Overview

⇒ Variables

• Top-down Proof Procedure with Variables
• Top-Down Reasoning Procedure
• Function Symbols
• Proof Procedures
• Top-Down Reasoning Procedure
Variables in Clauses

• Example KB

\[
\begin{align*}
\text{father}(\text{tim}, \text{steve}) & \quad \text{father}(\text{steve}, \text{john}) \\
\text{mother}(\text{pam}, \text{john}) & \quad \text{mother}(\text{susan}, \text{pam}) \\
\text{mother}(\text{helen}, \text{steve}) & \quad \text{mother}(\text{paula}, \text{tim}) \\
\text{parent}(X, Y) & \leftarrow \text{father}(X, Y) \\
\text{parent}(X, Y) & \leftarrow \text{mother}(X, Y) \\
\text{grandparent}(X, Y) & \leftarrow \text{parent}(X, Z) \land \text{parent}(Z, Y)
\end{align*}
\]

• Variables in KB useful for expressing knowledge

- Can derive \textit{parent} and \textit{grandparent} from \textit{father} and \textit{mother}, without having to specifying a lot of extra facts
- Only way to express an infinite amount of knowledge when we add function symbols
Handling Variables

• In order for a clause to be true for an interpretation, must be true in that interpretation for any variable assignment

• Could do proof procedure on all ground instances of the clauses
  - Include all constants in $KB$ and in query
  - If no constants, one (just one) needs to be invented
  - Only a finite number, so algorithm guaranteed to stop
  - Method is complete and sound for proving ground atoms

• Example

  \[
  q(a).
  q(b).
  r(a).
  s(W) \leftarrow r(W).
  p(X, Y) \leftarrow q(X) \land s(Y).
  \]
Need Alternative

• Number of ground instances of clauses could be huge
• Example

\[
\text{explained}(Room,\text{Now}) \leftarrow \text{hasdetector}(Room) \\
\land \text{lastmotion}(Room,\text{Prev}) \\
\land \text{subtract}(\text{Now},\text{Prev},\text{Diff}) \\
\land \text{motionlessinroom}(Room,\text{Time}) \\
\land \text{less}(\text{Diff},\text{Time})
\]

- Has 5 variables: Room Now Prev Diff Time
- If 100 constants in KB & Query,
  will be 100*100*100*100*100 = 10^{10} instances

• Need proof procedure to directly handle clauses with variables
Substitution

- **Substitution** is a finite set of the form \{V_1/t_1, ... V_n/t_n\}
  - Each $V_i$ is a distinct variable and each $t_i$ is a term
  - A substitution is in *normal form* if no $V_i$ appears in any $t_j$
  - \{X/Y, Y/a\} is not in normal form, but \{X/a, Y/a\} is

- **Application** of a substitution $\sigma = \{V_1/t_1, ..., V_n/t_n\}$ to expression $e$ written $e\sigma$ is the expression with every occurrence of $V_i$ in $e$ replaced by the corresponding $t_i$
  - $e\sigma$ is an *instance* of $e$
  - if $e\sigma$ is ground then it is called a ground instance of $e$

- Instance of clause represented as original clause + substitution
Examples

- $p(a, X) \{X/c\}$
- $p(Y, c) \{Y/a\}$
- $p(a, X) \{Y/a, Z/X\}$
- $p(X, X, Y, Y, Z) \{X/Z, Y/t\}$
- $p(X, Y) \leftarrow q(a, Z, X, Y, Z) \{X/Y, Z/a\}$
Unifiers

• Substitution $\sigma$ is a **unifier** of expressions $e_1$ and $e_2$ if $e_1\sigma$ is the same as $e_2\sigma$  
  - Example: $\{X/a, Y/b\}$ is a unifier of $t(a, Y, c)$ and $t(X, b, c)$

• Expressions have many unifiers  
  - Example: $p(X, Y)$ and $p(Z, Z)$

  - Which is best?
Most General Unifier

- **Most General Unifier (MGU)**
  - If $\sigma$ is a unifier of $e_1$ and $e_2$ giving $e$ and if for any other unifier of them, say giving $e'$, $e'$ is an instance of $e$

