Logic Programming

Logic Programming

- The kind of logic used in logic programming is the **first-order predicate calculus,** which is a way of formally expressing **logical statements,** that is, statements that are either true or false.
- Uses a set of logical assertions (i.e. statements that are either true or false), as a program (the facts).
- Execution is initiated by a *query* or *goal*, which the system attempts to prove true or false, based on the existing set of assertions.
- For this reason, logic programming systems are sometimes called *deductive databases*.
- No explicit functions, no explicit execution control.

English Statements

The following English statements are logical statements:

- 0 is a natural number.
- 2 is a natural number.
- For all x, if x is a natural number, then so is successor of x.
- -1 is a natural number

Logical statements

A translation into predicate calculus is as follows:

- natural(0).
- natural(2).
- For all x, natural(x)→natural(successor(x)).
- natural(-1).

axioms

- Among these logical statements, the first and third statements
  i. natural(0) and
  ii. For all x,natural(x)→natural(successor(x))

  can be viewed as **axioms** for the natural numbers:
- Statements that are assumed to be true are **axioms**.
- Indeed, the second statement can be proved from these axioms, since

  2 = successor(successor(0)) and

  natural(0) → natural(successor(0)) → natural(successor(successor (0))).
- Fourth statement cannot be proved from these axioms so it is false.

First-order Predicate Calculus

First-order Predicate Calculus starts with a set of axioms (true assertions), classifies the different parts of such statements as follows:

1. *Constants.* Usually numbers or names (atoms)
2. *Predicates.* Names for functions that are true or false, like Boolean functions in a program. Predicates can take a number of arguments. (Ex: natural function)
3. *Functions.* First-order predicate calculus distinguishes between functions that are true or false(these are the predicates) and all other functions, which represent non-Boolean values. (Ex: successor function)

Predicate Calculus (continued)

4. *Variables That Stand for* as yet Unspecified Quantities. (Ex: x is a variable)
5. *Connectives.* Operations and, or, and not; implication "→" and equivalence " ↔" These are not really new operations: *a → b means that b is* true whenever *a is true. This is equivalent to the statement b or not a. Also a ↔ b means the same as (a → b) and (b → a). (derived from the and, or & not operations)*
6. *Quantifiers.* These are operations that introduce variables: "for all" - the *universal quantifier*, and "there exists" - the *existential quantifier.*

   (Ex: For all x, natural(x) → natural(successor(x)) means that for every x in the universe, if x is a natural number, then the successor of x is also a natural number)
7. *Punctuation symbols:* left and right parentheses, the comma, and the period.

Note: Arguments to predicates and functions can only be **terms**: that is combinations of variables, constants, and functions.

English statements

The following are logical statements in English

- A horse is a mammal.
- A human is a mammal.
- Mammals have four legs and no arms, or two legs and two arms.
- A horse has no arms.
- A human has arms.
- A human has no legs

First order predicate calculus statements

A possible translation of these statements into first-order predicate calculus is as follows:

- mammal(horse).
- mammal(human).
- for all x, mammal(x)→legs(x,4) and arms(x,0) or legs(x,2) and arms(x,2).
- arms(horse,0).
- not arms(human,0).
- legs(human,0).
- In addition to the seven classes of symbols described, first-order predicate calculus has **inference rules,** which are ways of deriving or proving new statements from a given set of statements.

Inference rule

A typical inference rule is the following:

- From the statements *a → b and b → c, one can derive the statement a → c, or written more formally,*

  (a → b) and (b → c) as a → c
- These are statements that are always true whenever the original statements are true.
- For example, the first five statements about mammals in previous example allow us to derive the following statements:
  - legs(horse,4).
  - legs(human,2).
  - arms (human, 2).

  mammal(horse).
  mammal(human).
  for all x, mammal(x)→legs(x,4) and arms(x,0) or legs(x,2) and arms(x,2).
  arms(horse,0).
  not arms(human,0).
- Stated in the language of logic, these three statements become **theorems** derived from the first five statements or axioms of previous example.

Logic programming language

- This is the essence of logic programming: A collection of statements is assumed to be axioms, and from them a desired fact is derived by the application of inference rules in some automated way.

