Chapter 2 & 3 (Part 2): A Representation & Reasoning System & Using Definite Knowledge

- Proofs (2.7)
  (How to compute logical consequences ?)
- Functions Symbols: Language Extension (2.8)
- Representing Abstract Concepts (3.5)

D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*, Oxford University Press, January 1998

Proofs (2.7)

- A proof is a mechanically derivable demonstration that a formula logically follows from a knowledge base.
- Given a proof procedure, KB $\vdash$ g means g can be derived from knowledge base KB.
- Recall KB $\models$ g means g is true in all models of KB.
- A proof procedure is sound if KB $\vdash$ g implies KB $\models$ g (every thing derived from KB is a logical consequence).
- A proof procedure is complete if KB $\models$ g implies KB $\vdash$ g.

Proof (2.7) (cont.)

- Bottom-up Ground Proof Procedure
  - One rule of derivation, a generalized form of modus ponens:
    - If "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" is a clause in the knowledge base, and each bi has been derived, then h can be derived.
  - You are forward chaining on this clause (going forward from what is known but not backward from the query!).
  - (This rule also covers the case when m = 0.)

Proof (2.7) (cont.)

- Bottom-up proof procedure

$KB \vdash g$ if $g \in C$ at the end of this procedure:

$C := \{\}$; *(C: consequence set)*

repeat

  select clause "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" in KB such that

    $b_i \in C$ for all i, and

    $h \notin C$;

  $C := C \cup \{h\}$

until no more clauses can be selected.

Proof (2.7) (cont.)

◆**Example**

$a \leftarrow b \wedge c.$

$b \leftarrow d \wedge e.$

$b \leftarrow g \wedge e.$

$c \leftarrow e.$

$d.$

$e.$

$f \leftarrow a \wedge g.$

Proof (2.7) (cont.)

◆**Example (cont.)**

{}
{d}
{e,d}
{c,e,d}
{b,c,e,d}
{a,b,c,e,d}

Proof (2.7) (cont.)

- **Soundness of bottom-up proof procedure**

If KB $\vdash$ g then KB $\models$ g.

Suppose there is a g (atom) such that KB $\not\models$ g and KB $\vdash$ g.

Let h be the first atom added to C that's not true in every model of KB. Suppose h isn't true in model I of KB.There must be a clause in KB of form: $h \leftarrow b_1 \wedge \ldots \wedge b_m$ where:

Each b_i is true in I and h is false in I. So this clause is false in I.

Therefore I isn't a model of KB.

Contradiction: thus no such g exists !

Proof (2.7) (cont.)

- **Fixed Point**
  - The C generated at the end of the bottom-up algorithm is called a fixed point.
  - Let I be the interpretation in which every element of the fixed point is true and every other atom is false Then I is a model of KB.

    <u>Proof:</u> suppose $h \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB is false in I. The only way this could happen is if $b_1, b_2 \ldots, b_m \in C$ and $h \notin C$. Thus h can be added to C.

    Contradiction to C being the fixed point.
  - I is called a Minimal Model.

Proof (2.7) (cont.)

- **Completeness**
  - If KB $\models$ g then KB $\vdash$ g.

    Suppose KB $\models$ g. Then g is true in all models of KB.

    Thus g is true in the minimal model (defined by the fixed point).

    Thus g is generated by the bottom up algorithm.

    Thus KB $\vdash$ g.

Proof (2.7) (cont.)

- **Top-down Ground Proof Procedure**
  - Idea: search backward from a query to determine if it is a logical consequence of *KB*.
  - An answer clause is of the form:

$$yes \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m$$

  - The SLD Resolution of this answer clause on atom a_i with the clause:

$$a_i \leftarrow b_1 \wedge \ldots \wedge b_p$$

is the answer clause

$$yes \leftarrow a_1 \wedge \ldots \wedge a_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge a_{i+1} \wedge \ldots \wedge a_m$$

Proof (2.7) (cont.)

- **Top-down Ground Proof Procedure (cont.)**
  - **Derivations**
    - An answer is an answer clause with $m = 0$. That is, it is the answer clause *yes* ← .
    - A derivation of query "?$q_1 \wedge \ldots \wedge q_k$" from *KB* is a sequence of answer clauses $y_0; y_1, \ldots, y_n$ such that
      - y_0 is the answer clause *yes* ← $q_1 \wedge \ldots \wedge q_k$,
      - y_i is obtained by resolving y_{i-1} with a clause in *KB*, and
      - y_n is an answer.

Proof (2.7) (cont.)

- **Top-down ground proof procedure (cont.)**
  - Search backward from a query to determine if it is a logical consequence of the given clauses

```
Solve(q1 ∧ ∧… ∧ qk)
ac:="yes← q1 ∧ ∧… ∧ qk"
repeat
  select a conjunct ai from the body of ac
  choose clause C from KB with ai as head
  replace ai in the body of ac by the body of C
until ac is an answer
```

Proof (2.7) (cont.)

