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 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 b_i has been derived, then h can be derived.

You are forward chaining on this clause.

(This rule also covers the case when $m = 0$.)

Bottom-up proof procedure

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

$C := \{\}$;
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.

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

Example

$a \leftarrow b \wedge c.$

$a \leftarrow e \wedge f.$

$b \leftarrow f \wedge k.$

$c \leftarrow e.$

$d \leftarrow k.$

$e.$

$f \leftarrow j \wedge e.$

$f \leftarrow c.$

$j \leftarrow c.$

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 $KB \not\models g$.

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

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

Each b_i is true in I. 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.

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.

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

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 generated by the bottom up algorithm.

Thus $KB \vdash g$.

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:

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

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

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

Derivations

An answer is an answer clause with $m = 0$. That is, it is the answer clause $yes \leftarrow$.

A derivation of query "?$q_1 \land \ldots \land q_k$" from *KB* is a sequence of answer clauses $\gamma_0, \gamma_1, \ldots, \gamma_n$ such that

- γ_0 is the answer clause $yes \leftarrow q_1 \land \ldots \land q_k$,
- γ_i is obtained by resolving γ_{i-1} with a clause in *KB*, and
- γ_n is an answer.

Top-down definite clause interpreter

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

ac := "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$"
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.

Example: successful derivation

$a \leftarrow b \wedge c.$	$a \leftarrow e \wedge f.$	$b \leftarrow f \wedge k.$
$c \leftarrow e.$	$d \leftarrow k.$	$e.$
$f \leftarrow j \wedge e.$	$f \leftarrow c.$	$j \leftarrow c.$

Query: $?a$

$\gamma_0 : \quad yes \leftarrow a$
$\gamma_1 : \quad yes \leftarrow e \wedge f$
$\gamma_2 : \quad yes \leftarrow f$
$\gamma_3 : \quad yes \leftarrow c$
$\gamma_4 : \quad yes \leftarrow e$
$\gamma_5 : \quad yes \leftarrow$

Example: failing derivation

$a \leftarrow b \wedge c.$	$a \leftarrow e \wedge f.$	$b \leftarrow f \wedge k.$
$c \leftarrow e.$	$d \leftarrow k.$	$e.$
$f \leftarrow j \wedge e.$	$f \leftarrow c.$	$j \leftarrow c.$

Query: $?a$

$\gamma_0 : \quad yes \leftarrow a$

$\gamma_1 : \quad yes \leftarrow b \wedge c$

$\gamma_2 : \quad yes \leftarrow f \wedge k \wedge c$

$\gamma_3 : \quad yes \leftarrow c \wedge k \wedge c$

$\gamma_4 : \quad yes \leftarrow e \wedge k \wedge c$

$\gamma_5 : \quad yes \leftarrow k \wedge c$

Reasoning with Variables

- An $\boxed{\text{instance}}$ of an atom or a clause is obtained by uniformly substituting terms for variables.
- A $\boxed{\text{substitution}}$ 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 $\boxed{\text{application}}$ of a substitution $\sigma = \{V_1/t_1, \ldots, V_n/t_n\}$ 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:

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

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.

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:

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

Bottom-up procedure

- You can carry out the bottom-up procedure on the ground instances of the clauses.
- Soundness is a direct corollary of the ground soundness.
- For completeness, we build a canonical minimal model. We need a denotation for constants:

  Herbrand interpretation: The domain is the set of constants (we invent one if the KB or query doesn't contain one). Each constant denotes itself.

Definite Resolution with Variables

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

$$(yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge \ldots \wedge a_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge a_{i+1} \wedge \ldots \wedge a_m)\theta.$$

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_i 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_i and a.
  Fail if they don't unify;
 Set ac to $(yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge \ldots \wedge a_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge a_{i+1} \wedge \ldots \wedge a_m)\theta$
end while.

Example

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

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

$?live(A).$

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

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

$yes(w_6) \leftarrow live(w_5).$

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

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

$yes(w_6) \leftarrow .$

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.

With one function symbol and one constant we can refer to infinitely many individuals.

Lists

A list is an ordered sequence of elements.

Let's use the constant $\boxed{nil}$ to denote the empty list, and the function $\boxed{cons(H, T)}$ 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

$$cons(david, cons(alan, cons(randy, nil)))$$

$\boxed{append(X, Y, Z)}$ is true if list Z contains the elements of X followed by the elements of Y

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

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