- Thus, we can state the following definition:

- A ***logic programming language*** *is a notational system for writing logical statements together with* specified algorithms for implementing inference rules.

Logic program

- The set of logical statements that are taken to be axioms can be viewed as the logic program.
- The statements or the statements that are to be derived can be viewed as input that initiates the computation. Such inputs are also provided by the programmer and are called as queries or goals.
- For example to compute how many legs a human has, we would provide the following query:

  -Does there exist a *y such that y is the number of legs of a human?* or, in predicate calculus,

  -there exists y, legs(human,y)?

Write the following statements in the first order predicate calculus

- If it is raining or snowing, then there is precipitation.
- If it is freezing and there is precipitation, then it is snowing.
- If it is not freezing and there is precipitation, then it is raining.
- It is snowing.

- R or S → P
- F and P →S
- Not F and P→R
- S

Horn clause

- Unfortunately, automated deduction systems have difficulty handling all of first-order predicate calculus.
- First, there are too many ways of expressing the same statements, and second, there are too many inference rules.
- As a result, most logic programming systems restrict themselves to a particular subset of predicate calculus, called Horn clauses.
- A Horn clause (named after its inventor Alfred Horn) is a statement of the form:

  a_1 and a_2 and a_3 …. and $a_n \rightarrow b$

  where a_i are simple statements involving no connectives.
- Thus there are no OR connectives or no quantifiers in horn clauses.
- The preceding Horn clause says that *a_1 through a_n imply b, or that b is true if all the a_i are true*.
- b is called the **head** of the clause and a_i the **body** of the clause.

Facts

- In the horn clause number of a_i's may be zero in which case horn clause has the form:

$$\rightarrow b$$

- Such a clause means that b is always true, that is b is axiom and usually written without the connective. Such clauses are also called as facts.

Predicate calculus to horn clauses

- Horn clauses can be used to express most, but not all, logical statements.
- The basic idea is to remove *or connectives by writing* separate clauses, and
- to treat the lack of quantifiers by assuming that variables appearing in the head of a clause are universally quantified,
- while variables appearing in the body of a clause are existentially quantified.

Predicate calculus to horn clauses

- The following statements in first-order predicate calculus:

  natural(0).

  for all x, natural(x) → natural (successor (x)).
- These can be very simply translated into Horn clauses by dropping the quantifier:

  natural(0).

  natural(x) → natural (successor(x)).

GCD

- Consider the logical description for the Euclidian algorithm to compute the gcd of two positive integers *u and v:*

  The GCD of u and 0 is u.

  The GCD of u and v, if v is not 0,is the same as the GCD of v and the remainder of dividing v from u.

- Translating this into first-order predicate calculus gives:
  - for all u, gcd(u,0,u).

    for all u, for all v, for all w, not zero(v) and gcd(v, u mod v, w) → gcd(u, v, w).

- To translate these statements into Horn clauses, only drop the quantifiers:
  - gcd(u, 0, u).

    not zero(v) and gcd(v, u mod v, w) → gcd(u, v, w).

First Order predicate calculus statement to horn clause

for all x, for all y, (there exists z, parent(x, z) and parent(z, y))→grandparent(x, y).

- As a Horn clause this is expressed simply as:

  parent(x, z) and parent(z, y) → grandparent(x, y).

First Order predicate calculus statement

- To see how connectives are handled, consider the following statement:

  for all x, if x is a mammal then x has two or four legs.

- Translating in predicate calculus, we get:
  - for all x, mammal(x)→legs(x,2) or legs(x,4)

Horn Clause Equivalent

- mammal(x) and not legs(x,2)→legs(x,4)
- mammal(x) and not legs(x,4)→legs(x,2)

Procedural Interpretation

- If we write the horn clause in reverse order

  $b \leftarrow a_1$ and a_2 ……and a_n

  we can view this as a definition of procedure b: the body of *b is given by the body of the clause, namely* the operations indicated by the *ai's.*

Procedural Interpretation

- With the foregoing procedural interpretation in mind, most logic programming systems not only write Horn clauses backward but also drop the *and* connectives between the a_i, separating them with commas instead.
- Thus, the gcd clauses would appear as follows:
  - gcd(u, 0, u).
  - gcd(u, v, w) ← not zero(v), gcd(v, u mod v, w).
- This is beginning to look suspiciously like a more standard programming language expression for the gcd, such as
  - gcd(u, v) = if v = 0 then u else gcd(v, u mod v).
- From now on we will write Horn clauses in this form.

