MoreLogicProblems.mws

More example logic problems

Carl Devore <devore@math.udel.edu>

31 March 2001

```
> restart;
```

This application uses a Maple package called **LP** , contained in the file "LogicProblem.mpl", which is read by the next line. Make sure that file is in the same directory as this worksheet before executing.

```
> read "LogicProblem.mpl";
```

A simple problem showing the use of the "Less" constraint.

This problem is adapted from Varda Reisner, "Logic Problem 1: Delivery Please" in *Dell Math Puzzles and Logic Problems* (No. 49, November 2000).

For each day, Tuesday to Friday, of last week, Wendy eats dinner at the local diner. Each day, she orders a meat (one was pork), a soup, and a vegetable. Determine the meat, soup, and vegetable for each day.

1. The beef was chosen earlier in the week than the French onion soup, which was chosen earlier than the carrots.

2. The chicken was chosen earlier in the week than the cream of broccoli soup, but later in the week than the spinach.

3. She didn't have fish on Friday.

4. The cream of mushroom soup was not chosen on Tuesday or Friday.

5. The black bean soup was served with the cauliflower.

6. The chicken was not served with the cabbage.

```
> V:= ['day', 'meat', 'soup', 'veg'];
```

$$V := [day, meat, soup, veg]$$

```
> day:= [Tue, Wed, Thu, Fri];
```

$$day := [Tue, Wed, Thu, Fri]$$

```
> meat:= [pork, beef, chicken, fish];
```

$$meat := [pork, beef, chicken, fish]$$

```
> soup:= [onion, broc, mush, bean];
```

$$soup := [onion, broc, mush, bean]$$

```
> veg:= [carrots, spinach, cauliflower, cabbage];
```

$$veg := [carrots, spinach, cauliflower, cabbage]$$

```
> st:= time():
```

```
> lp2:= LogicProblem(V):
```

Setting infolevel[TCdisplay] to 6 will print the appropriate submatrix everytime an "equality" is concluded. Setting to 7 will also print it everytime an "inequality" is concluded. Setting it to 8 will also print it everytime an entry is "ruled out" because an equality has been placed in its row or column. All of this is usually way too much information, but it is sometimes useful for debugging.

Setting infolevel[TC] to 5 and infolevel[Constraints] to 4 will show the entry and arguements to each recursive procedure. The "Level" numbers printed are Maple's internal representation of the recursion depth. This can help you track exactly where in the solution process a particular conclusion is reached.

```
> infolevel['TCdisplay']:= 6: infolevel['TC']:= 5: infolevel['Constraints']:= 4:
```

```
> lp2:-CollectStats:= true:
```

```
> lp2:-Satisfy([Less([beef,onion,carrots], day) #Clue 1
,Less([spinach,chicken,broc], day) #Clue 2
,Fri<>fish #3
,mush<>Tue, mush<>Fri #4
,bean=cauliflower #5
,chicken<>cabbage #6
]);

Know:   Processing   [-2, Fri <> fish]

ParseAssertionTypes:   false, [-2, Fri <> fish]

Know/<>:   Fri, fish
```

```
&<!=>:   Conclusion   Fri <> fish   Level = 131

Elim:   1   2   Fri   fish   Level = 157

Elim:   2   1   fish   Fri   Level = 157

TransferX:   1   2   Fri   fish   Level = 152

TransferX:   2   1   fish   Fri   Level = 152

Know:   Processing   [-2, mush <> Tue]

ParseAssertionTypes:   false, [-2, mush <> Tue]

Know/<>:   mush, Tue

&<!=>:   Conclusion   mush <> Tue   Level = 131

Elim:   3   1   mush   Tue   Level = 157

Elim:   1   3   Tue   mush   Level = 157

TransferX:   3   1   mush   Tue   Level = 152

TransferX:   1   3   Tue   mush   Level = 152

Know:   Processing   [-2, mush <> Fri]

ParseAssertionTypes:   false, [-2, mush <> Fri]

Know/<>:   mush, Fri

&<!=>:   Conclusion   mush <> Fri   Level = 131

Elim:   3   1   mush   Fri   Level = 157

Elim:   1   3   Fri   mush   Level = 157

TransferX:   3   1   mush   Fri   Level = 152

TransferX:   1   3   Fri   mush   Level = 152

Know:   Processing   [-2, chicken <> cabbage]
```

```
ParseAssertionTypes:   false, [-2, chicken <> cabbage]

Know/<>:   chicken, cabbage

&<!=>:   Conclusion   chicken <> cabbage   Level = 131

Elim:   2   4   chicken   cabbage   Level = 157

Elim:   4   2   cabbage   chicken   Level = 157

TransferX:   2   4   chicken   cabbage   Level = 152

TransferX:   4   2   cabbage   chicken   Level = 152

Know:   Processing   [-3, bean = cauliflower]

ParseAssertionTypes:   false, [-3, bean = cauliflower]

Know/=:   bean, cauliflower

&<=>:   bean = cauliflower   Level = 131

TransClosure:   Conclusion   bean = cauliflower

RuleOut:   3   4   bean   cauliflower   Level = 178

RuleOut:   4   3   cauliflower   bean   Level = 178

Elim:   3   4   onion   cauliflower   Level = 183

Elim:   3   4   broc   cauliflower   Level = 183

Elim:   3   4   mush   cauliflower   Level = 183

Elim:   4   3   carrots   bean   Level = 183

Elim:   4   3   spinach   bean   Level = 183

Elim:   4   3   cabbage   bean   Level = 183

TransClosure:
```

	onion	broc	mush	bean
carrots	_	_	_	X
spinach	_	_	_	X
cauliflower	X	X	X	O
cabbage	_	_	_	X

```
Pivot:   3   4   bean   cauliflower   Level = 185

Pivot:   4   3   cauliflower   bean   Level = 185

Know:   Processing   [-6, Less([beef, onion, carrots],1)]

ParseAssertionTypes:   false, [-6, Less([beef, onion, carrots],1)]

Know/Less:   [beef, onion, carrots], 1

Less/OnePair:   beef, onion, 1

&<!=>:   Conclusion   beef <> onion   Level = 160

Elim:   2   3   beef   onion   Level = 186

Elim:   3   2   onion   beef   Level = 186

TransferX:   2   3   beef   onion   Level = 181

TransferX:   3   2   onion   beef   Level = 181

&<!=>:   Conclusion   onion <> Tue   Level = 160

Elim:   3   1   onion   Tue   Level = 186

Elim:   1   3   Tue   onion   Level = 186

TransferX:   3   1   onion   Tue   Level = 181

TransferX:   1   3   Tue   onion   Level = 181

&<!=>:   Conclusion   beef <> Fri   Level = 160

Elim:   2   1   beef   Fri   Level = 186
```

```
Elim:   1   2   Fri   beef   Level = 186

TransferX:   2   1   beef   Fri   Level = 181

TransferX:   1   2   Fri   beef   Level = 181

Less/RuleOutImpossible:   1   beef   onion

Less/OnePair:   beef, carrots, 1

&<!=>:   Conclusion   beef <> carrots   Level = 160

Elim:   2   4   beef   carrots   Level = 186

Elim:   4   2   carrots   beef   Level = 186

TransferX:   2   4   beef   carrots   Level = 181

TransferX:   4   2   carrots   beef   Level = 181

&<!=>:   Conclusion   carrots <> Tue   Level = 160

Elim:   4   1   carrots   Tue   Level = 186

Elim:   1   4   Tue   carrots   Level = 186

TransferX:   4   1   carrots   Tue   Level = 181

TransferX:   1   4   Tue   carrots   Level = 181

Less/RuleOutImpossible:   1   beef   carrots

Less/OnePair:   onion, carrots, 1

&<!=>:   Conclusion   onion <> carrots   Level = 160

Elim:   3   4   onion   carrots   Level = 186

Elim:   4   3   carrots   onion   Level = 186

TransferX:   3   4   onion   carrots   Level = 181

TransferX:   4   3   carrots   onion   Level = 181
```

```
&<!=>:   Conclusion   onion <> Fri   Level = 160

Elim:   3   1   onion   Fri   Level = 186

Elim:   1   3   Fri   onion   Level = 186

TransferX:   3   1   onion   Fri   Level = 181

TransferX:   1   3   Fri   onion   Level = 181

Less/RuleOutImpossible:   1   onion   carrots

&<!=>:   Conclusion   carrots <> Wed   Level = 188

Elim:   4   1   carrots   Wed   Level = 214

Elim:   1   4   Wed   carrots   Level = 214

TransferX:   4   1   carrots   Wed   Level = 209

TransferX:   1   4   Wed   carrots   Level = 209

Know:   Processing   [-6, Less([spinach, chicken, broc],1)]

ParseAssertionTypes:   false, [-6, Less([spinach, chicken, broc],1)]

Know/Less:   [spinach, chicken, broc], 1

Less/OnePair:   spinach, chicken, 1

&<!=>:   Conclusion   spinach <> chicken   Level = 160

Elim:   4   2   spinach   chicken   Level = 186

Elim:   2   4   chicken   spinach   Level = 186

TransferX:   4   2   spinach   chicken   Level = 181

TransferX:   2   4   chicken   spinach   Level = 181

&<!=>:   Conclusion   chicken <> Tue   Level = 160

Elim:   2   1   chicken   Tue   Level = 186
```

```
Elim:   1   2   Tue   chicken   Level = 186

TransferX:   2   1   chicken   Tue   Level = 181

TransferX:   1   2   Tue   chicken   Level = 181

&<!=>:   Conclusion   spinach <> Fri   Level = 160

Elim:   4   1   spinach   Fri   Level = 186

Elim:   1   4   Fri   spinach   Level = 186

TransferX:   4   1   spinach   Fri   Level = 181

TransferX:   1   4   Fri   spinach   Level = 181

Less/RuleOutImpossible:   1   spinach   chicken

Less/OnePair:   spinach, broc, 1

&<!=>:   Conclusion   spinach <> broc   Level = 160

Elim:   4   3   spinach   broc   Level = 186

Elim:   3   4   broc   spinach   Level = 186

TransferX:   4   3   spinach   broc   Level = 181

TransferX:   3   4   broc   spinach   Level = 181

&<!=>:   Conclusion   broc <> Tue   Level = 160

Elim:   3   1   broc   Tue   Level = 186

Elim:   1   3   Tue   broc   Level = 186

TransClosure:   Conclusion   Tue = bean

RuleOut:   1   3   Tue   bean   Level = 207

RuleOut:   3   1   bean   Tue   Level = 207

Elim:   1   3   Wed   bean   Level = 212
```

Elim: 1 3 Thu bean Level = 212

Elim: 1 3 Fri bean Level = 212

TransClosure: Conclusion Fri = broc

RuleOut: 1 3 Fri broc Level = 207

RuleOut: 3 1 broc Fri Level = 207

Elim: 1 3 Wed broc Level = 212

Elim: 1 3 Thu broc Level = 212

TransClosure:

$$\begin{bmatrix} & Tue & Wed & Thu & Fri \\ onion & X & \_ & \_ & X \\ broc & X & X & X & \bigcirc \\ mush & X & \_ & \_ & X \\ bean & \bigcirc & X & X & X \end{bmatrix}$$

Pivot: 1 3 Tue bean Level = 214

&<=>: Tue = cauliflower Level = 247

TransClosure: Conclusion Tue = cauliflower

RuleOut: 1 4 Tue cauliflower Level = 294

RuleOut: 4 1 cauliflower Tue Level = 294

Elim: 1 4 Wed cauliflower Level = 299

Elim: 1 4 Thu cauliflower Level = 299

Elim: 1 4 Fri cauliflower Level = 299

Elim: 4 1 spinach Tue Level = 299

Elim: 4 1 cabbage Tue Level = 299

TransClosure:

	Tue	Wed	Thu	Fri
carrots	X	X	_	_
spinach	X	_	_	X
cauliflower	○	X	X	X
cabbage	X	_	_	_

