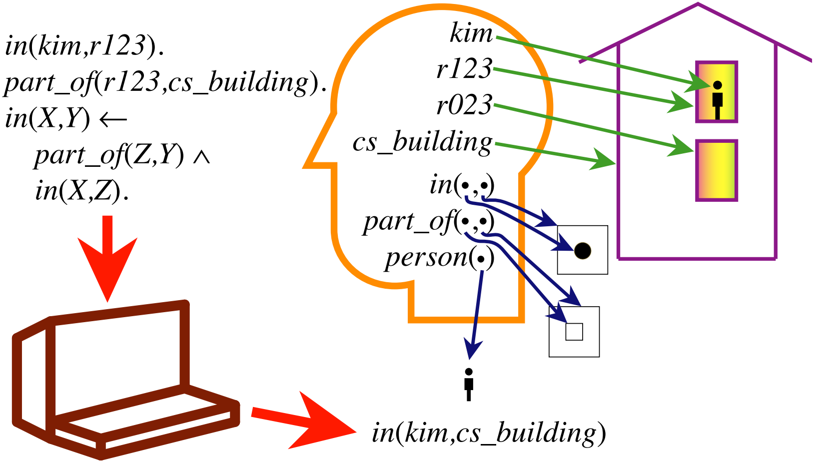
Chapter 12:

Reasoning about Individuals and Relations

Individuals and Relations

- It is useful to view the world as consisting of individuals (objects, things) and relations among individuals.
- Often features are made from relations among individuals and functions of individuals.
- Reasoning in terms of individuals and relationships can be simpler than reasoning in terms of features, if we can express general knowledge that covers all individuals.
- Sometimes we may know some individual exists, but not which one.
- Sometimes there are infinitely many individuals we want to refer to (e.g., set of all integers, or the set of all stacks of blocks).

Role of Semantics in Automated Reasoning



Features of Automated Reasoning

- Users can have meanings for symbols in their head.
- The computer doesn't need to know these meanings to derive logical consequence.
- Users can interpret any answers according to their meaning.

Automated Reasoning

- flat or modular or hierarchical
- explicit states or features or individuals and relations
- static or finite stage or indefinite stage or infinite stage
- fully observable or partially observable
- deterministic or stochastic dynamics
- goals or complex preferences
- single agent or multiple agents
- knowledge is given or knowledge is learned
- perfect rationality or bounded rationality

Representational Assumptions of Datalog

- An agent's knowledge can be usefully described in terms of *individuals* and *relations* among individuals.
- An agent's knowledge base consists of *definite* and *positive* statements.
- The environment is *static*.
- There are only a finite number of individuals of interest in the domain. Each individual can be given a unique name.

$\Longrightarrow$ Datalog

Syntax of Datalog

- A variable starts with upper-case letter.
- A constant starts with lower-case letter or is a sequence of digits (numeral).
- A predicate symbol starts with lower-case letter.
- A term is either a variable or a constant.
- An atomic symbol (atom) is of the form p or $p(t_1, \ldots, t_n)$ where p is a predicate symbol and t_i are terms.

Syntax of Datalog (cont)

- A definite clause is either an atomic symbol (a fact) or of the form:

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1 \wedge \cdots \wedge b_m}_{\text{body}}$$

  where a and b_i are atomic symbols.
- query is of the form $?b_1 \wedge \cdots \wedge b_m$.
- knowledge base is a set of definite clauses.

Example Knowledge Base

$$
\begin{aligned}
& in(kim, R) \leftarrow \\
& \quad teaches(kim, cs322) \wedge \\
& \quad in(cs322, R). \\
& grandfather(william, X) \leftarrow \\
& \quad father(william, Y) \wedge \\
& \quad parent(Y, X). \\
& slithy(toves) \leftarrow \\
& \quad mimsy \wedge borogroves \wedge \\
& \quad outgrabe(mome, Raths).
\end{aligned}
$$

Semantics: General Idea

A semantics specifies the meaning of sentences in the language.
An interpretation specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects & relations in world
  - constants denote individuals
  - predicate symbols denote relations

Formal Semantics

An interpretation is a triple $I = \langle D, \phi, \pi \rangle$, where

- D, the domain, is a nonempty set. Elements of D are individuals.
- ϕ is a mapping that assigns to each constant an element of D. Constant c denotes individual $\phi(c)$.
- π is a mapping that assigns to each n-ary predicate symbol a relation: a function from D^n into $\{TRUE, FALSE\}$.

