Declarative Programming
PROLOG (+ Bayesian Nets)

● Motivation
  ★ Warm Fuzzies
  ★ What is Logic? ... Logic Programming?

● Mechanics of Prolog
  ★ Terms, Substitution, Unification, Horn Clauses
  ★ Proof (procedure)
  ★ Example: List Processing

● Theoretical Foundations
  ★ Semantics
  ★ Logic / Theorem Proving ... Resolution

● Issues
  ★ Search Strategies
  ★ Declarative/Procedural, ...

● Other parts of Prolog
  ★ “Impure” Operators — NOT, !
  ★ Utilities

● Constraint Programming

● Bayesian Belief Nets
What is Logic?

Logic is *formal system for reasoning*

Reasoning is *inferring new facts from old*

Eg:  
\[
\begin{align*}
\text{Given: } & \{ \text{All men are mortal.} \\
& \text{Socrates is a man.} \}
\end{align*}
\]

*infer (conclude, reason that, ...)*

Socrates is mortal.

What is role of Logic within CS?

1. Foundation of discrete mathematics
2. Automatic theorem proving
3. Hardware design/debugging
4. Artificial intelligence (Cmput366)

Components: Syntax (What does it look like?)
Semantics (What does it mean?)
Reasoning/Proof Theory (New facts from old)
Logic Programming

- Program ≡ Logic Formula
- Execution of Program ≡ theorem proving

User: 1. Specifies WHAT is true
   2. Asks if something else follows

Prolog answers question.

- By comparison,
  using Procedural Programming (C, Pascal, ...):
User must
  – decide on data-structure
  – explicitly write procedure
    search, match, substitute
  – write diff programs for
    father(X, tom) vs father(tom, Y)
Logic in general

Logics are formal languages for representing information such that conclusions can be drawn.

Syntax defines the sentences in the language . . . what does it look like?

Semantics define “meaning” of sentences; i.e., define truth of a sentence in a world. How is it linked to the world?

Proof Theory “new facts from old” find implicit information . . . “pushing symbols”

Eg, wrt arithmetic

$\boxed{x + 2 \geq y}$ is sentence; $\boxed{x^2 + y >}$ is not

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number $y$

$x + 2 \geq y$ is true in a world where $x = 7$, $y = 1$

$x + 2 \geq y$ is false in a world where $x = 0$, $y = 6$
What are Parts of a Logic?

• Syntax: Set of Expressions

? Well-formed or not?
SEQUENCE of Symbols $\rightarrow \begin{cases} \text{Accept} \\ \text{Reject} \end{cases}$

Accept: The boys are at home.
at(X, home) :- boy(X).

Reject: boys. home the angrily democracy
X(at), x Boy(1X,( ) :-

• Proof Process:

Given Believed statements,
Determine other Believed statements.

$$\{s_1, \ldots, s_n\} \vdash P \quad s$$

( Semantics: Which expressions are Believed? )

John’s mother is (the individual) Mary. $\rightarrow \top$
John’s mother is (the individual) Fred. $\rightarrow \bot$
Colorless green ideas sleep furiously. $\rightarrow \bot$
“Logic Programming” Framework

Knowledge Base

(Facts about World)

Is $\sigma$ true?

Proof Procedure

Yes...No
Concept of PROLOG

PROgramming in LOGic
≈ Sound Reasoning Engine

1. User asserts true statements.
   User asserts \( \{ \text{All men are mortal.} \)
   \( \text{Socrates is a man.} \) \}

2. User poses query.
   A. User asks “Is Socrates mortal?”
   B. User asks “Who/what is mortal?”

3. Prolog provides answer (Y/N, binding).
   A. Prolog answers “Yes”.
   B. Prolog answers “Socrates”.
Tying Prolog to Logic

- Syntax: Horn Clauses
  (aka Rules, Facts; Axioms)
  - Terms

- Proof Process: Resolution
  - Substitution
  - Unification

( Semantics
  - Only in that Resolution is Sound )
Proof Process: Backward Chaining

To prove $X$, find FACT $X$ in database

To prove $X$, find RULE $Y \Rightarrow X$ in database, then prove $Y$.

• Actually...

To prove $X$, find FACT $X'$ in database (where $X' \approx X$)

To prove $X$, find RULE $Y \Rightarrow X'$ in database, (where $X' \approx X$) then prove $Y$.

• Need to define...