- If two expressions can be unified, they will have a MGU
  - Could be more than one

- Expression $e$ is *renaming* of $e'$ if differ only in names of vars
  - They are both instances of each other
  - Expressions resulting from applying MGU are renamings of each other

- **Example:** $p(X, Y)$ and $p(Z, Z)$
  - $\{X/Z, Y/Z\}$ is an MGU resulting in $p(Z, Z)$
  - $\{Y/X, Z/X\}$ is an MGU resulting in $p(X, X)$
Overview

• Variables

⇒ Top-down Proof Procedure with Variables

• Top-Down Reasoning Procedure

• Function Symbols

• Proof Procedures

• Top-Down Reasoning Procedure
Top-down Proof Procedure Recap

• Start with goal, work toward facts in $KB$
• Definite Clause Resolution for Ground Case

\[ \text{yes} \leftarrow a_1 \land \ldots \land a_m \]
\[ a_i \leftarrow b_1 \land \ldots \land b_p \]
\[ \text{yes} \leftarrow a_1 \land \ldots \land a_{i-1} b_1 \land \ldots \land b_p a_{i+1} \land \ldots \land a_m \]
Definite Resolution with Variables

- Generalized answer clause
  - \( \text{yes}(t_1, ..., t_k) \leftarrow a_1 \land ... \land a_m \)

- Resolution Rule
  \[
  \begin{align*}
  \text{yes}(t_1, ..., t_k) & \leftarrow a_1 \land ... \land a_m \\
  a & \leftarrow b_1 \land ... \land b_p
  \end{align*}
  \]

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

- Where \( \theta \) is the most general unifier of \( a \) and \( a_i \)
Derivation

• Sequence of $\gamma_0, \gamma_1, \ldots, \gamma_n$

• $\gamma_0$ is answer clause corresponding to original query

• $\gamma_i$ obtained by
  - Give $\gamma_{i-1}$ fresh variables
    + Ensures $\gamma_{i-1}$ does not have any variables in common with anything in KB
    + Captures how variables are locally scoped
  - Choose an atom in body of $\gamma_{i-1}$
  - Choose a clause in $KB$ whose head will unify with the chosen atom
  - Resolve $\gamma_{i-1}$ with clause

• $\gamma_n$ is an answer, and so is of the form $yes(t_1, \ldots, t_k)\leftarrow$.

• Specification of a proof procedure!
Example: Robot Delivery
Robot Delivery KB

\[
\begin{align*}
  &\text{west}(r101, r103). \\
  &\text{west}(r103, r105). \\
  &\text{west}(r105, r107). \\
  &\text{west}(r107, r109). \\
  &\text{west}(r109, r111). \\
  &\text{west}(r131, r129). \\
  &\text{west}(r129, r127). \\
  &\text{west}(r127, r125). \\
  &\text{east}(E, W) \leftarrow \text{west}(W, E). \\
  &\text{next\_door}(E, W) \leftarrow \text{east}(E, W). \\
  &\text{next\_door}(W, E) \leftarrow \text{west}(W, E). \\
  &\text{two\_east}(E, W) \leftarrow \text{east}(E, M) \land \text{east}(M, W). \\
  &?\text{two\_east}(R, r107)
\end{align*}
\]
Overview

- Variables
- Top-down Proof Procedure with Variables
  ⇒ Top-Down Reasoning Procedure
- Function Symbols
- Proof Procedures
- Top-Down Reasoning Procedure
Reasoning Procedure

• (Not in chapter 2)

• Reasoning procedure
  - Resolves the nondeterminism of proof procedure
  - Needs to be done through search
    + Search for the set of choices that reasoning procedure would have picked
  - Search space is large so need to search carefully

• Reasoning procedure might be incomplete because either
  - Proof procedure was incomplete
  - Search strategy can’t find answer (perhaps because space is too large)
Depth-first Search