Example Interpretation

Constants: *phone*, *pencil*, *telephone*.

Predicate Symbol: *noisy* (unary), *left_of* (binary).

- $D = \{$✂, ☎, ✎$\}$.
- $\phi(phone) =$ ☎, $\phi(pencil) =$ ✎, $\phi(telephone) =$ ☎.
- $\pi(noisy)$:

⟨✂⟩	*FALSE*	⟨☎⟩	*TRUE*	⟨✎⟩	*FALSE*

$\pi(left\_of)$:

⟨✂, ✂⟩	*FALSE*	⟨✂, ☎⟩	*TRUE*	⟨✂, ✎⟩	*TRUE*
⟨☎, ✂⟩	*FALSE*	⟨☎, ☎⟩	*FALSE*	⟨☎, ✎⟩	*TRUE*
⟨✎, ✂⟩	*FALSE*	⟨✎, ☎⟩	*FALSE*	⟨✎, ✎⟩	*FALSE*

Important points to note

- The domain D can contain real objects. (e.g., a person, a room, a course). D can't necessarily be stored in a computer.
- $\pi(p)$ specifies whether the relation denoted by the n-ary predicate symbol p is true or false for each n-tuple of individuals.
- If predicate symbol p has no arguments, then $\pi(p)$ is either *TRUE* or *FALSE*.

Truth in an interpretation

A constant c **denotes in I** the individual $\phi(c)$.
Ground (variable-free) atom $p(t_1, \ldots, t_n)$ is

- **true in interpretation I** if $\pi(p)(\langle\phi(t_1), \ldots, \phi(t_n)\rangle) = \textit{TRUE}$ in interpretation I and
- **false** otherwise.

Ground clause $h \leftarrow b_1 \wedge \ldots \wedge b_m$ is **false in interpretation I** if h is false in I and each b_i is true in I, and is **true in interpretation I** otherwise.

Example Truths

In the interpretation given before, which of following are true?

$$noisy(phone)$$
$$noisy(telephone)$$
$$noisy(pencil)$$
$$left\_of(phone, pencil)$$
$$left\_of(phone, telephone)$$
$$noisy(phone) \leftarrow left\_of(phone, telephone)$$
$$noisy(pencil) \leftarrow left\_of(phone, telephone)$$
$$noisy(pencil) \leftarrow left\_of(phone, pencil)$$
$$noisy(phone) \leftarrow noisy(telephone) \land noisy(pencil)$$

Example Truths

In the interpretation given before, which of following are true?

$noisy(phone)$	true
$noisy(telephone)$	true
$noisy(pencil)$	false
$left\_of(phone, pencil)$	true
$left\_of(phone, telephone)$	false
$noisy(phone) \leftarrow left\_of(phone, telephone)$	true
$noisy(pencil) \leftarrow left\_of(phone, telephone)$	true
$noisy(pencil) \leftarrow left\_of(phone, pencil)$	false
$noisy(phone) \leftarrow noisy(telephone) \wedge noisy(pencil)$	true

Models and logical consequences (recall)

- A knowledge base, KB, is true in interpretation I if and only if every clause in KB is true in I.
- A model of a set of clauses is an interpretation in which all the clauses are true.
- If KB is a set of clauses and g is a conjunction of atoms, g is a logical consequence of KB, written $KB \models g$, if g is true in every model of KB.
- That is, $KB \models g$ if there is no interpretation in which KB is true and g is false.

