**CONMAN REFINEMENT**
William Bricken
October 1987

We will have to put the data structure intelligence in, about Xi = Xj.

**ABOUT EQUAL** in the Possibility Calculus

We need typing across the variables, between letters and Carrys. The meaning of = changes depending on variable types.

Basically, Ci variables have no equality based constraints.

(type a) = letter and (type b) = letter and a = b
            ==> contradiction

(type a) = letter and (type b) = letter and a =/= b
            ==> tautology

These could also be implemented as special cases of linear equations:

(m*a, n*b, k*1) in standardized form,

      if k=0 and m+n=0, then fail.

=/= is then expressed as (not (if k=0 and m+n=0, then fail))

In terms of *if changes* demons in the data structure:

if Xi = {n}  (that is, the possibility list has one member)
      then for Xj (j =/= i), post Xj =/= n.

We could also put the termination condition into the if changes demons:

if Xi = { },
      then fail.

**INTEGERIZATION FUDGE**

Yes, a restriction of expressability. When we extend to ranges over continuous variables, possibility computation isn't as powerful.

## PICKING CONDITIONAL CONSTRAINTS

My feeling is that the same selection criterion that order all constraints should order conditional constraints by looking at the *then-constraint*.

So the ordering is:

```
Non-conditional of type A
Conditional with consequent of type A
Non-conditional of type B, B > A
...
```

The idea is to interleaf constraints and search by some criteria external to both.

## TYPESCRIPT FILE

Yes, the system falls into brute force search without the knowledge that e = n is not explorable. We do need smarts on the data structure.

I suspect the best approach is to put some processing capability into the data structure itself, rather than posting Xi =/= n constraints.

The philosophy is that the data structure permits a limited expressability. The *definition* of a letter is that it is unique in value. The letter possibility matrix data structure enforces definitions.

So the DATA-DEMONS:

```
Xi:   if Xi = { },
            then fail.

Xi:   if Xi = {n},
            then (modify-data-structure: all Xj, j =/= i, Xj =/= n)
```

DATA-DEMONS are just like if-changes demons except they send their constraints directly to the data structure, rather than to the constraint list. (I don't know how failure is handled, the first demon may be able to post FAIL to constraints, if you prefer.)

And in the possibility calculus, as a *filter on valid expressions*, not as a computable result:

```
Xi = Xj, i =/= j  ==> fail
Xi = Xj, i = j  ==> succeed (trivially, ie no changes)
Xi =/= Xj, i =/= j  ==> succeed (trivially)
Xi =/= Xj, i = j  ==>  fail
```