What $X$ is? “Term”
When $X' \approx X$? “Unification”
Terms

- BNF:

\[
\langle \text{term} \rangle \ ::= \langle \text{constant} \rangle \mid \langle \text{variable} \rangle \\
\quad \mid \langle \text{functor} \rangle
\]

\[
\langle \text{constant} \rangle \ ::= \langle \text{atom starting w/lower case} \rangle
\]

\[
\langle \text{variable} \rangle \ ::= \langle \text{atom starting w/upper case} \rangle
\]

\[
\langle \text{functor} \rangle \ ::= \langle \text{constant} \rangle(\langle \text{tlist} \rangle)
\]

\[
\langle \text{tlist} \rangle \ ::= "" \mid \langle \text{term} \rangle \{,\langle \text{tlist} \rangle\}
\]

- Examples of \langle \text{term} \rangle:

\[
a1 \ b \ fred \quad \langle \text{constant} \rangle
\]

\[
X \ Yc3 \ Fred \quad \langle \text{variable} \rangle
\]

\[
\text{married}(\text{fred}) \quad \text{g}(a, \ f(Yc3), \ b) \quad \langle \text{functor} \rangle
\]

- Ground Term \equiv \text{term with no variables}

\[
f(q) \quad \text{g}(f(w), w1(b, c)) \quad \text{are ground},
\]

\[
f(A) \quad \text{g}(f(w), w1(B, c)) \quad \text{are not}.
\]
Substitution

A Substitution is a set \( \{ v_1/t_1 \, v_2/t_2 \, \cdots \, v_n/t_n \} \)
where \( v_i \) are distinct variables
\( t_i \) are terms that do not use
any of the \( v_j \)'s.

Examples:

\[
\begin{align*}
\begin{array}{c}
\vdash \{ X/a \} \\
\{ X/a \, Y/b \, Z/f(a, W) \} \\
\{ X/W \, Y/f(W) \, Z/W \}
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\dashv \{ f(X)/a \} \\
\{ X/a \, X/b \} \\
\{ X/f(X) \} \\
\{ X/f(Y) \, Y/g(q) \}
\end{array}
\end{align*}
\]
Applying a Substitution

- Given \( \begin{cases} t & - a \text{ term} \\ \sigma & - a \text{ substitution} \end{cases} \)

"\( t\sigma \)" is the term resulting from applying substitution \( \sigma \) to term \( t \).

- Small Examples:
  \[
  X\{X/a\} = a \\
  f(X)\{X/a\} = f(a)
  \]

- Example: Using \( t = f(\ a, h(Y,b), X \) \)
  \[
  t\{X/b\} = f(\ a, h(Y,b), b ) \\
  t\{X/b\ Y/f(Z)\} = f(\ a, h(f(Z),b), b ) \\
  t\{X/Z\ Y/f(Z,a)\} = f(\ a, h(f(Z,a),b), Z ) \\
  t\{W/Z\} = f(\ a, h(Y,b), X )
  \]

- \( \sigma \) need not include all variables in \( t \);
  \( \sigma \) can include variables not in \( t \).
Composition of Substitutions

- Composition:
  \( \sigma \circ \theta \) is *composition* of substitutions \( \sigma, \theta \).
  For any term \( t \), \( t[\sigma \circ \theta] = (t\sigma)\theta \).

- Example:
  \[
  f(X)[\{X/Z\} \circ \{Z/a\}] = (f(X)[X/Z])\{Z/a\} \\
  = f(Z)\{Z/a\} \\
  = f(a)
  \]

- \( \sigma \circ \theta \) is a *substitution* (usually)

- Eg:
  \[
  \{X/a\} \circ \{Y/b\} = \{X/a, Y/b\} \\
  \{X/Z\} \circ \{Z/a\} = \{X/a, Z/a\}
  \]
**Unifiers**

- $t_1$ and $t_2$ are **unified by $\sigma$** iff

  \[ t_1\sigma = t_2\sigma. \]

  Then $\sigma$ is called a **unifier**

  $t_1$ and $t_2$ are **unifiable**

- **Examples:**

  \[
  \begin{array}{cccc}
  t_1 & t_2 & \text{unifier} & \text{term} \\
  \hline
  f(b,c) & f(b,c) & \{\} & f(b,c) \\
  f(X,b) & f(a,Y) & \{X/a\ Y/b\} & f(a,b) \\
  f(a,b) & f(c,d) & * & f(a,b) \\
  f(a,b) & f(X,X) & * & f(a,b) \\
  f(X,a) & f(Y,Y) & \{X/a\ Y/a\} & f(a,a) \\
  f(g(U),d) & f(X,U) & \{U/d\ X/g(d)\} & f(\ g(d),d\ ) \\
  f(X) & f(g(X)) & * & f(a,a) \\
  f(X,g(X)) & f(Y,Y) & * & f(a,a) \\
  f(X) & f(Y) & \{X/Y\} & f(Y) \\
  \end{array}
  \]

- **NB** $t_1$ and $t_2$ are symmetrical!

