CS 61A Week 15

Topic: Logic programming

Reading: Abelson & Sussman, Section 4.4.1–3

We are not assigning section 4.4.4, which discusses the implementation of the query system in detail. Feel free to read it just for interest; besides deepening your understanding of logic programming, it provides an example of using streams in a large project.

Homework:

Abelson & Sussman, exercises 4.56, 4.57, 4.58, 4.65

For all problems that involve writing queries or rules, test your solutions. To run the query system and load in the sample data:

```
% stk
> (load "~cs61a/lib/query.scm")
> (initialize-data-base microshaft-data-base)
> (query-driver-loop)
```

You're now in the query system's interpreter. To add an assertion:

```
(assert! (foo bar))
```

To add a rule:

```
(assert! (rule (foo) (bar)))
```

Anything else is a query.

Extra for experts:

The lecture notes for this week describe rules that allow inference of the `reverse` relation in one direction, i.e.,

```
;;; Query input:
(forward-reverse (a b c) ?what)

;;; Query results:
(FORWARD-REVERSE (A B C) (C B A))
```

Continued on next page.

Week 15 continued...

```
;;; Query input:
(forward-reverse ?what (a b c))

;;; Query results:
```
... infinite loop

or

```
;;; Query input:
(backward-reverse ?what (a b c))

;;; Query results:
(BACKWARD-REVERSE (C B A) (A B C))

;;; Query input:
(backward-reverse (a b c) ?what)

;;; Query results:
```
... infinite loop

Define rules that allow inference of the `reverse` relation in both directions, to produce the following dialog:

```
;;; Query input:
(reverse ?what (a b c))

;;; Query results:
(REVERSE (C B A) (A B C))

;;; Query input:
(reverse (a b c) ?what)

;;; Query results:
(REVERSE (A B C) (C B A))
```

Unix feature of the week: `perl`, `awk`, `sed`

Emacs feature of the week: `M-x shell-command-on-region`