Lecture 2

CS 475

January 31, 2003

1 Models and Logical Consequence

Given a KB and an atom a. a is a logical consequence of KB (or KB entails a), denoted by $KB \models a$, if a is true in every model of KB.

Example 1.1 *For KB consists of*

$$p \leftarrow q.$$
$$q.$$
$$r \leftarrow s.$$

Some interpretations, models of the KB:

	$\pi(p)$	$\pi(q)$	$\pi(r)$	$\pi(s)$	
$I1$	$TRUE$	$TRUE$	$TRUE$	$TRUE$	is a model of KB
$I2$	$FALSE$	$FALSE$	$FALSE$	$FALSE$	not a model of KB
$I3$	$TRUE$	$TRUE$	$FALSE$	$FALSE$	is a model of KB
$I4$	$TRUE$	$TRUE$	$TRUE$	$FALSE$	is a model of KB
$I5$	$TRUE$	$TRUE$	$FALSE$	$TRUE$	not a model of KB

There are 16 possible interpretations for the KB. Without listing all of them, we can conclude (why?):

$KB \models p, KB \models q, KB \not\models r, KB \not\models s$

2 Questions and Answers

A *query* is of the form

$$?b_1 \wedge \ldots \wedge b_m.$$

A query can contain variables.

A *ground instance* of an atom $p(t_1, \ldots, t_m)$ is a ground atom $p(v_1, \ldots, v_m)$ where $v_i = t_i$ if t_i is a constant.

An *answer* is either

- a *ground instance* of the query that is a logically consequence of the KB; or
- *no* if no instance of the query is a logically consequence of the KB.

Example:

$$in(alan, r1).$$
$$part\_of(r1, csb).$$
$$in(X, Y) \leftarrow in(X, Z) \wedge part\_of(Z, Y).$$

Query	*Answer*
$?part\_of(r1, B)$	$part\_of(r1, csb).$
$?part\_of(r2, csb)$	no
$?in(alan, r2)$	no
$?in(alan, X)$	$in(alan, r1)$
	$in(alan, csb)$

Note on Logical consequence

How can we realize that g is a logical consequence of KB? Atom g is a logical consequence of KB if and only if:

- g is a fact in KB, or
- there is a rule
$$g \leftarrow b_1 \wedge \ldots \wedge b_k$$
in KB such that each b_i is a logical consequence of KB.

What if we get a wrong conclusion? The intended model does entail some unintended conclusion; or does not entail some intended conclusions. Two cases:

- g is a fact in KB: this means that the fact is wrong.
- there is a rule
$$g \leftarrow b_1 \wedge \ldots \wedge b_k$$
in KB such that each b_i is true in the intended model, which means that the rule is wrong; otherwise, if some of the b_i is false in the intended model, the error is in b_i.

3 Computing the consequences: Proof Procedure

Answer the question: How to show that a KB entails a conclusion q?

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

A proof procedure allows us to prove things using computers (we can *implement* it)!

Given a proof procedure, $KB \vdash g$ means g can be derived from knowledge base KB. That is, there is a proof for g (according to the proof procedure).

(Recall $KB \models g$ means g is true in all models of KB.)

Important properties of a proof procedure:

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

3.1 Bottom-Up ground proof procedure

Use 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.

The technique is called *forward chaining.* When $m = 0$, we conclude h.

Bottom-up proof procedure: $KB \vdash g$ if $g \in C$ at the end of this procedure:

$C := \emptyset$
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.

Example: KB consists of the following rule

$$\begin{array}{l} a \leftarrow b \wedge c. \\ a \leftarrow e \wedge f. \\ b \leftarrow f \wedge k. \\ c \leftarrow e. \\ d \leftarrow k. \\ e. \\ f \leftarrow j \wedge e. \\ f \leftarrow c. \\ j \leftarrow c. \end{array}$$

The procedure goes through (one of the possible) steps:

$C = \emptyset$
$C = \{e\}$
$C = \{e, c\}$
$C = \{e, c, f, j\}$
$C = \{e, c, f, j, a\}$

Soundness of the bottom-up proof procedure: By contradiction. Suppose there is a g such that $KB \vdash g$ implies $KB \models g$. Let h be the first atom added to C that is 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 $h \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB such that b_i is true in I for all i. This means that this clause is false in I. Therefore I is not a model of KB. This contradicts what we assumed, thus no such g exists.

Before discussing the *completeness* of the bottom-up proof procedure, we define the notion of a *fix 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. We can show that I is a model of KB. Suppose that $h \leftarrow b_1 \wedge \ldots \wedge b_m$ 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. This contradicts the fact that C is the fixed point.

I is called a *minimal model.*

Soundness of the bottom-up proof procedure: Suppose that $KB \models g$. This means that g is true in the minimal model. This means that g is generated by the algorithm which implies $KB \vdash g$.

Complexity: linear in the size of the KB.

3.2 Top-down Ground Proof Procedure

Idea: Bottom-up proof procedure proceeds from the empty set, accumulates the consequences. Top-down proof procedure searchs backward from a query to determine if it is a logical consequence of KB.

An *answer clause* is of the form:

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

The rule used in derivation is called *SLD Resolution.* (SLD: linear resolution with a selection function for definite sentence)

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

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

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

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

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

A top-down definite clause interpreter: To solve the query $?q_1 \wedge \ldots \wedge q_k$ let

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

How to choose? non-deterministically selection in **choose clause**.

The choice that needs to be make in the above algorithms is nondeterministic (it is possible that many clauses have a_i as the head).