  (Both can have variables.)
**Multiple Unifiers**

- Unifier for \( t_1 = f(X) \) and \( t_2 = f(Y) \)

\[
\theta \quad t_1 \theta = t_2 \theta =
\]

\[
\{ X/Y \} \quad f(Y) \\
\{ Y/X \} \quad f(X) \\
\{ Y/a \quad X/a \} \quad f(a) \\
\{ Y/g(b,Z) \quad X/g(b,Z) \} \quad f(g(b(Z))) \\
\{ X/Y \quad W/f(q,Z) \} \quad f(Y)
\]

- \{Y/X\} and \{X/Y\} make sense, but
  \{ Y/a \quad X/a \} has irrelevant constant
  \{ X/Y \quad W/g \} has irrelevant binding (W)

- Adding irrelevant bindings: \( \infty \) unifiers!

? Is there a best one?
**Quest for Best Unifier**

- **Wish list:**
  - No irrelevant constants
    So \{Y/X\} preferred over \{Y/a, X/a\}
  - No irrelevant bindings
    So \{Y/X\} preferred over \{Y/X, W/f(4,Z)\}

- Spse \(\lambda_1\) has constant where \(\lambda_2\) has variable
  (Eg, \(\lambda_1 = \{X/a, Y/a\}, \lambda_2 = \{X/Y\}\))
  Then \(\exists\) subsitution \(\mu\) s.t. \(\lambda_2 \circ \mu = \lambda_1\)
  (Eg, \(\mu = \{Y/a\}: \{X/Y\}\circ\{Y/a\} = \{X/a, Y/a\}\))

- Spse \(\lambda_1\) has extra binding over \(\lambda_2\)
  (Eg, \(\lambda_1 = \{X/a, Y/b\}, \lambda_2 = \{X/a\}\))
  Then \(\exists\) subsitution \(\mu\) s.t. \(\lambda_2 \circ \mu = \lambda_1\)
  (Eg, \(\mu = \{Y/b\}: \{X/a\}\circ\{Y/b\} = \{X/a, Y/b\}\))

- **INFERIOR** unifier = composition of
  Good Unifier + another substitution
Most General Unifier

• $\sigma$ is a mgu for $t_1$ and $t_2$ iff
  
  $-$ $\sigma$ unifies $t_1$ and $t_2$, and
  
  $-$ $\forall \mu$: unifier of $t_1$ and $t_2$, 
  
  $\exists$ subsitution, $\theta$, s.t. $\sigma \circ \theta = \mu$.
  
  (Ie, for all terms $t$, $t\mu = (t\sigma)\theta$.)

• Example: $\sigma = \{X/Y\}$ is mgu for $f(X)$ and $f(Y)$.
  Consider unifier $\mu = \{X/a \ Y/a\}$.
  Use substitution $\theta = \{Y/a\}$:
  
  $f(X)\mu = f(X)\{X/a \ Y/a\}$
  
  $= f(a)$
  
  $f(X)[\sigma \circ \theta] = (f(X)\sigma) \theta$
  
  $= (f(X)\{X/Y\})\theta$
  
  $= f(Y)\{Y/a\}$
  
  $= f(a)$
  
  Similarly, $f(Y)\mu = f(a) = f(Y)[\sigma \circ \theta]$
  
  ($\mu$ is NOT a mgu, as $\exists \theta'$ s.t. $\mu \circ \theta' = \sigma$ !)
MGU — Example #2

A mgu for

\[ f(W,g(Z),Z) \land f(X,Y,h(X)) \]

is \( \{ X/W \ Y/g(h(W)) \ Z/h(W) \} \)
MGU (con’t)

• Notes:
  – If $t_1$ and $t_2$ are unifiable, then $\exists$ a mgu.
  – Can be more than 1 mgu
    but they differ only in variable names.
  – Not every unifier is a mgu.
  – A mgu uses constants only as necessary.