Pivot: 1 4 Tue cauliflower Level = 301

Pivot: 4 1 cauliflower Tue Level = 301

&<!=>: Conclusion cauliflower <> chicken Level = 338

Elim: 4 2 cauliflower chicken Level = 364

Elim: 2 4 chicken cauliflower Level = 364

TransClosure: Conclusion chicken = carrots

RuleOut: 2 4 chicken carrots Level = 385

RuleOut: 4 2 carrots chicken Level = 385

Elim: 2 4 pork carrots Level = 390

Elim: 2 4 fish carrots Level = 390

TransClosure:

	pork	beef	chicken	fish
carrots	X	X	○	X
spinach	_	_	X	_
cauliflower	_	_	X	_
cabbage	_	_	X	_

```
Pivot:   2   4   chicken   carrots   Level = 392

&<!=>:   Conclusion   chicken <> Wed   Level = 429

Elim:   2   1   chicken   Wed   Level = 455

Elim:   1   2   Wed   chicken   Level = 455

TransferX:   2   1   chicken   Wed   Level = 450

TransferX:   1   2   Wed   chicken   Level = 450

&<!=>:   Conclusion   chicken <> onion   Level = 429

Elim:   2   3   chicken   onion   Level = 455

Elim:   3   2   onion   chicken   Level = 455

TransferX:   2   3   chicken   onion   Level = 450

TransferX:   3   2   onion   chicken   Level = 450

&<!=>:   Conclusion   chicken <> bean   Level = 429

Elim:   2   3   chicken   bean   Level = 455

Elim:   3   2   bean   chicken   Level = 455

TransferX:   2   3   chicken   bean   Level = 450

TransferX:   3   2   bean   chicken   Level = 450

Pivot:   4   2   carrots   chicken   Level = 392

Pivot:   3   1   bean   Tue   Level = 214

Pivot:   1   3   Fri   broc   Level = 214

Pivot:   3   1   broc   Fri   Level = 214

&<!=>:   Conclusion   broc <> beef   Level = 251

Elim:   3   2   broc   beef   Level = 277
```

```
Elim:   2   3   beef   broc   Level = 277

TransferX:   3   2   broc   beef   Level = 272

TransferX:   2   3   beef   broc   Level = 272

&<!=>:   Conclusion   broc <> fish   Level = 251

Elim:   3   2   broc   fish   Level = 277

Elim:   2   3   fish   broc   Level = 277

TransferX:   3   2   broc   fish   Level = 272

TransferX:   2   3   fish   broc   Level = 272

Less/CheckForKnownPosition:   1   spinach   broc

Less/OnePair:   chicken, broc, 1

&<!=>:   Conclusion   chicken <> broc   Level = 160

Elim:   2   3   chicken   broc   Level = 186

Elim:   3   2   broc   chicken   Level = 186

TransClosure:   Conclusion   broc = pork

RuleOut:   3   2   broc   pork   Level = 207

RuleOut:   2   3   pork   broc   Level = 207

Elim:   3   2   onion   pork   Level = 212

Elim:   3   2   mush   pork   Level = 212

Elim:   3   2   bean   pork   Level = 212

TransClosure:   Conclusion   onion = fish

RuleOut:   3   2   onion   fish   Level = 207

RuleOut:   2   3   fish   onion   Level = 207
```

Elim: 3 2 mush fish Level = 212

Elim: 3 2 bean fish Level = 212

TransClosure: Conclusion bean = beef

RuleOut: 3 2 bean beef Level = 207

RuleOut: 2 3 beef bean Level = 207

Elim: 3 2 mush beef Level = 212

TransClosure: Conclusion mush = chicken

RuleOut: 3 2 mush chicken Level = 207

RuleOut: 2 3 chicken mush Level = 207

TransClosure: Conclusion chicken = mush

RuleOut: 2 3 chicken mush Level = 207

RuleOut: 3 2 mush chicken Level = 207

TransClosure:

$$\begin{bmatrix} & pork & beef & chicken & fish \\ onion & X & X & X & \bigcirc \\ broc & \bigcirc & X & X & X \\ mush & X & X & \bigcirc & X \\ bean & X & \bigcirc & X & X \end{bmatrix}$$

Pivot: 3 2 broc pork Level = 214

&<!=>: Conclusion broc <> carrots Level = 251

Elim: 3 4 broc carrots Level = 277

Elim: 4 3 carrots broc Level = 277

TransClosure: Conclusion carrots = mush

RuleOut: 4 3 carrots mush Level = 298

RuleOut: 3 4 mush carrots Level = 298

Elim: 4 3 spinach mush Level = 303

Elim: 4 3 cabbage mush Level = 303

TransClosure: Conclusion spinach = onion

RuleOut: 4 3 spinach onion Level = 298

RuleOut: 3 4 onion spinach Level = 298

Elim: 4 3 cabbage onion Level = 303

TransClosure: Conclusion cabbage = broc

RuleOut: 4 3 cabbage broc Level = 298

RuleOut: 3 4 broc cabbage Level = 298

TransClosure: Conclusion broc = cabbage

RuleOut: 3 4 broc cabbage Level = 298

RuleOut: 4 3 cabbage broc Level = 298

TransClosure:

	onion	*broc*	*mush*	*bean*
carrots	*X*	*X*	○	*X*
spinach	○	*X*	*X*	*X*
cauliflower	*X*	*X*	*X*	○
cabbage	*X*	○	*X*	*X*

Pivot: 4 3 carrots mush Level = 305

&<!=>: Conclusion carrots <> Fri Level = 342

```
Elim:   4   1   carrots   Fri   Level = 368

Elim:   1   4   Fri   carrots   Level = 368

TransClosure:   Conclusion   Fri = cabbage

RuleOut:   1   4   Fri   cabbage   Level = 389

RuleOut:   4   1   cabbage   Fri   Level = 389

Elim:   1   4   Wed   cabbage   Level = 394

Elim:   1   4   Thu   cabbage   Level = 394

TransClosure:   Conclusion   Wed = spinach

RuleOut:   1   4   Wed   spinach   Level = 389

RuleOut:   4   1   spinach   Wed   Level = 389

Elim:   1   4   Thu   spinach   Level = 394

TransClosure:   Conclusion   Thu = carrots

RuleOut:   1   4   Thu   carrots   Level = 389

RuleOut:   4   1   carrots   Thu   Level = 389

TransClosure:   Conclusion   carrots = Thu

RuleOut:   4   1   carrots   Thu   Level = 389

RuleOut:   1   4   Thu   carrots   Level = 389

TransClosure:
```

	carrots	spinach	cauliflower	cabbage
Tue	X	X	O	X
Wed	X	O	X	X
Thu	O	X	X	X
Fri	X	X	X	O

```
Pivot:   1   4   Fri   cabbage   Level = 396

&<!=>:   Conclusion   Fri <> chicken   Level = 433

Elim:   1   2   Fri   chicken   Level = 459

Elim:   2   1   chicken   Fri   Level = 459

TransClosure:   Conclusion   chicken = Thu

RuleOut:   2   1   chicken   Thu   Level = 480

RuleOut:   1   2   Thu   chicken   Level = 480

Elim:   2   1   pork   Thu   Level = 485

Elim:   2   1   beef   Thu   Level = 485

Elim:   2   1   fish   Thu   Level = 485

TransClosure:   Conclusion   Fri = pork

RuleOut:   1   2   Fri   pork   Level = 480

RuleOut:   2   1   pork   Fri   Level = 480

Elim:   1   2   Tue   pork   Level = 485

Elim:   1   2   Wed   pork   Level = 485

TransClosure:
```

	Tue	Wed	Thu	Fri
pork	X	X	X	○
beef	_	_	X	X
chicken	X	X	○	X
fish	_	_	X	X

```
Pivot:   2   1   chicken   Thu   Level = 487

Pivot:   1   2   Thu   chicken   Level = 487

&<=>:   Thu = mush   Level = 520

TransClosure:   Conclusion   Thu = mush

RuleOut:   1   3   Thu   mush   Level = 567

RuleOut:   3   1   mush   Thu   Level = 567

Elim:   1   3   Wed   mush   Level = 572

Elim:   3   1   onion   Thu   Level = 572

TransClosure:   Conclusion   onion = Wed

RuleOut:   3   1   onion   Wed   Level = 567

RuleOut:   1   3   Wed   onion   Level = 567

TransClosure:   Conclusion   Wed = onion

RuleOut:   1   3   Wed   onion   Level = 567

RuleOut:   3   1   onion   Wed   Level = 567

TransClosure:
```

	Tue	Wed	Thu	Fri
onion	X	O	X	X
broc	X	X	X	O
mush	X	X	O	X
bean	O	X	X	X

```
Pivot:   1   3   Thu   mush   Level = 574

Pivot:   3   1   mush   Thu   Level = 574

Pivot:   3   1   onion   Wed   Level = 574

Pivot:   1   3   Wed   onion   Level = 574

&<=>:   Wed = fish   Level = 607

TransClosure:   Conclusion   Wed = fish

RuleOut:   1   2   Wed   fish   Level = 654

RuleOut:   2   1   fish   Wed   Level = 654

Elim:   1   2   Tue   fish   Level = 659

Elim:   2   1   beef   Wed   Level = 659

TransClosure:   Conclusion   beef = Tue

RuleOut:   2   1   beef   Tue   Level = 654

RuleOut:   1   2   Tue   beef   Level = 654

TransClosure:   Conclusion   Tue = beef

RuleOut:   1   2   Tue   beef   Level = 654

RuleOut:   2   1   beef   Tue   Level = 654

TransClosure:
```

	Tue	Wed	Thu	Fri
pork	X	X	X	O
beef	O	X	X	X
chicken	X	X	O	X
fish	X	O	X	X

```
Pivot:   1   2   Wed   fish   Level = 661

Pivot:   2   1   fish   Wed   Level = 661

&<=>:   fish = spinach   Level = 694

TransClosure:   Conclusion   fish = spinach

RuleOut:   2   4   fish   spinach   Level = 741

RuleOut:   4   2   spinach   fish   Level = 741

Elim:   2   4   pork   spinach   Level = 746

Elim:   2   4   beef   spinach   Level = 746

Elim:   4   2   cauliflower   fish   Level = 746

Elim:   4   2   cabbage   fish   Level = 746

TransClosure:
```

	pork	beef	chicken	fish
carrots	X	X	O	X
spinach	X	X	X	O
cauliflower	_	_	X	X
cabbage	_	_	X	X

```
Pivot:   2   4   fish   spinach   Level = 748

Pivot:   4   2   spinach   fish   Level = 748
```

Pivot: 2 1 beef Tue Level = 661

&<=>: beef = cauliflower Level = 694

TransClosure: Conclusion beef = cauliflower

RuleOut: 2 4 beef cauliflower Level = 741

RuleOut: 4 2 cauliflower beef Level = 741

Elim: 2 4 pork cauliflower Level = 746

Elim: 4 2 cabbage beef Level = 746

TransClosure: Conclusion cabbage = pork

RuleOut: 4 2 cabbage pork Level = 741

RuleOut: 2 4 pork cabbage Level = 741

TransClosure: Conclusion pork = cabbage

RuleOut: 2 4 pork cabbage Level = 741

RuleOut: 4 2 cabbage pork Level = 741

TransClosure:

	pork	*beef*	*chicken*	*fish*
carrots	*X*	*X*	○	*X*
spinach	*X*	*X*	*X*	○
cauliflower	*X*	○	*X*	*X*
cabbage	○	*X*	*X*	*X*

