600.432/632 Declarative Methods - Prof. Jason Eisner
Answers to "Prolog Basics" discussion problems

1a. Unification succeeds:
A=3,
B=4,
C=baz(4)

1b. Unification succeeds:
Q=bar(4),
P=bar(4)

1c. Unification succeeds:
C=D,
D=C

1d. Does not unify. From the second argument:

f(C)=f(D)

which implies

C=D

but the first argument states that

C=f(f(D))

so C must be deeper than D.

HOWEVER, implementations of Prolog that skip the
occurs check will allow this unification to succeed,
with

C=D=X=f(f(f(f(...))))

1e. Unification succeeds:
A=C=g(3,D)
B=3
D is not further known (free variable)

1f. This "should" fail to unify, since there is no (finite) term X
such that foo(2,bar(3,X)) would equal foo(2,X).

However, in a Prolog implementation that skips the "occurs" check,
it will unify anyway, resulting in a cyclic data structure that
represents an infinite term. For example, ECLiPSe gives:

X = bar(3, bar(3, bar(3, bar(3, bar(...)))))

The "occurs check" will detect when a variable X is being unified
with a complex term in which X occurs. The simplest example is
X=f(X). This check is somewhat expensive, so most Prolog
implementations do not do it.

2a. The query is: loves(X,company).

2b. !!! WATCH OUT FOR VARIABLES !!!

John and Mary are capitalized, so Prolog will treat them as VARIABLES.

So, the first rule means that everything loves everything

The first answer will be: X = _G123, Yes;
or perhaps: X = X, Yes

In other words, the answer is that X loves company for any X.

If you hit ; and backtrack, you get the second answer,

```
X = misery
Yes
```

3a. The query is: plusone(Y,X), plusone(Z,Y)

3b. There are three solutions:

```
X=3, Y=2, Z=1 ;
X=4, Y=3, Z=2 ;
X=5, Y=4, Z=3
```

3c. One approach is to add additional rules:

```
indomain(1).
indomain(2).
indomain(3).
indomain(4).
indomain(5).
```

Then you can write the constraints as:

```
indomain(X), indomain(Y), indomain(Z).
```

3d. Let's consider some options:

Option A:

```
indomain(X), indomain(Y), indomain(Z), plusone(Y,X), plusone(Z,Y).
```

Prolog will backtrack through the 5*5*5 possible in-domain triples. For each one, it will test one or both of the plusone constraints in mode +,+, which takes time O(1) if the plusone facts are well-indexed, or longer if they're not.

Option B:

```
plusone(Y,X), plusone(Z,Y), indomain(X), indomain(Y), indomain(Z).
```

Prolog will backtrack through 2000 X,Y pairs. For each of those, it will consider one possible value for Z (namely Y-1), invoking the plusone constraint in mode -,+ to find that value of Z. Finding this value in O(1) time would require SECOND-argument indexing; many Prologs have only first-argument indexing and so will instead iterate over all 2000 of the plusone clauses. Once we have X,Y,Z, Prolog will test them to see if they're all in the domain 1..5 (usually they're not).

Option C:

```
plusone(Z,Y), plusone(Y,X), indomain(X), indomain(Y), indomain(Z)
```

This is a bit better because now the second plusone constraint is called in mode +,- rather than -,+, so can take advantage of first-argument indexing. But it still has to start by iterating over 2000 pairs.

Option D:

```
indomain(Z), plusone(Z,Y), indomain(Y), plusone(Y,X), indomain(X)
```

This is the best order! Try each of the 5 in-domain values of

Z. For each one, find the Y that equals Z+1. If that is out-of domain (e.g., Z=5, Y=6), quit. Otherwise, find the X that equals Y+1, and test that that is in-domain.

4a. The drawing looks like a garden trellis:
- a points to all of b,c,d
- each of b,c,d points to all of e,f,g
- each of e,f,g points to all of h,i,j, and so on.

4b. edge is a constraint on X,Y. It is the result of joining the three constraints shown in the body of the edge rule (plusone and two copies of vertex), and projecting this onto the XY plane.

4c. The first thing to realize is that variable names are local to a rule or a query. We could have written more sensibly

```
path(X,X).
path(X,Z) :- edge(X,Y), path(Y,Z).
```

and asked about the query

```
path(Anywhere,e).
```

There are 7 answers: e, b, c, d, a, a, a.
Note that a will be found 3 times because there are 3 paths from a to e.

Path(Start,n) will have a LOT of answers because the choices multiply out exponentially.

Prolog will be pretty inefficient in answering the query path(Anywhere,e), because it first tries to satisfy the subgoal edge(Anywhere,Y), which matches ALL of the edges, not just those that lead to e. Think about what happens after that ...