• Choice points
  - **Select** an atom in body of $\gamma_{i-1}$
  - **Choose** a clause in $KB$ whose head with unify with the chosen atom
• Always select first atom in body
  - We will have to consider each atom eventually, so just start with the first
• Choose first clause in KB whose head matches
  - Run with this as long as possible
  - If fail to produce an answer, backtrack to most recent choice, and pick next one
• Equivalent to Depth-first Search (but more lazily)
  - Nodes are derivations $\gamma$
  - Derivation has children of everything that can be derived from it, using different rules from the KB
Example

?two_east(_0,r107)

Answer clause corresponding

A: Use \textit{two\_east}(E,W) ← east(E,M) \land east(M,W)

B: Use \textit{east}(E,W) ← \textit{west}(W,E)

C: Use \textit{west}(r101,r103)

D: Use \textit{east}(E,W) ← \textit{west}(W,E)

Nothing unifies with \textit{west}(r107,r101).

Nothing else unifies with \textit{east}(r101,r107).

C: Use \textit{west}(r103,r105)

D: Use \textit{east}(E,W) ← \textit{west}(W,E)

Nothing unifies with \textit{west}(r107,r105).

Nothing else unifies with \textit{east}(r105,r107).

E: Use \textit{west}(r107,r109)

...
Overview

• Variables
• Top-down Proof Procedure with Variables
• Top-Down Reasoning Procedure

⇒ Function Symbols

• Proof Procedures
• Top-Down Reasoning Procedure
Function Symbols

• Predicate symbols used to assert that something is true or false
• constants refer to something in the domain
• variables refer to something in the domain
• functions also refer to something in the domain
  - constant mary could be mapped to Mary
  - function motherof(john) could also be mapped to Mary
• predicate mother(mary, john) versus function motherof(john)
  - predicate symbol captures truths about the world
    + that mary is john’s mother
  - function symbols just point to someone
Usefulness of Function Symbols

- Can talk about objects in the domain without having a constant symbol for them
- Might want to say \texttt{time(13,15)} to refer to 1:15pm
  - Just need 60 constant symbols rather than 24*60

- Keep in mind: term \textit{function} not used like it is elsewhere in CS
  - Does not capture anything about how time works
  - Capturing knowledge about time is up to predicate symbols and clauses
Further Usefullness of Function Symbols

• Want to reason about paths through a maze
  + How can we represent path 1, 2 versus 1, 2, 6
  + Make constant for each path: patha and pathb
    + Define facts: \texttt{start(pathb,1)}
      \texttt{after(pathb,1,2)}
      \texttt{after(pathb,2,6)}

• But all paths must be predefined in the KB
  - Infinite number of possible paths (including cycles)
Further Usefullness of Function Symbols

• Want to reason about paths through a maze
  + How can we represent path 1, 2 versus 1, 2, 6
  + Make constant for each path: **patha** and **pathb**
  + Define facts:
    - `start(pathb,1)`
    - `after(pathb,1,2)`
    - `after(pathb,2,6)`

• But all paths must be predefined in the KB
  - Infinite number of possible paths (including cycles)

• Can use functions to refer to a path by referring to its elements
  - Functions have a fixed number of arguements
    + So cannot use `path(1,2)`  `path(1,2,6)`
  - Instead, make path one cell at a time: `p(6,p(2,p(1,null)))`
    + constant **null** represents an empty path
    + function `p(T,R)` refers to path whose top element is `T` and rest of path is `R`
Function Syntax in Datalog

- **Function symbol** is a token starting with lowercase letter
- **Term** is either a variable, constant or of the form $f(t_1, ... t_n)$
  - Where $f$ is a function symbol and the $t_i$’s are terms
- Terms can only appear inside of predicates (arbitrarily nested)
  - Cannot appear alone in a KB, as part of a body, or as a head of a clause
Semantics of Function Symbols

