The Logic Language

The logic language was invented for Structure and Interpretation of Computer Programs.

- Based on Prolog (1972)
- Expressions are facts or queries, which contain relations
- Expressions and relations are Scheme lists
- For example, (likes john dogs) is a relation

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

Queries

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

Variables are symbols starting with ?

Each line is an assignment of variables to values

(likes john dogs)
Recursive Facts

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

\[
\text{logi} \text{c} \rightarrow ( \text{fact} (\text{ancestor} ?x ?y) (\text{parent} ?y ?z)) \\
\text{Success}:
\text{a}: \text{fillmore} \\
\text{a}: \text{eisenhower}
\]

Hierarchical Facts

Relations can contain relations in addition to symbols.

\[
\text{logi} \text{c} \rightarrow (\text{fact} (\text{dog} (\text{name} \text{abraham}) \text{fur} \text{long})) \\
\text{logi} \text{c} \rightarrow (\text{fact} (\text{dog} (\text{name} \text{clinton}) \text{fur} \text{long})) \\
\text{logi} \text{c} \rightarrow (\text{fact} (\text{dog} (\text{name} \text{eisenhower}) \text{fur} \text{short})) \\
\text{logi} \text{c} \rightarrow (\text{fact} (\text{dog} (\text{name} \text{fillmore}) \text{fur} \text{short})) \\
\text{logi} \text{c} \rightarrow (\text{fact} (\text{dog} (\text{name} \text{grover}) \text{fur} \text{short})) \\
\text{logi} \text{c} \rightarrow (\text{fact} (\text{dog} (\text{name} \text{herbert}) \text{fur} \text{long}))
\]

Variables can refer to symbols or whole relations.

\[
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{clinton}) \text{fur} \text{type}))) \\
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{fillmore}) \text{fur} \text{type}))) \\
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{clinton}) \text{fur} \text{long}))) \\
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{fillmore}) \text{fur} \text{long}))) \\
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{eisenhower}) \text{fur} \text{short}))) \\
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{fillmore}) \text{fur} \text{short}))) \\
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{grover}) \text{fur} \text{short}))) \\
\text{logi} \text{c} \rightarrow (\text{query} (\text{dog} (\text{name} \text{herbert}) \text{fur} \text{long})))
\]
Combining Multiple Data Sources

Which dogs have an ancestor of the same fur?

```
logic> (query (dog (name ?x) (fur ?fur))
  (ancestor ?y ?x)
  (dog (name ?y) (fur ?fur)))
```

Success!

- Barack: fur: short
- Eisenhower: fur: short
- Clinton: fur: long
- Abraham: fur: long
- Grover: fur: short
- Fillmore: fur: curly

Appending Lists

Lists in Logic

Expressions begin with `query` or `fact` followed by relations.

```
(fact (app () ?x ?x))
(fact (app (?a . ?r) ?y (?a . ?z))
  (app       ?r  ?y       ?z ))
```

```
(query (app ?left (c d) (e b c d))
```

Success!

```
left: (e b)
```

Unification

Pattern Matching

The basic operation of the Logic interpreter is to attempt to unify two relations. Unification is finding an assignment to variables that makes two relations the same.

```
{(a b) c (a b) }
{( ?x  ?x  ?x  ) True, (x: (a b))
{(a b) c (a b) }  {( ?x  ?x  ?x  ) True, (y: b, z: c)
{(a ?y) ?z (a b) }  {( ?x  ?x  ?x  ) False}
{(a b) c (a b) }  
{( ?x  ?x  ?x  ) }
```

Implementing Unification

```
def unify(e, f, env):
    e = lookup(e, env)
    f = lookup(f, env)
    if e == f:
        return True
    elif scheme_atom(e):
        env.define(e, f)
        return True
    elif scheme_atom(f):
        return False
    else:
        return unify(e.first, f.first, env) and unify(e.second, f.second, env)
```

Symbols/relations without variables only unify if they are the same.
Search

Searching for Proofs

The Logic interpreter searches the space of facts to find unifying facts and an env that prove the query to be true.

```
(fact (app () ?x ?x))
(query (app ?left (c d) (e b c d)))

(app ?left (c d) (e b c d))
(app (?a . ?r) ?y (?a . ?z))
{a: e, y: (c d), z: (b c d), left: (?a . ?r)}

(app ?r (c d) (b c d))
{r: (), x: (c d)}
```

```
Variables are local to facts & queries

?left: (e b)
```

Depth-First Search

The space of facts is searched exhaustively, starting from the query and following a depth-first exploration order.

Depth-first search: Each proof approach is explored exhaustively before the next.

```
def search(clauses, env):
    for fact in facts:
        env_head = an environment extending env
        if unify(conclusion of fact, first clause, env_head):
            for env_rule in search(hypotheses of fact, env_head):
                for result in search(rest of clauses, env_rule):
                    yield each successful result

- Limiting depth of the search avoids infinite loops.
- Each time a fact is used, its variables are renamed.
- Bindings are stored in separate frames to allow backtracking.

(Demo)
```

Addition

```
(Demo)
```