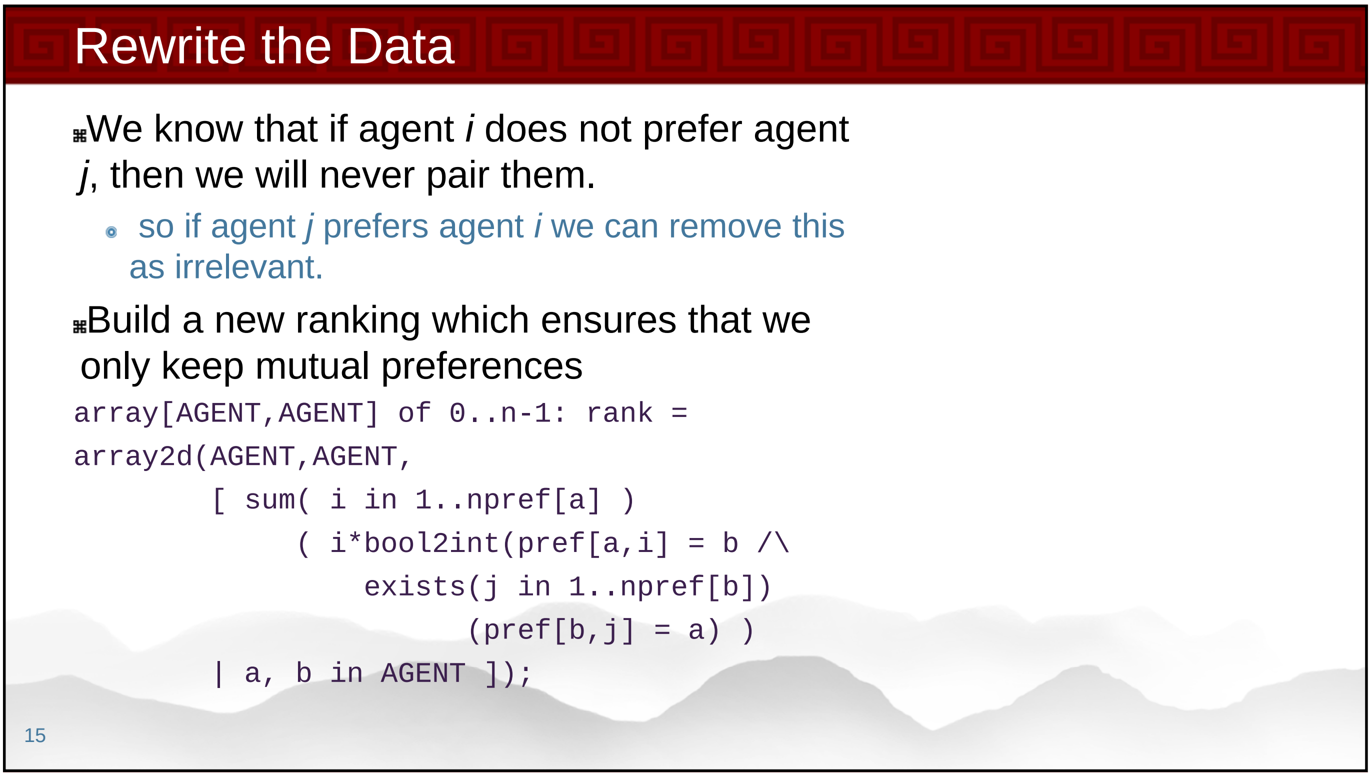
Stability Constraints

- Stability definition
  - if agent *i* is paired with agent *j* but prefers agent *k*, then agent *k* must be paired with an agent *l* they prefer to *i*

```
forall(i in AGENT, k in possible[i])
      (rank[i,k] < rank[i,pair[i]] ->
       (rank[k,i] = 0 \/
        rank[k,i] > rank[k,pair[k]]));
```

