COMP4418: Knowledge Representation and Reasoning

Prolog I

Maurice Pagnucco
School of Computer Science and Engineering

COMP4418, Week 3

Prolog

- Prolog — *Programming in Logic*
- Invented early 70s by Alain Colmeraurer et *al.*, University of Marseille
- *Declarative language*
  - Specify goal and interpreter/compiler will work out how to achieve it
  - Traditional (imperative) languages require you to specify how to solve problem
- Prolog program specifies:
  - facts about objects and their relationships
  - rules about objects and their relationships

Reference: Ivan Bratko, *Prolog Programming for Artificial Intelligence*, Addison-Wesley, 2001.

Starting Prolog

Good open source Prolog implementation: SWI Prolog

https://www.swi-prolog.org

```
$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
```

Relations

- Prolog programs specify relationships among objects and properties of objects
- When we say, “John owns the book”, we are declaring the ownership relation between two objects: John and the book
- When we ask, “Does John own the book?”, we are querying the relationship
- Relationships can also be rules such as:

  Two people are sisters if
  both are female
  they have the same parents

- This is a rule that allows us to find out about a relationship even if the relationship isn’t explicitly declared

Programming in Prolog

- Declare facts describing explicit relationships between objects and properties of objects
- Define rules describing implicit relationships between objects or implicit object properties
- Ask questions about relationships between objects and object properties

Representing Regulations

The rules for entry into a professional computer science society are set out below:

> *An applicant to the society is acceptable if he or she has been nominated by two established members of the society and is eligible under the terms below:*
>
> - *the applicant graduated with a university degree*
> - *the applicant has two years of professional experience*
> - *the applicant pays a joining fee of $200.*
>
> *An established member is one who has been a member for at least two years.*

Facts

- Properties of objects; relationshps between objects
- Example
  - “Maurice lectures in course COMP4418”
  - Prolog: `lectures(maurice, comp4418)`
- Notice
  - Names of properties/relationships begin with lower-case character
  - Name of relationship appears as first term, objects appear as arguments
  - Fact terminated by ‘.’
  - Objects (*atoms*) also begin with lower-case characters
- `lectures(maurice, 4418)` also called a *predicate*

Facts

Let us return to the regulations example:

```
experience(fred, 3).
fee_paid(fred).
graduated(fred, unsw).
university(unsw).
nominated_by(fred, jim).
nominated_by(fred, mary).
joined(jim, 2015).
joined(mary, 2016).
current_year(2021).
```

Prolog Database

A collection of facts about a hypothetical computer science department:

```
% lectures(X, Y): person X lectures in course Y
lectures(tony, comp1001).
lectures(andrew, comp2041).
lectures(john, comp2041).
lectures(gernot, comp3231).
lectures(arun, comp4141).
lectures(sowmya, comp4411).
lectures(claude, comp4411).
lectures(maurice, comp4418).
lectures(adnan, comp4418).
lectures(adnan, comp9518).
lectures(wayne, comp4418).
lectures(arthur, comp9020).

% studies(X, Y): person X studies course Y
```

Queries

- Once we have a database of facts (and, soon, rules) we need to be able to ask questions of the information that is stored
- `lectures(maurice, comp4418)?`
- Notice:
  - Query is terminated by a question mark '?'
  - To determine answer (`yes` or `no`), Prolog consults database checking whether this is a known fact
  - For example, `lectures(bob,comp4418)?`

    ```
    **no
    ```
  - If answer is `yes`, query *succeeded*; otherwise, if answer is `no`, query *failed*

Variables

- Suppose we want to ask, “What subject does John teach?”
- This could be phrased as:

  Is there a subject, X, that John teaches?
- The variable X stands for an object that the questioner does not yet know about
- To answer the question, Prolog has to find the value of X, if it exists
- As long as we do not know the value of the variable, it is said to be *unbound*
- When a value is found, the variable is *bound* to that value

Variables

- A variable must begin with a capital letter or ‘_’
- To ask Prolog to find the subject that John teaches, type:

```
: lectures(john, Subject)?

Subject = comp2041
```

- To ask which subjects that Adnan teaches, ask:

```
: lectures(adnan, X)?

X = comp4418

X = comp9518
```

Prolog can find all possible ways to satisfy a query

Conjunction in Queries

- How do we ask, “Does Arthur teach Jack?”
- This can be answered by finding out whether Arthur lectures in a subject that Jack studies:

```
lectures(arthur, Subject), studies(jack, Subject)?
```

- i.e., Arthur lectures in subject, `Subject`, and Jack studies subject, `Subject`.
- *Subject* is a variable
- The question consists of two goals
- To find the answer, Prolog must find a single value for *Subject* that satisfies both goals

Conjunctions

- Who does Adnan teach:

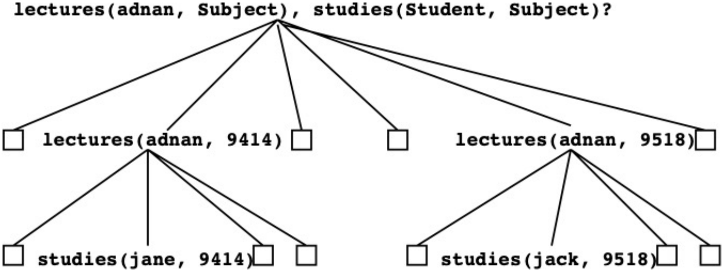
```
    : lectures(adnan, Subject), studies(Student, Subject)?
    Subject = comp4418
    Student = jane

    Subject = comp9518
    Student = jack
```