• Implementation:
  $\exists$ fast algorithm that computes
  a mgu of $t_1$ and $t_2$, if one exists;
  or reports failure.

( Slow part is verifying legal substitution:
  none of $v_i$ appear in any $t_j$.
  Avoid by resetting Prolog’s occurscheck parameter.)
MGU Procedure

Recursive Procedure MGU (x,y)
  If x=y then Return ()
  If Variable(x) then Return( MguVar(x,y) )
  If Variable(y) then Return( MguVar(y,x) )
  If Constant(x) or Constant(y) then Return( False )
  If Not(Length(x) = Length(y)) then Return( False )
  g ← []
  For i = 0 .. Length(x)
    s ← MGU( Part(x,i), Part(y,i) )
    g ← Compose(g,s)
    x ← Substitute(x,g)
    y ← Substitute(y,g)
  Return( g )
End

Procedure MguVar (v,e)
  If Includes(v,e) then Return( False )
  Return( [v/e] )
End
Backward Chaining

- Recall

<table>
<thead>
<tr>
<th>To prove $\text{X}$,</th>
</tr>
</thead>
<tbody>
<tr>
<td>find FACT $\text{X}'$ in database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To prove $\text{X}$,</th>
</tr>
</thead>
<tbody>
<tr>
<td>find RULE $\text{Y} \Rightarrow \text{X}'$ in database,</td>
</tr>
<tr>
<td>then prove $\text{Y}$.</td>
</tr>
</tbody>
</table>

- Prolog writes $\text{Y} \Rightarrow \text{X}'$ as $\text{X}' :- \text{Y}$

so always unifies $\text{X}'$ against “first part”...

| $\text{X}'$ | $\text{X}' :- \text{Y}$ |

- Issue: What if rule is $\text{Y}_1 \& \text{Y}_2 \Rightarrow \text{X}'$?
Prolog's Syntax

• BNF:

\[ \langle \text{Horn} \rangle ::= \langle \text{literal} \rangle . \mid \langle \text{literal} \rangle :- \langle \text{llist} \rangle . \]

\[ \langle \text{llist} \rangle ::= \langle \text{literal} \rangle \{ , \langle \text{llist} \rangle \} \]

\[ \langle \text{literal} \rangle ::= \langle \text{term} \rangle \]

• Examples:

\[
\begin{align*}
\text{father}(\text{john}, \text{sue}). \\
\text{father}(\text{odin}, X). \\
\text{parent}(X, Y) :- \text{father}(X, Y). \\
\text{gparent}(X, Z) :- \text{parent}(X, Y), \text{parent}(Y, Z).
\end{align*}
\]

• How to read as predicate calculus?

\[
\begin{align*}
\text{father}(\text{john}, \text{sue}) \\
\forall X. \text{father}(\text{odin}, X). \\
\forall X,Y. \text{father}(X,Y) \Rightarrow \text{parent}(X,Y). \\
\forall X,Y,Z. \text{parent}(X,Y) \& \text{parent}(Y,Z) \Rightarrow \text{gparent}(X,Z)
\end{align*}
\]
Relation to Predicate Calculus

• In general:

\[
\rightarrow \forall x_1 \ldots \forall x_m. t \\
[\text{called "atomic formula"}]
\]

\[
t \in t_1, t_2, \ldots, t_n. \\
\rightarrow \forall x_1, \ldots, x_m. t_1 \& t_2 \ldots \& t_n \Rightarrow t
\]

[\text{called "(production) rule"}]

• Set of Predicate Calculus Expressions = Knowledge Base \equiv

\text{Conjunctive Normal Form:}

\((A_1 \lor \neg A_2 \lor \neg A_7) \& (\neg A_1 \lor A_3 \lor A_4) \& \ldots \& (\neg A_2 \lor \neg A_4)\)

• Horn clause is disjunction with ONE Positive Literal

• \langle Horn \rangle Form is CNF, where every clause is Horn

\ldots has ONE Positive Literal

So \langle Horn \rangle \subset CNF.

\exists Predicate Calculus expressions which can\text{NOT} be written as Horn Clauses.

(Eg: \(A \lor B\))
Prolog’s Proof Process

- User provides
  - \( KB \): Knowledge Base
    (List of Horn Clauses — axioms)
  - \( \gamma \): Query (aka Goal, Theorem)
    (Literal) 1. Who is mortal? \( \text{mortal}(X) \).
    2. Is Socrates mortal? \( \text{mortal}(\text{soc}) \).

