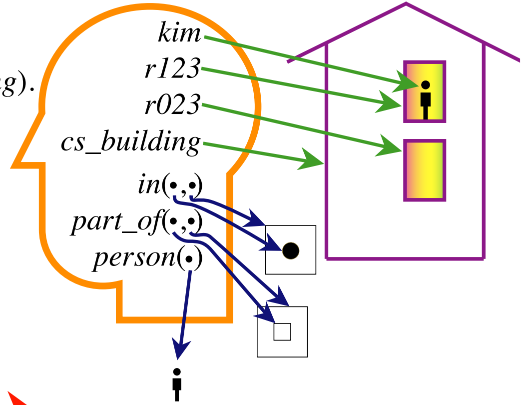
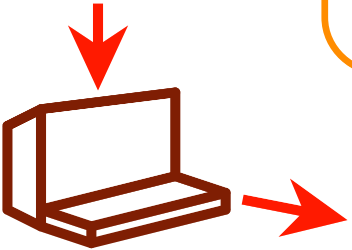
Objects and Relations

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

Role of Semantics in Automated Reasoning

in(*kim*,*r123*).
part_of(*r123*,*cs_building*).
in(*X*,*Y*) ←
  part_of(*Z*,*Y*) ∧
  in(*X*,*Z*).





in(*kim*,*cs_building*)

Features of Automated Reasoning

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

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

- variable starts with upper-case letter.
- constant starts with lower-case letter or is a sequence of digits (numeral).
- predicate symbol starts with lower-case letter.
- term is either a variable or a constant.
- 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)

- 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 $\{\mathit{TRUE}, \mathit{FALSE}\}$.

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 $\pi(p)(t'_1, \ldots, t'_n) = \mathit{TRUE}$, where t_i denotes t'_i in interpretation I and
- false in interpretation I if $\pi(p)(t'_1, \ldots, t'_n) = \mathit{FALSE}$.

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:

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

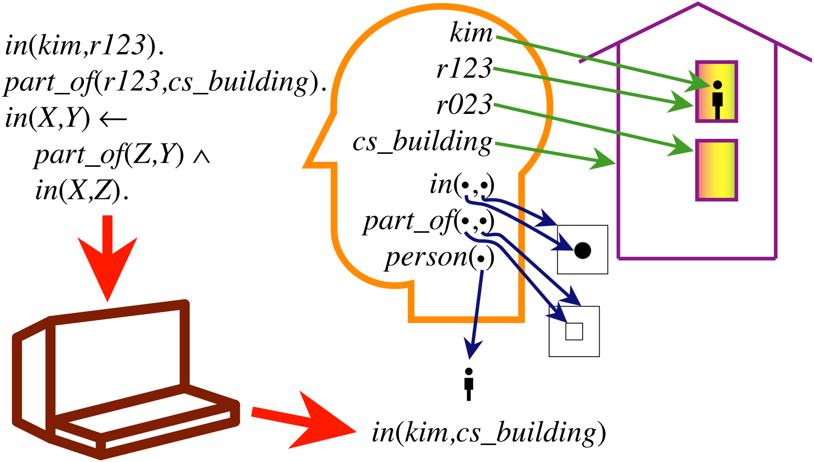
User's view of Semantics

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

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

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

Example Queries

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

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

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

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

Example Queries

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

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

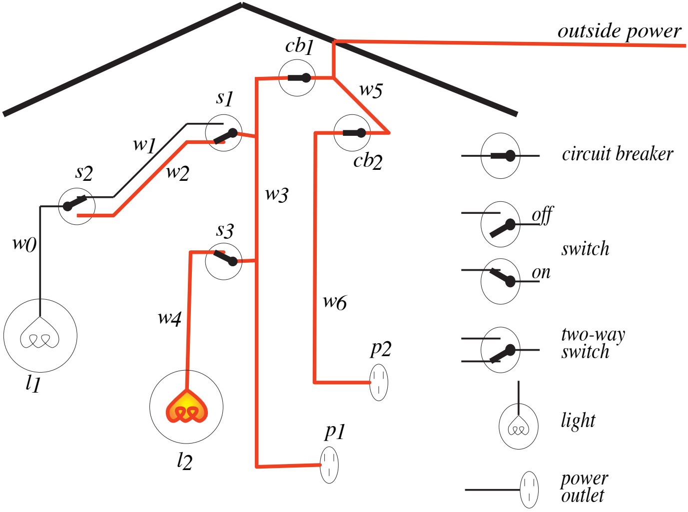
To debug answer g that is false in the intended interpretation:

- 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

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

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

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

$connected\_to(X, Y)$ 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

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

$connected\_to(X, Y)$ 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

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

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

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

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:

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

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

$p(A, b, C, D)$ 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.
- 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

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