Pivot: 2 4 beef cauliflower Level = 748

Pivot: 4 2 cauliflower beef Level = 748

Pivot: 4 2 cabbage pork Level = 748

```
Pivot:   2   4   pork   cabbage   Level = 748

Pivot:   2   4   pork   cabbage   Level = 748

Pivot:   4   2   cabbage   pork   Level = 748

Pivot:   1   2   Tue   beef   Level = 661

Pivot:   1   2   Tue   beef   Level = 661

Pivot:   2   1   beef   Tue   Level = 661

Pivot:   1   3   Wed   onion   Level = 574

Pivot:   3   1   onion   Wed   Level = 574

Pivot:   1   2   Fri   pork   Level = 487

Pivot:   2   1   pork   Fri   Level = 487

Pivot:   4   1   cabbage   Fri   Level = 396

Pivot:   1   4   Wed   spinach   Level = 396

Pivot:   4   1   spinach   Wed   Level = 396

Pivot:   1   4   Thu   carrots   Level = 396

Pivot:   4   1   carrots   Thu   Level = 396

Pivot:   4   1   carrots   Thu   Level = 396

Pivot:   1   4   Thu   carrots   Level = 396

Pivot:   3   4   mush   carrots   Level = 305

Pivot:   4   3   spinach   onion   Level = 305

Pivot:   3   4   onion   spinach   Level = 305

Pivot:   4   3   cabbage   broc   Level = 305

Pivot:   3   4   broc   cabbage   Level = 305
```

```
Pivot:   3   4   broc   cabbage   Level = 305

Pivot:   4   3   cabbage   broc   Level = 305

Pivot:   2   3   pork   broc   Level = 214

Pivot:   3   2   onion   fish   Level = 214

Pivot:   2   3   fish   onion   Level = 214

Pivot:   3   2   bean   beef   Level = 214

Pivot:   2   3   beef   bean   Level = 214

Pivot:   3   2   mush   chicken   Level = 214

Pivot:   2   3   chicken   mush   Level = 214

Pivot:   2   3   chicken   mush   Level = 214

Pivot:   3   2   mush   chicken   Level = 214

Less/CheckForKnownPosition:   1   chicken   broc

Less/Check:   Constraint   Less(chicken,broc,1)   completely satisfied.

Know:   Processing   [-7, Less(onion,carrots,1)]

ParseAssertionTypes:   false, [-7, Less(onion,carrots,1)]

Less/OnePair:   onion, carrots, 1

Less/Check:   Constraint   Less(onion,carrots,1)   completely satisfied.

Know:   Processing   [-7, Less(spinach,broc,1)]

ParseAssertionTypes:   false, [-7, Less(spinach,broc,1)]

Less/OnePair:   spinach, broc, 1

Less/Check:   Constraint   Less(spinach,broc,1)   completely satisfied.

Know:   Processing   [-7, Less(spinach,chicken,1)]
```

```
ParseAssertionTypes:   false, [-7, Less(spinach,chicken,1)]

Less/OnePair:   spinach, chicken, 1

Less/Check:   Constraint   Less(spinach,chicken,1)   completely satisfied.

Know:   Processing   [-7, Less(beef,carrots,1)]

ParseAssertionTypes:   false, [-7, Less(beef,carrots,1)]

Less/OnePair:   beef, carrots, 1

Less/Check:   Constraint   Less(beef,carrots,1)   completely satisfied.

Know:   Processing   [-7, Less(beef,onion,1)]

ParseAssertionTypes:   false, [-7, Less(beef,onion,1)]

Less/OnePair:   beef, onion, 1

Less/Check:   Constraint   Less(beef,onion,1)   completely satisfied.
```

$$Unsatisfied = [\ ]$$

```
NormalReturn:   Conclusions:   102
```

$$Unique\ solution:$$

$$\begin{bmatrix} Tue & beef & bean & cauliflower \\ Wed & fish & onion & spinach \\ Thu & chicken & mush & carrots \\ Fri & pork & broc & cabbage \end{bmatrix}$$

```
> time()-st;
```

.211

```
> lp2:-`&?`(ShowStats);
```

$$Sets = 6,\ Unsets = 27,\ RuleOuts = 60,\ Elims = 99,\ Transfers = 42,\ Pivots = 60,\ MaxLevel = 699,\ Guesses = 0,\ MaxDepth = 0$$

> **interface(rtablesize= 24);**

Note that if you do not use the "with" command, then you must call &? with quotes and parentheses or use a "use" statement. The same is true of any other procedure whose name begins with "&". The "use" statement is more elegant:

> **use lp2 in &? ShowStats end;**

$$Sets = 6,\ Unsets = 27,\ RuleOuts = 60,\ Elims = 99,\ Transfers = 42,\ Pivots = 60,\ MaxLevel = 699,\ Guesses = 0,\ MaxDepth = 0$$

> **use lp2 in &? day end;**

	pork	beef	chicken	fish	.	onion	broc	mush	bean	.	carrots	spinach	cauliflower	cabbage
Tue	X	O	X	X	.	X	X	X	O	.	X	X	O	X
Wed	X	X	X	O	.	O	X	X	X	.	X	O	X	X
Thu	X	X	O	X	.	X	X	O	X	.	O	X	X	X
Fri	O	X	X	X	.	X	O	X	X	.	X	X	X	O

An example with a lot of "Less" constraints and a relatively simple procedural constraint.

This problem is adapted from Frank Alward "Logic Problem 3: White-Water Rafting" in *Dell Math Puzzles and Logic Problems* (No. 49, November 2000).

In a marathon race, prizes were awarded (one was a DVD player) to the first 5 finishers (one was Mr. Young). Each of these 5 wore a different color jersey (one was yellow). Determine the first and last name, placement, jersey color, and prize of each of the 5.

1. The blue jersey was in 1st place.

2. The mountain bike was awarded to 5th place.

3. Mr. Bowker did not win the stereo.

4. The runner who won the VCR, whose jersey was neither red nor white, finished behind both the runner in the green jersey and the winner of the TV.

5. Mr. Sparks, who isn't Al, finished 2 places ahead of Otis.

6. Mr. Nolan and Mr. Sparks finished behind Mark, but all 3 finished before Mr. Stover and the winner of the stereo.

7. The runner in the red jersey finished before Del.

```
> V:= ['place', 'first', 'last', 'color', 'prize'];
```

$$V := [place, first, last, color, prize]$$

```
> place:= [$1..5];
```

$$place := [1, 2, 3, 4, 5]$$

```
> first:= [Al, Del, Larry, Mark, Otis];
```

$$first := [Al, Del, Larry, Mark, Otis]$$

```
> last:= [Bowker, Nolan, Sparks, Stover, Young];
```

$$last := [Bowker, Nolan, Sparks, Stover, Young]$$

```
> color:= [blue, green, red, white, yellow];
```

$$color := [blue, green, red, white, yellow]$$

```
> prize:= [bike, dvd, stereo, tv, vcr];
```

$$prize := [bike, dvd, stereo, tv, vcr]$$

```
> lp3:= LogicProblem(V):
```

$$Okay$$

Clue #5 cannot be completely handled by any of the constraint types that we've discussed so far. One approach is to use the following procedural constraint.

```
> ClueSparksOtis:= proc(M)
local sparks, otis, internal_place, place_sparks, place_otis;
use `&Soln`= M:-`&Soln`, ConstNum= M:-ConstNum, VarNum= M:-VarNum in
sparks,otis:= ConstNum(Sparks), ConstNum(Otis);
internal_place:= VarNum(place);
place_sparks:= sparks &Soln internal_place;
place_otis:= otis &Soln internal_place
end use;
if place_sparks > 0 and place_otis > 0 then
```

```
if place_otis - place_sparks = 2 then false,true #completely satisfied
else true #contradiction
end
#If we know the place of one of them, we can definitely specify the place of the other
elif place_sparks > 0 then false, false, [Otis = place_sparks+2]
elif place_otis > 0 then false, false, [Sparks = place_otis-2]
else false, false, [] #Consistent (so far), but not satisfied
end
end;
```

```
ClueSparksOtis := proc(M)
local sparks, otis, internal_place, place_sparks, place_otis;
    sparks, otis := M:-ConstNum(Sparks), M:-ConstNum(Otis);
    internal_place := M:-VarNum(place);
    place_sparks := M:-`&Soln`(sparks, internal_place);
    place_otis := M:-`&Soln`(otis, internal_place);
    if 0 < place_sparks and 0 < place_otis then if place_otis - place_sparks = 2 then false, true else true end if
    elif 0 < place_sparks then false, false, [Otis = place_sparks + 2]
    elif 0 < place_otis then false, false, [Sparks = place_otis - 2]
    else false, false, [ ]
    end if
end proc
```

```
> with(lp3);
```

[&!!, &-, &<, &>, &?, &G, &Soln, AutoGuess, CPV, CollectStats, ConstNum, Consts, ConstsInV, DifferentBlock, Equation, FreeGuess, GoBack, Guess, InternalRep, IsComplete, IsUnique, NC, NV, PrintConst, Quiet, Reinitialize, SameBlock, Satisfy, Separated, UniquenessProof, VarNum, VarNumC, X_O]

```
> infolevel['Constraints']:= 2: infolevel['TC']:= 1: infolevel['TCdisplay']:= 0: CollectStats:= true:
```

To specify clue 5, it is useful for the sake of efficiency (though it is not necessary) to say everything that can be said with the predefined constraint types in addition to giving the procedural constraint.

```
> Clue5:= Al<>Sparks, Less(Sparks, Otis, place), Sparks<>4, Otis<>2, Proc(ClueSparksOtis, []);
```

$$Clue5 := Al \neq Sparks, \mathrm{Less}(Sparks, Otis, [1, 2, 3, 4, 5]), Sparks \neq 4, Otis \neq 2, \mathrm{Proc}(ClueSparksOtis, [\ ])$$

```
> Constraints:=
[blue=1 #Clue 1
,bike=5 #Clue 2
,Bowker<>stereo #Clue 3

#The 5 parts of the next line are required to completely specify Clue 4
,vcr<>red, vcr<>white, Less(green, vcr, place), Less(tv, vcr, place), tv<>green

,_Clue5

#The next 5 items completely specify Clue 6.
,Less([Mark, Nolan, Stover], place), Less([Mark, Sparks, Stover], place)
,Less([Mark, Nolan, stereo], place), Less([Mark, Sparks, stereo], place)
,Stover<>stereo

#Clue 7
,Less(red,Del,place)
];
```

$$Constraints := [blue = 1, bike = 5, Bowker \neq stereo, vcr \neq red, vcr \neq white, Less(green, vcr, [1, 2, 3, 4, 5]), Less(tv, vcr, [1, 2, 3, 4, 5]), tv \neq green, \_Clue5, Less([Mark, Nolan, Stover], [1, 2, 3, 4, 5]), Less([Mark, Sparks, Stover], [1, 2, 3, 4, 5]), Less([Mark, Nolan, stereo], [1, 2, 3, 4, 5]), Less([Mark, Sparks, stereo], [1, 2, 3, 4, 5]), Stover \neq stereo, Less(red, Del, [1, 2, 3, 4, 5])]$$

```
> st:= time():

> Satisfy(subs(_Clue5= Clue5, Constraints));

Less/Check:   Constraint   Less(Mark,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(green,vcr,1)   completely satisfied.
```

$$Unsatisfied = [[-7, Less(Nolan, stereo, 1)], [-7, Less(red, Del, 1)], [-7, Less(Mark, Nolan, 1)], [-7, Less(Mark, Sparks, 1)], [-7, Less(tv, vcr, 1)], [-7, Less(Sparks, stereo, 1)], [-7, Less(Sparks, Otis, 1)], [-7, Less(Nolan, Stover, 1)], [-7, Less(Sparks, Stover, 1)], [-11, Proc(ClueSparksOtis, [])]]$$

```
Less/Check:   Constraint   Less(red,Del,1)   completely satisfied.
```

```
Less/Check:   Constraint   Less(tv,vcr,1)   completely satisfied.
```

$$Unsatisfied = [[-7, \text{Less}(Nolan, stereo, 1)], [-7, \text{Less}(Mark, Nolan, 1)], [-7, \text{Less}(Mark, Sparks, 1)], [-7, \text{Less}(Sparks, stereo, 1)], [-7, \text{Less}(Sparks, Otis, 1)], [-7, \text{Less}(Nolan, Stover, 1)], [-7, \text{Less}(Sparks, Stover, 1)], [-11, \text{Proc}(ClueSparksOtis, [\,])]]$$

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   Larry = 2   Depth=   1
```

$$Unsatisfied = [[-7, \text{Less}(Nolan, stereo, 1)], [-7, \text{Less}(Mark, Nolan, 1)], [-7, \text{Less}(Mark, Sparks, 1)], [-7, \text{Less}(Sparks, stereo, 1)], [-7, \text{Less}(Sparks, Otis, 1)], [-7, \text{Less}(Nolan, Stover, 1)], [-7, \text{Less}(Sparks, Stover, 1)], [-11, \text{Proc}(ClueSparksOtis, [\,])]]$$

```
Less/Check:   Constraint   Less(Nolan,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Nolan,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Sparks,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Nolan,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,Stover,1)   completely satisfied.
```

$$Unsatisfied = [[-7, \text{Less}(Sparks, Otis, 1)], [-11, \text{Proc}(ClueSparksOtis, [\,])]]$$

```
Less/Check:   Constraint   Less(Sparks,Otis,1)   completely satisfied.

