Variables

- Variables are \textit{universally quantified} in the scope of a clause.
- A \textit{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 \textit{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

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

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

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

<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></td>
</tr>
</tbody>
</table>
Example Queries

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

<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}(kim, r023)).</td>
<td>no</td>
</tr>
</tbody>
</table>
Example Queries

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

<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, cs_building)</td>
</tr>
<tr>
<td>?\text{part_of}(r023, cs_building)</td>
<td>no</td>
</tr>
<tr>
<td>?\text{in}(kim, r023)</td>
<td>no</td>
</tr>
<tr>
<td>?\text{in}(kim, B)</td>
<td>no</td>
</tr>
</tbody>
</table>

©D. Poole and A. Mackworth 2010 Artificial Intelligence, Lecture 12.3, Page 6
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 $KB$. 
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:

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

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 $b_i$. 
Electrical Environment

- **Switches:**
  - S1
  - S2
  - S3

- **Lights:**
  - L1
  - L2

- **Power Outlet:**
  - P1
  - P2

- **Circuit Breakers:**
  - CB1
  - CB2

- **Wires:**
  - W0
  - W1
  - W2
  - W3
  - W4
  - W5
  - W6
% 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). ➞
% 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). ➞
Axiomatizing the Electrical Environment

% light(L) is true if L is a light

\[ \text{light}(l_1). \quad \text{light}(l_2). \]

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

\[ \text{down}(s_1). \quad \text{up}(s_2). \quad \text{up}(s_3). \]

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

\[ \text{ok}(l_1). \quad \text{ok}(l_2). \quad \text{ok}(cb_1). \quad \text{ok}(cb_2). \]

\[ ?\text{light}(l_1). \quad \Rightarrow \quad \text{yes} \]

\[ ?\text{light}(l_6). \quad \Rightarrow \quad \text{no} \]

\[ ?\text{up}(X). \quad \Rightarrow \quad \]
% 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(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).  ⇒
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).  ⊨
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).  \implies  W = w_1
?connected\_to(w_1, W).  \implies  no
?connected\_to(Y, w_3).  \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).  ⇒
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

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

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

\[ live(Y) \leftarrow connected\_to(Y, Z) \land live(Z). \]
\[ live(outside). \]

This is a recursive definition of live.
Recursion and Mathematical Induction

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

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!