Logic programming

12.1-12.2

and other materials

Logic Programming

- Introduction
- Logic programming

Introduction

- A modeling view
  - The world can be taken as
    - *A set of objects,* and
    - *relations* among them
  - Traditional logic provides an excellent way to represent objects and define new relations using other relations
  - A subset of the traditional logic, Horn clauses, can be reasoned with efficiently

Roots of logic programming

- Logic
  - Provides ways of representing objects and defining relations
    - Example: above (blockx, blocky) if on(blockx, blocky)
- Artificial intelligence
  - John McCarthy, a giant in Computer Science and a Turing award winner, proposed to use logic as the basis of building intelligent agent.
  - In the process of implementing McCarthy's proposal, the ideas of logic programming was formed.

Problem solving using logic programming

- To solve a problem, one usually needs to
  - first *write a logic program*, called knowledge base, to describe the problem and
  - then form a *query* to this knowledge base.
  - The *answer* to the query is a solution to the original problem.

- Example. Problem: we try to know the relationship among the members of a big family.
- The knowledge base:
  - We have the following facts: who is male/female, who is whose father and mother
    - E.g., male(john). father(john, mary).
  - We have also the knowledge on parent: x is parent of y if x is father of y or mother of y
    - parent(X, Y) :- father(X, Y).   parent(X, Y) :- mother(X, Y).

- Quries:
  - ?father(john, mary)
  - ?parent(john, mary)
  - ?parent(X, mary) % who is parent of mary?
- Note this way of solving problems is different from the traditional imperative programming languages (where you have one executable file vs here a knowledge base and various queries)

- A logic programming interpreter is used to answer the queries with respect to a program (knowledge base)

Logic programming (LP)

- Objects
  - *constants (identifiers starts with small case letter).* "names for simple objects"
  - *function symbols (with arity): "name of* constructor for complex objects"
    - *usually, a function symbol is represented in this way: f/2*
  - *Terms:* "objects"
    - A constant is a *term*
    - If *t1, … tn* are terms and *f/n* a function symbol*, f(t1, …, tn)* is a *term*

- Example
  - How to represent natural numbers (i.e., give names to natural numbers)?
    - Signature:
      - Constant: 0
      - Function symbol: s/1
    - Example of terms: 0, s(0), …
    - Answer the question: then how to compute 100+100 under this representation?
  - How to represent a binary tree where each node has an integer key.
    - Signature
      - Constant: nil
      - Function symbol: treeNode(Key, L, R) : L left subtree, R: right subtree
    - Example
  - How to represent lists?
    - Signature
    - Example
    - More can be found later

Logic programming – relations

- Relations
  - *predicate symbols,* e.g., brother/2 (2 is the parameters of this predicate symbol)
  - *Atoms:*
    - *If t1, …, tn are terms and p/n is predicate, p(t1, …, tn) is called an atom.*
- Rules to define new relations
  - e.g., friend(X, Y) :- friend(Y, X)
  - Rule is of the following

    h :- g1, …, gn. where h, gi's are atoms (or their negations)

- How to read a rule:

  grandfather(X, Y) :- father(X, Z), father(Z, Y).

  For all x and y, x is grandfather of y if ***there is z*** such that x is father of z and z is father of y.

Example: objects

- Example. We can use terms to represent natural number.
  - function symbol: s/1
  - constant: nil
  - Informally, we can take nil as 0, s(nil) as 1, s(s(nil)) as 2, …
  - To define addition of two numbers (decomposition approach is used in forming this def)

```
add(X, Y, Z) :- X=nil, Z=Y.
add(X, Y, Z) :- X=s(T), add(T, Y, Z1), Z=s(Z1).
```

Example: rules to define new relations

- More exercises of defining relations
  - Given relations male(X), father(X, Y), mother(X,Y), define new relations
    - parent(X, Y) … (Note how disjunction is represented in logic programming)
      - To define “p if q or r” we use two rules:
      - p :- q.
      - p :- r.
    - brother(X,Y)
    - grandfather(X,Y) …

LP Object (data structure) – List

- Lists
  - [], [1,2,3], [[1],2,3]
  - [X|Y] represents a list whose first element is X and the rest forms a list of Y
    - e.g., [X|Y]=[1,2,3] % X=1, Y=[2,3] (note not 2, 3)
- Examples. (using decomposition of objects to write the definition)
  - Define member(X, L) which holds if X is a member of the list L
  - Define length(L, n) which holds if the length of L is n

Tree

- More interesting problem.
  - How to represent a tree in logic programming?
    - function symbol node/3.
    - A tree can be represented by a term *node(x, y, z)* where x is the key of the root node, y is the left subtree (thus a term starts with *node*) and z is the right subtree.
    - constant nil is used to represent an empty tree
  - Problem: find the number of nodes in a tree. We introduce the atom size(T, n) which holds if the number of nodes of tree T is n. Here we use the decomposition approach again

```
size(nil, 0).
size(Tree, n) :- Tree=node(X, L, R), size(L, nl), size(R, nr),
                 n=1+nl+nr.
```

How a query is answered

- Unification: solving term constraints
- Rewriting an atom/term constraint
- Construct an SLD tree of the query with respect to a program
- Traverse the tree find an answer of the query

Unification

- *Term constraints* are of the form t1=t2 where t1 and t2 are terms
- Solve term constraints: find a term for each variable in t1 and t2 such that after replacing the variables by these terms in t1 and t2, they are identical
- Example
  - brother(X, john) = brother(mat, john)

