analogy, java version

COMPILATION

1: Go to https://github.com/gsfk/Analogy_java and click on "clone or download" to download all files.

2: Download the theorem prover, Prover9. Go to https://www.cs.unm.edu/~mccune/mace4/download/ and select the most recent version.

3: Compile Prover9 using the instructions found on the download site.

4: Find your *absolute* path to the Prover9 executable. The simplest way is probably to navigate your way there in Terminal, then type the command pwd to print the full path, then copy it. It should be something along the lines of /Users/me/Documents/stuff/junk/LADR-2009-11A/bin/

5: The implementation needs to know where your copy of Prover9 is. Open the file "ProverPath.java" and replace the dummy path /your/path/to/prover9 with your full path to Prover9, and ending with the name of the executable itself, something like /Users/Me/Documents/stuff/LADR-2009-11A/bin/prover9 In the long term I hope to make this step easier.

6. To compile, type javac analogy.java. To run, type java analogy followed by the names of input file(s), see below.

OPERATION

When given a text file as representing a structure as input, analogy produces a list of true formulas for that structure, followed by a minimal subset.

More interesting output is produced by giving two files as input. The formulas for both input structures are given and compared, along with their respective minimal sets.

Several example input files are included. Example operations:

java analogy 3ng.txt
gives formulas and axioms for the "3-no-group" example from the paper "Two Ways of Analogy."

java analogy mod3.txt 3ng.txt
compares the 3-no-group with a group of the same order

java analogy g10acyclic.txt g10cyclic.txt
compares the cyclic group of order 10 with the acyclic group

java analogy 6.2.txt mod5.txt
compares an abelian (commutative) group with a non-abelian one.

In all cases you can add "-noprover" as a final argument to disconnect the theorem prover, but expect less than exciting output.

analogy will create input files for the prover in your current working directory, you can safely ignore these. For best results run the program from the folder where it resides.

MAKING YOUR OWN FILES

Input files should follow the format of the example files:

```
Elements:  0, 1, 2

Relation:  *

Facts:
(0,0,0)
(0,1,1)
(0,2,2)
(1,0,1)
(1,1,2)
(1,2,0)
(2,0,2)
(2,1,0)
(2,2,1)
end
```

Note the following limitations:

Elements should be separated by a comma and a space. Spaces are not necessary for facts.

Elements and relation names can be strings rather than single characters, they should not contain spaces or commas.

The implementation currently works only for binary operators, so all facts should contain 3 elements.

All example input uses "$*$" as the relation name. You are free to use a different name in your input, but do not try to compare axioms for input with different relation names.