- Careful:
  - what if *i* ranks *k* but *k* does not rank *i*
  - then rank[*k*,*i*] = 0
    - without the extra condition stability wont hold when it should

14

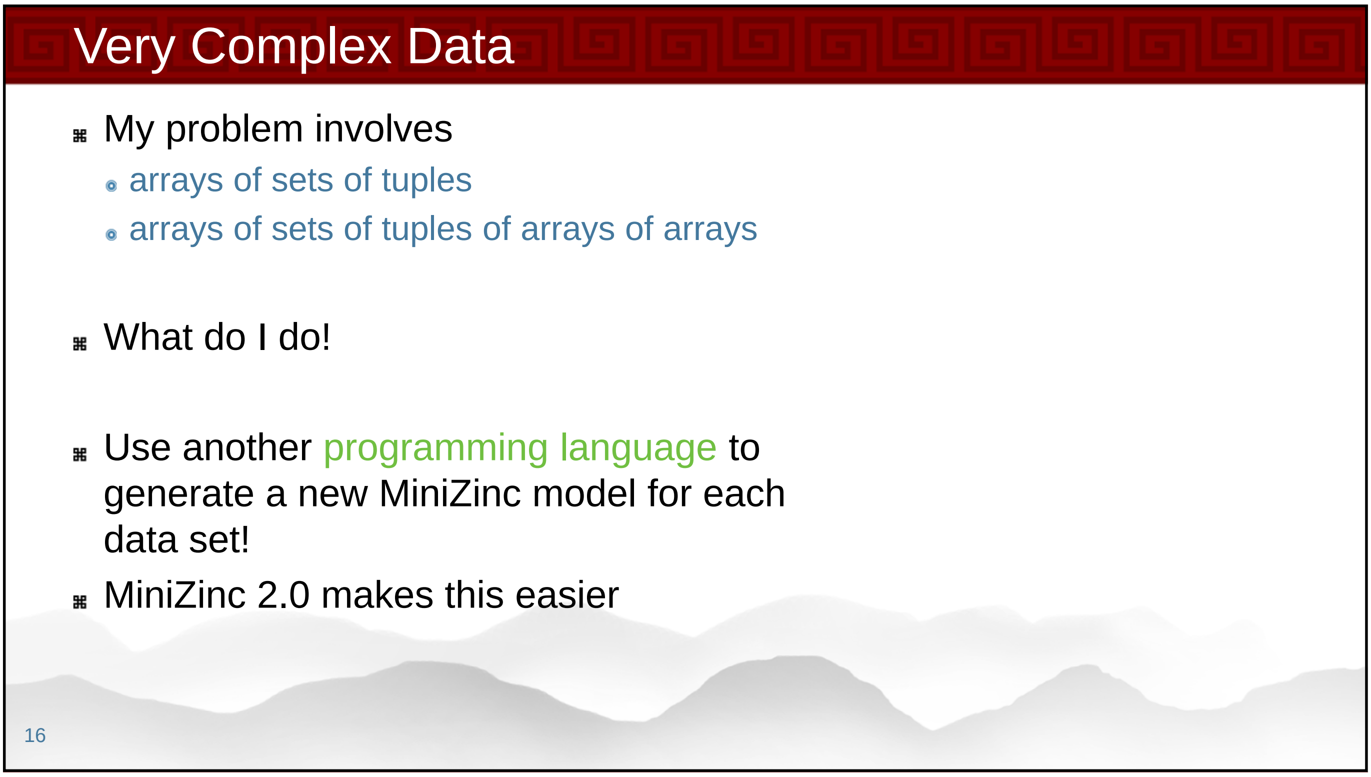


Rewrite the Data

- We know that if agent *i* does not prefer agent *j*, then we will never pair them.
  - so if agent *j* prefers agent *i* we can remove this as irrelevant.
- Build a new ranking which ensures that we only keep mutual preferences

```
array[AGENT,AGENT] of 0..n-1: rank =
array2d(AGENT,AGENT,
        [ sum( i in 1..npref[a] )
             ( i*bool2int(pref[a,i] = b /\
                 exists(j in 1..npref[b])
                       (pref[b,j] = a) )
        | a, b in AGENT ]);
```

