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).
\begin{align*}
  \text{in}(\text{kim}, \text{r123}). \\
  \text{part\_of}(\text{r123}, \text{cs\_building}). \\
  \text{in}(X,Y) & \leftarrow \\
  \text{part\_of}(Z,Y) \land \\
  \text{in}(X,Z).
\end{align*}
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.
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.
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.
definite clause is either an atomic symbol (a fact) or of the form:

\[ a \leftarrow b_1 \land \cdots \land b_m \]

where \( a \) and \( b_i \) are atomic symbols.

query is of the form \(?b_1 \land \cdots \land b_m\).

knowledge base is a set of definite clauses.
\begin{itemize}
\item \textit{in}(kim, R) \leftarrow \textit{teaches}(kim, cs322) \land \textit{in}(cs322, R).
\item \textit{grandfather}(william, X) \leftarrow \textit{father}(william, Y) \land \textit{parent}(Y, X).
\item \textit{slithy}(toves) \leftarrow \textit{mimsy} \land \textit{borogroves} \land \textit{outgrabe}(mome, Raths).
\end{itemize}
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
An interpretation is a triple $I = \langle D, \phi, \pi \rangle$, where

- $D$, the domain, is a nonempty set. Elements of $D$ are individuals.
- $\phi$ is a mapping that assigns to each constant an element of $D$. Constant $c$ denotes individual $\phi(c)$.
- $\pi$ 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).

- \( D = \{\text{phone}, \text{pencil}, \text{telephone}\} \).
- \( \phi(\text{phone}) = \text{phone}, \phi(\text{pencil}) = \text{pencil}, \phi(\text{telephone}) = \text{telephone} \).
- \( \pi(\text{noisy}) \):

  - \( \langle \text{phone} \rangle \) FALSE
  - \( \langle \text{pencil} \rangle \) TRUE
  - \( \langle \text{telephone} \rangle \) FALSE

- \( \pi(\text{left_of}) \):

  - \( \langle \text{phone}, \text{pencil} \rangle \) FALSE
  - \( \langle \text{phone}, \text{telephone} \rangle \) TRUE
  - \( \langle \text{pencil}, \text{phone} \rangle \) TRUE
  - \( \langle \text{pencil}, \text{pencil} \rangle \) FALSE
  - \( \langle \text{telephone}, \text{pencil} \rangle \) FALSE
  - \( \langle \text{telephone}, \text{telephone} \rangle \) TRUE
  - \( \langle \text{pencil}, \text{telephone} \rangle \) FALSE
  - \( \langle \text{pencil}, \text{pencil} \rangle \) FALSE
  - \( \langle \text{pencil}, \text{telephone} \rangle \) FALSE
  - \( \langle \text{telephone}, \text{pencil} \rangle \) FALSE
  - \( \langle \text{telephone}, \text{pencil} \rangle \) FALSE
  - \( \langle \text{telephone}, \text{telephone} \rangle \) TRUE
  - \( \langle \text{pencil}, \text{telephone} \rangle \) FALSE
  - \( \langle \text{pencil}, \text{pencil} \rangle \) FALSE
  - \( \langle \text{pencil}, \text{telephone} \rangle \) 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.
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 $\pi(p)(t'_1, \ldots, t'_n) = \text{TRUE}$, where $t_i$ denotes $t'_i$ in interpretation $I$ and
- **false in interpretation $I$** if $\pi(p)(t'_1, \ldots, t'_n) = \text{FALSE}$.

Ground clause $h \leftarrow b_1 \land \ldots \land b_m$ 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:

\[
\begin{align*}
\text{noisy}(\text{phone}) & \quad \text{true} \\
\text{noisy}(\text{telephone}) & \quad \text{true} \\
\text{noisy}(\text{pencil}) & \quad \text{false} \\
\text{left\_of}(\text{phone}, \text{pencil}) & \quad \text{true} \\
\text{left\_of}(\text{phone}, \text{telephone}) & \quad \text{false} \\
\text{noisy}(\text{pencil}) & \quad \text{left\_of}(\text{phone}, \text{telephone}) \quad \text{true} \\
\text{noisy}(\text{pencil}) & \quad \text{left\_of}(\text{phone}, \text{pencil}) \quad \text{false} \\
\text{noisy}(\text{phone}) & \quad \text{noisy}(\text{telephone}) \land \text{noisy}(\text{pencil}) \quad \text{true}
\end{align*}
\]
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.
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.
in(kim,r123).
part_of(r123,cs_building).
in(X,Y) ←
    part_of(Z,Y) ∧
in(X,Z).

