K.SUNDARAMOORTHI
Assistant Professor of CS

Knowledge Representation

Using Predicates Logic

1.Logical Representation Scheme

We are only going to look at one logical representation scheme, namely, first-order predicate logic. Predicate logic is more powerful than propositional logic as it allows for quantification. Facts are represented as logical propositions and additional information is deduced from these facts using backward chaining or resolution.

Converting facts to logical propositions

The first step is to convert facts in English to logical propositions called well-formed formulas (wffs). The wffs include the following symbols:

- Implies: →
- And: /\
- Or: \/
- Not: ~
- For all: ∀
- The exists: ∃

Example: Consider the following facts on Italian history:

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Roman.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers that they are not loyal to.
8. Marcus tried to assassinate Caesar.
9. All men are people.

These facts can be represented in predicate logic as the following wffs:

1. man(Marcus)
2. Pompeian(Marcus)
3. ∀x: Pompeian(x) → Roman(x)
4. ruler(Caesar)
5. ∀x: Roman(x) → loyalto(x,Caesar) \/ hate(x,Caesar)
6. ∀x ∃y: loyalto(x,y)
7. ∀x ∀y: person(x) /\ ruler(y) /\ tryassassinate(x,y) → ~loyalto(x,y)
8. tryassassinate(Marcus,Caesar)
9. ∀x: man(x) → person(x)

3.2. Backward Chaining

Suppose that given the facts 1 to 9 in the previous section, we want to answer the question 'Was Marcus loyal to Caesar"? We can prove this using backward chaining. Backward chaining matches a clause to the right-hand side of a rule (wff with an implies) and then attempts to prove the clauses on the left-hand of the rule. Alternatively, a clause can be matched against a single clause. Since wff 7 has a ~loyalto on the right-hand side we will attempt to prove ~loyalto(Marcus, Caesar):

~loyalto(Marcus, Caesar)
↓ 7
person(Marcus) /\ ruler(Caesar) /\ tryassasinate(Marcus,Caesar)
↓ 8
person(Marcus) /\ ruler(Caesar)
↓ 4
person(Marcus)
↓ 9
man(Marcus)
↓ 1
Nil

Using backward chaining we have proved that Marcus is not loyal to Caesar.

Exercise 3

1. Consider the following facts:

1. John likes all kinds of food.
2. Apples are food.
3. Chicken is food
4. Anything anyone eats and is not killed by is food.
5. Bill eats peanuts and is still alive.
6. Sue eats everything Bill eats.

a) Convert these facts to wffs in predicate logic.
b) Using backward chaining prove the "John likes peanuts".

2. Consider the following facts:

1. Steve only likes easy courses.
2. Science courses are hard.
3. All the courses in the basketweaving department are easy.
4. BK301 is a basketweaving course.

a) Convert these facts to wffs in predicate logic.
b) Use backward chaining to answer the question "What course would Steve like?".

Computable Functions and Predicates

It may be necessary to compute functions as part of a fact. In these cases a computable predicate is used. A computable predicate may include computable functions such as +, -, *, etc. For example, gt(x-y,10) →bigger(x) contains the computable predicate *gt* which performs the greater than function. Note that this computable predicate uses the computable function subtraction.

Example: Consider the following statements:

1. Marcus was a man.
2. Marcus was Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 1991.
8. Alive means not dead.
9. If someone dies, he is dead at all later times.

The wffs for these facts are:

1. man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. $\forall x$: man(x) → mortal(x)
5. erupted(volcano,79) /\ $\forall x$: Pompeian(x) → died(x,79)
6. $\forall x\ \forall t_1\ \forall t_2$: mortal(x) /\ born(x,$t_1$) /\ gt(t2-t1,150) → dead(x,t_2)
7. now=1991
8. $\forall x\ \forall t$: [alive(x,t) → ~dead(x,t)] /\ [~dead(x,t) →alive(x,t)]
9. $\forall x\ \forall t_1\ \forall t_2$:died(x,$t_1$) /\ gt($t_2$,$t_1$) →dead(x,$t_2$)

Suppose we want to answer the question “Is Marcus alive now?”. We can do this by either proving alive(Marcus,now) or ~alive(Marcus,now). Let us try the latter:

~alive(Marcus,now)
↓ 8
~[~dead(Marcus,now)]
↓ negation operation
dead(Marcus,now)
↓ 9
died(Marcus,t_1) /\ gt(now,t_1)
↓ 5
erupted(volcano,79) /\ Pompeian(Marcus) /\ gt(now,79)
↓fact, 2
gt(now,79)
↓
gt(1991,79)
↓ compute gt
nil

Resolution

In the previous sections facts were proved or queries were answered using backward chaining. In this section we will examine the use of resolution for this purpose. Resolution proves facts and answers queries by refutation. This involves assuming the fact/query is untrue and reaching a contradiction which indicates that the opposite must be true. The wffs must firstly be firstly converted to clause form before using resolution. The algorithm for converting wffs to clause form and the resolution algorithm are listed below.

Algorithm: Converting wffs to Clause Form

1. Remove all implies, i.e. → by applying the following: a → b is equivalent to ~a ∨ b.

2. Use the following rules to reduce the scope of each negation operator to a single term:
   - ~(~a) = a
   - ~(a ∧ b) = ~a ∨ ~b
   - ~(a ∨ b) = ~a ∧ ~b
   - ~∀x: p(x) = ∃x: ~p(x)
   - ~∃x: p(x) = ∀x: ~p(x)

3. Each quantifier must be linked to a unique variable. For example, consider ∀x: p(x) ∨ ∀x: q(x). In this both quantifiers are using the same variable and one must changed to another variable: ∀x: p(x) ∨ ∀y: q(y).