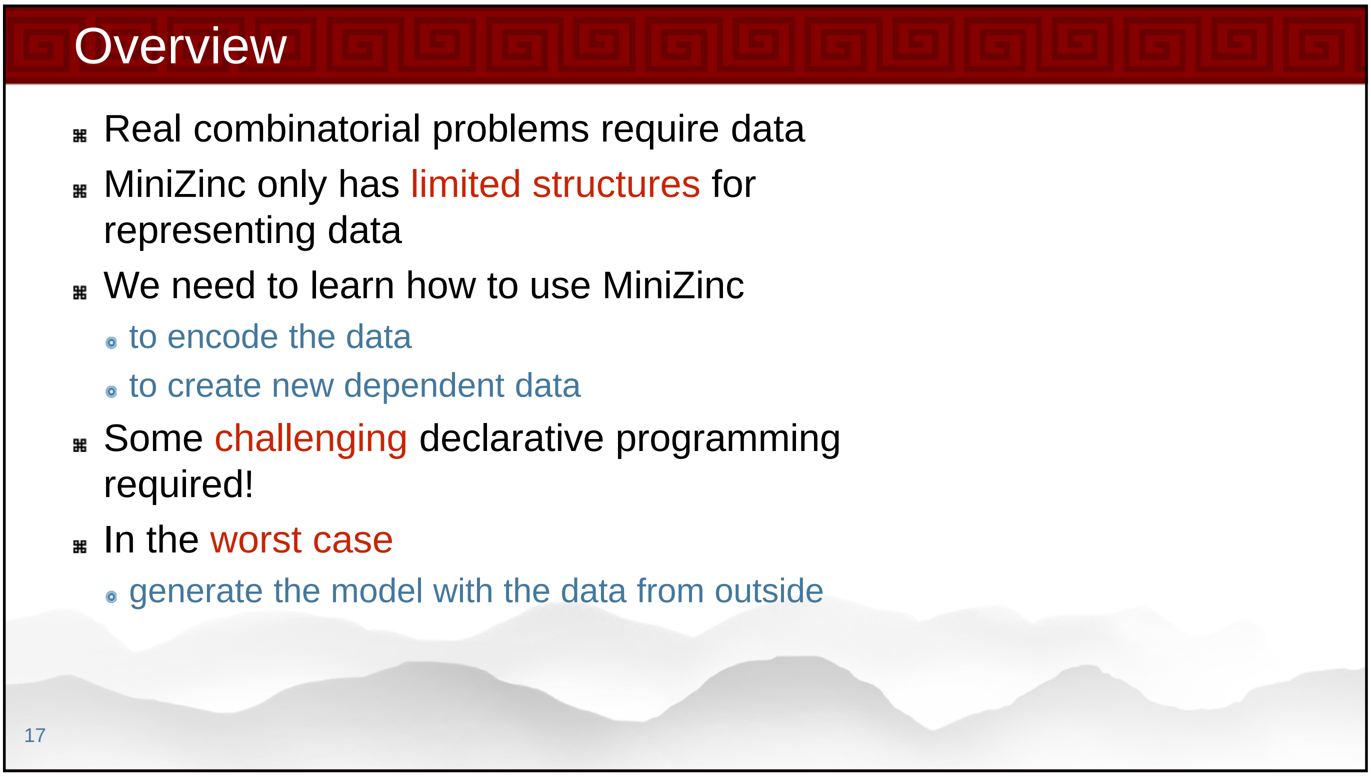
15



Very Complex Data

- My problem involves
  - arrays of sets of tuples
  - arrays of sets of tuples of arrays of arrays
- What do I do!
- Use another programming language to generate a new MiniZinc model for each data set!
- MiniZinc 2.0 makes this easier

