61A Lecture 31

Wednesday, November 20

Announcements

- Project 4 due Thursday 11/21 @ 11:59pm.
- Extra reader office hours in 405 Soda this week.
  - Wednesday: 5:30pm–7pm
  - Thursday: 5:30pm–7pm
- Homework 10 due Tuesday 11/26 @ 11:59pm.
- Recursive art contest entries will be due Monday 12/2 @ 11:59pm (After Thanksgiving).

Declarative Languages

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

toy_id	toy	color	cost	weight
2	whiffleball	yellow	2.20	0.40
5	frisbee	yellow	1.50	0.20
10	yoyo	yellow	1.50	0.20

Each row is a record

SQL is an example of a declarative programming language.

It separates *what* to compute from *how* it is computed.

The language interpreter is free to compute the result in any way it wants.

http://www.headfirstlabs.com/sql_hands_on/

Declarative Programming

Characteristics of **declarative languages**:

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

The Logic Language

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
- Expressions and relations are Scheme lists.
- For example, **(likes john dogs)** is a relation.
- Implementation fits on a single sheet of paper (next lecture).

Today's theme:

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f38833015388dfa8f19970b-800wi

Simple Facts

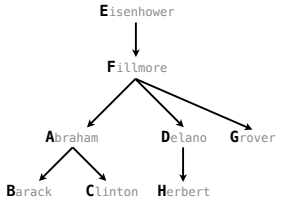
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
```



Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3) **1**

(add 1 ? 3) **2**

(add 1 2 ?) **3**

(? 1 2 3) **add**

Queries

Queries

A *query* contains one or more relations that may contain variables.

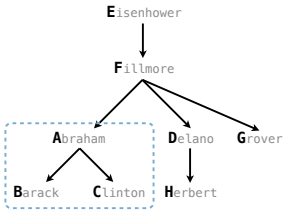
Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
Success!
puppy: barack
puppy: clinton
```

A variable can have any name

Each line is an assignment of variables to values



(Demo)

Compound Facts and Queries

Compound Facts

A fact can include multiple relations and variables as well.

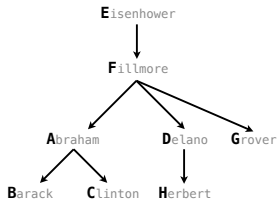
(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.
logic> (query (child ?kid fillmore))
Success!
kid: abraham
kid: delano
kid: grover
```



Compound Queries

An assignment must satisfy all relations in a query.

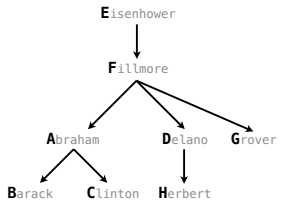
(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham

logic> (query (child ?y ?x)
              (child ?x eisenhower))
Success!
y: abraham    x: fillmore
y: delano     x: fillmore
y: grover     x: fillmore
```



Recursive Facts

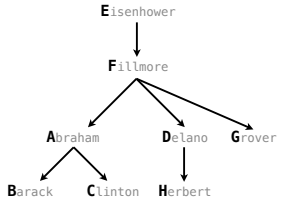
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (query (ancestor ?a barack)
              (ancestor ?a herbert))
Success!
a: fillmore
a: eisenhower
```



Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore ⇦
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

(parent delano herbert)     ; (1), a simple fact
(ancestor delano herbert)   ; (2), from (1) and the 1st ancestor fact
(parent fillmore delano)    ; (3), a simple fact
(ancestor fillmore herbert) ; (4), from (2), (3), & the 2nd ancestor fact
```


Hierarchical Facts

Hierarchical Facts

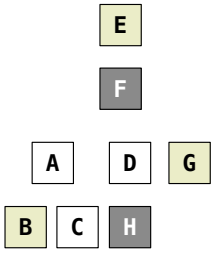
Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
Success!
color: white

logic> (query (dog (name clinton) ?stats))
Success!
stats: (color white)
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
              (dog (name ?y) (color ?fur)))
Success!
x: barack    fur: tan     y: eisenhower
x: clinton   fur: white   y: abraham
x: grover    fur: tan     y: eisenhower
x: herbert   fur: gray    y: fillmore
```