EXTENDING A THEOREM PROVER

4

Program: name and where it is to be found
On Gould: /ai/ai2teach/projects/prover

Description of the program
It is a heuristic search resolution theorem prover.

References
"Computer Modelling of Mathematical Reasoning", A.Bundy, Academic Press, 1983, especially appendix 1 which documents and lists the program. All references below refer to this book. Copies of appendix are available from your tutor.

Specific questions/tasks to be addressed
a. Run the program on the simple example provided.

b. Comment the program more thoroughly, i.e. roughly one comment per line, rather than one per procedure.

c. Prepare 6 more examples and run them on the program. Suggestions can be found in the book. Include a non-Horn clause example, e.g. page 73 (6.6).

Proposals for extending the project
1) Impose the input restriction and run the program on the previous examples. Compare the behaviour.

2) Experiment with alternative versions of the evaluation function, see page 78 (6.7.3).

3) Modify the unification algorithm to include the occurs check: see page 241 (17.3) and Mark Uribe's 1983/4 AI2 project.

4) Modify the theorem prover to do paramodulation as well as resolution and factoring, see page 62 (5.3).

5) Follow up any ideas that you may generate yourself.

[set by Alan Bundy, SB]

EXPERT SYSTEMS/PROLOG

1

Program: name and where it is to be found
On Gould: /ai/ai2teach/projects/ks299

Description of the program
Simple expert system shell - with backchaining, certainty factors, limited help, trace, explanation.

References
See the documentation on-line in the ks299 directory.
Mycin - in Artificial Intelligence 8 (1977) pp 15-45
by Davis, Buchanan and Shortliffe

Specific questions/tasks to be addressed
a. Write a one/two page summary of how the expert system shell works.

b. Construct and test a new knowledge base for the shell. Use it in addition to the original one to test your modifications to the shell.

c. Do exercise (b) and one of exercises (h) to (k) from Peter Ross's handout of projects and exercises to do with KS299 (enclosed).

d. Do a critical analysis of your system. Describe its strengths/weaknesses. Show how it could be made better. (Be explicit, don't make sweeping essentially content-free generalisations.)

Proposals for extending the project
1) Elaborate the knowledge base. Interesting interaction between the knowledge bits, rules, etc. counts more than simply having lots of things and lots of rules.

2) Address questions (c) to (f) on the sheet.
Useful features include:
conflict resolution strategies
meta-rules
better explanation

3) Address other exercises from (h) to (o). It is better to do a few in depth rather than many superficially.

[set by Mike Uschold, South Bridge]

MATCHING 2D BOUNDARY DESCRIPTIONS OF FLAT POLYGONAL OBJECTS

Program: name and where it is to be found
The program does not yet exist.

Description of the program
This program will match 2d boundary descriptions of flat polygonal objects. A data description consists of a set of straight line segments. A model description consists of either a set of straight line segments, or line segments plus larger structures defined by straight line segments. Because the data position and orientation may be different from that of the model, the object's position and orientation will need to be estimated. This project will match the models to the data and extract the location and orientation.

References
AI2 vision notes.

Specific questions/tasks to be addressed
a. Write the program to match the above specification.

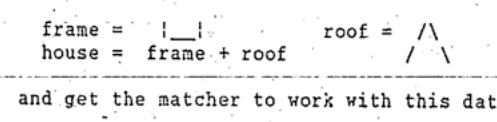
b. Define a small set of polygonal models in terms of line segments only. Specify an instance of each model as data at different positions and orientations.

c. Get your matcher program to correctly match, locate and orient this data.

Proposals for extending the project
1) Extend the matcher to allow a specified amount of point deviation due to noise.

2) Extend the matcher to match curved segments.

3) Change models to have hierarchical structure as:



and get the matcher to work with this data.

[set by Bob Fisher, FHill]

UNDERSTANDING A COMPUTER LANGUAGE: LOGO

Program: name and where it is to be found
Understanding A Computer Language: LOGO
(program does not exist)

Description of the program
The language LOGO is now beginning to be used quite a lot in UK schools. It is a language specially designed to be easy to learn, yet powerful, and it is based loosely on LISP. The most common introduction to LOGO is through using the turtle graphics commands it provides, with some other basic ones such as PRINT and MAKE:

```
FORWARD 100
RIGHT 90
PENDOWN
REPEAT 3 [FORWARD 100 RIGHT 90]
MAKE 'X 60
REPEAT 6 [FORWARD 50 LEFT :X]
```

(note: LOGO does not require any form of declaration, e.g. MAKE 'X 3 will if necessary create a variable called X first.)
Variables have no type - there are only three data types anyway, namely 'word' (such as 'Hello)
'number' (such as 23.417)
and 'list ' (an ordered collection of data, such as [Hello 23.417 [More stuff] 66])).
A good LOGO provides several hundred basic commands, but only a very small subset gets used by beginners. However, beginners do use the procedure-building mechanism:

```
TO TRIANGLE :SIDE
  REPEAT 3 [FORWARD :SIDE RIGHT 120]
END
TRIANGLE 93
```

They tend to make simple mistakes, such as
- using a procedure before they have defined it: TRINGLE 93
- referring to a non-existant variable: TRIANGLE :XX
- missing out a bracket: REPEAT 3 PRINT 'HELLO]
- giving the wrong number of arguments: TRIANGLE 9 3
- omitting some important syntactic marker, such as a colon or quote (you'd need to know a little more about LOGO to know what they do)

Task
ò Devise a Prolog program, probably devised round a Definite Clause Grammar, to spot a variety of such simple mistakes, and perhaps offer SENSIBLE comments.

References
- loads of books on LOGO, in the Dept. library or in Thins
- lots of local LOGO expertise; ask Peter Ross
- loads of transcripts of LOGO work by schoolchildren, all in machine-readable form

[set by Peter Ross, SB]