- **Example**

a ← b ∧ c.

b ← d ∧ e.

b ← g ∧ e.

c ← e.

d.

e.

f ← a ∧ g.

?a.

Proof (2.7) (cont.)

- **Example (cont.)**

yes ← a.

yes ← b ∧ **c.** (select b and choose b ← d ∧ e)

yes ← d ∧ e ∧ **c.** (or g ∧ e: failure: no rule chosen headed by g!)

yes ← e ∧ c.

yes ← c.

yes ← e.

yes ← .

"The two operations involved are "select" and "choose".

Proof (2.7) (cont.)

- Nondeterministic Choice
  - "Don't-care nondeterminism": If one selection doesn't lead to a solution, there is no point trying other alternatives. select
  - "Don't-know nondeterminism": If one choice doesn't lead to a solution, other choices may. choose

Proof (2.7) (cont.)

- Reasoning with Variables
  - An instance of an atom or a clause is obtained by uniformly substituting terms for variables.
  - A substitution is a finite set of the form

    $$\{V_1/ t_1, \ldots, V_n/ t_n\},$$

    where each V_i is a distinct variable and each t_i is a term.
  - The application of a substitution $\sigma = \{V_1/ t_1, \ldots, V_n / t_n\}$ to an atom or clause e, written e, is the instance of e with every occurrence of V_i replaced by t_i.

Proof (2.7) (cont.)

Application Examples

The following are substitutions:

- $\sigma_1 = \{X/A, Y/b, Z/C, D/e\}$
- $\sigma_2 = \{A/X, Y/b, C/Z, D/e\}$
- $\sigma_3 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\}$

The following shows some applications:

- $p(A, b, C, D)\sigma_1 = p(A, b, C, e)$
- $p(X, Y, Z, e)\sigma_1 = p(A, b, C, e)$
- $p(A, b, C, D)\sigma_2 = p(X, b, Y, e)$
- $p(X, Y, Z, e)\sigma_2 = p(X, b, Y, e)$
- $p(A, b, C, D)\sigma_3 = p(V, b, W, e)$
- $p(X, Y, Z, e)\sigma_3 = p(V, b, W, e)$

Proof (2.7) (cont.)

- Unifiers
  - Substitution σ is a unifier of e_1 and e_2 if $e_1\sigma = e_2\sigma$.
  - Substitution is a most general unifier (mgu) of e_1 and e_2 if
    - σ is a unifier of e_1 and e_2; and
    - if substitution σ' also unifies e_1 and e_2, then $e\sigma'$ is an instance of $e\sigma$ for all atoms e.
  - If two atoms have a unifier, they have a most general unifier.

Proof (2.7) (cont.)

- Unification Example

$p(A, b, C, D)$ and $p(X, Y, Z, e)$ have as unifiers:

- $\sigma_1 = \{X/A, Y/b, Z/C, D/e\}$
- $\sigma_2 = \{A/X, Y/b, C/Z, D/e\}$
- $\sigma_3 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\}$
- $\sigma_4 = \{A/a, X/a, Y/b, C/c, Z/c, D/e\}$
- $\sigma_5 = \{X/A, Y/b, Z/A, C/A, D/e\}$
- $\sigma_6 = \{X/A, Y/b, Z/C, D/e, W/a\}$

The first three are most general unifiers.

The following substitutions are not unifiers:

- $\sigma_7 = \{Y/b, D/e\}$
- $\sigma_8 = \{X/a, Y/b, Z/c, D/e\}$

Proof (2.7) (cont.)

- Most General Unifiers
  - Example:

    Atoms e_1= p(X,Y) and e_2=p(Z,Z) have as unifiers: σ_1={X/b, Y/b, Z/b}, σ_2={X/c, Y/c, Z/c} and σ_3={X/Z, Y/Z}. The σ_3 unifier is mgu, since:

    (i) σ_3 unifies e_1 and e_2

    (ii) If $\sigma_1 = \sigma'$ then $e_1\sigma'$ =p(b,b) is an instance of $e_1\sigma_3$=p(Z,Z)

    Similarly, $e_2\sigma'$ =p(b,b) is an instance of $e_2\sigma_3$=p(Z,Z)

    If $\sigma_2 = \sigma'$ then $e_1\sigma'$ =p(c,c) is an instance of $e_1\sigma_3$=p(Z,Z)

    Similarly, $e_2\sigma'$ =p(c,c) is an instance of $e_2\sigma_3$=p(Z,Z)

Proof (2.7) (cont.)

- **Bottom-up Procedure with Variables**
  - A ground instance of a clause is obtained by uniformly substituting constants appearing in a KB or in any query for the variables in the clause

Proof (2.7) (cont.)

- **Example**

q(a).
q(b).
r(a).
s(W)← r(W).
p(X,Y)← q(X) ∧ s(Y).

The set of all ground instances is:
q(a).
q(b).
r(a).
s(a)← r(a)
s(b)← r(b)

Proof (2.7) (cont.)