Know/Proc:   Proc(ClueSparksOtis,[])   satisfied.

Satisfy:
```

Complete, consistent solution found. Attempting to prove uniqueness.

```
GoBack:   Restoring saved state from before guess   Larry = 2   Depth=   0
```

$$Unsatisfied = [[-7, \text{Less}(Nolan, stereo, 1)], [-7, \text{Less}(Mark, Nolan, 1)], [-7, \text{Less}(Mark, Sparks, 1)], [-7, \text{Less}(Sparks, stereo, 1)], [-7, \text{Less}(Sparks, Otis, 1)], [-7, \text{Less}(Nolan, Stover, 1)], [-7, \text{Less}(Sparks, Stover, 1)], [-11, \text{Proc}(ClueSparksOtis, [\,])]]$$

```
Less/Check:   Constraint   Less(Nolan,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Nolan,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Sparks,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,Otis,1)   completely satisfied.

Less/Check:   Constraint   Less(Nolan,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,Stover,1)   completely satisfied.

Know/Proc:   Proc(ClueSparksOtis,[])   contradicted.
```

Unique solution:

$$\begin{bmatrix} 1 & Mark & Bowker & blue & tv \\ 2 & Larry & Sparks & green & dvd \\ 3 & Al & Nolan & yellow & vcr \\ 4 & Otis & Young & red & stereo \\ 5 & Del & Stover & white & bike \end{bmatrix}$$

> **time()-st;**

.190

> **&? ShowStats;**

$$Sets = 33,\ Unsets = 70,\ RuleOuts = 168,\ Elims = 290,\ Transfers = 116,\ Pivots = 168,\ MaxLevel = 662,\ Guesses = 1,\ MaxDepth = 1$$

Note that Clue 5 actually specifies a relationship between two constants and a variable. This is the same situation as with constraint types Succ, Less, and NextTo. In this case, it is simpler and more efficient to specify the constraint via the Relational/Procedure contraint type. We can consider this particular relationship as being "AheadBy2".

> **AheadBy2:= (M, a, b) -> `if`(b=0, {a+2}, `if`(a=0, {b-2}, evalb(b-a<>2)));**

$$AheadBy2 := (M, a, b) \rightarrow \text{`if`}(b = 0, \{a + 2\}, \text{`if`}(a = 0, \{b - 2\}, \text{evalb}(b - a \neq 2)))$$

Note how much simpler it is to specify the constraint in this form.

> **Clue5;**

$$Al \neq Sparks, \text{Less}(Sparks, Otis, [1, 2, 3, 4, 5]), Sparks \neq 4, Otis \neq 2, \text{Proc}(ClueSparksOtis, [\ ])$$

> **Clue5:= Al<>Sparks, Rel(AheadBy2, Sparks, Otis, place, []);**

$$Clue5 := Al \neq Sparks, \text{Rel}(AheadBy2, Sparks, Otis, [1, 2, 3, 4, 5], [\ ])$$

> **Reinitialize();**

$$Okay$$

> **st:= time():**

> **Satisfy(subs(_Clue5= Clue5, Constraints));**

```
Less/Check:   Constraint   Less(Mark,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(green,vcr,1)   completely satisfied.

Less/Check:   Constraint   Less(tv,vcr,1)   completely satisfied.
```

$$Unsatisfied = [[-7, \text{Less}(Nolan, Stover, 1)], [-7, \text{Less}(Sparks, Stover, 1)], [-7, \text{Less}(Mark, Sparks, 1)], [-7, \text{Less}(Nolan, stereo, 1)], [-7, \text{Less}(red, Del, 1)], [-7, \text{Less}(Mark, Nolan, 1)], [-7, \text{Less}(Sparks, stereo, 1)], [-10, \text{Rel}(AheadBy2, Sparks, Otis, 1, [\ ])]]$$

```
Less/Check:   Constraint   Less(red,Del,1)   completely satisfied.

Rel/Check:   Constraint   Rel(AheadBy2,Sparks,Otis,1,[])   satisfied.
```

$$Unsatisfied = [[-7, \text{Less}(Nolan, Stover, 1)], [-7, \text{Less}(Sparks, Stover, 1)], [-7, \text{Less}(Mark, Sparks, 1)], [-7, \text{Less}(Nolan, stereo, 1)], [-7, \text{Less}(Mark, Nolan, 1)], [-7, \text{Less}(Sparks, stereo, 1)]]$$

```
Less/Check:   Constraint   Less(Nolan,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Sparks,1)   completely satisfied.

Less/Check:   Constraint   Less(Nolan,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Nolan,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,stereo,1)   completely satisfied.
```

Unique solution:

$$\begin{bmatrix} 1 & Mark & Bowker & blue & tv \\ 2 & Larry & Sparks & green & dvd \\ 3 & Al & Nolan & yellow & vcr \\ 4 & Otis & Young & red & stereo \\ 5 & Del & Stover & white & bike \end{bmatrix}$$

> **time()-st;**

.100

> **&? ShowStats;**

$Sets = 23, Unsets = 69, RuleOuts = 120, Elims = 269, Transfers = 118, Pivots = 120, MaxLevel = 607, Guesses = 0, MaxDepth = 0$

Note how much more efficient it is to use the Relational/Procedural constraint type. A Proc/Procedural clue that is hard to satisfy leads to inefficient guessing and backtracking.

The relationship expressed by "AheadBy2" already exists as a predefined constraint type "Equation".

> **Clue5;**

$Al \neq Sparks, \mathrm{Rel}(AheadBy2, Sparks, Otis, [1, 2, 3, 4, 5], [\ ])$

> **Clue5:= Al<>Sparks, Rel(Equation, Sparks, Otis, place, [{_A+2}, {_B-2}]);**

$$Clue5 := Al \neq Sparks, \mathrm{Rel}(Equation, Sparks, Otis, [1, 2, 3, 4, 5], [\{\_A + 2\}, \{\_B - 2\}])$$

The first expression in brackets represents the solution set for the second constant's (in this case Otis) position given that the first constant's (in this case Sparks) position is known; the second expression represents to solutions for the first constant's position given that the second constant's position is known. Note the the symbols _A and _B must be used.

> **Reinitialize();**

$$Okay$$

> **st:= time():**

> **Satisfy(subs(_Clue5= Clue5, Constraints));**

```
Less/Check:   Constraint   Less(Mark,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(green,vcr,1)   completely satisfied.

Less/Check:   Constraint   Less(tv,vcr,1)   completely satisfied.
```

$$Unsatisfied = [[-7, \mathrm{Less}(Nolan, Stover, 1)], [-7, \mathrm{Less}(Sparks, Stover, 1)], [-7, \mathrm{Less}(Mark, Sparks, 1)], [-7, \mathrm{Less}(Nolan, stereo, 1)], [-7, \mathrm{Less}(red, Del, 1)], [-7, \mathrm{Less}(Mark, Nolan, 1)], [-7, \mathrm{Less}(Sparks, stereo, 1)], [-10, \mathrm{Rel}(Equation, Sparks, Otis, 1, [\{\_A + 2\}, \{\_B - 2\}])]]$$

```
Less/Check:   Constraint   Less(red,Del,1)   completely satisfied.

Rel/Check:   Constraint   Rel(Equation,Sparks,Otis,1,[{_A+2}, {_B-2}])   satisfied.
```

$$Unsatisfied = [[-7, \mathrm{Less}(Nolan, Stover, 1)], [-7, \mathrm{Less}(Sparks, Stover, 1)], [-7, \mathrm{Less}(Mark, Sparks, 1)], [-7, \mathrm{Less}(Nolan, stereo, 1)], [-7, \mathrm{Less}(Mark, Nolan, 1)], [-7, \mathrm{Less}(Sparks, stereo, 1)]]$$

```
Less/Check:   Constraint   Less(Nolan,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,Stover,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Sparks,1)   completely satisfied.
```

```
Less/Check:   Constraint   Less(Nolan,stereo,1)   completely satisfied.

Less/Check:   Constraint   Less(Mark,Nolan,1)   completely satisfied.

Less/Check:   Constraint   Less(Sparks,stereo,1)   completely satisfied.
```

Unique solution:

$$\begin{bmatrix} 1 & Mark & Bowker & blue & tv \\ 2 & Larry & Sparks & green & dvd \\ 3 & Al & Nolan & yellow & vcr \\ 4 & Otis & Young & red & stereo \\ 5 & Del & Stover & white & bike \end{bmatrix}$$

> **time()-st;**

.141

> **&? ShowStats;**

Sets = 23, *Unsets* = 69, *RuleOuts* = 120, *Elims* = 269, *Transfers* = 118, *Pivots* = 120, *MaxLevel* = 607, *Guesses* = 0, *MaxDepth* = 0

An example problem with dummy variables and what to do when some of the variables are not the right size.

This problem is adapted from Diane Yoko "Logic Problem 7: Christmastime Already!?!" in *Dell Math Puzzles and Logic Problems* (No. 49, November 2000)

Dan and five of his friends each bought distinct types of Christmas trees (one was a spruce) from distinct vendors on not-necessarily-distinct days, Monday to Friday, of last week. Determine each friends' first and last name, type of tree, vendor, and day of purchase.

1. A person's sex can be determined in the natural way from their first name.

2. Anne bought her tree from a private farm.

3. M. Turner and Eva bought trees on Wednesday.

4. M. Turner is married.

5. M. Turner didn't buy the white pine.

6. A single woman bought a tree from Andrews' Nursery.

7. The Douglas fir wasn't the tree Mr. Stevens bought on Monday.

8. Sally and M. Jansen are both married; Irene and M. Hall are both single; four distinct persons are mentioned in this clue.

9. M. Eden didn't buy the white pine or the Douglas fir.

10. Theo wasn't the man who bought the Scotch pine on Thursday.

11. The Christmas store did not sell the Douglas fir.

12. The balsam was bought by a woman on Tuesday.

13. Sally's tree wasn't from Caine's Nursery.

14. Mr. Conner bought his tree from Bank's Nursery.

15. The tree from the department store was bought on Friday.

16. The tree from the department store wasn't the artificial one.

In order to use the Logic Puzzle program, we need to state the problem in terms of equivalence relations. The first problem with that is that we need 6 equivalence classes, but there are only five days Monday to Friday. A quick reading of the clues shows that a tree was purchased on each of those days and two were purchased on Wednesday. Thus, we will put two tokens for Wednesday in the varibable 'day'. The second problem is that sex and marital status are obviously important pieces of information, though they are not explicitly asked for. We can figure out everyone's sex by clue 1. But we can't figure out everyone's marital status. By clue 8, there are at least two married persons and at least two single persons. We create a variable 'status' that assumes the values M1, M2, S1, S2, U1, U2. These stand for "married person 1", "married person 2", "single person 1", "single person 2", "unknown person 1", "unknown person 2". It is possible to completely solve the problem as stated without explicitly determining the marital status of those two unknowns.

```
> V:= ['first', 'last', 'vendor', 'tree', 'day', 'sex', 'status'];
```

$$V := [first, last, vendor, tree, day, sex, status]$$

```
> first:= [Anne, Dan, Eva, Irene, Sally, Theo];
```

$$first := [Anne, Dan, Eva, Irene, Sally, Theo]$$

```
> last:= [Conner,Eden,Hall,Jansen,Stevens,Turner];
```

$$last := [Conner, Eden, Hall, Jansen, Stevens, Turner]$$

```
> vendor:= [Andrews, Banks, Caines, Christmas, Dept, farm];
```

$$vendor := [Andrews, Banks, Caines, Christmas, Dept, farm]$$

```
> tree:= [art, balsam, fir, scotch, spruce, white];
```

$$tree := [art, balsam, fir, scotch, spruce, white]$$

```
> day:= [Mon, Tue, Wed1, Wed2, Thu, Fri];
```

$$day := [Mon, Tue, Wed1, Wed2, Thu, Fri]$$

2 males and 4 females.

