COMP2411 Lecture 10: Propositional Logic Programming

Note: This material is not covered in the book

Resolution Applied to Horn Clauses

Consider two Horn clauses

$$p \vee \neg p_1 \vee \ldots \neg p_n \quad \text{and}$$
$$q \vee \neg q_1 \vee \ldots \neg q_m$$

Suppose these resolve, with $p = q_1$, say. Then the result is

$$q \vee \neg p_1 \vee \ldots \neg p_n \vee \neg q_1 \vee \ldots \neg q_m$$

which is a Horn clause.

Thus: resolution applied to Horn clauses produces only Horn clauses.

In the book's notation:

$$\frac{p_1 \wedge \ldots \wedge p_n \longrightarrow p \qquad q_1 \wedge \ldots \wedge q_m \longrightarrow q \qquad p = q_1}{p_1 \wedge \ldots \wedge p_n \wedge q_2 \wedge \ldots \wedge q_m \longrightarrow q}$$

Definite Horn Clauses

A Horn clause is said to be *definite* if it contains a positive literal.

(In the book's notation, a Horn clause $L \longrightarrow R$ is definite if R is not $\bot$.)

Example:

- $\neg C \vee \neg EF \vee I$ (i.e. $C \wedge EF \longrightarrow I$) is definite
- C (i.e. $\top \longrightarrow C$) is definite
- $\neg D \vee \neg N$ (i.e. $D \wedge N \longrightarrow \bot$) is not definite.

A set of definite Horn clauses is called a *logic program.*

When P is a logic program and $G_1 \wedge \ldots \wedge G_n$ is a conclusion such that we we would like to derive $P \models G_1 \wedge \ldots \wedge G_n$, the clause $G_1 \wedge \ldots \wedge G_n \longrightarrow \bot$ (i.e. $\neg G_1 \vee \ldots \vee \neg G_n$) added to P is called a *goal.*

Linear Resolution

A resolution proof starting from a set of premises (clauses) $\{P_1, \ldots, P_n\}$ is said to be *linear* if it can be written in the form of a sequence of clauses

$$P_1, \ldots, P_n, Q_1, \ldots Q_m$$

such that Q_1 is derived by resolution from some pair of premises, and for every $i = 2 \ldots m$, the clause Q_i is derived from

- Q_{i-1} and
- some premise in $\{P_1, \ldots, P_n\}$

(See figure 1 for an example of a linear resolution proof based on the medical database form the last lecture)

Theorem: (Completeness of linear resolution for logic programs) Let P be a logic program and let G be a goal. Suppose that $P, G \models \bot$. Then there exists a linear resolution proof of $\bot$ from the clauses P, G in which every clause is descended from the goal.

Moreover, we can construct this proof so that, at each step, the last clause generated will have the form

$$L_1 \wedge \ldots L_n \longrightarrow \bot$$

and the next step is to resolve this with some rule of the form $N \longrightarrow L_1$, producing

$$N \wedge L_2 \wedge \ldots L_n \longrightarrow \bot$$

That is: it suffices to always choose the *left-most* atom in the current descendent of the goal to resolve against.

Proofs in this form are called *SLD proofs with leftmost selection rule.* (S = selection, L= linear, for D = definite clauses)

This leaves *only one* choice to be made at each step: when there are several rules of the form $N \longrightarrow L_1$, which one to use?

If we consider all possibilities, we get a tree in which each branch corresponds to an attempted linear proof with leftmost selection.

(See figure 2 for the complete tree obtained in the medical database using the left-most slection rule.)

Most logic programming systems, like Prolog, search this tree in depth first fashion from left to right. Moreover the input clauses are ordered, and the possibilities are considerd from first to last.

Deductive Database Systems, such as Coral use optimized versions of the algorithm from the book.

Other systems, such as XSB, use a hybrid approach

More on this later.....

Predicate Calculus

Reading: Huth & Ryan, Section 2.1 - 2.2

Declarative sentences in natural language have a lot more structure than we have captured so far. Moreover, this structure plays an essential role in intuitively valid arguments:

> Margaret is John's supervisor.
> Thus, John has a supervisor.

If we express this in logic as two assertions P, Q, we find that $P \not\models Q$.

More examples ....

Dilbert is an employee. Every employee except the CEO has a supervisor. Dilbert is not the CEO. Thus, Dilbert has a supervisor.

Every employee's supervisor is out of his depth. Dilbert is an employee. Thus, Dilbert's supervisor is out of his depth.

Every pixel that is not contained in some window has the background colour. There is a pixel not contained in any window. Thus, there is a pixel that has the background colour.

All transactions in the database have been checked. Every transaction made yesterday is in the database. Thus, every transaction made yesterday has been checked.

What sorts of structure is there in these sentences that might account for the validity of these arguments?

- Names for objects: "John", "Margaret", "Dilbert"
- Complex expressions for objects: "Dilbert's supervisor"
- Properties of objects: "has a supervisor", "is in the database"
- Relationships between objects: "... is the manager of ...", "... is contained in ..."
- Quantifiers: "Every", "Somebody", "All", "there is"

Predicate Logic or *First Order Logic*, attempts to formalize these sorts of constructs.

Syntax of Predicate Logic, and Intuitive Semantics

Predicate Logic deals with some domain of discourse. *Terms* refer to objects in this domain.

Terms are constructed from:

- Constant terms: $a, b, c, \ldots$
  — these represent names, such as "Dilbert"
- Variables: $u, v, w, x, y, z \ldots$
  — these act as place holders for expressions referring to objects
- Function Symbols: $f, g, h, \ldots$
  — these represent functional operations such as "'s supervisor"
  — Each function symbol has an *arity*, i.e. the number of arguments it takes.

