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)

Proofs (2.7)

◆ A proof is a mechanically derivable demonstration that a formula logically follows from a knowledge base.

◆ Given a proof procedure, KB ⊢ g means g can be derived from knowledge base KB.

◆ Recall KB ⊨ g means g is true in all models of KB.

◆ A proof procedure is sound if KB ⊢ g implies KB ⊨ g (every thing derived from KB is a logical consequence).

◆ A proof procedure is complete if KB ⊬ g implies KB ⊢ 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 \land \ldots \land b_m” is a clause in the knowledge base, and each b_i 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

\[ KB \vdash g \text{ if } g \in C \text{ at the end of this procedure:} \]

\[
C := \{\}; \quad (C: \text{consequence set})
\]

repeat

select clause “\( h \leftarrow b_1 \land \ldots \land b_m \)” 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 \land c.
  
  b \leftarrow d \land e.
  
  b \leftarrow g \land e.
  
  c \leftarrow e.
  
  d.
  
  e.
  
  f \leftarrow a \land 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 $\text{KB} \not\models g$, then $\text{KB} \models g$.
Suppose there is a $g$ (atom) such that $\text{KB} \not\models g$ and $\text{KB} \not\models g$.
Let $h$ be the first atom added to $C$ that’s not true in every model of $\text{KB}$. Suppose $h$ isn’t true in model $I$ of $\text{KB}$. There must be a clause in $\text{KB}$ of form: $h \leftarrow b_1 \land \ldots \land b_m$ 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 $\text{KB}$.
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 $\text{KB}$.

Proof: suppose $h \leftarrow b_1 \land \ldots \land b_m$ 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 \( \text{KB} \models g \) then \( \text{KB} \vdash g \).

Suppose \( \text{KB} \models g \). Then \( g \) is true in all models of \( \text{KB} \).
Thus \( g \) is true in the minimal model (defined by the fixed point).
Thus \( g \) is generated by the bottom up algorithm.
Thus \( \text{KB} \vdash g \).

Proof (2.7) (cont.)

◆ Top-down Ground Proof Procedure

◆ Idea: search backward from a query to determine if it is a logical consequence of \( \text{KB} \).

◆ An answer clause is of the form:

\[
\text{yes} \leftarrow a_1 \land a_2 \land \ldots \land a_m
\]

◆ The SLD Resolution of this answer clause on atom \( a_i \) with the clause:

\[
a_i \leftarrow b_1 \land \ldots \land b_p
\]

is the answer clause

\[
\text{yes} \leftarrow a_1 \land \ldots \land a_{i-1} \land b_1 \land \ldots \land b_p \land a_{i+1} \land \ldots \land a_m
\]
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 \( \text{yes} \leftarrow \).

◆ A derivation of query “\(?q_1 \land \ldots \land q_k\)” from \( KB \) is a sequence of answer clauses \( y_0, y_1, \ldots, y_n \) such that