```
> sex:= [m1,m2,f1,f2,f3,f4];
```

$$sex := [m1, m2, f1, f2, f3, f4]$$

```
> status:= [M1,M2,U1,U2,S1,S2];
```

$$status := [M1, M2, U1, U2, S1, S2]$$

```
> Constraints:=
[Dan=m1, Theo=m2, Eva=f1, Irene=f2, Sally=f3, Anne=f4 #Clue 1
,Anne=farm #Clue 2
,Turner=Wed1, Eva=Wed2 #Clue 3
,Turner=M1 #Clue 4
,Turner<>white #Clue 5
,Distinct([S1, {m1,m2}]) #Clue 6 "a single woman..."
,S1=Andrews #Clue 6
,Distinct([Stevens, {f||(1..4)}]) #Clue 7: Stevens is male. We can't say whether he is m1 or m2.
#Note that I could have said OR([Stevens=m1, Stevens=m2]), but the Distinct contraint type is far more efficient
,Stevens=Mon, Stevens<>fir #Clue 7

#The next 5 items are required to specify Clue 8
,Distinct([Sally,Jansen,Irene,Hall])
,Distinct([{Sally, Jansen}, {U1,U2,S1,S2}])
,Distinct([{Irene, Hall}, {M1,M2,U1,U2}])

,Distinct([Eden, {white, fir}]) #Clue 9
```

```
,scotch=Thu, Distinct([Thu, {Theo, f||(1..4)}]) #Clue 10
,Christmas<>fir #Clue 11
,balsam=Tue, Distinct([balsam, {m1,m2}]) #Clue 12
,Sally<>Caines #Clue 13
,Distinct([Conner, {f||(1..4)}]), Conner=Banks #Clue 14
,Dept=Fri #Clue 15
,Dept<>art #Clue 16
];
```

$$Constraints := [Dan = m1, Theo = m2, Eva = f1, Irene = f2, Sally = f3, Anne = f4, Anne = farm, Turner = Wed1, Eva = Wed2, Turner = M1, Turner \neq white, \text{Distinct}([S1, \{m1, m2\}]), S1 = Andrews, \text{Distinct}([Stevens, \{f3, f4, f1, f2\}]), Stevens = Mon, Stevens \neq fir, \text{Distinct}([Sally, Jansen, Irene, Hall]), \text{Distinct}([\{Jansen, Sally\}, \{U1, U2, S1, S2\}]), \text{Distinct}([\{Irene, Hall\}, \{M1, M2, U1, U2\}]), \text{Distinct}([Eden, \{fir, white\}]), scotch = Thu, \text{Distinct}([Thu, \{f3, f4, Theo, f1, f2\}]), Christmas \neq fir, balsam = Tue, \text{Distinct}([balsam, \{m1, m2\}]), Sally \neq Caines, \text{Distinct}([Conner, \{f3, f4, f1, f2\}]), Conner = Banks, Dept = Fri, Dept \neq art]$$

```
> lp4:= LogicProblem(V):
```

Okay

```
> st:= time():
```

```
> infolevel['Constraints']:= 1: infolevel['TC']:= 1:
```

```
> lp4:-Satisfy(Constraints);
```

Incomplete solution; need more constraints to complete:

$$\begin{bmatrix} Anne & Jansen & farm & balsam & Tue & f4 & M2 \\ Dan & Conner & Banks & scotch & Thu & m1 & \_ \\ Eva & Hall & Andrews & fir & Wed2 & f1 & S1 \\ Irene & Eden & Dept & spruce & Fri & f2 & S2 \\ Sally & Turner & Christmas & art & Wed1 & f3 & M1 \\ Theo & Stevens & Caines & white & Mon & m2 & \_ \end{bmatrix}$$

```
> time()-st;
```

.341

> **lp4:-`&?`(CountGuesses);**

0

Since there are no undisproved guesses, it is clear that the required solution on the first five variables is complete and unique.

A more complex example with a more complicated uniqueness proof.

This problem is adapted from Jean Hannagan "Logic Problem 10: Letters to the Editor" in *Dell Math Puzzles and Logic Problems* (No. 49, November 2000).

Last month, each of five freelance food writers (one's first name is Kit) for a magazine received a letter from a reader pointing out a mistake in a recent article they had written. Each writer lives in a different town (one lives in Forest Park). One writer decided to quit, and each of the others mailed a correction. Each article was about a different subject (one was a new restaurant). Determine each writers full name, the subject of their article, the town they live in, and the order in which their complaint letters were received. Also determine which writer decided to quit.

1. The sexes cannot be determined from the first names.

2. Kevin didn't write the article about garlic.

3. Kevin received his letter sometime after the one who wrote about herbs.

4. Kevin mailed his correction along with a new story (i.e., Kevin did not quit).

5. Lee isn't M. Ware.

6. Lee received a letter after the one who wrote about desserts.

7. The one who wrote about roasting vegetables received her letter before the Cedar Grove resident.

8. Lee isn't M. Landis.

9. M. Landis got a letter sometime after the one who wrote an article on desserts.

10. Sal's letter arrived sometime after M. Turner's.

11. The recipient of the first letter did not write about herbs.

12. The recipient of the first letter did not quit.

13. The letter about roasting vegetables came after the one about herbs.

14. The article about herbs wasn't written by the man who lives in Sunrise Acres.

15. Kevin's letter arrived before the one for the resident of Cedar Grove.

16. The writer from Cedar Grove did not write about garlic.

17. Sal's letter arrived before the Spring Ridge resident received his.

18. The Pine Landing resident isn't M. Banks.

19. The Pine Landing resident received a letter before M. Ware.

20. M. Banks received a letter before Lou.

21. Neither M. Banks nor Lou decided to quit.

```
> V:= ['first','last','subject','city','When','quitter','sex'];
```

$$V := [first, last, subject, city, When, quitter, sex]$$

```
> first:= [Kevin,Kit,Lee,Lou,Sal];
```

$$first := [Kevin, Kit, Lee, Lou, Sal]$$

```
> last:= [Banks,Colson,Landis,Turner,Ware];
```

$$last := [Banks, Colson, Landis, Turner, Ware]$$

```
> subject:= [dessert,garlic,herbs,New,roast];
```

$$subject := [dessert, garlic, herbs, New, roast]$$

```
> city:= [Cedar,Forest,Pine,Spring,Sunrise];
```

$$city := [Cedar, Forest, Pine, Spring, Sunrise]$$

```
> When:= [$1..5];
```

$$When := [1, 2, 3, 4, 5]$$

We need four dummy constants for the four writers who did not quit.

```
> quitter:= [q,n1,n2,n3,n4];
```

$$quitter := [q, n1, n2, n3, n4]$$

There are clearly at least two men and at least one woman. The other two are of unknown sex.

```
> sex:= [m1,m2,u1,u2,f];
```

$$sex := [m1, m2, u1, u2, f]$$

```
> Constraints:=
[Kevin<>garlic #Clue 2
,Less(herbs,Kevin,When) #Clue 3
,Kevin<>q, Distinct([Kevin, {u1,u2,f}]) #Clue 4
,Lee<>Ware #Clue 5
,Less(dessert,Lee,When) #Clue 6
,Less(roast,Cedar,When), roast=f #Clue 7
,Lee<>Landis #Clue 8
,Less(dessert,Landis,When) #Clue 9
,Less(Turner,Sal,When) #Clue 10
,1<>herbs #Clue 11
,1<>q #Clue 12
,Less(herbs,roast,When) #Clue 13
,herbs<>Sunrise, Distinct([Sunrise, {u1,u2,f}]) #Clue 14
,Less(Kevin,Cedar,When) #Clue 15
,Cedar<>garlic #Clue 16
,Less(Sal,Spring,When), Distinct([Spring, {u1,u2,f}]) #Clue 17
,Pine<>Banks #Clue 18
,Less(Pine,Ware,When) #Clue 19
,Less(Banks,Lou,When) #Clue 20
,Banks<>q, Lou<>q #Clue 21
];
```

$$Constraints := [Kevin \neq garlic, \mathrm{Less}(herbs, Kevin, [1, 2, 3, 4, 5]), Kevin \neq q, \mathrm{Distinct}([Kevin, \{u1, u2, f\}]), Lee \neq Ware,$$
$$\mathrm{Less}(dessert, Lee, [1, 2, 3, 4, 5]), \mathrm{Less}(roast, Cedar, [1, 2, 3, 4, 5]), roast = f, Lee \neq Landis, \mathrm{Less}(dessert, Landis, [1, 2, 3, 4, 5]),$$
$$\mathrm{Less}(Turner, Sal, [1, 2, 3, 4, 5]), 1 \neq herbs, 1 \neq q, \mathrm{Less}(herbs, roast, [1, 2, 3, 4, 5]), herbs \neq Sunrise, \mathrm{Distinct}([Sunrise, \{u1, u2, f\}]),$$
$$\mathrm{Less}(Kevin, Cedar, [1, 2, 3, 4, 5]), Cedar \neq garlic, \mathrm{Less}(Sal, Spring, [1, 2, 3, 4, 5]), \mathrm{Distinct}([Spring, \{u1, u2, f\}]), Pine \neq Banks,$$
$$\mathrm{Less}(Pine, Ware, [1, 2, 3, 4, 5]), \mathrm{Less}(Banks, Lou, [1, 2, 3, 4, 5]), Banks \neq q, Lou \neq q]$$

```
> infolevel['TC']:= 1: infolevel['Constraints']:= 2: infolevel['all']:= 0:
```

```
> lp5:= LogicProblem(V):
```

Okay

```
> st:= time():
```

```
> lp5:-Satisfy(Constraints);
```

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   Spring = Kevin   Depth=   1
```

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

Backup: Guessing q = roast Depth= 2

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Satisfy:

Attempting to fulfill unsatisfied constraints by guessing.

Backup: Guessing q = Lee Depth= 3

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Less/Check: Constraint Less(Turner,Sal,5) completely satisfied.

Less/Check: Constraint Less(Pine,Ware,5) completely satisfied.

Less/Check: Cannot satisfy Less(dessert,Lee,5)

NegateLastGuess: Backing up to last guess. Level = 95 Depth = 3

GoBack: Restoring saved state from before guess q = Lee Depth= 2

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Less/Check: Constraint Less(dessert,Landis,5) completely satisfied.

Less/Check: Cannot satisfy Less(dessert,Lee,5)

NegateLastGuess: Backing up to last guess. Level = 95 Depth = 2

GoBack: Restoring saved state from before guess q = roast Depth= 1

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Satisfy:

Attempting to fulfill unsatisfied constraints by guessing.

Backup: Guessing q = 5 Depth= 2

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Satisfy:

Attempting to fulfill unsatisfied constraints by guessing.

Backup: Guessing q = Cedar Depth= 3

Unsatisfied = [[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Sal*, *Spring*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)]]

Less/Check: Constraint Less(Turner,Sal,5) completely satisfied.

Less/Check: Cannot satisfy Less(dessert,Landis,5)

NegateLastGuess: Backing up to last guess. Level = 95 Depth = 3

GoBack: Restoring saved state from before guess q = Cedar Depth= 2

&<=>: Contradicts roast <> Spring

NegateLastGuess: Backing up to last guess. Level = 90 Depth = 2

GoBack: Restoring saved state from before guess q = 5 Depth= 1

$$Unsatisfied = [[-7, Less(Turner, Sal, 5)], [-7, Less(dessert, Landis, 5)], [-7, Less(Pine, Ware, 5)], [-7, Less(dessert, Lee, 5)], [-7, Less(herbs, roast, 5)], [-7, Less(roast, Cedar, 5)], [-7, Less(Sal, Spring, 5)], [-7, Less(Banks, Lou, 5)], [-7, Less(herbs, Kevin, 5)], [-7, Less(Kevin, Cedar, 5)]]$$

Satisfy:

Attempting to fulfill unsatisfied constraints by guessing.

Backup: Guessing 5 = Cedar Depth= 2

$$Unsatisfied = [[-7, Less(Turner, Sal, 5)], [-7, Less(dessert, Landis, 5)], [-7, Less(Pine, Ware, 5)], [-7, Less(dessert, Lee, 5)], [-7, Less(herbs, roast, 5)], [-7, Less(roast, Cedar, 5)], [-7, Less(Sal, Spring, 5)], [-7, Less(Banks, Lou, 5)], [-7, Less(herbs, Kevin, 5)], [-7, Less(Kevin, Cedar, 5)]]$$

Less/Check: Constraint Less(dessert,Lee,5) completely satisfied.

Less/Check: Constraint Less(herbs,roast,5) completely satisfied.

Less/Check: Constraint Less(roast,Cedar,5) completely satisfied.

Less/Check: Constraint Less(Sal,Spring,5) completely satisfied.