• $\phi$ used to just map constants to objects in the domain
• $\phi$ also maps $n$-ary function $f$ to $D^n \rightarrow D$
  - Notice that it is defined as mapping $D^n$ to $D$, not constants $^n$
  - Hence, there can be objects in the domain that might not have a constant for them, but can only be referred to with function symbols
• Interpretations no longer finite
  - One 1-ary function symbol can name an infinite number of objects
  - Example
    + Constant 0
    + Successor function $s : D \rightarrow D$
    + Can specify all of the natural numbers: $0, s(0), s(s(0)), s(s(s(0))), ...$
Defining Functions

• Any knowledge about functions must be defined by clauses
• What knowledge of numbers might we want?

• What knowledge of paths (lists) might we want?
Knowledge about Lists

• Can represent lists by
  - constant `null` represents an empty path
  - function `p(T,R)` refers to path where `T` is top element and `R` is rest of path
  - example: `p(6,p(2,p(1,null)))`

• Can we write a predicate `member(X,List)`
  - True if `X` is in list `List`
Knowledge about Lists

• Can represent lists by
  - constant **null** represents an empty path
  - function \( p(T,R) \) refers to path where \( T \) is top element and \( R \) is rest of path
  - example: \( p(6,p(2,p(1,null))) \)

• Can we write a predicate **member**(\( X, List \))
  - True if \( X \) is in list \( List \)

\[
\text{member(Top, p(Top, Rest))} \\
\text{member(X, p(Top, Rest))} ← \text{member(X, Rest)}
\]
Building Data Structures

• Can use function symbols to build other data structures

• Tree data structure:
  - A labeled tree is either a node \( \text{node}(\text{Name}, \text{LeftTree}, \text{RightTree}) \)
    or a leaf \( \text{l}(\text{Name}) \)
  - Example:
    \[
    \text{node}(n_1, \text{node}(n_2, \text{l}(l_1), \text{l}(l_2)), \text{node}(n_3, \text{l}(l_3), \text{node}(n_4, \text{l}(l_4), \text{l}(l_5))))
    \]
    \[
    \begin{array}{c}
    n_1 \\
    \text{n2} \quad \text{n3} \\
    \text{l1} \quad \text{l2} \quad \text{l3} \quad \text{n4} \\
    \text{l4} \quad \text{l5}
    \end{array}
    \]
Clauses about Trees

• \( \text{has\_leaf}(L, T) \) is true if \( L \) is the label of a leaf in tree \( T \)

\[
\text{has\_leaf}(L, l(L)).
\]

\[
\text{has\_leaf}(L, \text{node}(N, LT, RT)) \leftarrow \text{has\_leaf}(L, LT).
\]

\[
\text{has\_leaf}(L, \text{node}(N, LT, RT)) \leftarrow \text{has\_leaf}(L, RT).
\]
Clauses about Numbers

• Let $lt(X, Y)$ be true when $X < Y$
  - To define it in Datalog, need to capture facts and rules about it that capture its entire meaning
  - What is a comprehensive fact about $lt$ that we can write?
    + Fact should have $lt$ as its predicate
    + Should include an $s(\ldots)$
    + Make it as general as possible
  - What is a rule that we can write about $lt$
    + Should have $lt$ on right and left hand side
    + Atom on right hand side should be simpler than left hand side
    + By repeatedly applying the rule, should end at fact
    + Think of this as the induction step in a proof by induction
Overview

• Variables
• Top-down Proof Procedure with Variables
• Top-Down Reasoning Procedure
• Function Symbols
⇒ Proof Procedures
• Top-Down Reasoning Procedure
Bottom-Up Proof Procedure with Variables

• Previously, had bottom-up proof procedure replace clauses with variables with all ground instances
• But, function symbols cause infinite number of terms
• But it is countable
  - There is a way to enumerate all terms
  - Just as there is a way to enumerate all rational numbers
• Make sure procedure \textit{fairly} introduces ground instances
Top Down Proof Procedure

• Just have to make sure procedure that determines MGU works with function symbols