4. Move all quantifiers, in order, to the left of each wff.

5. Remove existential quantifiers by using Skolem constants or functions. For example, ∃x: p(x) becomes p(s1) and ∀x ∃y: q(x,y) is replaced with ∀x: q(s2(x), x).

6. Drop the quantifier prefix.

7. Apply the associative property of disjunctions: a ∨ (b ∨ c) = (a ∨ b) ∨ c and remove brackets giving a ∨ b ∨ c.

8. Remove all disjunctions of conjunctions from predicates, i.e. create conjunctions of disjunctions instead, by applying the following rule iteratively: (a ∧ b) ∨ c = (a ∨ c) ∧ (b ∨ c).

9. Create a separate clause for each conjunction.

10. Rename variables in the clauses resulting from step 9. to ensure that no two clauses refer to the same variable.

Algorithm: Resolution

1. Convert the wffs to clause form.

2. Add the fact (or query) P to be proved to the set of clauses:
   i. Negate P.
   ii. Convert negated P to clause form.
   iii. Add the result of ii to the set of clauses.

3. Repeat

   i. Select two clauses.
   ii. Resolve the clauses using unification.
   iii. If the resolvent clause is the empty clause, then a contradiction has been reached. If not add the resolvent to the set of clauses.

   Until (a contradiction is found OR no progress can be made)

Consider the following wffs:

1. man(Marcus)
2. Pompeian(Marcus)
3. ∀x: Pompeian(x) → Roman(x)
4. ruler(Caesar)
5. ∀x: Roman(x) → loyalto(x,Caesar) ∨ hate(x,Caesar)
6. ∀x ∃y: loyalto(x,y)
7. ∀x ∀y: person(x) ∧ ruler(y) ∧ tryassassinate(x,y) → ~loyalto(x,y)
8. tryassassinate(Marcus,Caesar)
9. ∀x: man(x) → person(x)

Converting these to clause form gives:

1. man(Marcus)
2. Pompeian(Marcus)
3. ~Pompeian(x) ∨ Roman(x)
4. ruler(Caesar)
5. ~Roman(x1) ∨ loyalto(x1,Caesar) ∨ hate(x1,Caesar)
6. loyalto(x2,s1(x2))
7. ~person(x3) ∨ ~ruler(y) ∨ ~tryassassinate(x3,y) ∨ ~loyalto(x3,y)
8. tryassassinate(Marcus, Caesar)
9. ~man(x4) ∨ person(x4)

Suppose that we want to prove that Marcus hates Caesar. We firstly convert this to a wff: hate(Marcus,Caesar). The wff is then negated and converted to clause form: ~hate(Marcus,Caesar). This clause is added to the set of clauses and the resolution is algorithm is applied:

~hate(Marcus,Caesar)
↓ 5
~Roman(Marcus) \/ loyalto(x1,Caesar)
↓ 3
~Pompeian(Marcus) \/ loyalto(Marcus,Caesar)
↓ 2
loyalto(Marcus,Caesar)
↓ 7
~person(Marcus) \/ ~ruler(Caesar) \/ ~tryassassinate(Marcus,Caesar)
↓ 4
~person(Marcus) \/ ~tryassassinate(Marcus,Caesar)
↓ 8
~person(Marcus)
↓ 9
~man(Marcus)
↓ 1
□

Thus, we have a contradiction and the original statement, i.e. Marcus hates Caesar must be true.

Consider the wffs we created above:

1. man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. ∀x: man(x) → mortal(x)
5. erupted(volcano,79) /\ ∀x: Pompeian(x) → died(x,79)
6. ∀x ∀t_1 ∀t_2: mortal(x) /\ born(x,t_1) /\ gt(t2-t1,150) → dead(x,t_2)
7. now=1991
8. ∀x ∀t: [alive(x,t) → ~dead(x,t)] /\ [~dead(x,t) →alive(x,t)]
9. ∀x ∀t_1 ∀t_2:died(x,t_1) /\ gt(t_2,t_1) →dead(x,t_2)

Suppose we now want to use resolution to prove that “Marcus is not alive now”. We firstly have to convert these statements to clause form:

1. man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. ~man(x) \/ mortal(x)
5. erupted(volcano,79)
6. ~Pompeian(x1) \/ died(x1,79)
7. ~mortal(x2) \/ ~born(x2,t1) \/ ~gt(t2-t1,150) \/ dead(x2,t2)
8. now=1991
9. ~alive(x3,t) \/ ~dead(x3,t)
10. dead(x4,t3) \/ alive(x4,t3)

11. ~died(x5,t4) ∨ ~gt(t5,t4) ∨ dead(x5,t5)

We want to prove ~alive(Marcus,now). We negate this and convert it clause form: alive(Marcus, now) and find a contradication:

alive(Marcus,now)
↓10
dead(Marcus,now)
↓11
~died(Marcus,t4) ∨~gt(now,t4)
↓6
~Pompeian(Marcus) ∨~gt(now,79)
↓ 2
~gt(now,t4)
↓8
~gt (1991,79)
↓
□

Exercise 4

1. Consider the following facts:

1. John likes all kinds of food.
2. Apples are food.
3. Chicken is food
4. Anything anyone eats and is not killed by is food.
5. Bill eats peanuts and is still alive.
6. Sue eats everything Bill eats.

a) Convert the wffs for these facts (which you created in Exercise 3) to clause form.
b) Using resolution prove that “John likes peanuts”.

2. Consider the following facts:

1. Steve only likes easy courses.
2. Science courses are hard.
3. All the courses in the basketweaving department are easy.
4. BK301 is a basketweaving course.

a) Convert the wffs for these facts (which you created in Exercise 3) to clause form.
b) Use resolution to answer the question “What course would Steve like?”.