- Unification algorithm for solving term constraints
  - Any term constraint is processed in the following way
    - Case 1: it is of form c1=c2 where c1 and c2 are constants
      - If c1 is identical to c2, discard it; otherwise report no solution
    - Case 2: it is of form x=t1 (x is variable and t1 a term)
      - Assign t1 to x, discard this constraint, and replace all x, in all term constraints and assignments made, by t1 *if t1 has no x inside it*; otherwise report no solution
    - Case 3: it is of the form f(s1, …, sn) = g(t1, …, tn) (f,g: function symbols, si and ti: terms)
      - If f is identical to g, replace the term constraint by constraints s1=t1 and s2=t2, ……, sn=tn; otherwise: no solution
  - Repeat this process until no more constraints left or a report of no-solution exists.

- Examples
  - $f(X) = f(f(X))$
  - $f(X, Y, g(a))=f(g(Y), Z,X)$

Rewriting an atom*

- *Rewrite* an atom p(t1, …, tn) using a rule
  p(s1, …, sn) :- q(u1, …, un).
  - rename all variables in the rule to new variables
  - replace p(t1, …, tn) by
    t1=s1, …, tn=sn, q(u1, …, un)
- Example
  - program:
    - parent(X, Y) :- father(X, Y).
    - parent(X, Y) :- mother(X, Y).
    - father(john, mary)
    - mother(mary, peter)
  - Rewrite parent(X, john) using the first rule:

- We introduce a state <q1, …, qn | C> where qi are atoms or term constraints, and C is a set of term constraints that have a solution. C is also called a *constraint store.*
- State transition rules. Given a state <q1, …, qn | C>,
  - case 1: q1 is a term constraint t1=t2,
    - A new state will be <q2, …, qn | C U {t1=t2}> if C U {t1=t2} has a solution
    - Otherwise, produce a new state <fail |_>
  - case 2: q1 is an atom p(t1, …, tn) , rewrite q1 using a rule p(s1, …, sn) :- B1, …, Bn, leading to
    - A new state <t1=s1, …, tn=sn, B1, …,Bn, q2, …, qn|C>

- Example
  - state transition
    - <parent(X, john) | {} >, new state?

- SLD resolution tree of a query p1,…,pn, with respect to a knowledge base.
  - The root is state <p1, …, pn | C>
  - A child node is is state that can be transited from the root state by the state transition rules
  - Repeat this process
- A state <|C> is successful is C has solution; A state <fail | _> is *fail.*
- To answer a query is to find a success state from the SLD resolution tree of the query with respect to a program

- Example
  - parent(X, peter)
  - SLD resolution tree of this query
  - How an answer is obtained?

A logic programming example: answer set programming

- Answer set programming is an instance of logic programming. (The logic underlying it is non-monotonic)
- Reference: Dr. Gelfond’s book (avail in his web page)
- It has same rules as we introduced before and the following one
  - :-B1, …, Bn. Which reads as B1, …, Bn can not hold at the same time.
- We use it to model an interesting application

Mothodology to model problems using LP

- Identify the objects and relations in the problem
- Using program to describe the relations in your problem
- Write a query whose answer against the program gives an solution of the original problem. (In answer set programming, the set of all beliefs specified by the program gives a solution of the original problem).

SPARC

- http://ec2-52-25-88-7.us-west-2.compute.amazonaws.com/
- Components of SPARC
  - Sorts

    #people = {a, b}.
  - Predicates

    brother(#people, #people).
  - Rules

    brother(a,b).

    -brother(b,a).

- Example on rules

  A family: Bob is father, Joanne is the mother, and John, Sam and Sara are the kids.

  See example online beside the slides

- Example

A lawyer in Paris has a brother in Toronto who is a doctor, but the doctor in Toronto has no brother in Paris.

sorts

#people = {a, b}.

predicates:

% brother/2 isLawyer/1 isDoctor/1 …

rules

isLawyer(a).

% answer set?

% what is needed for us to believe there is

% a contradiction?

SUDOKU*

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

- Objects: numbers, locations of the table
- Relation: we want to know which number is in a grid
  - atLocation(n, x, y) holds if number n is at location (x,y).

- Constraints over the relations location/3:
  - Given by the initial situation

    atLocation(1, 2, 6). % number 6 is at location (1,2)

    …
  - Enforced by the problem
    - All numbers in the same column are different

      :- location(N1, X, Y), location(N2, X, Y1), Y != Y1, N1=N2.
    - All numbers in the same row are different
    - All numbers in the same small box are different

- all numbers in a small box are different

  :- differentLoc(X1, Y1, X2, Y2), sameBox(X1, Y1, X2, Y2), location(N1, X1, Y1), location(N2, X2, Y2), N1=N2.

  differentLoc(X1, Y1, X2, Y2) :- % (english description first and then to a more precise "code")

  sameBox(X1, Y1, X2, Y2) :- …

- We can't have more than one number in any location

  :- location(N1, R, C), location(N2, R, C), N1 != N2.

- There must be a number at any location

  location(1, R, C) v location(2, R, C) v … v location(9, R, C).

Summary

- Know basics of logic programming
  - terms, rules
- Know how to model a problem in logic programming
  - objects, relations
  - how to define disjunctions in the body
  - Applications such as a family and Sudoku*
- Know how a query is answered: (SLD resolution tree*) and unification