Reasoning with Variables

- An instance of an atom or a clause is obtained by uniformly substituting terms for variables. Every instance of the same variable is replaced by the same term.
- A substitution is a finite set of the form $\{V_1/t_1, \dots, V_n/t_n\}$, where each V_i is a distinct variable and each t_i is a term.
- The application of a substitution $\sigma = \{V_1/t_1, \dots, V_n/t_n\}$ to an atom or clause e, written $e\sigma$, is the instance of e with every occurrence of V_i replaced by t_i.

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:

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

Unifiers

- Substitution σ is a unifier of e_1 and e_2 if $e_1\sigma = e_2\sigma$.
- Substitution σ 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.
- If there are multiple most general unifiers, they only differ in the names of the variables.

procedure *unify*(t_1, t_2) ▷ Returns mgu of t_1 and t_2 or $\bot$.

    $E := \{t_1 = t_2\}$ ▷ Set of equality statements
    $S := \{\}$ ▷ Substitution
    while $E \neq \{\}$ **do**
        select and remove $x = y$ from E
        if y is not identical to x **then**
            if x is a variable **then**
                replace x with y in E and S
                $S := \{x/y\} \cup S$
            else if y is a variable **then**
                replace y with x in E and S
                $S := \{y/x\} \cup S$
            else if x is $p(x_1, \ldots, x_n)$ and y is $p(y_1, \ldots, y_n)$ **then**
                $E := E \cup \{x_1 = y_1, \ldots, x_n = y_n\}$
            else
                return $\bot$ ▷ t_1 and t_2 do not unify
    return S ▷ S is mgu of t_1 and t_2

Examples

- unify $p(A, b, C, D)$ and $p(X, Y, Z, e)$
  $\{A/X, Y/b, C/Z, D/e\}$
- unify $p(A, b, A, D)$ and $p(X, X, Z, Z)$
  $\{A/b, X/b, Z/b, D/b\}$
- unify $p(A, b, A, d)$ and $p(X, X, Z, Z)$
  $\bot$
- unify $n([sam, likes, prolog], L2, I, C1, C2)$ and
  $n([P|R], R, P, [person(P)|C], C)$
  $\{P/sam,\ R/[likes, prolog],\ L2/[likes, prolog],\ I/sam,$
  $C1/[person(sam)|C2],\ C/C2\}$

Logical Consequence

Atom g is a logical consequence of KB if and only if:

- g is an instance of a fact in KB, or
- there is an instance of a rule

$$g \leftarrow b_1 \wedge \ldots \wedge b_k$$

in KB such that each b_i is a logical consequence of KB.

Aside: Debugging false conclusions

To debug answer g that is false in the intended interpretation:

- If g is a fact in KB, this fact is wrong.
- Otherwise, suppose g was proved using the rule:

$$g \leftarrow b_1 \wedge \ldots \wedge b_k$$

  where each b_i is a logical consequence of KB.
  - If each b_i is true in the intended interpretation, this clause is false in the intended interpretation.
  - If some b_i is false in the intended interpretation, debug b_i.

Proofs

- 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 $KB \models g$.
- A proof procedure is complete if $KB \models g$ implies $KB \vdash g$.

Bottom-up proof procedure

$KB \vdash g$ if there is g' added to C in this procedure where $g = g'\theta$:

$C := \{\}$;
repeat
  select clause "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" in KB such that
    there is a substitution θ such that
    for all i, there exists $b'_i \in C$ and θ'_i where $b_i\theta = b'_i\theta'_i$ and
    there is no $h' \in C$ and θ' such that $h'\theta' = h\theta$
  $C := C \cup \{h\theta\}$
until no more clauses can be selected.

Example

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

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

$$
\begin{aligned}
C = \{ & live(outside),\\
& connected\_to(w_6, w_5),\\
& connected\_to(w_5, outside),\\
& live(w_5),\\
& live(w_6)\}
\end{aligned}
$$

Soundness of bottom-up proof procedure

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

- Suppose there is a g such that $KB \vdash g$ and $KB \not\models g$.
- Then there must be a first atom added to C that has an instance that isn't true in every model of KB. Call it h.
- Suppose h isn't true in model I of KB.
- There must be an instance of clause in KB of form

$$h' \leftarrow b_1 \wedge \ldots \wedge b_m$$

  where $h = h'\theta$ and $b_i\theta$ is an instance of an element of C.
  - Each $b_i\theta$ is true in I.
  - h is false in I.
  - So an instance of this clause is false in I.
  - Therefore I isn't a model of KB.
  - Contradiction.

Fixed Point

- The C generated by the bottom-up algorithm is called a fixed point.
- C can be infinite; we require the selection to be fair.
- Herbrand interpretation: The domain is the set of constants. We invent a constant if the KB or query doesn't contain one. Each constant denotes itself.
- Let I be the Herbrand interpretation in which every ground instance of every element of the fixed point is true and every other atom is false.
- I is a model of KB.
  Proof: suppose $h \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB is false in I. Then h is false and each b_i is true in I. Thus h can be added to C. Contradiction to C being the fixed point.
- I is called a Minimal Model.

Completeness for Datalog

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.
- Thus g is in the fixed point.
- Thus g is generated by the bottom up algorithm.
- Thus $KB \vdash g$.

— In Datalog, bottom-up procedure always halts.
— With function symbols, it may go on indefitely.
Gödel's theorem implies it can't be both sound and complete.
Consider "this statement cannot be proved".
Prolog can represent this, and so cannot be both sound and complete.