- Prolog finds
  - a Proof of \( \gamma \), from \( KB \), if one exists
    & substitution for \( \gamma \)'s variables: \( \sigma \)
    \[
    KB \vdash_P \gamma(\sigma) \quad KB_1 \vdash_P \text{mortal}(X)(\{X/\text{soc}\})
    \]

- Failure (otherwise)

- Returns bindings
  Finds “Top-Down” (refutation) Proof
  Actually returns LIST of \( \sigma \)'s [one for each proof]
  \[ \{X/\text{soc}\} \quad \{X/\text{plato}\} \quad \{X/\text{freddy}\} \quad \ldots \]
Examples of Proofs: I

- Using Knowledge Base, \( KB_1 = \)
  \[
  \begin{align*}
  \text{on}(a, b). & \quad (1) \\
  \text{on}(b, c). & \quad (2) \\
  \text{above}(X, Y) & \text{:-} \ \text{on}(X, Y). \quad (3)
  \end{align*}
  \]

- Query \( \gamma_1: \ \text{on}(a,b) \)

  \[
  \begin{array}{c}
  \text{on}(a,b) \\
  \text{success} \\
  \end{array}
  \]

  \( (1) \)

  Hence, \( KB_1 \models_P \ \text{on}(a,b) \) \[
  \begin{array}{c}
  \text{empty substitution} \\
  \end{array}
  \]

  ( Like Data Base retrieval )
Examples of Proof: II (variables)

- Using Knowledge Base, \( KB_1 \)

- Query \( \gamma_2: \text{on}(a,Y) \)
  
  \[
  \begin{align*}
  \text{on}(a,Y) \\
  \text{(1) } Y = b \\
  \text{success} - \{Y/b\}
  \end{align*}
  \]

  (Say \( KB_1 \vdash_P \text{on}(a,Y)\{Y/b\} \))

- Query \( \gamma_3: \text{on}(X,Y) \)
  
  \[
  \begin{align*}
  \text{on}(X,Y) \\
  X=a, \ Y=b \quad (1) \quad (2) \ X=b, \ Y=c \\
  \text{success} - \{X/a, Y/b, \} \quad \text{success} - \{X/b, Y/c, \}
  \end{align*}
  \]

\[
\begin{align*}
(KB_1 \vdash_P \text{on}(X,Y)\{X/a, Y/b\} & \quad \rightarrow \quad KB_1 \vdash_P \text{on}(a,b) \\
(KB_1 \vdash_P \text{on}(X,Y)\{X/b, Y/c\} & \quad \rightarrow \quad KB_1 \vdash_P \text{on}(b,65) \\
\end{align*}
\]
Examples of Proof: III (failures)

( Using Knowledge Base, \( KB_1 \) )

- Query \( \gamma_4 \): on(a, b10)

  \[
  \text{on(a, b10)}
  \]
  \[
  \times
  \]

  (Hence, \( KB_1 \not\vdash_P \text{on(a, b10)} \) )

- Query \( \gamma_5 \): on(X, b10)

  \[
  \text{on(X, b10)}
  \]
  \[
  \times
  \]

  (Hence, \( KB_1 \not\vdash_P \text{on(X, b10)}, \) for any value of X.)
Examples of Proof: IV (rules)

(Using $KB_1$)

- Query $\gamma_6$: above(b,c)

\[
\text{above}(b,c) \quad (3) \quad X = b, \ Y = c \\
\text{on}(b,c) \quad (2) \\
\text{success}
\]

(Hence, $KB_1 \vdash_P \text{above}(b,c)$)

- Query $\gamma_7$: above(b,W)

\[
\text{above}(b,W) \quad (3) \quad X = b, \ Y = W \\
\text{on}(b,W) \quad (2) \quad W = c \\
\text{success} - \{W/c\}
\]

(Hence, $KB_1 \vdash_P \text{above}(b,W)\{W/c\}$

$\rightarrow \quad KB_1 \vdash_P \text{above}(b,c)$)
Examples of Proof: $V$ (big)