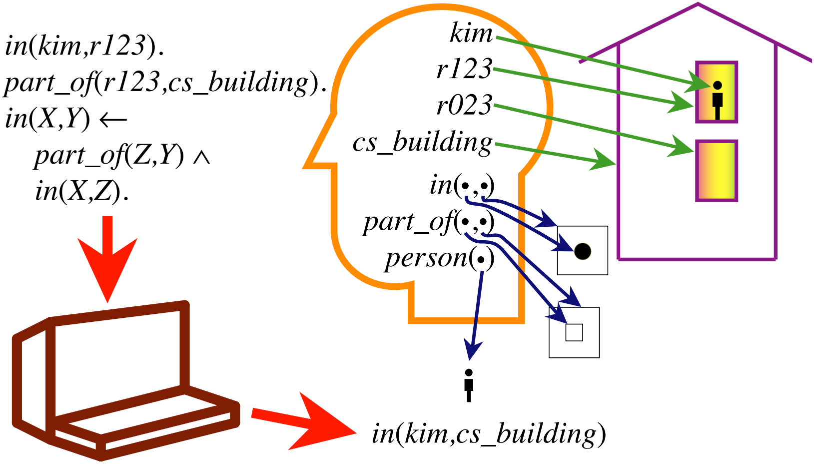
User's view of Semantics

1. Choose a task domain: intended interpretation.
2. Associate constants with individuals you want to name.
3. For each relation you want to represent, associate a predicate symbol in the language.
4. Tell the system clauses that are true in the intended interpretation: axiomatizing the domain.
5. Ask questions about the intended interpretation.
6. If $KB \models g$, then g must be true in the intended interpretation.

Computer's view of semantics

- The computer doesn't have access to the intended interpretation.
- All it knows is the knowledge base.
- The computer can determine if a formula is a logical consequence of KB.
- If $KB \models g$ then g must be true in the intended interpretation.
- If $KB \not\models g$ then there is a model of KB in which g is false. This could be the intended interpretation.

Role of Semantics in an RRS



Variables

- Variables are universally quantified in the scope of a clause.
- A variable assignment is a function from variables into the domain.
- Given an interpretation and a variable assignment,
  each term denotes an individual and
  each clause is either true or false.
- A clause containing variables is true in an interpretation if it is true for all variable assignments.

Queries and Answers

A query is a way to ask if a body is a logical consequence of the knowledge base:

$$?b_1 \wedge \cdots \wedge b_m.$$

An answer is either

- an instance of the query that is a logical consequence of the knowledge base KB, or
- no if no instance is a logical consequence of KB.

Example Queries

$$KB = \begin{cases} in(kim, r123). \\ part\_of(r123, cs\_building). \\ in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z). \end{cases}$$

Query	Answer
$?part\_of(r123, B).$	

Example Queries

$$KB = \begin{cases} in(kim, r123). \\ part\_of(r123, cs\_building). \\ in(X, Y) \leftarrow part\_of(Z, Y) \land in(X, Z). \end{cases}$$

Query	Answer
$?part\_of(r123, B).$	$part\_of(r123, cs\_building)$
$?part\_of(r023, cs\_building).$	

Example Queries

$$KB = \begin{cases} in(kim, r123). \\ part\_of(r123, cs\_building). \\ in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z). \end{cases}$$

Query	Answer
$?part\_of(r123, B).$	$part\_of(r123, cs\_building)$
$?part\_of(r023, cs\_building).$	*no*
$?in(kim, r023).$	

Example Queries

$$KB = \left\{ \begin{array}{l} in(kim, r123). \\ part\_of(r123, cs\_building). \\ in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z). \end{array} \right.$$

Query	Answer
$?part\_of(r123, B).$	$part\_of(r123, cs\_building)$
$?part\_of(r023, cs\_building).$	no
$?in(kim, r023).$	no
$?in(kim, B).$	

Example Queries

$$KB = \begin{cases} in(kim, r123). \\ part\_of(r123, cs\_building). \\ in(X, Y) \leftarrow part\_of(Z, Y) \land in(X, Z). \end{cases}$$

Query	Answer
$?part\_of(r123, B).$	$part\_of(r123, cs\_building)$
$?part\_of(r023, cs\_building).$	no
$?in(kim, r023).$	no
$?in(kim, B).$	$in(kim, r123)$
	$in(kim, cs\_building)$

Logical Consequence

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.

Debugging false conclusions

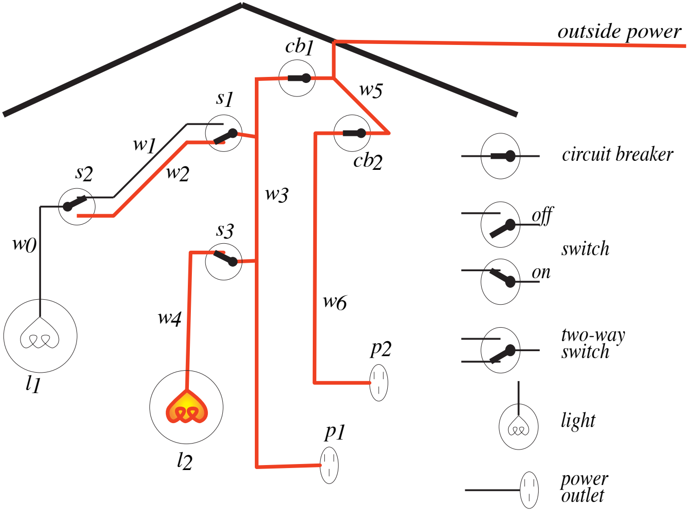
To debug answer g that is false in the intended interpretation:

- 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.