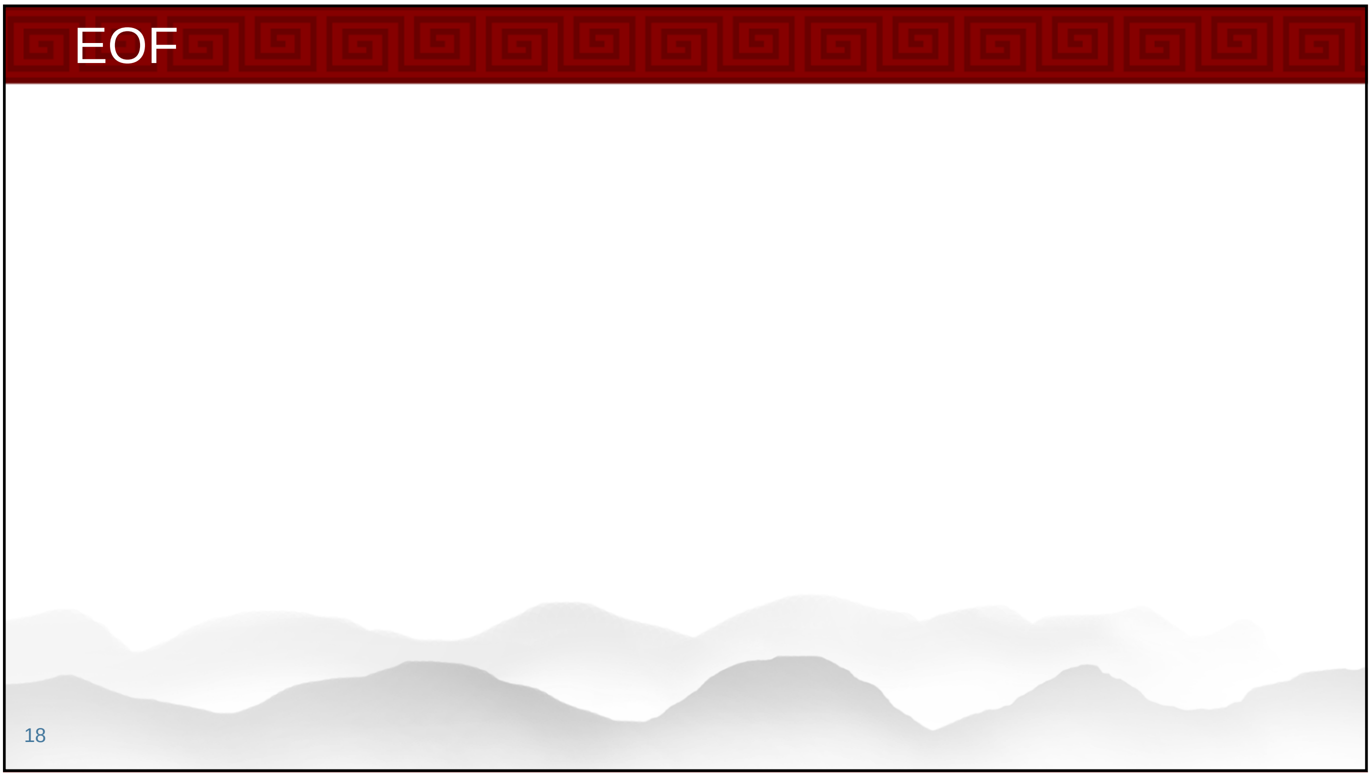
16



Overview

- Real combinatorial problems require data
- MiniZinc only has limited structures for representing data
- We need to learn how to use MiniZinc
  - to encode the data
  - to create new dependent data
- Some challenging declarative programming required!
- In the worst case
  - generate the model with the data from outside

17



EOF

18