Introduction to Logic Programming

Michael McThrow

San Jose State University
Computer Science Department
CS 152 – Programming Paradigms

October 25, 2021

Table of Contents

Table of Contents

In this course, we have explored two programming paradigms: procedural programming and functional programming. The next paradigm we will be exploring is *logic programming*.

Imperative and Declarative Programming

Programming language paradigms can be split into two categories: *imperative* and *declarative*.

Imperative and Declarative Programming

Programming language paradigms can be split into two categories: *imperative* and *declarative*.

- Imperative programming languages emphasize *how* to perform a computation.

Imperative and Declarative Programming

Programming language paradigms can be split into two categories: *imperative* and *declarative*.

- Imperative programming languages emphasize *how* to perform a computation.
- Declarative programming languages emphasize *what* to compute.

Imperative vs. Declarative Example

Example: Let's find all people under 25 years old.

```
# Imperative Example in Python
results = []
for person in people:
  if person.age < 25:
    results.add(person)

-- Declarative Example in SQL
SELECT * FROM people WHERE age < 25;
```

Is functional programming declarative?

Is functional programming declarative? Yes, provided that we avoid side-effects and mutation.

Table of Contents

Logic programming is a declarative programming paradigm that is rooted in *first-order logic*, also known as *predicate logic*.

Models of Computation

- Turing machines $\rightarrow$ Procedural programming
- Lambda calculus $\rightarrow$ Functional programming
- First-order logic $\rightarrow$ Logic programming

Like functional programming, logic programming is an alternative to the von Neumann architecture and the imperative programming paradigm it promotes.

What is Logic Programming

Instead of providing explicit execution steps as in imperative programming, we instead provide *knowledge* that the runtime environment treats as logical axioms. We then *query* this knowledge base in order to solve problems.

Quote from *The Art of Prolog*

"In its ultimate and purest form, logic programming suggests that even explicit instructions for operation not be given but rather that the knowledge about the problem and assumptions sufficient to solve it be stated explicitly, as logical axioms" [Sterling and Shapiro, p. 3].

How Execution Works in Logic Programming

- "The program can be executed by providing it with a problem formalized as a logical statement to be proved, called a goal statement" (p. 3).
- "The execution is an attempt to solve the problem, that is, to prove the goal statement, given the assumptions in the logic program" (p. 3-4).

Structure of Goal Statements

- "There exists a list `X` such that sorting the list `[3, 1, 2]` gives `X`" (p. 4).
- "In this example, assuming the logic function contains appropriate axioms defining the `sort` relation, the output of the computation would be `X = [1, 2, 3]`" (p. 4).

Summary of Logic Programming (from *The Art of Prolog*)

- A *program* is a set of axioms.
- A *computation* is a constructive proof of a goal statement from the program.

Table of Contents

Facts

Definition (Fact)

A *fact* is a statement of the relationship between objects. Note that "a finite set of facts constitutes a program" (p. 12).

Examples of Facts

```
evolution(bulbasaur,ivysaur).
evolution(ivysaur,venusaur).
evolution(charmander,charmeleon).
evolution(charmeleon,charizard).
evolution(squirtle,wartortle).
evolution(wartortle,blastoise).
```

These *relations* (or *predicates*) specify Pokémon evolutions (e.g., Bulbasaur evolves into Ivysaur). Note that the lowercase text and the period at the end matter.

More Examples of Facts

```
fire(charmander).
fire(charmeleon).
fire(charizard).

flying(charizard).

plus(0,0,0).
plus(0,1,1).
plus(1,0,1).
plus(1,1,2).
```

Queries

We use *queries* to get information from a program. Queries end with a question mark ? instead of a period.

Difference Between a Fact and a Query

- `fire(charizard).`
  - This is a fact.
  - Statement that `fire(charizard)` is true (i.e., Charizard is a fire-type Pokémon).
- `fire(charizard)?`
  - This is a query.
  - Query that asks whether `fire(charizard)` is true (i.e., is Charizard a fire-type Pokémon?).

Examples of Queries

- `fire(charizard)?` $\Rightarrow$ yes
- `fire(venusaur)?` $\Rightarrow$ no

IMPORTANT: A *no* does not mean the query is false; it means it couldn't be proved from the program.

Another Query Example

Based on the facts we have provided thus far, `fire(vulpix)?` would result in *no* because it hasn't been defined in the program. However, all Pokémon fans know that Vulpix is a fire-type Pokémon. Thus, a *no* does not mean that the query is false; it simply means that the query could not be proved from the program.

Queries with Variables

How could we find out what a Pokémon evolves into?

Queries with Variables

How could we find out what a Pokémon evolves into?

We can use a variable to find this out.

Queries with Variables

How could we find out what a Pokémon evolves into?

We can use a variable to find this out.

`evolution(bulbasaur,X)?` results in `X = ivysaur`.

Queries with Variables

How could we find out what a Pokémon evolves into?

We can use a variable to find this out.

`evolution(bulbasaur,X)?` results in `X = ivysaur`.

`evolution(venusaur,X)?` results in *no* (Venusaur cannot evolve, and for the more serious Pokémon fans, I'm ignoring Mega Evolution).

Queries with Variables

Sometimes a query could yield multiple answers.

Example: In the query `fire(X)?`, `X = charmander, charmeleon, charizard.`

Facts with Variables

We can use variables to assign universal facts.

Example: `likes(X,pomegrantes).` means for all `X`, `X` likes `pomegrantes`.

Conjunctive Queries

We can use a comma to join multiple queries using a logical AND.

Example: The conjunctive query `fire(X),flying(X)?` retrieves `X = charizard` since Charizard is both fire- and flying-type.

Rules

Definition (Rule)

A *rule* is a statement of the following form:

`A` $\leftarrow$ `B1,B2,...,Bn`

where `A` is the *head* and `B1,B2,...,Bn` is the *body*.

- Rules, facts, and queries are examples of Horn caluses.
- $\leftarrow$ is used to denote logical implication.
- A logic program is a finite set of rules.

Table of Contents

On Wednesday, I will be covering recursion in logic programming, which is an important and powerful construct.