Fact , Query or goals

- In fact, a query is exactly the opposite of a fact—a Horn clause with no head:

  mammal(human)← ---- a fact

  ←mammal(human) ---- a query or goal

- A Horn clause without a head could also include a sequence of queries separated by commas:

  ←mammal(x), legs(x, y)

Resolution

- Resolution is an inference rule for Horn clauses and states that

- if we have two horn clauses, and we can match the head of the first horn clause with the one of the statements in the body of the second clause, then the first clause can be used to replace its head in the second clause by its body.

Resolution

- $a \leftarrow a_1, \ldots a_n$.
- $b \leftarrow b_1, \ldots, b_m$. and b_i *matches a, then we can infer the clause:*
- $b \leftarrow b_1 \ldots b_{i-1}, a_1, \ldots a_n, b_{i+1}, \ldots b_m$

Resolution

- The simplest example of this occurs when the body of the Horn clauses contains only single statements, such as in:

  $b \leftarrow a.$

  and:

  $c \leftarrow b.$

- In this case, resolution says that we may infer the following clause:

  $c \leftarrow a.$

Resolution

- Another way of looking at resolution is to combine LHS and RHS of both horn clauses and cancel those statements that match on both the sides.

Resolution

- Thus, for the simplest example,

$$b \leftarrow a.$$

and:

$$c \leftarrow b.$$

give:

$$b, c \leftarrow a, b.$$

and canceling the *b*,

$$\cancel{b}, c \leftarrow a, \cancel{b}$$

gives:

$$c \leftarrow a.$$

Logic Programming

- Logic programming treats goal or list of goals as a horn clause without a head.
- The system attempts to apply resolution by matching one of the goals in the body of the headless clause with the head of a known clause.
- It then replaces the matched goal with the body of that clause, creating new list of goals which it continues to modify in the same way. The new goals are called subgoals.

Logic Programming

- ←a
- a←a1,…..an
- ←a1,a2,…..an
- If the system succeeds in eliminating all goals---thus deriving the empty horn clause----then the original statement has been proved.

Axioms

- legs(x,2)←mammal(x),arms(x,2).
- legs(x,4)←mammal(x),arms(x,0).
- mammal(horse).
- arms(horse,0).

query

- ←legs(horse,4).

using resolution and unification

- legs(x,4)←mammal(x),arms(x,0),legs(horse,4).
- Replace x by horse, the process of replacing variables by terms is called unification.
- ~~legs(horse,4)~~←mammal(horse),arms(horse,0),~~legs(horse,4).~~

using resolution and unification

- ←mammal(horse),arms(horse,0)
- ~~mammal(horse)~~←~~mammal(horse)~~,arms(horse,0).
- ←arms(horse,0)
- ~~arms(horse,0)~~← ~~arms(horse,0).~~
  ←

Thus horse has four legs has been proved.

Axioms

- gcd(u,0,u)
- gcd(u,v,w)←not zero(v),gcd(v,u mod v, w).

Query

- ←gcd(15,10,x)

Using resolution and Unification

- ~~gcd(15,10,x)~~←not zero(10), gcd(10,15 mod 10, x), ~~gcd(15,10,x).~~
- ←gcd(10,5,x)
- ~~gcd(10,5,x)~~←not zero(5), gcd(5,10 mod 5, x), ~~gcd(10,5,x).~~
- ←gcd(5,0,x).
- gcd(5,0,5)←gcd(5,0,x)
- ~~gcd(5,0,5)~~←~~gcd(5,0,5)~~
- Yes: x=5

References

Text book

- Kenneth C. Louden “Programming Languages Principles and Practice” second edition Thomson Brooks/Cole Publication.

Reference Books:

- Terrence W. Pratt, Masvin V. Zelkowitz “Programming Languages design and Implementation” Fourth Edition Pearson Education.
- Allen Tucker, Robert Noonan “Programming Languages Principles and Paradigms second edition Tata MC Graw –Hill Publication.