CompSci.367

The Practice of Artificial Intelligence

Assoc. Prof. Ian Watson

Symbolic reasoning

- GOFAI* relies on the Physical Symbol System Hypothesis:
  - Intelligent activity is achieved through the use of
    - symbol patterns to represent the problem
    - operations on those patterns to generate potential solutions
    - search to select a solution among the possibilities
  - An AI representation language must
    - handle qualitative knowledge
    - allow new knowledge to be inferred from facts & rules
    - allow representation of general principles
    - capture complex semantic meaning
    - allow for meta-level reasoning (reasoning about reasoning)
    - e.g., Predicate Calculus (also, the basis of Prolog)

*** GOFAI = "Good Old Fashioned AI**

Predicate Calculus

- Predicate calculus (PC) is a language for representing knowledge, amenable to reasoning using inference rules
- the syntax of a language defines the form of statements
  - the building blocks of statements in the PC are terms and predicates
  - *terms* denote objects and properties
    - truth symbols `true` `false`
    - constant symbols `dave` `redBlock` `happy`
    - variable symbols `X` `Person` `Answer1`
    - function expressions `mother(bob)` `plus(1,3)`
  - *predicates* define relationships between objects (*arity* defines the number of arguments)
    - `mother/1` `plus/2`

Predicate calculus: sentences

- sentences are statements about the world
  - propositions (predicates applied to terms) are sentences
  - if S, S1 and S2 are sentences, then so are
    - ¬S (negation – NOT)
    - S1 ∧ S2 (conjunction – AND)
    - S1 ∨ S2 (disjunction – OR)
    - S1 ⇒ S2 (implication – IF-THEN)
    - ∀X S (universal quantification – FOR ALL X...)
    - ∃X S (existential quantification – THERE EXISTS an X...)

Predicate calculus: sentences

- male(dave)
- parent(dave, jack)
- ¬happy(chris)
- parent(dave, jack) ∧ parent(dave, charlie)
- happy(chris) ∨ ¬happy(chris)
- healthy(kelly) ⇒ happy(kelly)
- ∀X (healthy(X) ⇒ happy(X))
- ∃X parent(dave, X)

Predicate calculus: semantics

- the semantics of a language defines the meaning of statements
- an interpretation assigns meaning to terms/sentences
  - must focus on a particular domain (universe of objects)
  - terms are assigned values from the domain
    - constant → an object in the domain
    - variable → a subset of the domain
    - function symbol → a function mapping args to an object in the domain
  - predicate symbols are assigned mappings from args to true/false
    - e.g. DOMAIN: students in this class
    - patrick, bryanP, john, bryanJ, scott : mapped to actual people
    - friend function : maps a students to his/her friend
    - friend(patrick) → bryanP, friend(bryanP) → patrick
    - undergrad/1: maps a student to true if an undergrad, else false
    - grad/1: maps a student to true if a grad student, else false
    - male/1: maps a student to true if male, else false
    - female/1: maps a student to true if female, else false

Predicate calculus: semantics

- interpretation assigns true/false value to sentences
  - proposition assigned true/false according to predicate mapping
  - ¬S — true if S is false, else false
  - S1 ∧ S2 — true if both S1 and S2 are true, else false
  - S1 ∨ S2 — true if either S1 or S2 are true, else false
  - S1 ⇒ S2 — false if S1 is true and S2 is false, else true
  - ∀X S — true if S is true for all assignments to X
  - ∃X S — true if S is true for any assignment to X

Predicate calculus: semantics

- e.g. the following are all assigned true under the previous interpretation
  - undergrad(patrick) grad(scott) male(john)
  - ¬undergrad(john) ¬female(bryanP)
  - undergrad(patrick) ∧ undergrad(friend(patrick))
  - undergrad(bryanP) ∨ undergrad(bryanJ)
  - ∀X male(X) ∃G grad(G)
  - ∀X (undergrad(friend(X)) ⇒ undergrad(X))

Predicate calculus: logical consequence

- the semantics of the predicate calculus provides a basis for a formal theory of logical inference
- an interpretation that makes a sentence true satisfies it
  - a set of expressions {S1, ..., Sn} logically implies S if
    - every interpretation that satisfies {S1, ..., Sn} satisfies S
    - equivalently, we could say S is a logical consequence of {S1, ..., Sn}
    - shorthand notation: {S1, ..., Sn} ⊨ S
    - e.g.,
    - {itRains⇒getWet, goSwim⇒getWet, itRains∨goSwim } ⊨ getWet
    - {∀P(human(P)⇒mortal(P)), human(socrates)} ⊨ mortal(socrates)

Predicate calculus: inference

- proving logical consequence via interpretations is difficult
  - requires reasoning over all interpretations
- alternatively, a proof procedure can generate logical consequences
  - a proof procedure is a combination of inference rules and an algorithm for applying the rules to generate logical consequences
  - example inference rules:
  - *Modus Ponens:* if S1 and S1⇒S2 are true, then infer S2
  - *And Elimination:* if S1∧S2 is true, then infer S1 and infer S2
  - *And Introduction:* if S1 and S2 are true, then infer S1∧S2
  - *Universal Instantiation:* if ∀X p(X) is true, then infer p(*a*) for any *a*