• Need to be careful about normal form
  - Substitution is a finite set of the form \( \{V_1/t_1, \ldots, V_n/t_n\} \)
  - Each \( V_i \) is a distinct variable and each \( t_i \) is a term
  - A substitution is in normal form if no \( V_i \) appears in any \( t_j \)

• Most substitutions can be put into normal form
  \( \{X/Z, Z/a\} \Rightarrow \{X/a, Z/a\} \)
  \( \{X/Z, Z/X\} \Rightarrow \{X/Z\} \)

• Can any substitution be put into normal form?
  - What about \( \{X/f(X)\} \)?
Normal Form of Substitutions

• \{X/f(X)\} cannot be put into normal form.
  - So is normal form too restrictive?
  - What would this substitution even mean?

• Consider \( KB = lt(X, s(X)) \)
  \[ lt(X, s(Y)) \leftarrow lt(X, Y). \]

• Does \( lt(X, X) \) follow from \( KB \)
  - Does \( lt(X_1, X_1) \) unify with \( lt(X, s(X)) \)?
    + Note we made up new variables so we don’t get confused
Normal Form of Substitutions

• \{X/f(X)\} cannot be put into normal form.
  - So is normal form too restrictive?
  - What would this substitution even mean?

• Consider $KB = lt(X, s(X))$
  
  $lt(X, s(Y)) ← lt(X, Y)$.

• Does $lt(X, X)$ follow from $KB$
  
  - Does $lt(X_1, X_1)$ unify with $lt(X, s(X))$?
    + Note we made up new variables so we don’t get confused
  
  - The unifier \{X_1/X, X/s(X)\} sort of makes them the same
    + But this cannot be put into normal form
    + Good thing, otherwise, we would have an example of an unsound inference
    + Checking for this is called \textit{occurs check}
Algorithm for Finding MGU (Not in textbook)

• Take two expressions (no variables in common)
  - Compare them token for token (left to right)

• If one has a connector, other must have same one

• If one has \( n \)-ary symbol \( p \), other must as well

• For each term of predicates and functions
  - If both terms are same variable, don’t need to do anything
  - If one has variable \( V \) and other has term \( t \), add \( V/t \) to substitution
    + \( t \) should not contain \( V \) (occurs check)
    + Apply \( V/t \) to rest of both expressions and to any terms in substitution list
    + Variable \( V \) should now only be in substitution once (on left hand side)
  - Otherwise, if one has constant \( c \), other must as well
  - Otherwise, both are functions, and make sure they unify (recursive)
Examples

\[ p(X, Y) \text{ and } p(Z, Z) \]

\[ p(X, X) \text{ and } p(f(A, c), B) \]

\[ p(X, X) \text{ and } p(B, f(A, c)) \]

\[ p(X, X) \text{ and } p(B, f(A, B)) \]
Overview

• Variables
• Top-down Proof Procedure with Variables
• Top-Down Reasoning Procedure
• Function Symbols
• Proof Procedures

⇒ Top-Down Reasoning Procedure
Top-Down Proof Procedure (Repeat)

• Sequence of $\gamma_0, \gamma_1, \ldots, \gamma_n$

• $\gamma_0$ is answer clause corresponding to original query

• $\gamma_i$ obtained by
  - Give $\gamma_{i-1}$ fresh variables
    + Ensures $\gamma_{i-1}$ does not have any variables in common with anything in KB
    + Captures how variables are locally scoped
  - Choose an atom in body of $\gamma_{i-1}$
  - Choose clause in $KB$ whose head will unify with the chosen atom
  - Resolve $\gamma_{i-1}$ with clause

• $\gamma_n$ is an answer, and so is of the form $yes(t_1, \ldots, t_k)\leftarrow$. 
Example Proof with Functions

• Defined $\text{has\_leaf}(L, T)$ as true if $L$ is label of leaf in tree $T$
  $\text{has\_leaf}(L, l(L))$.
  $\text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, LT)$.
  $\text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, RT)$.

