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:

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

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

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

$\sigma_2 = \{Y/b, D/e\}$
$\sigma_5 = \{X/a, Y/b, Z/c, D/e\}$
1: **procedure** *unify*(*t_1*, *t_2*)  \[\triangleright\text{Returns mgu of } t_1 \text{ and } t_2 \text{ or } \bot.\]
2: \[E \leftarrow \{t_1 = t_2\}\] \[\triangleright\text{Set of equality statements}\]
3: \[S \leftarrow \{}\] \[\triangleright\text{Substitution}\]
4: **while** \(E \neq \{}\) **do**
5: \hspace{1cm} select and remove \(x = y\) from \(E\)
6: \hspace{2cm} **if** \(y\) is not identical to \(x\) **then**
7: \hspace{3cm} **if** \(x\) is a variable **then**
8: \hspace{4cm} replace \(x\) with \(y\) in \(E\) and \(S\)
9: \hspace{4cm} \(S \leftarrow \{x/y\} \cup S\)
10: \hspace{2cm} **else if** \(y\) is a variable **then**
11: \hspace{3cm} replace \(y\) with \(x\) in \(E\) and \(S\)
12: \hspace{3cm} \(S \leftarrow \{y/x\} \cup S\)
13: \hspace{2cm} **else if** \(x\) is \(p(x_1, \ldots, x_n)\) and \(y\) is \(p(y_1, \ldots, y_n)\) **then**
14: \hspace{3cm} \(E \leftarrow E \cup \{x_1 = y_1, \ldots, x_n = y_n\}\)
15: \hspace{2cm} **else**
16: \hspace{3cm} **return** \(\bot\) \[\triangleright\text{ } t_1 \text{ and } t_2 \text{ do not unify}\]
17: \hspace{2cm} **return** \(S\) \[\triangleright S \text{ is mgu of } t_1 \text{ and } t_2\]

©D. Poole and A. Mackworth 2016  \[\quad\]\[Artificial Intelligence, Lecture 13.3, Page 12\]
Atom $g$ is a logical consequence of $KB$ if and only if:

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

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

in $KB$ such that each $b_i$ is a logical consequence of $KB$. 
Aside: 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 $b_i$. 
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 “$h \leftarrow b_1 \land \ldots \land b_m$” in $KB$ such that there is a substitution $\theta$ such that for all $i$, there exists $b'_i \in C$ and $\theta'_i$ where $b_i\theta = b'_i\theta'_i$ and there is no $h' \in C$ and $\theta'$ such that $h'\theta' = h\theta$

$C := C \cup \{h\theta\}$

until no more clauses can be selected.
Example

\[\text{live}(Y) \leftarrow \text{connected}_\text{to}(Y, Z) \land \text{live}(Z). \quad \text{live(\text{outside}).} \]

\[\text{connected}_\text{to}(w_6, w_5). \quad \text{connected}_\text{to}(w_5, \text{outside}).\]
Example

\[
live(Y) \leftarrow connected\_to(Y, Z) \land live(Z).\ live(outside).
\]
\[
connected\_to(w_6, w_5).\ connected\_to(w_5, outside).
\]
\[
C = \{ live(outside),
\quad connected\_to(w_6, w_5),
\quad connected\_to(w_5, outside),
\quad live(w_5),
\quad live(w_6)\}\]
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 $h$. 
If $KB \vdash g$ then $KB \models g$.

- Suppose there is a $g$ such that $KB \vdash g$ and $KB \models \neq g$.
- 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 an instance of clause in $KB$ of form

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

where
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 an instance of clause in $KB$ of form

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

  where $h = h'\theta$ and $b_i\theta$ is an instance of an element of $C$.
  - Each $b_i\theta$ is true in $I$.
  - $h$ is false in $I$.
  - So an instance of this clause is false in $I$.
  - Therefore $I$ isn’t a model of $KB$.
  - Contradiction.
- 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 a constant if the KB or query doesn’t contain one. Each constant denotes itself.
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 a constant 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.
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 a constant 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:
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 a constant 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 \( h \leftarrow b_1 \land \ldots \land b_m \) 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 $KB \models g$.

- Suppose $KB \models g$. Then $g$ is true in all models of $KB$.

Thus $g$ is in the fixed point.

Thus $g$ is generated by the bottom up algorithm.

Thus $KB \models g$.
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.
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.
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.
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 $KB \vdash g$. 
A generalized answer clause is of the form

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

where \( t_1, \ldots, t_k \) are terms and \( a_1, \ldots, a_m \) are atoms.
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 \(yes(V_1, \ldots, V_k) \leftarrow B\)

\(\textbf{while } ac \text{ is not an answer } \textbf{do}\)

- Suppose \(ac\) is \(yes(t_1, \ldots, t_k) \leftarrow a_1 \land a_2 \land \ldots \land a_m\)
- \(\textbf{select} \) atom \(a_i\) in the body of \(ac\)
- \(\textbf{choose} \) 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 \((yes(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_{i-1} \land b_1 \land \ldots \land b_p \land a_{i+1} \land \ldots \land a_m)\theta\)

\(\textbf{end while}\).

Answer is \(V_1 = t_1, \ldots, V_k = t_k\)

where \(ac\) is \(yes(t_1, \ldots, t_k) \leftarrow \)
Example

\[\text{live}(Y) \leftarrow \text{connected}_\text{to}(Y, Z) \land \text{live}(Z). \quad \text{live(outside)}.\]
\[\text{connected}_\text{to}(w_6, w_5). \quad \text{connected}_\text{to}(w_5, \text{outside}).\]
\[?\text{live}(A)\].
Example

\[ \text{live}(Y) \leftarrow \text{connected\_to}(Y, Z) \land \text{live}(Z). \quad \text{live(outside)}. \]

\[ \text{connected\_to}(w_6, w_5). \quad \text{connected\_to}(w_5, \text{outside}). \]

\[ \text{?live}(A). \]

\[ \text{yes}(A) \leftarrow \text{live}(A). \]

\[ \text{yes}(A) \leftarrow \text{connected\_to}(A, Z_1) \land \text{live}(Z_1). \]

\[ \text{yes}(w_6) \leftarrow \text{live}(w_5). \]

\[ \text{yes}(w_6) \leftarrow \text{connected\_to}(w_5, Z_2) \land \text{live}(Z_2). \]

\[ \text{yes}(w_6) \leftarrow \text{live(outside)}. \]

\[ \text{yes}(w_6) \leftarrow . \]
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 $\textit{nil}$ to denote the empty list, and the function $\textit{cons}(H, T)$ 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

$$\textit{cons}(sue, \textit{cons}(kim, \textit{cons}(randy, \textit{nil})))$$

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


Consider a knowledge base consisting of one fact:

\[ \text{lt}(X, s(X)). \]

Should the following query succeed?

\[ \text{ask} \quad \text{lt}(Y, Y). \]
Consider a knowledge base consisting of one fact:

\[ \text{lt}(X, s(X)). \]

Should the following query succeed?

\[ \text{ask} \quad \text{lt}(Y, Y). \]

What does the top-down proof procedure give?
Unification with function symbols

- Consider a knowledge base consisting of one fact:

\[ \text{lt}(X, s(X)). \]

- Should the following query succeed?

\[ \text{ask } \text{lt}(Y, Y). \]

- What does the top-down proof procedure give?

- Solution: variable \( X \) should not unify with a term that contains \( X \) inside.
  E.g., \( X \) should not unify with \( s(X) \).
  Simple modification of the unification algorithm, which Prolog does not do!