Top-down Propositional Proof Procedure (recall)

- 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_1 with the clause in the knowledge base:

$$a_1 \leftarrow b_1 \wedge \ldots \wedge b_p$$

  is the answer clause

$$yes \leftarrow b_1 \wedge \cdots \wedge b_p \wedge a_2 \wedge \cdots \wedge a_m.$$

  A fact in the knowledge base is considered as a clause where $p = 0$.

Top-down Proof procedure

- 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.
- Select atom in body to resolve against, say a_1.
- The SLD resolution of this generalized answer clause on a_1 with the clause

$$a \leftarrow b_1 \wedge \ldots \wedge b_p$$

  where a_1 and a have most general unifier θ, is

$$(yes(t_1, \ldots, t_k) \leftarrow b_1 \wedge \ldots \wedge b_p \wedge a_2 \wedge \ldots \wedge a_m)\theta$$

Top-down propositional definite clause interpreter (review)

To solve the query $?q_1 \wedge \ldots \wedge q_k$:

ac := "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$"
repeat
  select atom a_1 from the body of ac
  choose clause C from KB with a_1 as head
  replace a_1 in the body of ac by the body of C
until ac is an answer.

Top-down Proof Procedure

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

Set ac to generalized answer clause $yes(V_1, \ldots, V_k) \leftarrow B$
while ac is not an answer **do**
  Suppose ac is $yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m$
  select atom a_1 in the body of ac
  choose clause $a \leftarrow b_1 \wedge \ldots \wedge b_p$ in KB
  Rename all variables in $a \leftarrow b_1 \wedge \ldots \wedge b_p$
  Let θ be the most general unifier of a_1 and a.
    Fail if they don't unify
  Set ac to $(yes(t_1, \ldots, t_k) \leftarrow b_1 \wedge \ldots \wedge b_p \wedge a_2 \wedge \ldots \wedge a_m)\theta$
end while.

Suppose ac is generalized answer clause $yes(t_1, \ldots, t_k) \leftarrow$
Answer is $V_1 = t_1, \ldots, V_k = t_k$

Example

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

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

$?live(A).$

$$yes(A) \leftarrow live(A).$$
$$yes(A) \leftarrow connected\_to(A, Z_1) \wedge live(Z_1).$$
$$yes(w_6) \leftarrow live(w_5).$$
$$yes(w_6) \leftarrow connected\_to(w_5, Z_2) \wedge live(Z_2).$$
$$yes(w_6) \leftarrow live(outside).$$
$$yes(w_6) \leftarrow .$$

Example

```
elem(E, set(E,_,_)).
elem(V, set(E,LT,_)) :-
    V #< E,
    elem(V,LT).
elem(V, set(E,_,RT)) :-
    E #< V,
    elem(V,RT).
?- elem(3,S),elem(8,S).
```

```
yes(S) :- elem(3,S),elem(8,S)
yes(set(3,S1,S2)) :- elem(8, set(3,S1,S2))
yes(set(3,S1,S2)) :- 3 #< 8, elem(8,S2)
yes(set(3,S1,S2)) :- elem(8,S2)
yes(set(3,S1,set(8,S3,S4))) :-
```

Answer is $S = set(3, S1, set(8, S3, S4))$

Clicker Question

What is the resolution of the generalized answer clause:

$$yes(B, N) \leftarrow append(B, [a, N|R], [b, a, c, d]).$$

with the clause

$$append([], L, L).$$

A $yes([], c) \leftarrow append(B, R, [d])$

B $yes([b], c) \leftarrow$

C $yes([b|T1], N) \leftarrow append(T1, [a, N|R], [a, c, d]).$

D $yes([b], N) \leftarrow append([], [a, N|R], [a, c, d]).$

E the resolution fails (they do not resolve)

Clicker Question

What is the resolution of the generalized answer clause:

$$yes(B, N) \leftarrow append(B, [a, N|R], [b, a, c, d]).$$

with the clause

$$append([H1 \mid T1], A1, [H1 \mid R1]) \leftarrow$$
$$append(T1, A1, R1).$$

A $yes([], c) \leftarrow append(B, R, [d])$

B $yes([b], c) \leftarrow$

C $yes([b|T1], N) \leftarrow append(T1, [a, N|R], [a, c, d]).$

D $yes([b], N) \leftarrow append([], [a, N|R], [a, c, d]).$

E the resolution fails (they do not resolve)

Clicker Question

What is the resolution of the generalized answer clause:

$$yes([b|T1], N) \leftarrow append(T1, [a, N|R], [a, c, d]).$$

with the clause

$$append([], L, L).$$

A $yes([], c) \leftarrow append(B, R, [d])$

B $yes([b], c) \leftarrow$

C $yes([b|T1], N) \leftarrow append([], [a, c, d], [a, c, d]).$

D $yes([b], N) \leftarrow append([], [a, N|R], [a, c, d]).$

E the resolution fails (they do not resolve)

Function Symbols

- Often we want to refer to individuals in terms of components.
- Examples: 4:55 p.m. English sentences. A classlist.
- 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.
- One function symbol and one constant can refer to infinitely many individuals.

Lists

- 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 *sue*, *kim* and *randy* is

$$cons(sue, cons(kim, cons(randy, nil)))$$

- $append(X, Y, Z)$ is true if list Z contains the elements of X followed by the elements of Y

$$append(nil, Z, Z).$$

$$append(cons(A, X), Y, cons(A, Z)) \leftarrow append(X, Y, Z).$$

Unification with function symbols

- Consider a knowledge base consisting of one fact:

  $lt(X, s(X)).$

- Should the following query succeed?

  ask $lt(Y, Y)$.

- What does the top-down proof procedure give?
- Solution: variable X should not unify with a term that contains X inside. "Occurs check"
  E.g., X should not unify with $s(X)$.
  Simple modification of the unification algorithm, which Prolog does not do!