$KB_2 = \begin{cases} 
\text{on}(a, b). & (1) \\
\text{on}(b, c). & (2) \\
\text{above}(X, Y) : - \text{on}(X, Y). & (3) \\
\text{above}(X, Y) : - \text{on}(X, Z), \text{above}(Z, Y). & (4) 
\end{cases}$

\begin{center}
\begin{tikzpicture}
  \node (KB2) {above(a,c)};
  \node (onac) [below of=KB2] {on(a,c)};
  \node (onaZaboveZc) [below of=KB2] {on(a,Z), above(Z,c)};
  \node (Xac) [left of=KB2] {X=a, Y=c (3)};
  \node (XabYc) [right of=KB2] {X=a, Y=c (4)};
  \node (Zb) [below of=KB2] {Z=b};
  \node (onbc) [below of=Zb] {above(b,c)};
  \node (onbcXbYc) [left of=KB2] {X=b, Y=c (3)};
  \node (onbcXbYc) [right of=KB2] {X=b, Y=c (4)};
  \node (success) [below of=KB2] {success};
  \node (onccXcYc) [left of=KB2] {X=c, Y=c (3)};
  \node (onccXcYc) [right of=KB2] {X=c, Y=c (4)};
  \node (oncc) [below of=onccXcYc] {on(c,c)};
  \node (onccYcaboveYc) [below of=onccXcYc] {on(c,Y), above(Y,c)};
  \draw (KB2) -- (onac);
  \draw (KB2) -- (onaZaboveZc);
  \draw (onac) -- (Xac);
  \draw (onac) -- (XabYc);
  \draw (onaZaboveZc) -- (Zb);
  \draw (Zb) -- (onbc);
  \draw (onbc) -- (onbcXbYc);
  \draw (onbc) -- (onbcXbYc);
  \draw (onac) -- (success);
  \draw (onaZaboveZc) -- (success);
  \draw (success) -- (oncc);
  \draw (success) -- (onccYcaboveYc);
  \draw (oncc) -- (onccXcYc);
  \draw (onccYcaboveYc) -- (onccXcYc);
\end{tikzpicture}
\end{center}
Examples of Proof: VI (many answers)

• Using $KB_3 =$

\[
\begin{align*}
on(a, b). & \quad (1) \\
on(b, c). & \quad (2) \\
above(X, Y) & \quad : - \quad on(X,Y). & \quad (3) \\
above(X, Y) & \quad : - \quad on(X,Z), above(Z,Y). & \quad (4) \\
above(c1, c2). & \quad (5)
\end{align*}
\]

Query $\gamma_9$: $above(X,Y)$

• Answers:

- $[X=a,Y=b]$ \quad $above(a,b)$ \quad (3), (1)
- $[X=b,Y=c]$ \quad $above(b,c)$ \quad (3), (2)
- $[X=a,Y=c]$ \quad $above(a,c)$ \quad (4), (1), (3), (2)
- $[X=c1,Y=c2]$ \quad $above(c1,c2)$ \quad (5)
Prolog’s Proof Process

• A *goal* is either
  – a sequence of literals (conjunction),
  – the special goal “success”

(eg, \texttt{on(X,Y)}, \texttt{p(X,5), q(X)}, \texttt{success} \ldots)

• The sequence of goals

\[
\langle G_1, G_2, \ldots, G_n \rangle
\]

is a *top-down proof* of \( G_1 \)
(from the knowledge base, \( KB \)) iff

1. \( G_n = \text{success} \), and

2. \( G_i \) is a *SUBGOAL* (in \( KB \)) of \( G_{i-1} \),
   \[ i = 2, 3, \ldots n \]
Subgoals

Subgoals of $G = \{ g_1, \ldots, g_r \}$ in $KB$:

Rule 1 If atomic axiom “$t$” in $KB$
where $t$ and $g_i$ have mgu $\sigma$, then
$$\{ g_1\sigma, \ldots, g_{i-1}\sigma, g_{i+1}\sigma, \ldots, g_r\sigma \}$$ is a subgoal of $G$.

(If $r = 1$, then “success” is subgoal of $G$.)

Rule 2 If axiom “$t :- t_1, \ldots, t_k$” in $KB$
where $t$ and $g_i$ have mgu $\sigma$, then
$$\{ t_1\sigma, \ldots, t_k\sigma, g_1\sigma, \ldots, g_{i-1}\sigma, g_{i+1}\sigma, \ldots g_r\sigma \}$$ is a subgoal of $G$. 
Example of Subgoals — I

\[ KB_3 = \begin{cases} 
(1) & \text{on}(a, b). \\
(2) & \text{on}(b, c). \\
(3) & \text{above}(X, Y) \iff \text{on}(X,Y). \\
(4) & \text{above}(X, Y) \iff \text{on}(X,Z), \text{above}(Z,Y). \\
(5) & \text{above}(c1, c2). 
\end{cases} \]

Subgoals of...