- **Example (cont'd)**

$p(a,a) \leftarrow q(a) \wedge s(a).$
$p(a,b) \leftarrow q(a) \wedge s(b).$
$p(b,a) \leftarrow q(b) \wedge s(a).$
$p(b,b) \leftarrow q(b) \wedge s(b)$

the bottom-up procedure will derive:

q(a), q(b), r(a), s(a), p(a,a) and p(b,a).

Proof (2.7) (cont.)

- Definite Resolution with Variables: Top-down Derivation
  - A generalized answer clause is of the form
  $$yes(t_1,\ldots,t_k) \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m$$
  where $t_1, \ldots, t_k$ are terms and $a_1, \ldots, a_m$ are atoms.
  - The SLD resolution of this generalized answer clause on a_i with the clause
  $$a \leftarrow b_1 \wedge \ldots \wedge b_p$$
  - where a_i and a have most general unifier θ, is the clause:
  $$(yes\ (t_1, \ldots, t_k) \leftarrow$$
  $$a_1 \wedge \ldots \wedge a_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge a_{i+1} \wedge \ldots \wedge a_m\theta$$

Proof (2.7) (cont.)

- To solve query ?*B* with variables $V_1, \ldots, V_k$

```
Set ac to generalized answer clause yes(V1, …, Vk)← B;
while ac is not an answer do \\ (yes ←.) is false
   Suppose ac is yes(t1, …,  tk) ← a1 ∧ a2 ∧ … ∧ am
   Select atom ai in the body of ac;
   Choose clause a ← b1 ∧ … ∧ bp in KB;
   Rename all variables in a ←  b1 ∧ … ∧bp to have new names;
   Let θ be the most general unifier of ai and a.
   Fail if they don't unify;
   Set ac to
   (yes(t1, …,  tk) ← a1 ∧ … ∧ ai-1∧ b1 ∧ …∧ bp ∧ ai+1 ∧ …∧ am)θ
end while.
```

Proof (2.7) (cont.)

- Example1

Consider the database of Figure 2.3 (page 44) and the query:

?two_doors_east(R,r107).

Proof (2.7) (cont.)

Solution:

?two_doors_east(R, r107).

yes(R)←two_doors_east(R, 107) % this is ac

% resolve with two_doors_east(E_1,W_1)←imm_east(E_1,W_1) ∧ imm_east(M_1,W_1)

% substitution: {E_1/R, W_1/r107} a_i

yes(R)←imm_east(R,M_1) ∧ imm_east(M_1, r107) a

% select leftmost conjunct and resolve with imm_east(E_2,W_2)←imm_east(W_2,E_2)

% substitution θ: {E_2/R, W_2/M_1} (θ mgu for a_i and a)

yes(R)←imm_west(M_1, R) ∧ imm_east(M_1, r107)

% select leftmost conjunct and resolve with imm_west(r109,r111) *****

% substitution: {M_1/r109, R/r111}

yes(r111)←imm_east(r109, r107)

% resolve with imm_east(E_3,W_3)←imm_west(W_3,E_3)

% substitution: {E_3/r109, W_3/r107}

yes(r111)←imm_west(r107, r109)

% resolve with imm_west(r107, r109) and substitution: {}

yes(r111)←.

Example2 (Class Exercise)

$live(Y) \leftarrow connected\_to(Y, Z) \wedge live(Z).$ $live(outside)$

$connected\_to(w_6, w_5).$ $connected\_to(w_5, outside).$

$?live(A).$

Resolve and Substitute: {Y_1/A}

$yes(A) \leftarrow live(A).$

$yes(A) \leftarrow connected\_to(A, Z_1) \wedge live(Z_1).$

$yes(w_6) \leftarrow live(w_5).$ Resolve with connected_to(w_6,w_5)

$yes(w_6) \leftarrow connected\_to(w_5, Z_2) \wedge live(Z_2).$

$yes(w_6) \leftarrow live(outside).$

$yes(w_6) \leftarrow .$

Function Symbols: Language Extension (2.8)

- Often we want to refer to individuals in terms of components.
- Examples: 4:55 p.m. English sentences. A class list.
- We extend the notion of term . So that a term can be $f(t_1, \ldots, t_n)$ where f is a function symbol and the t_i are terms.
- In an interpretation and with a variable assignment, term $f(t_1, \ldots, t_n)$ denotes an individual in the domain.
- With one function symbol and one constant we can refer to infinitely many individuals.

Representing Abstract Concept: Lists (3.5)

- A list is an ordered sequence of elements.
- Let's use the constant nil to denote the empty list, and the function cons(H, T) to denote the list with first element H and rest-of-list T. These are not built-in.
- The list containing david, alan and randy is

  Cons(david, cons(alan, cons(randy, nil)))
- append(X, Y, Z) is true if list Z contains the elements of X followed by the elements of Y

  append(nil, Z, Z).

  append(cons(A, X), Y, cons(A, Z)) ← append(X, Y, Z).