Announcements

- HW12 due Wednesday
- Scheme project, contest due next Monday

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.

```sql
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.


Declarative Programming

The main 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 procedural 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 all 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 both Scheme lists

For example, (likes Amir 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 containing `fact` followed by one or more relations
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`
- 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 (and how) they are all simultaneously satisfied

Queries may contain variables: 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 ?child))
Success!
child: barack
child: clinton
```

Compound Facts

A fact can include multiple relations and variables as well

```
(fact <conclusion> <hypothesis0> <hypothesis1> ... <hypothesisN>)
```

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

```
logic> (fact (child ?a ?b) (parent ?a ?b))
logic> (query (child herbert delano))
Success!
child: herbert
child: delano
```

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> (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 a satisfying assignment

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

Relations can contain relations in addition to atoms

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

\[
\text{logic}\quad (\text{fact}\ (\text{dog}\ (\text{name}\ \text{barack})\ (\text{color}\ \text{tan}))))
\]

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

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

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

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

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

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

Variables can refer to atoms or relations

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

Success!

\[
\text{color}:\ \text{white}
\]

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

Success!

\[
\text{info}:(\text{color}\ \text{white})
\]

Example: Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

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

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

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

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
  
  \[
  ()\ (a\ b\ c)\ (a\ b\ c)
  \]

- Both of the following hold:
  - List 1 and 3 have the same first element
  - The rest of list 2 append to the rest of list 3

\[
\text{logic}\quad (\text{fact}\ (\text{append-to-form}\ ()\ ?x\ ?x))
\]

\[
\text{logic}\quad (\text{fact}\ (\text{append-to-form}\ (?a\ .\ ?r)\ ?y\ (?a\ .\ ?z)))
\]

\[
(\text{append-to-form}\ ?r\ ?y\ ?z))
\]