Terms are defined recursively: A terms is

- a constant c, or
- a variable x, or
- an expression $f(t_1, \ldots, t_n)$, where f is an n-ary function symbol and $t_1, \ldots, t_n$ are terms.

Examples:

1. `dilbert`
2. `supervisor(dilbert)`
3. `salary(dilbert)`
4. `sum(salary(dilbert),salary(supervisor(dilbert)))`
5. `sum(salary(x),salary(supervisor(x)))`

Intuitively,

1. refers to Dilbert
2. refers to Dilbert's supervisor
3. refers to Dilbert's salary
4. refers to the sum of Dilbert's salary and the salary of his supervisor
5. refers to the sum of the salary of the person referred to as "x" and the salary of the supervisor of that person. Once we decide what "x" refers to we can calculate this.

Predicates

The language of predicate logic also contains a set of predicate symbols: $P, Q, R, \ldots$

Each has an arity, i.e. a number of arguments that it takes.

Intuitively, a predicate symbol represents a relationship that may hold between its arguments.

Examples:

- `Stingy` of arity one
- `Supervises` of arity 2
- `Walked` of arity 3 (.. took .. for a walk at place ...)

An atomic formula of predicate logic has the form $P(t_1, \ldots, t_n)$, where P is a predicate symbol and $t_1, \ldots, t_n$ are terms.

Intuitively, this states that the relationship referred to by P applies to (is true of) the objects referred to by $t_1, \ldots, t_n$ (in that order).

Examples:

1. `Stingy(supervisor(dilbert))`
2. `Tiny(salary(dilbert))`
3. `Supervises(snoopy, dilbert)`
4. `Walked(charlie,snoopy,park)`
5. `Supervises(snoopy, supervisor(x))`

Note that in the last case, `x` is a variable, so we can't determine a truth value until we say what this refers to.

Intuitively, these formulas state that

1. Dilberts supervisor is stingy.

2. Dilberts salary is tiny.

3. Snoopy supervises Dilbert.

4. Charlie took Snoopy for a walk in the park.

5. Snoopy supervises the supervisor of x.

From the basis of atomic formulas, we can form more complex formulas using the operators of propositional logic.

Examples

1. $\neg$`Supervises(snoopy, dilbert)`
   Snoopy does not supervise Dilbert.
2. `Supervises(snoopy, dilbert)`$\longrightarrow$`Tiny(salary(dilbert))`
   (If Snoopy supervises Dilbert then Dilbert's salary is tiny.)
3. `Walked(charlie,snoopy,park)`$\vee$`Walked(snoopy,charlie,park)`
   (Either Charlie took Snoopy for a walk in the park or Snoopy took Charlie for a walk in the park.)
4. `Supervises(snoopy, dilbert)`$\longrightarrow$`Tiny(salary(x))`
   (If Snoopy supervises Dilbert then $\ldots_x$'s salary is tiny.)

Universal Quantifier

The universal quantifier "$\forall$" is used to capture expressions such as "All ... ." and "Every ... .".

Formally, if ϕ is a formula and x is a variable, then $\forall x(\phi)$ is a formula.

Intuitively, this states that ϕ holds for *every* possible value of the variable x.

Examples:

- Every project of Dilbert fails.
  $\forall x(\texttt{ProjectOf}(\texttt{Dilbert}, x) \longrightarrow \texttt{Fails}(x))$
- All employees have tiny salaries.
  $\forall x(\texttt{Employee}(x) \longrightarrow \texttt{Tiny}(\texttt{salary}(x)))$
- Every owner walks all their dogs in the park
  $\forall x(\texttt{Owner}(x) \wedge \forall y(\texttt{DogOf}(x, y) \longrightarrow \texttt{Walks}(x, y, \texttt{park})))$
  Or ...
  $\forall x \forall y(\texttt{Owns}(x, y) \longrightarrow \texttt{Walks}(x, y, \texttt{park}))$

Existential Quantifier

The quantifier "$\exists$" is used to capture expressions such as "Some... ...." "There is ... that ...." and "There exists... that ....".

Formally, if ϕ is a formula and x is a variable, then $\forall x(\phi)$ is a formula.

Intuitively, this states that ϕ holds for *some* possible value of the variable x.

Examples:

- Some project of Dilbert fails.
  $\exists x(\texttt{ProjectOf}(\texttt{dilbert}, x) \wedge \texttt{Fails}(x))$
- There is an employee who has a tiny salary.
  $\exists x(\texttt{Employee}(x) \wedge \texttt{Tiny}(\texttt{salary}(x)))$
- Someone owns a dog and walks it in the park
  $\exists x(\texttt{Owner}(x) \wedge \exists y(\texttt{DogOf}(x, y) \wedge \texttt{Walks}(x, y, \texttt{park})))$
  Or ...
  $\exists x \exists y(\texttt{Owns}(x, y) \wedge \texttt{Dog}(y) \wedge \texttt{Walks}(x, y, \texttt{park}))$
- Someone walks all their dogs in the park.
  $\exists x(\texttt{Person}(x) \wedge \forall y(\texttt{DogOf}(x, y) \longrightarrow \texttt{Walks}(x, y, \texttt{park})))$

Note the general pattern for these translations:

- "All A's B" is translated as $\forall x(A(x) \longrightarrow B(x))$
- "Some A B's" is translated as $\exists x(A(x) \wedge B(x))$