© D. Poole and A. Mackworth 2008
Artificial Intelligence, Lecture 12.2, Page 10
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.
A **query** is a way to ask if a body is a logical consequence of the knowledge base:

\[ ?b_1 \land \cdots \land b_m. \]

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 = \[
\begin{align*}
\text{in}(\text{kim}, r123). \\
\text{part}\_\text{of}(r123, \text{cs}\_\text{building}). \\
\text{in}(X, Y) & \leftarrow \text{part}\_\text{of}(Z, Y) \land \text{in}(X, Z). 
\end{align*}
\]

<table>
<thead>
<tr>
<th>Query</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{?part}_\text{of}(r123, B).</td>
<td>\text{no}</td>
</tr>
</tbody>
</table>
Example Queries

\[ KB = \{ \begin{align*}
    &in(kim, r123). \\
    &part\_of(r123, cs\_building). \\
    &in(X, Y) \leftarrow part\_of(Z, Y) \land in(X, Z). 
\end{align*}\] 

<table>
<thead>
<tr>
<th>Query</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>?part_of(r123, B).</td>
<td>part_of(r123, cs_building)</td>
</tr>
<tr>
<td>?part_of(r023, cs_building).</td>
<td></td>
</tr>
</tbody>
</table>

©D. Poole and A. Mackworth 2008 Artificial Intelligence, Lecture 12.3, Page 4
Example Queries

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

<table>
<thead>
<tr>
<th>Query</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>?part_of(r123, B)</td>
<td>part_of(r123, cs_building)</td>
</tr>
<tr>
<td>?part_of(r023, cs_building)</td>
<td>no</td>
</tr>
<tr>
<td>?in(kim, r023)</td>
<td>no</td>
</tr>
</tbody>
</table>
Example Queries

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

<table>
<thead>
<tr>
<th>Query</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>?\text{part_of}(r123, B).</td>
<td>\text{part_of}(r123, \text{cs_building})</td>
</tr>
<tr>
<td>?\text{part_of}(r023, \text{cs_building}).</td>
<td>no</td>
</tr>
<tr>
<td>?\text{in}(\text{kim}, r023).</td>
<td>no</td>
</tr>
<tr>
<td>?\text{in}(\text{kim}, B).</td>
<td>no</td>
</tr>
</tbody>
</table>
Example Queries

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

<table>
<thead>
<tr>
<th>Query</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>?part_of(r123, B).</td>
<td>part_of(r123, cs_building)</td>
</tr>
<tr>
<td>?part_of(r023, cs_building).</td>
<td>no</td>
</tr>
<tr>
<td>?in(kim, r023).</td>
<td>no</td>
</tr>
<tr>
<td>?in(kim, B).</td>
<td>in(kim, r123)</td>
</tr>
<tr>
<td></td>
<td>in(kim, cs_building)</td>
</tr>
</tbody>
</table>
Atom $g$ is a logical consequence of $KB$ if and only if:

- $g$ is a fact in $KB$, or
- there is a rule

$$g \leftarrow b_1 \land \ldots \land b_k$$

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

$$g \leftarrow b_1 \land \ldots \land b_k$$

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 $b_i$. 

© D. Poole and A. Mackworth 2008
Artificial Intelligence, Lecture 12.3, Page 9
% light(L) is true if L is a light
light(l₁). light(l₂).

% down(S) is true if switch S is down
down(s₁). up(s₂). up(s₃).

% ok(D) is true if D is not broken
ok(l₁). ok(l₂). ok(cb₁). ok(cb₂).

?light(l₁). →
% *light*(L) is true if L is a light


% *down*(S) is true if switch S is down

*down*(s₁).  *up*(s₂).  *up*(s₃).

% *ok*(D) is true if D is not broken


? *light*(l₁).  ⇒  yes

? *light*(l₆).  ⇒
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). $$\implies$$ yes
?light(l_6). $$\implies$$ no
?up(X). $$\implies$$
Axiomatizing the Electrical Environment

% light(L) is true if L is a light
light(l₁). light(l₂).
% down(S) is true if switch S is down
down(s₁). up(s₂). up(s₃).
% ok(D) is true if D is not broken
ok(l₁). ok(l₂). ok(cb₁). ok(cb₂).