Two possible treatment:

- *Don't-care nondeterminism*: If one selection doesn't lead to a solution, there is no point trying other alternatives. (in bottom-up procedure)

- *Don't-know nondeterminism*: If one choice doesn't lead to a solution, other choices may. (in top-down procedure)

Example: KB consists of the following rule

$$
\begin{aligned}
&a \leftarrow b \wedge c.\\
&a \leftarrow e \wedge f.\\
&b \leftarrow f \wedge k.\\
&c \leftarrow e.\\
&d \leftarrow k.\\
&e.\\
&f \leftarrow j \wedge e.\\
&f \leftarrow c.\\
&j \leftarrow c.
\end{aligned}
$$

Query: $?a$

Successful derivation: $a \Rightarrow e \wedge f \Rightarrow f \Rightarrow c \Rightarrow e$ ($x \Rightarrow y$ means that we use SLD resolution to reduce from x to y)

Failing derivation: $a \Rightarrow b \wedge c \Rightarrow f \wedge k \wedge c \Rightarrow c \wedge k \wedge c \Rightarrow e \wedge k \wedge c \Rightarrow k \wedge c$

4 Reasoning with variables

An *instance* of an atom or a clause is obtained by uniformly (or simultaneously) 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. V_i/t_i is called a *binding* for the variable V_i. A substitution is in *normal form* if it V_i does not appear in t_j for every pair of i and j.

We only work with normal form substitutions.

The 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. If $e\sigma$ is ground we say that it is a ground instance of e.

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

Substitutions can apply to clauses, terms, and atoms. For example, the application of the substitution $\{X/Y, Z/a\}$ to the clause

$$p(X, Y) \leftarrow q(a, Z, X, Y, Z)$$

is the clause

$$p(Y, Y) \leftarrow q(a, a, Y, Y, a).$$

Unifiers Substitution σ is a unifier of $e1$ and $e2$ if $e1\sigma = e2\sigma$.

Substitution σ is a most general unifier (mgu) of $e1$ and $e2$ if

- σ is a unifier of $e1$ and $e2$; and
- if substitution σ_0 also unifies $e1$ and $e2$, then σ_0 is an instance of $e\sigma$ for all atoms e.

If two atoms have a unifier, they have a most general unifier.

Unification Example

$\sigma_1 = \{X/A, Y/b, Z/C, D/e\}$
$\sigma_2 = \{A/X, Y/b, C/Z, D/e\}$
$\sigma_3 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\}$
$\sigma_4 = \{A/a, X/a, Y/b, C/c, Z/c, D/e\}$
$\sigma_5 = \{X/A, Y/b, Z/A, C/A, D/e\}$
$\sigma_6 = \{X/A, Y/b, Z/C, D/e, W/a\}$
The first three are most general unifiers. The following substitutions are not unifiers: $\sigma_7 = \{Y/b, D/e\}$
$\sigma_8 = \{X/a, Y/b, Z/c, D/e\}$

4.1 Bottom-up procedure (for queries with variables)

You can carry out the bottom-up procedure on the ground instances of the clauses.

Example: For KB consisting of

$$\begin{array}{l} q(a). \\ q(b). \\ r(a). \\ s(W) \leftarrow r(W). \\ p(X, Y) \leftarrow q(X) \wedge s(Y). \end{array}$$

the set of all ground instances is

$$\begin{array}{l} q(a). \\ q(b). \\ r(a). \\ s(a) \leftarrow r(a). \\ s(b) \leftarrow r(b). \\ p(a, a) \leftarrow q(a) \wedge s(a). \\ p(a, b) \leftarrow q(a) \wedge s(b). \\ p(b, a) \leftarrow q(b) \wedge s(a). \\ p(b, b) \leftarrow q(b) \wedge s(b). \end{array}$$

Using the bottom-up proof procedure, we can derive: $q(a), q(b), r(a), s(a), p(a, a), p(a, b)$.

What happens if there is no constants? We introduce one, say a, and proceed as above.

Example: For KB consisting of

$$\begin{array}{l} p(X, Y). \\ q \leftarrow p(W, W). \end{array}$$

We introduce a new constant a and get the set of all ground instances is

$$p(a, a).$$
$$q \leftarrow p(a, a).$$

that allows us to conclude that q is entailed by the KB.

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.

4.2 Top-Down Procedure with Variables

Definite Resolution with Variables

A *generalized answer clause* is of the form

$$yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge \ldots \wedge a_m.$$

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 generalized answer clause

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

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

- γ_0 is the answer clause $yes(V_1, \ldots, V_k) \leftarrow q_1 \wedge \ldots \wedge q_k$ where V_i are the variables occurring in the query,
- γ_i is obtained by resolving γ_{i-1} with a copy of a clause in KB, and
- γ_n is an answer, i.e., it is of the form $yes(t_1, \ldots, t_k) \leftarrow$. This gives us the answer $V_i = t_i$.

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

Set $ac := yes(V_1, \ldots, V_k) \leftarrow B$
while ac is not an answer do
    Suppose ac is $yes(t_1, \ldots, r_k) \leftarrow a_1 \wedge \ldots \wedge a_m$
    select a conjunct a_i from 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 $(yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge \ldots a_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge a_{i+1} \ldots \wedge a_m)\theta$.
end while

Example:

$$live(Y) \leftarrow connected_t o(Y, Z) \land live(Z).$$
$$live(outside).$$
$$connected_t o(w6, w5).$$
$$connected_t o(w5, outside).$$

$?live(A).$
$yes(A) \leftarrow live(A).$
$yes(A) \leftarrow connected_t o(A, Z1) \land live(Z1).$
$yes(w6) \leftarrow live(w5).$
$yes(w6) \leftarrow connected_t o(w5, Z2) \land live(Z2).$
$yes(w6) \leftarrow live(outside).$
$yes(w6) \leftarrow .$