Less/Check: Constraint Less(Banks,Lou,5) completely satisfied.

Less/Check: Constraint Less(herbs,Kevin,5) completely satisfied.

Less/Check: Constraint Less(Kevin,Cedar,5) completely satisfied.

$$Unsatisfied = [[-7, Less(Turner, Sal, 5)], [-7, Less(dessert, Landis, 5)], [-7, Less(Pine, Ware, 5)]]$$

Less/Check: Constraint Less(Turner,Sal,5) completely satisfied.

Less/Check: Constraint Less(dessert,Landis,5) completely satisfied.

Less/Check: Constraint Less(Pine,Ware,5) completely satisfied.

Incomplete solution; need more constraints to complete:

$$\begin{bmatrix} Kevin & Ware & dessert & Spring & 3 & \_ & \_ \\ Kit & Turner & garlic & Sunrise & 1 & \_ & \_ \\ Lee & Banks & roast & Forest & 4 & \_ & f \\ Lou & Landis & New & Cedar & 5 & \_ & \_ \\ Sal & Colson & herbs & Pine & 2 & q & \_ \end{bmatrix}$$

> time()-st;

.872

> lp5:-`&?`(ActiveGuesses);

$$[Spring = Kevin, 5 = Cedar]$$

We have a complete answer to the stated problem. It is not necessarily unique because of the guesses. Let's see if we can prove uniqueness for a restriction of the above problem. Then I will extend the proof to the whole problem. I will eliminate the quitter variable. Obviously, there is no way to distiguish m1 and m2. Let's be definite and say that the man from Sunrise Acres is m1 and the man from Spring Ridge is m2. Also, we can't distinguish u1 and u2. By clue 7, the known woman is not from Cedar Grove. Let's say the person from Cedar Grove is u1. The rest is cut-and-paste from above.

> V2:= ['first','last','subject','city','When','sex'];

$$V2 := [first, last, subject, city, When, sex]$$

> Constraints:=
[Kevin<>garlic #Clue 2
,Less(herbs,Kevin,When) #Clue 3
 # ,Kevin<>q
,Distinct([Kevin, {u1,u2,f}]) #Clue 4
,Lee<>Ware #Clue 5
,Less(dessert,Lee,When) #Clue 6
,Less(roast,Cedar,When), roast=f, Cedar=u1 #Clue 7 #new info added here
,Lee<>Landis #Clue 8
,Less(dessert,Landis,When) #Clue 9
,Less(Turner,Sal,When) #Clue 10
,1<>herbs #Clue 11
 # ,1<>q #Clue 12
,Less(herbs,roast,When) #Clue 13

```
,herbs<>Sunrise, Sunrise=m1 #Clue 14 #new info added here
,Less(Kevin,Cedar,When) #Clue 15
,Cedar<>garlic #Clue 16
,Less(Sal,Spring,When), Spring=m2 #Clue 17 #new info added here
,Pine<>Banks #Clue 18
,Less(Pine,Ware,When) #Clue 19
,Less(Banks,Lou,When) #Clue 20
# ,Banks<>q, Lou<>q #Clue 21
];
```

$$Constraints := [Kevin \neq garlic, \mathrm{Less}(herbs, Kevin, [1, 2, 3, 4, 5]), \mathrm{Distinct}([Kevin, \{u1, u2, f\}]), Lee \neq Ware, \mathrm{Less}(dessert, Lee, [1, 2, 3, 4, 5]), \mathrm{Less}(roast, Cedar, [1, 2, 3, 4, 5]), roast = f, Cedar = u1, Lee \neq Landis, \mathrm{Less}(dessert, Landis, [1, 2, 3, 4, 5]), \mathrm{Less}(Turner, Sal, [1, 2, 3, 4, 5]), 1 \neq herbs, \mathrm{Less}(herbs, roast, [1, 2, 3, 4, 5]), herbs \neq Sunrise, Sunrise = m1, \mathrm{Less}(Kevin, Cedar, [1, 2, 3, 4, 5]), Cedar \neq garlic, \mathrm{Less}(Sal, Spring, [1, 2, 3, 4, 5]), Spring = m2, Pine \neq Banks, \mathrm{Less}(Pine, Ware, [1, 2, 3, 4, 5]), \mathrm{Less}(Banks, Lou, [1, 2, 3, 4, 5])]$$

This shows why it is useful to use the module structure to have two problems active at the same time.

```
> lp5_2:= LogicProblem(V2):
```

$$Okay$$

```
> st:= time():
```

```
> lp5_2:-Satisfy(Constraints);
```

$$Unsatisfied = [[-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)]]$$

$$Unsatisfied = [[-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)]]$$

```
Satisfy:
```

$$Attempting\ to\ fulfill\ unsatisfied\ constraints\ by\ guessing.$$

```
Backup:   Guessing   5 = Cedar   Depth=   1
```

$$Unsatisfied = [[-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)]]$$

$$Unsatisfied = [[-7, Less(Turner, Sal, 5)], [-7, Less(dessert, Lee, 5)], [-7, Less(Pine, Ware, 5)], [-7, Less(Banks, Lou, 5)], [-7, Less(herbs, roast, 5)], [-7, Less(Sal, Spring, 5)], [-7, Less(dessert, Landis, 5)], [-7, Less(Kevin, Cedar, 5)], [-7, Less(roast, Cedar, 5)], [-7, Less(herbs, Kevin, 5)]]$$

$$Unsatisfied = [[-7, Less(Turner, Sal, 5)], [-7, Less(dessert, Lee, 5)], [-7, Less(Pine, Ware, 5)], [-7, Less(Banks, Lou, 5)], [-7, Less(herbs, roast, 5)], [-7, Less(Sal, Spring, 5)], [-7, Less(dessert, Landis, 5)], [-7, Less(Kevin, Cedar, 5)], [-7, Less(roast, Cedar, 5)], [-7, Less(herbs, Kevin, 5)]]$$

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   New = Landis   Depth=   2
```

$$Unsatisfied = [[-7, Less(Turner, Sal, 5)], [-7, Less(dessert, Lee, 5)], [-7, Less(Pine, Ware, 5)], [-7, Less(Banks, Lou, 5)], [-7, Less(herbs, roast, 5)], [-7, Less(Sal, Spring, 5)], [-7, Less(dessert, Landis, 5)], [-7, Less(Kevin, Cedar, 5)], [-7, Less(roast, Cedar, 5)], [-7, Less(herbs, Kevin, 5)]]$$

```
Less/Check:   Constraint   Less(Turner,Sal,5)   completely satisfied.

Less/Check:   Constraint   Less(dessert,Lee,5)   completely satisfied.

Less/Check:   Constraint   Less(Pine,Ware,5)   completely satisfied.

Less/Check:   Constraint   Less(Banks,Lou,5)   completely satisfied.

Less/Check:   Constraint   Less(herbs,roast,5)   completely satisfied.

Less/Check:   Constraint   Less(Sal,Spring,5)   completely satisfied.

Less/Check:   Constraint   Less(dessert,Landis,5)   completely satisfied.

Less/Check:   Constraint   Less(Kevin,Cedar,5)   completely satisfied.

Less/Check:   Constraint   Less(roast,Cedar,5)   completely satisfied.

Less/Check:   Constraint   Less(herbs,Kevin,5)   completely satisfied.
```

$$Unsatisfied = [\ ]$$

```
Satisfy:
```

Complete, consistent solution found. Attempting to prove uniqueness.

```
GoBack:   Restoring saved state from before guess   New = Landis   Depth=   1
```

$$Unsatisfied = [[-7, Less(Turner, Sal, 5)], [-7, Less(dessert, Lee, 5)], [-7, Less(Pine, Ware, 5)], [-7, Less(Banks, Lou, 5)], [-7, Less(herbs, roast, 5)], [-7, Less(Sal, Spring, 5)], [-7, Less(dessert, Landis, 5)], [-7, Less(Kevin, Cedar, 5)], [-7, Less(roast, Cedar, 5)], [-7, Less(herbs, Kevin, 5)]]$$

```
Less/Check:   Constraint   Less(Turner,Sal,5)   completely satisfied.

Less/Check:   Constraint   Less(dessert,Lee,5)   completely satisfied.

Less/Check:   Cannot satisfy   Less(Pine,Ware,5)

NegateLastGuess:   Backing up to last guess.   Level = 95   Depth = 1

GoBack:   Restoring saved state from before guess   5 = Cedar   Depth=   0

&<!=>:   Contradicts   4 = Cedar
```

Unique solution:

$$\begin{bmatrix} Kevin & Ware & dessert & Spring & 3 & m2 \\ Kit & Turner & garlic & Sunrise & 1 & m1 \\ Lee & Banks & roast & Forest & 4 & f \\ Lou & Landis & New & Cedar & 5 & u1 \\ Sal & Colson & herbs & Pine & 2 & u2 \end{bmatrix}$$

```
> time()-st;
```

.330

```
> with(lp5);

Warning, these names have been rebound: &!!, &-, &<, &>, &?, &G, &Soln, CollectStats, ConstNum, Consts, ConstsInV,
DifferentBlock, FreeGuess, GoBack, Guess, InternalRep, IsComplete, IsUnique, NC, NV, PrintConst, Reinitialize, SameBlock,
Satisfy, Separated, VarNum, VarNumC
```

[&!!, &-, &<, &>, &?, &G, &Soln, AutoGuess, CPV, CollectStats, ConstNum, Consts, ConstsInV, DifferentBlock, Equation, FreeGuess, GoBack, Guess, InternalRep, IsComplete, IsUnique, NC, NV, PrintConst, Quiet, Reinitialize, SameBlock, Satisfy, Separated, UniquenessProof, VarNum, VarNumC, X_O]

> **Reinitialize();**

Okay

Now we go back to the original problem using the dummy variable assignments from the restricted problem and putting the quitter constraints back in:

```
> Constraints:=
[Kevin<>garlic #Clue 2
,Less(herbs,Kevin,When) #Clue 3
,Kevin<>q
,Distinct([Kevin, {u1,u2,f}]) #Clue 4
,Lee<>Ware #Clue 5
,Less(dessert,Lee,When) #Clue 6
,Less(roast,Cedar,When), roast=f, Cedar=u1 #Clue 7
,Lee<>Landis #Clue 8
,Less(dessert,Landis,When) #Clue 9
,Less(Turner,Sal,When) #Clue 10
,1<>herbs #Clue 11
,1<>q #Clue 12
,Less(herbs,roast,When) #Clue 13
,herbs<>Sunrise, Sunrise=m1 #Clue 14
,Less(Kevin,Cedar,When) #Clue 15
,Cedar<>garlic #Clue 16
,Less(Sal,Spring,When), Spring=m2 #Clue 17
,Pine<>Banks #Clue 18
,Less(Pine,Ware,When) #Clue 19
,Less(Banks,Lou,When) #Clue 20
,Banks<>q, Lou<>q #Clue 21
];
```

Constraints := [*Kevin* ≠ *garlic*, Less(*herbs*, *Kevin*, [1, 2, 3, 4, 5]), *Kevin* ≠ *q*, Distinct([*Kevin*, {*u1*, *u2*, *f*}]), *Lee* ≠ *Ware*, Less(*dessert*, *Lee*, [1, 2, 3, 4, 5]), Less(*roast*, *Cedar*, [1, 2, 3, 4, 5]), *roast* = *f*, *Cedar* = *u1*, *Lee* ≠ *Landis*, Less(*dessert*, *Landis*, [1, 2, 3, 4, 5]), Less(*Turner*, *Sal*, [1, 2, 3, 4, 5]), 1 ≠ *herbs*, 1 ≠ *q*, Less(*herbs*, *roast*, [1, 2, 3, 4, 5]), *herbs* ≠ *Sunrise*, *Sunrise* = *m1*, Less(*Kevin*, *Cedar*, [1, 2, 3, 4, 5]), *Cedar* ≠ *garlic*, Less(*Sal*, *Spring*, [1, 2, 3, 4, 5]), *Spring* = *m2*, *Pine* ≠ *Banks*, Less(*Pine*, *Ware*, [1, 2, 3, 4, 5]), Less(*Banks*, *Lou*, [1, 2, 3, 4, 5]), *Banks* ≠ *q*, *Lou* ≠ *q*]

> **infolevel['TC']:= 1: infolevel['Constraints']:= 2:**

> **Satisfy(Constraints);**

Unsatisfied = [[-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   Sunrise = Kit   Depth=   1
```

Unsatisfied = [[-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   New = Landis   Depth=   2
```

Unsatisfied = [[-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)],

[-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

```
Less/Check:   Constraint   Less(Pine,Ware,5)   completely satisfied.