- \textbf{above}(A,B) are
  - \textbf{on}(A,B): \sigma = \{ X/A, Y/B \}
    using Rule 2, (3)
  - \textbf{on}(A,Z), \textbf{above}(Z,B): \sigma = \{ X/A, Y/B \}
    using Rule 2, (4)
  - \textbf{success}: \sigma = \{ A/c1, B/c2 \}
    using Rule 1, (5)
Example of Subgoals – II

\[ KB_3 = \begin{cases} 
(1) & \text{on}(a, b). \\
(2) & \text{on}(b, c). \\
(3) & \text{above}(X, Y) :\rightarrow \text{on}(X, Y). \\
(4) & \text{above}(X, Y) :\rightarrow \text{on}(X, Z), \text{above}(Z, Y). \\
(5) & \text{above}(c1, c2). 
\end{cases} \]

Subgoals of …

- \{ \text{on}(A, Z_1), \text{above}(Z_1, B) \} are

  - \textcolor{red}{\textbf{above}(b, B)}: \quad \sigma = \{ A/a, Z_1/b \}
    using Rule 1, (1) [1st literal]

  - \textcolor{red}{\textbf{above}(c, B)}: \quad \sigma = \{ A/b, Z_1/c \}
    using Rule 1, (2) [1st literal]

  - \textcolor{red}{\textbf{on}(A, c1)}: \quad \sigma = \{ Z_1/c1, B/c2 \}
    using Rule 1, (5) [2nd literal]

  - \textcolor{red}{\textbf{on}(Z_1, B), \text{on}(A, Z_1)}: \quad \sigma = \{ X/Z_1, Y/B \}
    using Rule 2, (3) [2nd literal]

  - \textcolor{red}{\textbf{on}(Z_1, Z), \text{above}(Z, B), \text{on}(A, Z_1)}: \quad \sigma = \{ X/Z_1, Y/B \}
    using Rule 2, (4) [2nd literal]
Comments wrt Prolog’s Proof Procedure

- \{ \text{Variable bindings, Unifier} \} \text{ found during proof}

- Prolog returns these overall mgu’s \textbf{1-by-1}

- Which “strategy”?  
  - Within “frontier” of subgoal-sets, which to expand?  
  - Given specific subgoal-set, which literal?  
  - Given specific literal (within subgoal-set), which rule/assertion?

  \textit{(Prolog uses “SLD Resolution” strategy)}

- Does Prolog work correctly?

- Does Prolog run efficiently?
What User Really Types

> sicstus
SICStus 3.11.2 (x86-linux-glibc2.3): Wed Jun 2 11:44:50 CEST
Licensed to cs.ualberta.ca
| ?- [user].       % For user to enter ‘‘Assert-fact’’ mode.
| on(a,b).        % Prolog’s answer to most operations.
| on(b,c).
| above(X,Y) :- on(X,Y).
| ^D user con...  % Typing ‘‘^D exits ‘‘assert’’ mode.
yes
| ?- on(a,b).      % User asks a question.
yes
| ?- on(a,Y).      % User’s second question.
Y = b_          % Prolog’s answer: a binding list.
    % User types CR.
yes
| ?- on(X,Y).      % User’s third question.
X = a
Y = b;           % Prolog’s binding list
    % User asks for ANOTHER answer
    % by typing ‘‘;’’.
X = b
Y = c;           % Prolog supplies another binding list
    % Still not satisfied, user asks for
    % yet ANOTHER answer by typing ‘‘;’’.
no
| ?-
> sicstus
SICStus 3.11.2 (x86-linux-glibc2.3): Wed Jun 2 11:44:50 CEST
Licensed to cs.ualberta.ca
| ?- [file1]. % File ‘‘file1’’ contains propositions
file1 consulted 120 bytes 0.0333333 sec.
yes % Prolog’s answer to this operation.
| ?- on(a,b10). % User asks a question.
no % Prolog’s answer: not derivable.
| ?- on(X,b10). % User’s next question.
no % Again, no answer.
| ?- above(b,c).
yes % Prolog can find a proof
% Notice: needs more than simple lookup.

| ?- above(b,W).
W = c; % Prolog find an answer.
no % … but only one answer.
| ?-