Announcements

• Homework 30 due Tuesday 11/26 @ 11:59pm
• No lecture on Wednesday 11/27 or Friday 11/29
• No discussion section Wednesday 11/27 through Friday 11/29
• Lab will be held on Wednesday 11/27
• Recursive art contest entries due Monday 12/2 @ 11:59pm

Lists in Logic

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

Expressions and their relations are Scheme lists.

(1) \((\text{append-to-form} () ?x ?x)\)
(2) \((\text{append-to-form} (?a . ?r) ?y (?a . ?z))\)

In a `fact`, the first relation is the conclusion and the rest are hypotheses.

In a `query`, all relations must be satisfied.

The interpreter lists all bindings of variables to values that it can find to satisfy the query.

Appending Lists

(Demo)

Anagrams in Logic

A permutation (i.e., anagram) of a list is:

1. The empty list for an empty list.
2. The first element of the list inserted into an anagram of the rest of the list.

(1) \((\text{anagram} ())())\)
(2) \((\text{anagram} (?a . ?r) ?b)\)
(3) \((\text{anagram} ?a ?b)\)
(4) \((\text{anagram} ?a . ?r) ?b\)

(Demo)
Unification

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.

\[
\begin{align*}
(a \ b) \ c &\quad (a \ b) \\
(\ ?x) \ c &\quad ?x \\
\end{align*}
\]

True, \{x: (a b)\}

\[
\begin{align*}
(a \ b) \ c &\quad (a \ b) \\
(\ ?y) \ ?z &\quad (a \ b) \\
\end{align*}
\]

True, \{y: b, z: c\}

\[
\begin{align*}
(a \ b) \ c &\quad (a \ b) \\
(\ ?x) \ ?x &\quad ?x \\
\end{align*}
\]

False

Unification recursively unifies each pair of corresponding elements in two relations, accumulating an assignment.

1. Look up variables in the current environment.
2. Establish new bindings to unify elements.

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

Two relations that contain variables can be unified as well.

\[
\begin{align*}
(\ ?x) &\quad ?x \\
(a \ y \ c) &\quad (a \ b \ ?z) \\
\end{align*}
\]

True, \{x: (a \ y \ c), y: b, z: c\}

Substituting values for variables may require multiple steps.

This process is called grounding. Two unified expressions have the same grounded form.

\[
\begin{align*}
\text{lookup}(\ ?x') &\rightarrow (a \ y \ c) \\
\text{lookup}(\ ?y') &\rightarrow b \\
\text{ground}(\ ?x') &\rightarrow (a \ b \ c)
\end{align*}
\]

Implementing Unification

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

Variables are local to facts & queries

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
```

Environment now contains new unifying bindings

- 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.

Addition

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

(Demo)

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

(Demo)