Electrical Environment



Axiomatizing the Electrical Environment

% $light(L)$ is true if L is a light
$light(l_1)$. $light(l_2)$.
% $down(S)$ is true if switch S is down
$down(s_1)$. $up(s_2)$. $up(s_3)$.
% $ok(D)$ is true if D is not broken
$ok(l_1)$. $ok(l_2)$. $ok(cb_1)$. $ok(cb_2)$.

$?light(l_1)$. $\Longrightarrow$

Axiomatizing the Electrical Environment

% $light(L)$ is true if L is a light
$light(l_1)$. $light(l_2)$.
% $down(S)$ is true if switch S is down
$down(s_1)$. $up(s_2)$. $up(s_3)$.
% $ok(D)$ is true if D is not broken
$ok(l_1)$. $ok(l_2)$. $ok(cb_1)$. $ok(cb_2)$.

$?light(l_1)$. $\Longrightarrow$ *yes*
$?light(l_6)$. $\Longrightarrow$

Axiomatizing the Electrical Environment

% $light(L)$ is true if L is a light
$light(l_1)$. $light(l_2)$.
% $down(S)$ is true if switch S is down
$down(s_1)$. $up(s_2)$. $up(s_3)$.
% $ok(D)$ is true if D is not broken
$ok(l_1)$. $ok(l_2)$. $ok(cb_1)$. $ok(cb_2)$.

$?light(l_1)$. $\Longrightarrow$ *yes*
$?light(l_6)$. $\Longrightarrow$ *no*
$?up(X)$. $\Longrightarrow$

Axiomatizing the Electrical Environment

% $light(L)$ is true if L is a light
$light(l_1)$. $light(l_2)$.
% $down(S)$ is true if switch S is down
$down(s_1)$. $up(s_2)$. $up(s_3)$.
% $ok(D)$ is true if D is not broken
$ok(l_1)$. $ok(l_2)$. $ok(cb_1)$. $ok(cb_2)$.

$?light(l_1)$. $\Longrightarrow$ *yes*
$?light(l_6)$. $\Longrightarrow$ *no*
$?up(X)$. $\Longrightarrow$ $up(s_2)$, $up(s_3)$

$connected\_to(X, Y)$ is true if component X is connected to Y

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$
$$connected\_to(w_0, w_2) \leftarrow down(s_2).$$
$$connected\_to(w_1, w_3) \leftarrow up(s_1).$$
$$connected\_to(w_2, w_3) \leftarrow down(s_1).$$
$$connected\_to(w_4, w_3) \leftarrow up(s_3).$$
$$connected\_to(p_1, w_3).$$

$?connected\_to(w_0, W).$ $\Longrightarrow$

$connected\_to(X, Y)$ is true if component X is connected to Y

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$
$$connected\_to(w_0, w_2) \leftarrow down(s_2).$$
$$connected\_to(w_1, w_3) \leftarrow up(s_1).$$
$$connected\_to(w_2, w_3) \leftarrow down(s_1).$$
$$connected\_to(w_4, w_3) \leftarrow up(s_3).$$
$$connected\_to(p_1, w_3).$$

$?connected\_to(w_0, W). \implies W = w_1$

$?connected\_to(w_1, W). \implies$

$connected\_to(X, Y)$ is true if component X is connected to Y

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$
$$connected\_to(w_0, w_2) \leftarrow down(s_2).$$
$$connected\_to(w_1, w_3) \leftarrow up(s_1).$$
$$connected\_to(w_2, w_3) \leftarrow down(s_1).$$
$$connected\_to(w_4, w_3) \leftarrow up(s_3).$$
$$connected\_to(p_1, w_3).$$

$?connected\_to(w_0, W).$ $\Longrightarrow$ $W = w_1$

$?connected\_to(w_1, W).$ $\Longrightarrow$ no

$?connected\_to(Y, w_3).$ $\Longrightarrow$

$connected\_to(X, Y)$ is true if component X is connected to Y

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$
$$connected\_to(w_0, w_2) \leftarrow down(s_2).$$
$$connected\_to(w_1, w_3) \leftarrow up(s_1).$$
$$connected\_to(w_2, w_3) \leftarrow down(s_1).$$
$$connected\_to(w_4, w_3) \leftarrow up(s_3).$$
$$connected\_to(p_1, w_3).$$

