[holmes overview in the 3rd edition that was cut on the fourth]

This book does not cover holmes in detail due to lack of space; see the *PP4E\AI\ExpertSystem* directory in this book's examples distribution for its code and documentation. But by way of introduction, holmes is an inference engine that performs forward and backward chaining deduction on rules that you supply. For example, the rule:

```
rule pylike if ?X likes coding, ?X likes spam then ?X likes Python
```

can be used both to prove whether someone likes Python (backward, from "then" to "if"), and to deduce that someone likes Python from a set of known facts (forward, from "if" to "then"). Deductions may span multiple rules, and rules that name the same conclusion represent alternatives. holmes also performs simple pattern-matching along the way to assign the variables that appear in rules (e.g., `?X`), and it is able to explain its work.

To make all of this more concrete, let's step through a simple holmes session. The `+=` interactive command adds a new rule to the rule base by running the rule parser, and `@@` prints the current rule base:

```
C:..\PP4E\Ai\ExpertSystem\holmes\holmes>python holmes.py
-Holmes inference engine-
holmes> += rule pylike if ?X likes coding, ?X likes spam then ?X likes Python
holmes> @@
rule pylike if ?X likes coding, ?X likes spam then ?X likes Python.
```

Now, to kick off a backward-chaining proof of a goal, use the `?-` command. A proof explanation is shown here; holmes can also tell you why it is asking a question. Holmes pattern variables can show up in both rules and queries; in rules, variables provide generalization; in a query, they provide an answer:

```
holmes> ?- mel likes Python
is this true: "mel likes coding" ? y
is this true: "mel likes spam" ? y
yes: (no variables)

show proof ? yes
  "mel likes Python" by rule pylike
      "mel likes coding" by your answer
      "mel likes spam" by your answer
more solutions? n

holmes> ?- ann likes ?X
is this true: "ann likes coding" ? y
is this true: "ann likes spam" ? y
yes: ann likes Python
```

Forward chaining from a set of facts to conclusions is started with a `+-` command. Here, the same rule is being applied but in a different way:

```
holmes> +- chris likes spam, chris likes coding
I deduced these facts...
    chris likes Python
I started with these facts...
    chris likes spam
    chris likes coding
time: 0.0
```

More interestingly, deductions chain through multiple rules when part of a rule's "if" is mentioned in another rule's "then":

```
holmes> += rule 1 if thinks ?x then human ?x
holmes> += rule 2 if human ?x then mortal ?x
holmes> ?- mortal bob
is this true: "thinks bob" ? y
yes: (no variables)

holmes> +- thinks bob
```

```
I deduced these facts...
    human bob
    mortal bob
I started with these facts...
    thinks bob
time: 0.0
```

Finally, the `@=` command is used to load files of rules that implement more sophisticated knowledge bases; the rule parser is run on each rule in the file. Here is a file that encodes animal classification rules (other example files are available in the book's examples distribution, if you'd like to experiment):

```
holmes> @= ..\kbases\zoo.kb
holmes> ?- it is a penguin
is this true: "has feathers" ? why
to prove "it is a penguin" by rule 17
this was part of your original query.
is this true: "has feathers" ? y
is this true: "able to fly" ? n
is this true: "black color" ? y
yes: (no variables)
```

Type `stop` to end a session and `help` for a full commands list; see the text files in the holmes directories for more details.