  ◆ \( y_0 \) is the answer clause \( \text{yes} \leftarrow q_1 \land \ldots \land q_k \),
  ◆ \( y_i \) 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(q_1 \land \ldots \land q_k)
ac:="yes\leftarrow q_1 \land \ldots \land q_k"
repeat
  select a conjunct \( a_i \) from the body of ac
  choose clause C from KB with \( a_i \) as head
  replace \( a_i \) in the body of ac by the body of C
until ac is an answer
```
Proof (2.7) (cont.)

◆Example

\[ a \leftarrow b \land c. \]
\[ b \leftarrow d \land e. \]
\[ b \leftarrow g \land e. \]
\[ c \leftarrow e. \]
\[ d. \]
\[ e. \]
\[ f \leftarrow a \land g. \]
\[ ?a. \]

Proof (2.7) (cont.)

◆Example (cont.)

\[ \text{yes} \leftarrow a. \]
\[ \text{yes} \leftarrow b \land c. \ (\text{select } b \text{ and choose } b \leftarrow d \land e) \]
\[ \text{yes} \leftarrow d \land e \land c. \ (\text{or } g \land e: \text{failure: no rule chosen headed by } g!) \]
\[ \text{yes} \leftarrow e \land c. \]
\[ \text{yes} \leftarrow c. \]
\[ \text{yes} \leftarrow e. \]
\[ \text{yes} \leftarrow. \]

"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.)

The following are substitutions:
- \( \sigma_1 = \{X/A, Y/b, Z/C, D/e\} \)
- \( \sigma_2 = \{A/X, Y/b, C/Z, D/e\} \)
- \( \sigma_3 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\} \)

Application Examples

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 \( \sigma \) 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
    - \( \sigma \) is a unifier of \( e_1 \) and \( e_2 \); and
    - if substitution \( \sigma' \) 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:
- \( \sigma_1 = \{X/b, Y/b, Z/b\} \)
- \( \sigma_2 = \{X/c, Y/c, Z/c\} \)
- \( \sigma_3 = \{X/Z, Y/Z\} \). The \( \sigma_3 \) unifier is mgu, since:

1. \( \sigma_3 \) unifies \( e_1 \) and \( e_2 \)
2. If \( \sigma = \sigma' \) 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 \( \sigma = \sigma' \) 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

Example

\[ \begin{align*}
q(a). \\
q(b). \\
r(a). \\
s(W) &\leftarrow r(W). \\
p(X,Y) &\leftarrow q(X) \land s(Y). \\
\end{align*} \]

The set of all ground instances is:

\[ \begin{align*}
q(a). \\
q(b). \\
r(a). \\
s(a) &\leftarrow r(a) \\
s(b) &\leftarrow r(b) \\
\end{align*} \]
Proof (2.7) (cont.)

**Example (cont’d)**

\[ p(a, a) \iff q(a) \land s(a). \]
\[ p(a, b) \iff q(a) \land s(b). \]
\[ p(b, a) \iff q(b) \land s(a). \]
\[ p(b, b) \iff q(b) \land s(b). \]

the bottom-up procedure will derive:

\[ q(a), q(b), r(a), s(a), p(a, a) \text{ and } p(b, a). \]

---

Proof (2.7) (cont.)

**Definite Resolution with Variables: Top-down Derivation**

A generalized answer clause is of the form

\[ \text{yes}(t_1, ..., t_k) \iff a_1 \land a_2 \land ... \land a_m \]

where \( t_1, ..., t_k \) are terms and \( a_1, ..., a_m \) are atoms.

The SLD resolution of this generalized answer clause on \( a_i \) with the clause

\[ a \iff b_1 \land ... \land b_p \]

where \( a_i \) and \( a \) have most general unifier \( \theta \), is the clause:

\[ \text{yes}(t_1, ..., t_k) \iff a_1 \land ... \land a_{i-1} \land b_1 \land ... \land b_p \land a_{i+1} \land ... \land a_m \theta \]
Proof (2.7) (cont.)

◆ To solve query \( B \) with variables \( V_1, \ldots, V_k \)

Set \( ac \) to generalized answer clause \( \text{yes}(V_1, \ldots, V_k) \leftarrow B \);
while \( ac \) is not an answer do \( \text{(yes } \leftarrow \text{)} \) is false

Suppose \( ac \) is \( \text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land a_2 \land \ldots \land a_m \)
Select atom \( a_1 \) in the body of \( ac \);
Choose clause \( a \leftarrow b_1 \land \ldots \land b_p \) in KB;
Rename all variables in \( a \leftarrow b_1 \land \ldots \land b_p \) to have new names;
Let \( \theta \) be the most general unifier of \( a_1 \) and \( a \).
Fail if they don’t unify;
Set \( ac \) to
\[
(\text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_{i-1} \land b_1 \land \ldots \land b_p \land a_{i+1} \land \ldots \land a_m) \theta
\]
end while.

Proof (2.7) (cont.)

◆ Example1

Consider the database of Figure 2.3 (page 44) and the query:
\(? \text{two}_\text{doors}_\text{east}(R, r107)\).
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(E1, W1) < imm_east(E1, W1) ^ imm_east(M1, W1)
% substitution: {E1/R, W1/r107}
yes(R) < imm_east(R, M1) ^ imm_east(M1, r107)
% select leftmost conjunct and resolve with imm_east(E2, W2) < imm_east(W2, E2)
% substitution: {E2/R, W2/M1} (θ mgu for a_i and a)
yes(R) < imm_west(M1, R) ^ imm_east(M1, R107)
% select leftmost conjunct and resolve with imm_west(r109, r111) *****
% substitution: {M1/r109, R/r111}
yes(r111) < imm_west(r109, r107)
% resolve with imm_west(E3, W3) < imm_west(W3, E3)
% substitution: {E3/r109, W3/r107}
yes(r111) < imm_west(r107, r109)
% resolve with imm_west(r107, r109) and substitution: {}
yes(r111).
```

Example 2 (Class Exercise)

```
live(Y) ← connected_to(Y, Z) ∧ live(Z).  live(outside)
connected_to(w6, w5).  connected_to(w5, outside).

?live(A).

yes(A) ← live(A).  Resolve and Substitute: {Y/A}
yes(A) ← connected_to(A, Z1) ∧ live(Z1).
yes(w6) ← live(w5).  Resolve with connected_to(w6, w5)
yes(w6) ← connected_to(w5, Z2) ∧ live(Z2).
yes(w6) ← live(outside).
yes(w6) ← .
```
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)) $\iff$ append(X, Y, Z).