$?connected\_to(w_0, W).$ $\Longrightarrow$ $W = w_1$

$?connected\_to(w_1, W).$ $\Longrightarrow$ *no*

$?connected\_to(Y, w_3).$ $\Longrightarrow$ $Y = w_2,\ Y = w_4,\ Y = p_1$

$?connected\_to(X, W).$ $\Longrightarrow$

$connected\_to(X, Y)$ is true if component X is connected to Y

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$
$$connected\_to(w_0, w_2) \leftarrow down(s_2).$$
$$connected\_to(w_1, w_3) \leftarrow up(s_1).$$
$$connected\_to(w_2, w_3) \leftarrow down(s_1).$$
$$connected\_to(w_4, w_3) \leftarrow up(s_3).$$
$$connected\_to(p_1, w_3).$$

$?connected\_to(w_0, W).$ $\Longrightarrow$ $W = w_1$

$?connected\_to(w_1, W).$ $\Longrightarrow$ no

$?connected\_to(Y, w_3).$ $\Longrightarrow$ $Y = w_2,\ Y = w_4,\ Y = p_1$

$?connected\_to(X, W).$ $\Longrightarrow$ $X = w_0, W = w_1, \ldots$

% $lit(L)$ is true if the light L is lit

$$lit(L) \leftarrow light(L) \land ok(L) \land live(L).$$

% $live(C)$ is true if there is power coming into C

$$\begin{aligned}
&live(Y) \leftarrow \\
&\qquad connected_to(Y, Z) \land \\
&\qquad live(Z). \\
&live(outside).
\end{aligned}$$

This is a recursive definition of *live*.

Recursion and Mathematical Induction

$$above(X, Y) \leftarrow on(X, Y).$$
$$above(X, Y) \leftarrow on(X, Z) \wedge above(Z, Y).$$

This can be seen as:

- Recursive definition of *above*: prove *above* in terms of a base case (*on*) or a simpler instance of itself; or
- Way to prove *above* by mathematical induction: the base case is when there are no blocks between X and Y, and if you can prove *above* when there are n blocks between them, you can prove it when there are $n + 1$ blocks.

Limitations

Suppose you had a database using the relation:

$$enrolled(S, C)$$

which is true when student S is enrolled in course C.
You can't define the relation:

$$empty\_course(C)$$

which is true when course C has no students enrolled in it.
This is because $empty\_course(C)$ doesn't logically follow from a set of *enrolled* relations. There are always models where someone is enrolled in a course!

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 $e\sigma$, 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:

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

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.

Unification Example

Which of the following are unifiers of $p(A, b, C, D)$ and $p(X, Y, Z, e)$:

$$\sigma_1 = \{X/A, Y/b, Z/C, D/e\}$$
$$\sigma_2 = \{Y/b, D/e\}$$
$$\sigma_3 = \{X/A, Y/b, Z/C, D/e, W/a\}$$
$$\sigma_4 = \{A/X, Y/b, C/Z, D/e\}$$
$$\sigma_5 = \{X/a, Y/b, Z/c, D/e\}$$
$$\sigma_6 = \{A/a, X/a, Y/b, C/c, Z/c, D/e\}$$
$$\sigma_7 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\}$$
$$\sigma_8 = \{X/A, Y/b, Z/A, C/A, D/e\}$$

Which are most general unifiers?

Unification Example

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

$\sigma_1 = \{X/A, Y/b, Z/C, D/e\}$
$\sigma_4 = \{A/X, Y/b, C/Z, D/e\}$
$\sigma_7 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\}$
$\sigma_6 = \{A/a, X/a, Y/b, C/c, Z/c, D/e\}$
$\sigma_8 = \{X/A, Y/b, Z/A, C/A, D/e\}$
$\sigma_3 = \{X/A, Y/b, Z/C, D/e, W/a\}$

The first three are most general unifiers.

The following substitutions are not unifiers:

$\sigma_2 = \{Y/b, D/e\}$
$\sigma_5 = \{X/a, Y/b, Z/c, D/e\}$

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 $b'_i \in C$ where $b_i\theta = b'_i\theta$ and
    there is no $h' \in C$ such that h' is more general than $h\theta$
  $C := C \cup \{h\theta\}$
until no more clauses can be selected.