Less/Check:   Constraint   Less(dessert,Landis,5)   completely satisfied.

Less/Check:   Constraint   Less(herbs,roast,5)   completely satisfied.

Less/Check:   Constraint   Less(Banks,Lou,5)   completely satisfied.

Less/Check:   Constraint   Less(Turner,Sal,5)   completely satisfied.

Less/Check:   Constraint   Less(Kevin,Cedar,5)   completely satisfied.

Less/Check:   Constraint   Less(dessert,Lee,5)   completely satisfied.

Less/Check:   Constraint   Less(roast,Cedar,5)   completely satisfied.

Less/Check:   Constraint   Less(herbs,Kevin,5)   completely satisfied.

Less/Check:   Constraint   Less(Sal,Spring,5)   completely satisfied.
```

$$Unsatisfied = [\ ]$$

Incomplete solution; need more constraints to complete:

$$\begin{bmatrix} Kevin & Ware & dessert & Spring & 3 & \_ & m2 \\ Kit & Turner & garlic & Sunrise & 1 & \_ & m1 \\ Lee & Banks & roast & Forest & 4 & \_ & f \\ Lou & Landis & New & Cedar & 5 & \_ & u1 \\ Sal & Colson & herbs & Pine & 2 & q & u2 \end{bmatrix}$$

I want to know what was known about the quitter before any guesses were made. Procedure GoBack restores the problem state to what is was before the last guess.

> **to &? CountGuesses do GoBack() end;**

```
GoBack:   Restoring saved state from before guess   New = Landis   Depth=   1
```

$$New = Landis$$

```
GoBack:   Restoring saved state from before guess   Sunrise = Kit   Depth=   0
```

$$Sunrise = Kit$$

The return values of GoBack() are the guesses. I don't care about them for this problem.

Let's see what was known about the quitter:

> **&? q;**

$$q \neq [Kevin, Kit, Lou, Banks, 1]$$

Kevin, Kit, and Lou are obviously distinct entities (they are all first names). There is no way to distinguish n1, n2, n3, and n4. So we arbitrarily assign Kevin=n1, Kit= n2, Lou= n3.

> **st:= time():**

> **Reinitialize();**

$$Okay$$

> **Constraints:=**
[Kevin<>garlic #Clue 2
,Less(herbs,Kevin,When) #Clue 3
,Kevin= n1, Kit= n2, Lou= n3
,Distinct([Kevin, {u1,u2,f}]) #Clue 4
,Lee<>Ware #Clue 5
,Less(dessert,Lee,When) #Clue 6
,Less(roast,Cedar,When), roast=f, Cedar=u1 #Clue 7
,Lee<>Landis #Clue 8
,Less(dessert,Landis,When) #Clue 9
,Less(Turner,Sal,When) #Clue 10
,1<>herbs #Clue 11
,1<>q #Clue 12
,Less(herbs,roast,When) #Clue 13
,herbs<>Sunrise, Sunrise=m1 #Clue 14
,Less(Kevin,Cedar,When) #Clue 15
,Cedar<>garlic #Clue 16
,Less(Sal,Spring,When), Spring=m2 #Clue 17
,Pine<>Banks #Clue 18

,Less(Pine,Ware,When) #Clue 19
,Less(Banks,Lou,When) #Clue 20
,Banks<>q, Lou<>q #Clue 21
];

$$Constraints := [Kevin \neq garlic, \mathrm{Less}(herbs, Kevin, [1,2,3,4,5]), Kevin = n1, Kit = n2, Lou = n3, \mathrm{Distinct}([Kevin, \{u1, u2, f\}]), Lee \neq Ware, \mathrm{Less}(dessert, Lee, [1,2,3,4,5]), \mathrm{Less}(roast, Cedar, [1,2,3,4,5]), roast = f, Cedar = u1, Lee \neq Landis, \mathrm{Less}(dessert, Landis, [1,2,3,4,5]), \mathrm{Less}(Turner, Sal, [1,2,3,4,5]), 1 \neq herbs, 1 \neq q, \mathrm{Less}(herbs, roast, [1,2,3,4,5]), herbs \neq Sunrise, Sunrise = m1, \mathrm{Less}(Kevin, Cedar, [1,2,3,4,5]), Cedar \neq garlic, \mathrm{Less}(Sal, Spring, [1,2,3,4,5]), Spring = m2, Pine \neq Banks, \mathrm{Less}(Pine, Ware, [1,2,3,4,5]), \mathrm{Less}(Banks, Lou, [1,2,3,4,5]), Banks \neq q, Lou \neq q]$$

> **Satisfy(Constraints);**

$$Unsatisfied = [[-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)]]$$

$$Unsatisfied = [[-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)]]$$

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   n1 = Spring   Depth=   1
```

$$Unsatisfied = [[-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)]]$$

$$Unsatisfied = [[-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)]]$$

$$Unsatisfied = [[-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)],$$

[-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Satisfy:

Attempting to fulfill unsatisfied constraints by guessing.

Backup: Guessing 1 = Sunrise Depth= 2

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Satisfy:

Attempting to fulfill unsatisfied constraints by guessing.

Backup: Guessing New = Landis Depth= 3

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Less/Check: Constraint Less(dessert,Lee,5) completely satisfied.

Less/Check: Constraint Less(herbs,roast,5) completely satisfied.

Less/Check: Constraint Less(herbs,Kevin,5) completely satisfied.

Less/Check: Constraint Less(Kevin,Cedar,5) completely satisfied.

Less/Check: Constraint Less(Pine,Ware,5) completely satisfied.

Less/Check: Constraint Less(dessert,Landis,5) completely satisfied.

Less/Check: Constraint Less(roast,Cedar,5) completely satisfied.

Less/Check: Constraint Less(Turner,Sal,5) completely satisfied.

Less/Check: Constraint Less(Sal,Spring,5) completely satisfied.

$$Unsatisfied = [[-7, \mathrm{Less}(Banks, Lou, 5)]]$$

Less/Check: Constraint Less(Banks,Lou,5) completely satisfied.

Satisfy:

Complete, consistent solution found. Attempting to prove uniqueness.

GoBack: Restoring saved state from before guess New = Landis Depth= 2

$$Unsatisfied = [[-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)]]$$

Less/Check: Constraint Less(dessert,Lee,5) completely satisfied.

Less/Check: Constraint Less(herbs,roast,5) completely satisfied.

Less/Check: Constraint Less(Banks,Lou,5) completely satisfied.

Less/Check: Constraint Less(herbs,Kevin,5) completely satisfied.

Less/Check: Constraint Less(Kevin,Cedar,5) completely satisfied.

Less/Check: Cannot satisfy Less(Pine,Ware,5)

NegateLastGuess: Backing up to last guess. Level = 95 Depth = 2

GoBack: Restoring saved state from before guess 1 = Sunrise Depth= 1

&<!=>: Contradicts herbs = Cedar

NegateLastGuess: Backing up to last guess. Level = 90 Depth = 1

GoBack: Restoring saved state from before guess n1 = Spring Depth= 0

```
&<=>:   Contradicts   2 <> Kevin
```

Unique solution:

$$\begin{bmatrix} Kevin & Ware & dessert & Spring & 3 & n1 & m2 \\ Kit & Turner & garlic & Sunrise & 1 & n2 & m1 \\ Lee & Banks & roast & Forest & 4 & n4 & f \\ Lou & Landis & New & Cedar & 5 & n3 & u1 \\ Sal & Colson & herbs & Pine & 2 & q & u2 \end{bmatrix}$$

Voila.

> **time()-st;**

.570

A large portion of the time is often taken by the uniqueness proof. Often a uniqueness proof is unnecessary. For example, you may already know that the solution is unique, or you may not care if the solution is unique. Let's redo the same problem without getting a uniqueness proof.

> **UniquenessProof:= false:**

> **Reinitialize();**

Okay

> **st:= time():**

> **Satisfy(Constraints);**

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   n1 = Spring   Depth=   1
```

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   1 = Sunrise   Depth=   2
```

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

Unsatisfied = [[-7, Less(*dessert*, *Lee*, 5)], [-7, Less(*herbs*, *roast*, 5)], [-7, Less(*Banks*, *Lou*, 5)], [-7, Less(*herbs*, *Kevin*, 5)], [-7, Less(*Kevin*, *Cedar*, 5)], [-7, Less(*Pine*, *Ware*, 5)], [-7, Less(*dessert*, *Landis*, 5)], [-7, Less(*roast*, *Cedar*, 5)], [-7, Less(*Turner*, *Sal*, 5)], [-7, Less(*Sal*, *Spring*, 5)]]

```
Satisfy:
```

Attempting to fulfill unsatisfied constraints by guessing.

```
Backup:   Guessing   New = Landis   Depth=   3
```

$$Unsatisfied = [[-7, \mathrm{Less}(dessert, Lee, 5)], [-7, \mathrm{Less}(herbs, roast, 5)], [-7, \mathrm{Less}(Banks, Lou, 5)], [-7, \mathrm{Less}(herbs, Kevin, 5)], [-7, \mathrm{Less}(Kevin, Cedar, 5)], [-7, \mathrm{Less}(Pine, Ware, 5)], [-7, \mathrm{Less}(dessert, Landis, 5)], [-7, \mathrm{Less}(roast, Cedar, 5)], [-7, \mathrm{Less}(Turner, Sal, 5)], [-7, \mathrm{Less}(Sal, Spring, 5)]]$$

```
Less/Check:   Constraint   Less(dessert,Lee,5)   completely satisfied.

Less/Check:   Constraint   Less(herbs,roast,5)   completely satisfied.

Less/Check:   Constraint   Less(herbs,Kevin,5)   completely satisfied.

Less/Check:   Constraint   Less(Kevin,Cedar,5)   completely satisfied.

Less/Check:   Constraint   Less(Pine,Ware,5)   completely satisfied.

Less/Check:   Constraint   Less(dessert,Landis,5)   completely satisfied.

Less/Check:   Constraint   Less(roast,Cedar,5)   completely satisfied.

Less/Check:   Constraint   Less(Turner,Sal,5)   completely satisfied.

Less/Check:   Constraint   Less(Sal,Spring,5)   completely satisfied.
```

$$Unsatisfied = [[-7, \mathrm{Less}(Banks, Lou, 5)]]$$

```
Less/Check:   Constraint   Less(Banks,Lou,5)   completely satisfied.
```

Complete solution, not necessarily unique:

$$\begin{bmatrix} Kevin & Ware & dessert & Spring & 3 & n1 & m2 \\ Kit & Turner & garlic & Sunrise & 1 & n2 & m1 \\ Lee & Banks & roast & Forest & 4 & n4 & f \\ Lou & Landis & New & Cedar & 5 & n3 & u1 \\ Sal & Colson & herbs & Pine & 2 & q & u2 \end{bmatrix}$$

```
> time()-st;
```

.440

> **&? ShowStats;**

$$Sets = 0,\ Unsets = 0,\ RuleOuts = 0,\ Elims = 0,\ Transfers = 0,\ Pivots = 0,\ MaxLevel = 0,\ Guesses = 0,\ MaxDepth = 0$$

>

A puzzle with a two-dimensional spatial relationships and several complex procedural clues.

"Tailgate Party" in *Dell Champion: The Best of Logic Puzzles* (Feb. 1998).

Ina Rush was caught in a terrific traffic jam last Wednesday around lunch time. Traffic was completely stopped for nearly an hour, but Ina made the best of it by introducing herself to the drivers of the eight cars around hers, and encouraging them to treat the situation as an impromptu picnic. Four of the eight drivers were male (Carl, Joe, Rudy, and Otto), and four were female (Bea, Dee, Elaine, and Emma). From this information and the following clues, match each driver's full name (one last name is Poeler) with the color of his or her car, and locate the car on the chart below.

1 2 3

4 I 5

6 7 8

[The printed chart indicates that Ina is in the middle.]

1. The car just in front of Ina was the same color as the car just to her left, while the car just behind Ina was the same color as the car just to her right.

2. Ms. Kamuter shared some of her lunch with Mr. Blough, who was stopped in the same lane as her, and with Elaine, who was in a different lane.