?light(l₁). ⟷ yes
?light(l₆). ⟷ no
?up(X). ⟷ up(s₂), up(s₃)
connected_to(X, Y) is true if component X is connected to Y

connected_to(w_0, w_1) ← up(s_2).
connected_to(w_0, w_2) ← down(s_2).
connected_to(w_1, w_3) ← up(s_1).
connected_to(w_2, w_3) ← down(s_1).
connected_to(w_4, w_3) ← up(s_3).
connected_to(p_1, w_3).

? connected_to(w_0, W).  ⇒
connected_to(X, Y) is true if component X is connected to Y

connected_to(w₀, w₁) ← up(s₂).
connected_to(w₀, w₂) ← down(s₂).
connected_to(w₁, w₃) ← up(s₁).
connected_to(w₂, w₃) ← down(s₁).
connected_to(w₄, w₃) ← up(s₃).
connected_to(p₁, w₃).

?connected_to(w₀, W).  ⇒  W = w₁
?connected_to(w₁, W).  ⇒  W = w₁
connected_to(X, Y) is true if component X is connected to Y

connected_to(w_0, w_1) ← up(s_2).
connected_to(w_0, w_2) ← down(s_2).
connected_to(w_1, w_3) ← up(s_1).
connected_to(w_2, w_3) ← down(s_1).
connected_to(w_4, w_3) ← up(s_3).
connected_to(p_1, w_3).

?connected_to(w_0, W).  ⇒  W = w_1
?connected_to(w_1, W).  ⇒  no
?connected_to(Y, w_3).  ⇒  
connected_to(X, Y) is true if component X is connected to Y

\[
\begin{align*}
\text{connected_to}(w_0, w_1) & \leftarrow up(s_2). \\
\text{connected_to}(w_0, w_2) & \leftarrow down(s_2). \\
\text{connected_to}(w_1, w_3) & \leftarrow up(s_1). \\
\text{connected_to}(w_2, w_3) & \leftarrow down(s_1). \\
\text{connected_to}(w_4, w_3) & \leftarrow up(s_3). \\
\text{connected_to}(p_1, w_3). & \\
\end{align*}
\]

?connected_to(w_0, W). \implies W = w_1 \\
?connected_to(w_1, W). \implies no \\
?connected_to(Y, w_3). \implies Y = w_2, Y = w_4, Y = p_1 \\
?connected_to(X, W). \implies
connected_to(X, Y) is true if component X is connected to Y

connected_to(w₀, w₁) ← up(s₂).
connected_to(w₀, w₂) ← down(s₂).
connected_to(w₁, w₃) ← up(s₁).
connected_to(w₂, w₃) ← down(s₁).
connected_to(w₄, w₃) ← up(s₃).
connected_to(p₁, w₃).

?connected_to(w₀, W).  ⬝→  W = w₁
?connected_to(w₁, W).  ⬝→  no
?connected_to(Y, w₃).  ⬝→  Y = w₂, Y = w₄, Y = p₁
?connected_to(X, W).  ⬝→  X = w₀, W = w₁, . . .
% lit(L) is true if the light L is lit

\[
\text{lit}(L) \leftarrow \text{light}(L) \land \text{ok}(L) \land \text{live}(L).
\]

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

\[
\begin{align*}
\text{live}(Y) & \leftarrow \\
& \quad \text{connected\_to}(Y, Z) \land \\
& \quad \text{live}(Z).
\end{align*}
\]

\[
\text{live(outside)}.
\]

This is a \textbf{recursive definition} of \textit{live}. 
Recursion and Mathematical Induction

\[
\text{above}(X, Y) \leftarrow \text{on}(X, Y).
\]
\[
\text{above}(X, Y) \leftarrow \text{on}(X, Z) \land \text{above}(Z, Y).
\]

This can be seen as:

- Recursive definition of \textit{above}: prove \textit{above} in terms of a base case (\textit{on}) or a simpler instance of itself; or
- Way to prove \textit{above} by mathematical induction: the base case is when there are no blocks between \(X\) and \(Y\), and if you can prove \textit{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:

\[ \text{enrolled}(S, C) \]

which is true when student \( S \) is enrolled in course \( C \).