Example

$$live(Y) \leftarrow connected\_to(Y, Z) \wedge live(Z).$$
$$live(outside).$$
$$connected\_to(w_5, outside),$$
$$connected\_to(w_6, w_5).$$

Example

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

$$live(outside).$$

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

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

$$C = \{live(outside),$$

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

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

$$live(w_5),$$

$$live(w_6)\}$$

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 a clause in KB of form

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

  where $h = h'\theta$. 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.

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 one 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

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 $KB \vdash g$.

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.

- 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

$$live(Y) \leftarrow connected\_to(Y, Z) \land live(Z).$$
$$live(outside).$$
$$connected\_to(w_6, w_5).$$
$$connected\_to(w_5, outside).$$
$$?live(A).$$

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.
- 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

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

Natural Language Understanding

- We want to communicate with computers using natural language (spoken and written).
  - unstructured natural language — allow any statements, but make mistakes or failure.
  - controlled natural language — only allow unambiguous statements that can be interpreted (e.g., in supermarkets or for doctors).
- There is a vast amount of information in natural language.
- Understanding language to extract information or answering questions is more difficult than getting extracting gestalt properties such as topic, or choosing a help page.
- Many of the problems of AI are explicit in natural language understanding. "AI complete".

Syntax, Semantics, Pragmatics

- Syntax describes the form of language (using a grammar).
- Semantics provides the meaning of language.
- Pragmatics explains the purpose or the use of language (how utterances relate to the world).

Examples:

- *This lecture is about natural language.*
- *The green frogs sleep soundly.*
- *Colorless green ideas sleep furiously.*
- *Furiously sleep ideas green colorless.*

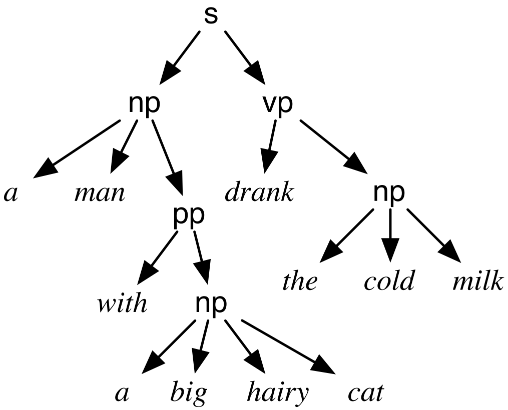
Beyond N-grams

- *A man with a big hairy cat drank the cold milk.*
- Who or what drank the milk?

Beyond N-grams

- *A man with a big hairy cat drank the cold milk.*
- Who or what drank the milk?

Simple syntax diagram:



Context-free grammar

- A terminal symbol is a word (perhaps including punctuation).
- A non-terminal symbol can be rewritten as a sequence of terminal and non-terminal symbols, e.g.,

$$sentence \longmapsto noun\_phrase, verb\_phrase$$

$$verb\_phrase \longmapsto verb, noun\_phrase$$

$$verb \longmapsto [drank]$$

- Can be written as a logic program, where a sentence is a sequence of words:

$$sentence(S) \leftarrow noun\_phrase(N), verb\_phrase(V), append(N, V, S).$$

To say word "drank" is a verb:

$$verb([drank]).$$

Difference Lists

- Non-terminal symbol s becomes a predicate with two arguments, $s(T_1, T_2)$, meaning:
  - T_2 is an ending of the list T_1
  - all of the words in T_1 before T_2 form a sequence of words of the category s.
- Lists T_1 and T_2 together form a difference list.
- "the student" is a noun phrase:

$$noun\_phrase([the, student, passed, the, course], [passed, the, course])$$

Difference Lists

- Non-terminal symbol s becomes a predicate with two arguments, $s(T_1, T_2)$, meaning:
  - T_2 is an ending of the list T_1
  - all of the words in T_1 before T_2 form a sequence of words of the category s.
- Lists T_1 and T_2 together form a difference list.
- "the student" is a noun phrase:

$$noun\_phrase([the, student, passed, the, course], [passed, the, course])$$

- The word "drank" is a verb:

$$verb([drank|W], W).$$

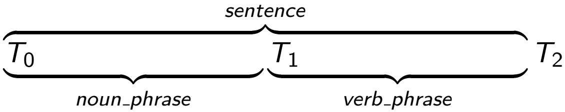
Definite clause grammar

The grammar rule

$$sentence \longmapsto noun\_phrase, verb\_phrase$$

means that there is a sentence between T_0 and T_2 if there is a noun phrase between T_0 and T_1 and a verb phrase between T_1 and T_2:

$$\begin{aligned} sentence(T_0, T_2) \leftarrow \\ noun\_phrase(T_0, T_1) \wedge \\ verb\_phrase(T_1, T_2). \end{aligned}$$



