Comp 205: Comparative Programming Languages

Declarative Programming Languages

- Logic Programming
- Horn-Clause Logic
- Prolog

Lecture notes, exercises, etc., can be found at:
www.csc.liv.ac.uk/~grant/Teaching/COMP205/

Declarative vs Imperative

Declarative Languages

- the programmer specifies what is to be computed

Imperative Languages

- the programmer specifies how this is to be computed

The Imperative Paradigm

Key features of the imperative paradigm:

- storage
- variables
- state
- instructions
- flow of control

The Declarative Paradigm

Key features of the declarative paradigm:

- expressions
- referential transparency
  (an expression denotes a value irrespective of context)
- absence of state
- implicit operational semantics
  (term-rewriting, unification/resolution)

Declarative Languages

Functional

- functional relationship between input and output (one output for each input)

Relational

- logical relationships between entities in the problem domain
  (many possible solutions - programs are more like database queries)

Relational Languages

Prolog is the main language

- theorem proving (Colmerauer & Roussel)
- programming (Kowalski)
  (Program = Logic + Flow-of-control)

Other languages: KL-1, Eqlog.
Logic Programming

- OBJ is First-Order Equational Logic
- Haskell is (restricted) Higher-Order Equational Logic
- Prolog is First-Order **Horn-Clause Logic**

What is a Horn Clause?

A Horn Clause is a conditional rule

\[ P \text{ if } C_1, \ldots, C_n \]

that states that a proposition \( P \) is true if all the conditions \( C_1, \ldots, C_n \) are true.

If \( n=0 \), then \( P \) is unconditionally true; a Horn clause of this form is just written:

\[ P \]

Predicates

Propositions and conditions are predicates. These are formed from:

- Predicate symbols
- Terms

In Prolog, terms are expressions built from operators and constants. Constants and predicate symbols are just names, and are not declared, just used.

Predicates

A predicate is formed by "applying" a predicate symbol to one or more terms. For example:

- isHappy(mary)
- watching(peter, football)
- marriedBy(joseph, charlotte, fatherJack)

None of these has any intrinsic meaning.

Horn Clauses

In Prolog, the main proposition ("head") of a Horn clause is separated from the conditions by ":-"

```
happy(mary) :- stocked(mary), knitting(mary).
stocked(mary) :- has(mary, sweets), has(mary, stout).
has(mary, sweets).
has(mary, stout).
knitting(mary).
```

Queries

A Prolog "program" is a list of Horn clauses sometimes such a list is called a "database", or "knowledgebase".

Computation in Prolog is achieved by solving "queries" to the database, where a query is just a predicate.

```
?- happy(mary).
```
Solving Queries

Prolog solves a query by logical inferences, using its database. It maintains a list of goals, where each goal is a predicate, and attempts to infer that each goal is true. Initially, the list contains just the predicate in the query.

?- happy(mary).

Prolog infers that the goal is true ("satisfied") if both stocked(mary) and knitting(mary) are true; its list of goals now becomes stocked(mary), knitting(mary).

Yes, She's Happy

The process continues until (in this case) all the goals are seen to be satisfied.

happy(mary) :- stocked(mary), knitting(mary).

stocked(mary) :- has(mary, sweets), has(mary, stout).

has(mary, sweets).

has(mary, stout).

knitting(mary)

Variables and General Rules

General rules that apply to all constants (Prolog is not typed) are expressed by Horn clauses with variables (these begin with capital letters in Prolog)

happy(X) :- stocked(X), knitting(X).

This states that everyone is happy if they are stocked and knitting. (I.e., variables are universally quantified.)

Is She Still Happy?

Given the query "?- happy(mary)"", the constant mary will match with the variable X.

happy(X) :- stocked(X), knitting(X).

going again rise the list of goals stocked(mary), knitting(mary).

Is Everyone Happy?

Given the query "?- happy(john)", the constant john will match with the variable X.

happy(X) :- stocked(X), knitting(X).

giving rise to the list of goals stocked(john), knitting(john). In this case, these goals (and hence the original query) are not satisfied, and the answer is no. This is referred to as negation as failure.
Constructing Solutions

Queries can also contain variables.

A query containing a variable is seen as a request to find an instantiation of the variable that makes the predicate true.

For example, the query "?-happy(X)", is a request to find someone that (according to the database) is happy.

The result will be an instantiation of the variable X, for example, X = mary.

Who's Happy?

Given the query "?-happy(X)", the variable X will unify with the variable Y,

```
happy(Y) :- stocked(Y), knitting(Y).
```

giving rise to the list of goals

`stocked(X), knitting(X).`

The process of unification is a two-way process of finding matches for variables both in goals and in rules.

So Who's Stocked?

Given the goals `stocked(X), knitting(X),`

the variable X will unify with the variable Z,

```
stocked(Z) :- has(Z,sweets), has(Z,stout).
```

giving rise to the list of goals

`has(X, sweets), has(X, stout), knitting(X).`

Could It Be Mary?

Given the goals

```
has(X, sweets), has(X, stout), knitting(X).
```

the variable X will match with the constant mary,

```
has(mary, sweets).
```

giving rise to the list of goals

`has(mary, stout), knitting(mary).`

Alone of All Her X

The goals

```
has(mary, stout), knitting(mary).
... has(mary, stout).
knotting(mary).
```

will be seen to be satisfied by the database

and so a solution to the query is

X = mary

Summary

- Horn-clause logic
- Variables in queries are existentially quantified
- Computation as inference

Next: Unification and Backtracking;
Summary of declarative languages