- Prolog solves problems by proceedings left to right and then *backtracking*
- Given the initial query, Prolog tries to solve
  `lectures(adnan, Subject)`
- There are twelve `lectures` clauses but only two have `adnan` as first argument
- Prolog chooses the first clause containing a reference to `adan` i.e.,
  `lectures(adnan, 4418)`

Proof Tree

- With `Subject = 4418`, it then tries to satisfy the next goal, viz `studies(Student, 4418)`
- After the solution is found, Prolog retraces its steps and looks for alternative solutions
- It may now go down the branch containing `lectures(adnan, 9518)` and try `studies(Student, 9518)`



Rules

- The previous question can be restated as a general rule:

  One person, `Teacher` *teaches another person,* `Student` **if**
  `Teacher` *lectures subject,* `Subject` **and**
  `Student` *studies* `Subject`

- In Prolog this is written as the:

```
teaches(Teacher, Student) :- % This is a clause
    lectures(Teacher, Subject),
    studies(Student, Subject).

teaches(adnan, Student)?
```

- Facts are *unit clauses* and rules are *non-unit clauses*

Rules

```
acceptable(Applicant) :-
    nominated(Applicant),
    eligible(Applicant).

nominated(Applicant) :-
    nominated_by(Applicant, Member1),
    nominated_by(Applicant, Member2),
    Member1 \= Member2,
    current_year(ThisYear),
    joined(Member1, Year1), ThisYear >= Year1 + 2,
    joined(Member2, Year2), ThisYear >= Year2 + 2,.

eligible(Applicant) :-
    graduated(Applicant, University), university(University),
    experience(Applicant, Experience), Experience >= 2,
    fee_paid(Applicant).
```

Clause Syntax

- ‘`:-`’ means “if” or “is implied by”. Also called “neck”
- The left hand side of the neck is the *head*
- The right hand side is called the *body*
- The comma, ‘,’ separating the goals stands for *and*

```
more_advanced(Student1, Student2) :-
    year(Student1, Year1),
    year(Student2, Year2),
    Year1 > Year2.
```

- Note the use of the *predefined predicate* ‘>’

```
more_advanced(jane, mary)?
more_advanced(jack, X)?
```

Structures

- Functional terms can be used to construct complex data structures
- E.g., to say that John owns the book *Foundation*, this may be expressed as:

  ```
  owns(john, ’Foundation’).
  ```

- Often objects have a number of attributes
- A book may have a title and an author:

  ```
  owns(john, book(’Foundation’, asimov)).
  ```

- To be more accurate we should give the author’s family and given names:

  ```
  owns(john, book(’Foundation’, author(asimov, isaac))).
  ```

Asking Questions with Structures

- How do we ask:
  “What books does John own that were written by someone called “Asimov”?

```
: owns(john, book(Title, author(asimov, GivenName)))?
Title = Foundation
GivenName = isaac

: owns(john, Book)?
Book = book(Foundation, author(asimov, isaac))

: owns(john, book(Title, Author))?
Title = Foundation
Author = author(asimov, isaac)
```

Databases

- A database of books in a library contains facts of the form:
  - `book(CatNo, Title, author(Family, Given)).`
  - `member(MemNo, name(Family, Given), Address).`
  - `loan(CatNo, MemNo, Borrowed, Due).`
- A member of the library may borrow a book
- A “loan” records:
  - the catalogue number of the book
  - the number of the member
  - the borrow date
  - the due date

Database Structures

- Dates are stored as structures:

  ```
  date(Year, Month, Day).
  ```

- E.g., `date(2001, 9, 8)` represents `8 September 2001`
- Names and addresses are all stored as character strings
- Which books has a member borrowed?

  ```
  has_borrowed(MemFamily, Title, CatNo) :-
      memb(MemNo, name(MemFamily, _), _),
      loan(CatNo, MemNo, _, _),
      book(CatNo, Title, _).
  ```

- Which books are overdue?

Overdue Books

```
later(date(Y, M, D1), date(Y, M, D2)) :- D1 > D2.
later(date(Y, M1, _), date(Y, M2, _)) :- M1 > M2.
later(date(Y1, _, _), date(Y2, _, _)) :- Y1 > Y2.

later(date(2001, 12, 3), date(1999, 8, 3))?

overdue(Today, Title, CatNo, MemFamily) :-
    loan(CatNo, MemNo, _, DueDate),
    later(Today, DueDate),
    book(CatNo, Title, _),
    memb(MemNo, name(MemFamily, _), _).
```

Due Date

```
due_date(date(Y, M1, D), date(Y, M2, D)) :-
    M1 < 12,
    M2 is M1 + 1.
due_date(date(Y1, 12, D), date(Y2, 1, D)) :-
    Y2 is Y1 + 1.
```

- `is` accepts two arguments
- The right hand argument must be an evaluable arithmetic expression
- The term is evaluated and unified with the left hand argument
- It *is not* an assignment statement
- Variables *cannot* be reassigned values
- Arguments of comparison operators can also be arithmetic expressions