You can’t define the relation:

\[ \text{empty\_course}(C) \]

which is true when course \( C \) has no students enrolled in it. This is because \( \text{empty\_course}(C) \) doesn’t logically follow from a set of \( \text{enrolled} \) 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 \( t_i \).
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 $\sigma$ is a **unifier** of $e_1$ and $e_2$ if $e_1\sigma = e_2\sigma$.
- Substitution $\sigma$ is a **most general unifier** (mgu) of $e_1$ and $e_2$ if
  - $\sigma$ is a unifier of $e_1$ and $e_2$; and
  - if substitution $\sigma'$ 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

\[ p(A, b, C, D) \text{ and } p(X, Y, Z, e) \text{ have as unifiers:} \]
\begin{itemize}
  \item \( \sigma_1 = \{ X/A, Y/b, Z/C, D/e \} \)
  \item \( \sigma_2 = \{ A/X, Y/b, C/Z, D/e \} \)
  \item \( \sigma_3 = \{ A/V, X/V, Y/b, C/W, Z/W, D/e \} \)
  \item \( \sigma_4 = \{ A/a, X/a, Y/b, C/c, Z/c, D/e \} \)
  \item \( \sigma_5 = \{ X/A, Y/b, Z/A, C/A, D/e \} \)
  \item \( \sigma_6 = \{ X/A, Y/b, Z/C, D/e, W/a \} \)
\end{itemize}

The first three are most general unifiers. The following substitutions are not unifiers:
\begin{itemize}
  \item \( \sigma_7 = \{ Y/b, D/e \} \)
  \item \( \sigma_8 = \{ X/a, Y/b, Z/c, D/e \} \)
\end{itemize}
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.
A **generalized answer clause** is of the form

\[
\text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land a_2 \land \ldots \land a_m,
\]

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

\[
a \leftarrow b_1 \land \ldots \land b_p,
\]

where \(a_i\) and \(a\) have most general unifier \(\theta\), is

\[
(\text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_{i-1} \land b_1 \land \ldots \land b_p \land a_{i+1} \land \ldots \land a_m)\theta.
\]
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 \(yes(t_1, \ldots, t_k) \leftarrow a_1 \land a_2 \land \ldots \land a_m\)

Select atom \(a_i\) in the body of \(ac\);

Choose clause \(a \leftarrow b_1 \land \ldots \land b_p\) in \(KB;\)

Rename all variables in \(a \leftarrow b_1 \land \ldots \land b_p;\)

Let \(\theta\) be the most general unifier of \(a_i\) and \(a.\)

Fail if they don’t unify;

Set \(ac\) to \((yes(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_{i-1} \land b_1 \land \ldots \land b_p \land a_{i+1} \land \ldots \land a_m)\theta\)

end while.
Example

\[
\begin{align*}
\text{live}(Y) & \leftarrow \text{connected_to}(Y, Z) \land \text{live}(Z). \quad \text{live(outside)}. \\
\text{connected_to}(w_6, w_5). \quad \text{connected_to}(w_5, \text{outside}). \\
? \text{live}(A). \\
\text{yes}(A) & \leftarrow \text{live}(A). \\
\text{yes}(A) & \leftarrow \text{connected_to}(A, Z_1) \land \text{live}(Z_1). \\
\text{yes}(w_6) & \leftarrow \text{live}(w_5). \\
\text{yes}(w_6) & \leftarrow \text{connected_to}(w_5, Z_2) \land \text{live}(Z_2). \\
\text{yes}(w_6) & \leftarrow \text{live(outside)}. \\
\text{yes}(w_6) & \leftarrow .
\end{align*}
\]
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.
A list is an ordered sequence of elements.

Let’s use the constant \texttt{nil} to denote the empty list, and the function \texttt{cons}(H, T) to denote the list with first element \textit{H} and rest-of-list \textit{T}. \textbf{These are not built-in.}

The list containing \textit{sue}, \textit{kim} and \textit{randy} is

\[
\text{cons}(\text{sue, cons(kim, cons(randy, nil))})
\]

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

\[
\text{append}(\text{nil, Z, Z}).
\]

\[
\text{append}(\text{cons(A, X), Y, cons(A, Z))} \leftarrow \text{append}(X, Y, Z).
\]