Definite clause grammar rules

The rewriting rule

$$h \longmapsto b_1, b_2, \ldots, b_n$$

says that h is b_1 then b_2, ..., then b_n:

$$\begin{aligned} h(T_0, T_n) \leftarrow & \\ & b_1(T_0, T_1) \wedge \\ & b_2(T_1, T_2) \wedge \\ & \vdots \\ & b_n(T_{n-1}, T_n). \end{aligned}$$

using the interpretation

$$\overbrace{\underbrace{T_0 \qquad\quad}_{b_1} \underbrace{T_1 \qquad\quad}_{b_2} T_2 \cdots \underbrace{T_{n-1} \qquad\quad}_{b_n}}^{h} T_n$$

Terminal Symbols

Non-terminal h gets mapped to the terminal symbols, $t_1, ..., t_n$:

$$h([t_1, \cdots, t_n | T], T)$$

using the interpretation

$$\overbrace{t_1, \cdots, t_n}^{h} T$$

Thus, $h(T_1, T_2)$ is true if $T_1 = [t_1, ..., t_n | T_2]$.

Complete Context Free Grammar Example

see
`http://artint.info/code/Prolog/ch12/cfg_simple.pl`

What will the following query return?

$noun\_phrase([the, student, passed, the, course, with, a, computer], R).$

Complete Context Free Grammar Example

see
`http://artint.info/code/Prolog/ch12/cfg_simple.pl`

What will the following query return?

$noun\_phrase([the, student, passed, the, course, with, a, computer], R).$

How many answers does the following query have?

$sentence([the, student, passed, the, course, with, a, computer], R).$

Augmenting the Grammar

Two mechanisms can make the grammar more expressive:

- extra arguments to the non-terminal symbols
- arbitrary conditions on the rules.

We have a Turing-complete programming language at our disposal!

Building Structures for Non-terminals

Add an extra argument representing a parse tree:

$$\begin{aligned}&sentence(T_0, T_2, s(NP, VP)) \leftarrow \\ &\quad noun\_phrase(T_0, T_1, NP) \land \\ &\quad verb\_phrase(T_1, T_2, VP).\end{aligned}$$

Enforcing Constraints

Add an argument representing the number (singular or plural), as well as the parse tree:

$$sentence(T_0, T_2, Num, s(NP, VP)) \leftarrow$$
$$noun\_phrase(T_0, T_1, Num, NP) \wedge$$
$$verb\_phrase(T_1, T_2, Num, VP).$$

The parse tree can return the determiner (definite or indefinite), number, modifiers (adjectives) and any prepositional phrase:

$$noun\_phrase(T, T, Num, no\_np).$$
$$noun\_phrase(T_0, T_4, Num, np(Det, Num, Mods, Noun, P$$
$$det(T_0, T_1, Num, Det) \wedge$$
$$modifiers(T_1, T_2, Mods) \wedge$$
$$noun(T_2, T_3, Num, Noun) \wedge$$
$$pp(T_3, T_4, PP).$$

Complete Example

see
`http://artint.info/code/Prolog/ch12/nl_numbera.pl`

Question-answering

- How can we get from natural language to a query or to logical statements?
- Goal: map natural language to a query that can be asked of a knowledge base.
- Add arguments representing the individual and the relations about that individual. E.g.,

$$noun\_phrase(T_0, T_1, O, C_0, C_1)$$

  means
  - $T_0 - T_1$ is a difference list forming a noun phrase.
  - The noun phrase refers to the individual O.
  - C_0 is list of previous relations.
  - C_1 is C_0 together with the relations on individual O given by the noun phrase.

Example natural language to query

see
`http://artint.info/code/Prolog/ch12/nl_interface.pl`

Context and world knowledge

The student took many courses. Two computer science courses and one mathematics course were particularly difficult. The mathematics course...

Context and world knowledge

The student took many courses. Two computer science courses and one mathematics course were particularly difficult. The mathematics course. . .

Who was the captain of the Titanic?
Was she tall?