Databases

A database is a collection of records (tuples) and an interface for adding, editing, and retrieving records.

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

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

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 deems appropriate.

Logical/Declarative Programming

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

By contrast, in imperative languages such as Python & Scheme:
- A "program" is a description of procedures.
- The interpreter carries out execution/evaluation rules.

Building a universal problem solver is a difficult task.

Declarative programming languages compromise by solving only a subset of problems.

They typically trade off data scale for problem complexity.

The Logic Language

The Logic language is invented for this course.
- Based on the Scheme project + ideas from Prolog
- 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:

Simple Facts

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

Let's say I want to track my many dogs' ancestry.

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.
- In Scheme, we write (abs -3) to call abs on -3. It returns 3.
- In Logic, (abs -3 3) asserts that the abs of -3 is 3.

For example, if we wanted to assert that 1 + 2 = 3,

```
(add 1 2 3)
```

Why declare knowledge in this way? It will allow us to solve problems in two directions:

```
(add 1 2 _)
(add _ 2 3)
(add 1 _ 3)
```
Queries

A query contains one or more relations. The Logic interpreter returns whether (& how) they are all simultaneously satisfied.

Queries may contain variables: symbols starting with ?

\[
\text{logic} > (\text{fact (parent abraham barack)})
\]

\[
\text{logic} > (\text{fact (parent abraham clinton)})
\]

\[
\text{logic} > (\text{fact (parent delano herbert)})
\]

\[
\text{logic} > (\text{fact (parent fillmore abraham)})
\]

\[
\text{logic} > (\text{fact (parent fillmore delano)})
\]

\[
\text{logic} > (\text{fact (parent fillmore grover)})
\]

\[
\text{logic} > (\text{fact (parent eisenhower fillmore)})
\]

\[
\text{logic} > (\text{query (parent abraham ?child)})
\]

**Success!**

color: tan

color: white

color: brown

color: brown

color: white

Compound Facts

A fact can include multiple relations and variables as well.

\[
(\text{fact <conclusion> <hypothesis1> <hypothesis2> ... <hypothesisn>})
\]

Means <conclusion> is true if all <hypothesis> are true.

\[
\text{logic} > (\text{fact (child ?c ?p) (parent ?p ?c)})
\]

\[
\text{logic} > (\text{query (child herbert delano)})
\]

**Success!**

color: brown

color: tan

color: white

color: brown

Recursive Facts

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

\[
\text{logic} > (\text{fact (ancestor ?a ?y) (parent ?a ?y)})
\]

\[
\text{logic} > (\text{fact (ancestor ?a ?y) (parent ?a ?y) (ancestor ?a ?herbert)})
\]

**Success!**

color: tan

color: white

color: brown

Searching to Satisfy Queries

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

\[
\text{logic} > (\text{fact (parent delano herbert)})
\]

**Success!**

color: fillmore

color: eisenhower

Hierarchical Facts

Relations can contain relations in addition to atoms.

\[
\text{logic} > (\text{fact (dog (name abraham) (color white)))}
\]

\[
\text{logic} > (\text{fact (dog (name clinton) (color white)))}
\]

\[
\text{logic} > (\text{fact (dog (name delano) (color white)))}
\]

\[
\text{logic} > (\text{fact (dog (name eisenhower) (color tan)))}
\]

\[
\text{logic} > (\text{fact (dog (name fillmore) (color brown)))}
\]

\[
\text{logic} > (\text{fact (dog (name grover) (color tan)))}
\]

\[
\text{logic} > (\text{fact (dog (name herbert) (color brown)))}
\]

Variables can refer to atoms or relations.

\[
\text{logic} > (\text{query (dog (name clinton) (color ?color))})
\]

**Success!**

color: white

\[
\text{logic} > (\text{query (dog (name clinton) ?info})
\]

**Success!**

info: color white

Example: Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

\[
\text{logic} > (\text{query (dog (name ?name) (color ?color))})
\]

\[
(\text{ancestor ?name ?name})
\]

\[
(\text{dog (name ?ancestors) (color ?color))}
\]

**Success!**

name: barack color: tan ancestor: eisenhower

name: clinton color: white ancestor: abraham

name: grover color: tan ancestor: eisenhower

name: herbert color: brown ancestor: fillmore

Example: Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

\[
\text{logic} > (\text{query (dog (name ?name) (color ?color))})
\]

\[
(\text{ancestor ?name ?name})
\]

\[
(\text{dog (name ?ancestors) (color ?color))}
\]

**Success!**

name: barack color: tan ancestor: eisenhower

name: clinton color: white ancestor: abraham

name: grover color: tan ancestor: eisenhower

name: herbert color: brown ancestor: fillmore
Example: Appending Lists

Two lists append to form a third list if:
- The first list is empty and the second and third are the same
- The rest of 1 and 2 append to form the rest of 3

logic> (fact (append-to-form () ?x ?x))
logic> (fact (append-to-form (?a . ?r) ?y (?a . ?z))
 (append-to-form ?r ?y ?z))

Deno