Inference example

- initial knowledge:
  { ∀P(human(P) ⇒ mortal(P)),
  human(socrates) }

  ⇓ extend using Universal Instantiation

  { ∀P(human(P) ⇒ mortal(P)),
  human(socrates),
  human(socrates) ⇒ mortal(socrates) }

  ⇓ extend using Modus Ponens

  { ∀P(human(P) ⇒ mortal(P)),
  human(socrates),
  human(socrates) ⇒ mortal(socrates),
  mortal(socrates) }

Inference example 2

- initial knowledge:

{∀P(student(P) ⇒ tired(P)),
∀S(csmajor(S) ⇒ overworked(S)),
∀X(tired(X) ∧ overworked(X) ⇒ testy(X)),
student(patrick),
csmajor(patrick)}

AI programming languages

- there are 2 major programming language used for AI research
  - LISP (List Processing)
    - older (1957), more established in the U.S.
    - uses a functional style (but is still declarative)
    - CLIPS is based on LISP
  - Prolog (Programming in Logic)
    - newer (1971), more widely used in Europe & Asia
    - uses a totally declarative style, a.k.a. logic programming
    - attractive features: built-in notion of search
    - general data structures
    - powerful primitives for symbol manipulation

AI programming languages

- Prolog evolved out of the automated deduction community – 2 drivers:
  - (1) focus on a subset of predicate calculus
    - *programs are collections of logical statements & relations*
  - (2) implement a simple but efficient proof procedure
    - *Prolog interpreter applies inference rules to perform deduction*
  - logic programming: *computation = logical deduction from program statements*

Prolog

- Prolog programs are statements from the Horn clause subset of predicate calculus
- facts (i.e., propositions)
  - all variables are assumed to be universally quantified, so ∀ is implicit
  - terminate each fact with a period

```
male(dave).
parent(dave, jack).
mortal(X).
```

Prolog

- rules (i.e., implications of the form: P1 ∧ ... ∧ Pn ⇒ C)
  - again, ∀ is implicit & terminate rule with a period
  - slightly different notation in Prolog (to suit standard keyboards)
    - (1) conclusion is on the left (the Horn clause form)
    - (2) :- replaces ⇐
    - (3) comma replaces ∧
    - e.g., C :- P1, ..., Pn.

```
happy(chris) :- healthy(chris).
```
If Chris is healthy, Chris is happy
```
mortal(X) :- human(X).
father(F, C) :- parent(F, C), male(F).
grandfather(F, G) :- father(F, C), parent(C, G).
```


Prolog's basic model of computation

- the programmer states relations between objects as facts & rules
  - `parent(dave, charlie).` `parent(dave, jack).`
  - `parent(laura, charlie).` `parent(laura, jack).`
  - `male(dave).` `male(charlie).` `male(jack).` `female(laura).`
  - `father(dave, charlie) :- parent(dave, charlie), male(dave).`
  - `mother(M, C) :- parent(M, C), female(M).`

Prolog's basic model of computation

- the job of the Prolog interpreter is to receive queries and determine whether they are logical consequences of the facts & rules

in simple case, merely looks up facts
```
?- parent(dave, charlie).
Yes
```

more generally, may need to perform inferences using the rules
```
?- father(dave, charlie).
Yes
?- father(charlie, dave).
No
```

may even require picking the right instances of rules
```
?- mother(laura, charlie).
Yes
```

Prolog's basic model of computation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% family.pro
%%%
%%% Encodes family relations.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

parent(dave, charlie).
parent(dave, jack).
parent(laura, charlie).
parent(laura, jack).

male(dave).
male(charlie).
male(jack).

female(laura).

father(F, C) :-
    parent(F, C), male(F).

mother(M, C) :-
    parent(M, C), female(M).
```

- A Prolog program is just a database of facts & rules that define relations between objects
- % begins a comment in Prolog
- name files with .pro or .pl extensions
- good practice to group all definitions of the same relation together
  (some Prolog interpreters complain otherwise)
- since a period marks the end of a rule, can split across lines for readability

Prolog environment

```
Welcome to SWI-Prolog (Version 4.0.11)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

?- consult('a:/family.pro')).
% a:/family.pro compiled 0.00 sec, 1,516 bytes

Yes

?- parent(laura, charlie).

Yes

?- mother(laura, charlie).

Yes

?- mother(dave, charlie).

No
```

- SWI-Prolog is a free Prolog interpreter/environment
  - online HTML reference manual:
  - http://www.swi-prolog.org/
- program files are simply text files, use your favorite text editor
- once a file is created, its knowledge (facts & rules) can be loaded by consulting that file
- once the facts & rules have been consulted, can enter queries and the interpreter determines logical consequence