3. Ms. Stucke passed a can of soda to the driver of a blue car just to her right.

4. Otto shared his potato salad with the two women on his immediate left and right, who were (not necessarily respectively) Ms. Toots and the driver of a gray car.

5. Emma and Ms. Stucke both were driving red cars.

6. Carl (who isn't Mr. Blough) gave his extra slice of apple pie to the driver of a green car that was somewhere ahead of him in the same lane.

7. Ms. Trapt and Dee (neither of whom drove a red car) traded half of their sandwiches with each other.

8. Rudy and Mr. Cruize were driving black cars.

9. Mr. Horne shared a small carton of ice cream with the driver of a white car,

With a careful reading, we see that the colors are 2 red, 1 blue, 2 black, 1 white, 1 gray, and 1 green. This is the type of thing that must be determined by the user before the Logic Problem program can even begin to be used.

```
> V:= ['first', 'last', 'color', 'place']:

> Men:= {Carl, Joe, Rudy, Otto}: Women:= {Bea, Dee, Elaine, Emma}:

> first:= [op(Men), op(Women)]:

> last:= [Poeler, Kamuter, Blough, Stucke, Toots, Trapt, Cruize, Horne]:

> color:= [red1, red2, blue, black1, black2, white, gray, green]:

> setattribute(black1, black): setattribute(black2, black):

> setattribute(red1, red): setattribute(red2, red):

> place:= [$1..8]:

> Row1:= {1,2,3}: Row3:= {6,7,8}: Lane1:= {1,4,6}: Lane2:= {2,7}: Lane3:= {3,5,8}:

> Rows:= [Row1, Row3]: Lanes:= [Lane||(1..3)]:
```

We won't need Row2.

```
> Cars:= LogicProblem(V):
```

Okay

```
> with(Cars):

Warning, these names have been rebound: &!!, &-, &<, &>, &?, &G, &Soln, CPV, ConstNum, Consts, ConstsInV, DifferentBlock,
FreeGuess, GoBack, Guess, InternalRep, IsComplete, IsUnique, NC, NV, PrintConst, Reinitialize, SameBlock, Satisfy, Separated,
UniquenessProof, VarNum, VarNumC
```

We need this "Proc" type constraint for clue 1.

```
> SameColor:= proc(M, positions)
local pos1, pos2;
use `&Soln`= M:-`&Soln`, VarNum= M:-VarNum, ConstNum= M:-ConstNum, Consts= M:-Consts in
pos1:= ConstNum(positions[1]) &Soln VarNum(color);
```

```
pos2:= ConstNum(positions[2]) &Soln VarNum(color);
if pos1=0 and pos2=0 then
false, false
elif pos1<>0 and pos2<>0 then
if attributes(Consts[pos1]) = attributes(Consts[pos2]) then false,true else true fi
elif pos1<>0 then
if pos1 = ConstNum(black1) then false, false, [positions[2] = black2]
elif pos1 = ConstNum(black2) then false, false, [positions[2] = black1]
elif pos1 = ConstNum(red1) then false, false, [positions[2] = red2]
elif pos1 = ConstNum(red2) then false, false, [positions[2] = red1]
else true
fi
else #pos2<>0
if pos2 = ConstNum(black1) then false, false, [positions[1] = black2]
elif pos2 = ConstNum(black2) then false, false, [positions[1] = black1]
elif pos2 = ConstNum(red1) then false, false, [positions[1] = red2]
elif pos2 = ConstNum(red2) then false, false, [positions[1] = red1]
else true
fi
fi
end use
end:

> Constraints:=
[#Classify by sex:
Distinct([{Kamuter, Stucke, Toots, Trapt}, Men])
,Distinct([{Horne, Poeler, Blough, Cruize}, Women])

#Clue 1:
,Distinct([{2, 4, 5, 7}, {green, gray, white, blue}])
,Proc(SameColor, [2,4]), Proc(SameColor, [5,7])

#Clue 2:
,Kamuter<>Elaine
#Here I introduce the partition/membership constraint types SameBlock and DifferentBlock.
,Rel(SameBlock, Kamuter, Blough, place, Lanes)
,Rel(DifferentBlock, Kamuter, Elaine, place, Lanes)
,Rel(DifferentBlock, Blough, Elaine, place, Lanes)

#Clue 3
,Distinct([Stucke, {4} union Lane3])
,Distinct([blue, {5} union Lane1])
#We can use a "Succ" constraint to specify that the blue car is just to the right of Stucke.
#But note that this only works because of the limitations already placed on their positions by the two Distinct constraints.
,Succ(blue, Stucke, place)
```

```
#Clue 4
,Otto<>gray
,Distinct([Otto, Lane1 union Lane3])
,Distinct([Toots, gray, 2,7,4,5])
,Distinct([gray, Men])
,Rel(SameBlock, [Otto, Toots, gray], place, Rows)

#Clue 5
,Emma=red1, Stucke=red2

#Clue 6
,Carl<>Blough, Carl<>green
,Rel(SameBlock, Carl, green, place, Lanes)
,Distinct([Carl, Row1])
,Distinct([green, Row3])

#Clue 7
,Distinct([Trapt,Dee,red1,red2])

#Clue 8
,Rudy=black1, Cruize=black2

#Clue 9
,Horne<>white

#The next constraint illustrates how the Dummy constraint type can be used to print out whatever infomation the user wants.
#It will be called before each pass through the Unsatisfied list.
,Dummy(proc(M) use `&?`= M:-`&?` in print(&? ShowStats) end; false, false end, [])
];
```

```
Constraints := [Distinct([{Kamuter, Trapt, Toots, Stucke}, {Otto, Rudy, Carl, Joe}]),
Distinct([{Poeler, Horne, Cruize, Blough}, {Dee, Emma, Elaine, Bea}]), Distinct([{2, 4, 5, 7}, {gray, white, green, blue}]), Proc(SameColor, [2, 4]),
Proc(SameColor, [5, 7]), Kamuter ≠ Elaine, Rel(SameBlock, Kamuter, Blough, [1, 2, 3, 4, 5, 6, 7, 8], [{1, 4, 6}, {2, 7}, {3, 5, 8}]),
Rel(DifferentBlock, Kamuter, Elaine, [1, 2, 3, 4, 5, 6, 7, 8], [{1, 4, 6}, {2, 7}, {3, 5, 8}]),
Rel(DifferentBlock, Blough, Elaine, [1, 2, 3, 4, 5, 6, 7, 8], [{1, 4, 6}, {2, 7}, {3, 5, 8}]), Distinct([Stucke, {3, 4, 5, 8}]), Distinct([blue, {1, 4, 5, 6}]),
Succ(blue, Stucke, [1, 2, 3, 4, 5, 6, 7, 8]), Otto ≠ gray, Distinct([Otto, {1, 3, 4, 5, 6, 8}]), Distinct([Toots, gray, 2, 7, 4, 5]),
Distinct([gray, {Otto, Rudy, Carl, Joe}]), Rel(SameBlock, [Otto, Toots, gray], [1, 2, 3, 4, 5, 6, 7, 8], [{1, 2, 3}, {6, 7, 8}]), Emma = red1,
Stucke = red2, Carl ≠ Blough, Carl ≠ green, Rel(SameBlock, Carl, green, [1, 2, 3, 4, 5, 6, 7, 8], [{1, 4, 6}, {2, 7}, {3, 5, 8}]),
Distinct([Carl, {1, 2, 3}]), Distinct([green, {6, 7, 8}]), Distinct([Trapt, Dee, red1, red2]), Rudy = black1, Cruize = black2, Horne ≠ white,
Dummy(proc(M) print(M:-`&?`(ShowStats)); false, false end proc, [])]
```

```
> infolevel['Constraints']:= 0: infolevel['TC']:= 0: infolevel['TCdisplay']:= 0: CollectStats:= true:

> Reinitialize():

> st:= time():

> Satisfy(Constraints);
```

$$Sets = 0,\ Unsets = 0,\ RuleOuts = 0,\ Elims = 0,\ Transfers = 0,\ Pivots = 0,\ MaxLevel = 0,\ Guesses = 0,\ MaxDepth = 0$$

$$Sets = 4,\ Unsets = 110,\ RuleOuts = 8,\ Elims = 267,\ Transfers = 220,\ Pivots = 8,\ MaxLevel = 227,\ Guesses = 0,\ MaxDepth = 0$$

$$Sets = 6,\ Unsets = 116,\ RuleOuts = 14,\ Elims = 289,\ Transfers = 232,\ Pivots = 14,\ MaxLevel = 314,\ Guesses = 1,\ MaxDepth = 1$$

$$Sets = 6,\ Unsets = 120,\ RuleOuts = 14,\ Elims = 297,\ Transfers = 240,\ Pivots = 14,\ MaxLevel = 314,\ Guesses = 1,\ MaxDepth = 1$$

$$Sets = 12,\ Unsets = 124,\ RuleOuts = 30,\ Elims = 349,\ Transfers = 246,\ Pivots = 30,\ MaxLevel = 575,\ Guesses = 2,\ MaxDepth = 2$$

$$Sets = 27,\ Unsets = 153,\ RuleOuts = 94,\ Elims = 523,\ Transfers = 284,\ Pivots = 94,\ MaxLevel = 621,\ Guesses = 2,\ MaxDepth = 2$$

$$Sets = 27,\ Unsets = 160,\ RuleOuts = 96,\ Elims = 541,\ Transfers = 296,\ Pivots = 96,\ MaxLevel = 621,\ Guesses = 2,\ MaxDepth = 2$$

$$Sets = 34,\ Unsets = 177,\ RuleOuts = 124,\ Elims = 625,\ Transfers = 324,\ Pivots = 124,\ MaxLevel = 720,\ Guesses = 2,\ MaxDepth = 2$$

$$Sets = 37,\ Unsets = 185,\ RuleOuts = 156,\ Elims = 660,\ Transfers = 332,\ Pivots = 156,\ MaxLevel = 720,\ Guesses = 2,\ MaxDepth = 2$$

$$Sets = 39,\ Unsets = 186,\ RuleOuts = 174,\ Elims = 667,\ Transfers = 332,\ Pivots = 174,\ MaxLevel = 720,\ Guesses = 2,\ MaxDepth = 2$$

$$Sets = 40,\ Unsets = 193,\ RuleOuts = 180,\ Elims = 690,\ Transfers = 344,\ Pivots = 180,\ MaxLevel = 720,\ Guesses = 2,\ MaxDepth = 2$$

$$Sets = 46,\ Unsets = 196,\ RuleOuts = 196,\ Elims = 745,\ Transfers = 348,\ Pivots = 196,\ MaxLevel = 720,\ Guesses = 3,\ MaxDepth = 2$$

$$Sets = 61,\ Unsets = 224,\ RuleOuts = 260,\ Elims = 922,\ Transfers = 384,\ Pivots = 260,\ MaxLevel = 720,\ Guesses = 3,\ MaxDepth = 2$$

$$Sets = 61,\ Unsets = 231,\ RuleOuts = 262,\ Elims = 940,\ Transfers = 396,\ Pivots = 262,\ MaxLevel = 720,\ Guesses = 3,\ MaxDepth = 2$$

$$Sets = 68,\ Unsets = 248,\ RuleOuts = 290,\ Elims = 1024,\ Transfers = 424,\ Pivots = 290,\ MaxLevel = 720,\ Guesses = 3,\ MaxDepth = 2$$

$$Sets = 73,\ Unsets = 257,\ RuleOuts = 340,\ Elims = 1066,\ Transfers = 432,\ Pivots = 340,\ MaxLevel = 720,\ Guesses = 3,\ MaxDepth = 2$$

Unique solution:

$$\begin{bmatrix} Otto & Cruize & black2 & 2 \\ Rudy & Horne & black1 & 4 \\ Carl & Poeler & white & 6 \\ Joe & Blough & blue & 8 \\ Dee & Toots & green & 1 \\ Emma & Kamuter & red1 & 5 \\ Elaine & Stucke & red2 & 7 \\ Bea & Trapt & gray & 3 \end{bmatrix}$$

> **time()-st;**

.660

In this exmaple, we have seen how the program can handle multi-dimensional relationships. Please proceed to the worksheet ColorASquare.mws for a significantly more complex and more abstract two-dimensional problem.

>