• Prove $l_4$ is a leaf of $n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$

- Query?
Example Proof with Functions

• Defined $\text{has_leaf}(L, T)$ as true if $L$ is label of leaf in tree $T$
  
  $\text{has_leaf}(L, l(L))$.
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, LT)$.
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, RT)$.

• Prove $l4$ is a leaf of $n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5))))$
  
  $\text{yes} \leftarrow \text{has_leaf}(l4, n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5))))))$.
  1st clause in KB does not unify
  2nd clause in KB unifies $\text{yes} \leftarrow \text{has_leaf}(l4, n(n1, l(l1), l(l2)))$. A
    1st clause in KB does not unify
    2nd clause in KB unifies $\text{yes} \leftarrow \text{has_leaf}(l4, l(l1))$. B
    No clause in KB unifies. Backtrack to B.
Example Proof with Functions

- Defined \( \text{has\_leaf}(L, T) \) as true if \( L \) is label of leaf in tree \( T \)

\[
\text{has\_leaf}(L, l(L)).
\]

\[
\text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, LT).
\]

\[
\text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, RT).
\]

- Prove \( l4 \) is a leaf of \( n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5)))) \)

\[
yes \leftarrow \text{has\_leaf}(l4, n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5)))).
\]

1st clause in \( KB \) does not unify

2nd clause in \( KB \) unifies \( yes \leftarrow \text{has\_leaf}(l4, n(n1, l(l1), l(l2))). \)

\[
A
\]

1st clause in \( KB \) does not unify

2nd clause in \( KB \) fails

\[
B
\]

© P. Heeman, 2020  38 of 40  CS560  Class 04: Top-Down Reasoning Procedure
Example Proof with Functions

• Defined $\text{has_leaf}(L, T)$ as true if $L$ is label of leaf in tree $T$
  
  $\text{has_leaf}(L, l(L))$.
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, LT)$.
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, RT)$.

• Prove $l_4$ is a leaf of $n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$
  
  $\text{yes} \leftarrow \text{has_leaf}(l_4, n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))).$

  1st clause in KB does not unify
  2nd clause in KB unifies $\text{yes} \leftarrow \text{has_leaf}(l_4, n(n_1, l(l_1), l(l_2))).$  \hspace{1cm} A
  1st clause in KB does not unify
  2nd clause in KB fails $\text{B}$
  3rd clause in KB unifies $\text{yes} \leftarrow \text{has_leaf}(l_4, l(l_2)).$
Example Proof with Functions

• Defined \( \text{has\_leaf}(L, T) \) as true if \( L \) is label of leaf in tree \( T \)
  
  \[ \text{has\_leaf}(L, l(L)). \]
  
  \[ \text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, LT). \]
  
  \[ \text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, RT). \]

• Prove \( l_4 \) is a leaf of \( n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5)))) \)

\[ \text{yes} \leftarrow \text{has\_leaf}(l_4, n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))). \]

1st clause in \( KB \) does not unify

2nd clause in \( KB \) unifies \( \text{yes} \leftarrow \text{has\_leaf}(l_4, n(n_1, l(l_1), l(l_2))). \)

1st clause in \( KB \) does not unify

2nd clause in \( KB \) fails

3rd clause in \( KB \) unifies \( \text{yes} \leftarrow \text{has\_leaf}(l_4, l(l_2)). \)

No clause in \( KB \) unifies. Backtrack to A.
Example Proof with Functions

- Defined $\text{has_leaf}(L, T)$ as true if $L$ is label of leaf in tree $T$
  
  $\text{has_leaf}(L, l(L))$.
  
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, LT)$.
  
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, RT)$.

- Prove $l_4$ is a leaf of $n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$

  $\text{yes} \leftarrow \text{has_leaf}(l_4, n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$.

  1st clause in KB does not unify

  2nd clause in KB fails

© P. Heeman, 2020  38 of 40  CS560 Class 04: Top-Down Reasoning Procedure
Example Proof with Functions

• Defined $\text{has_leaf}(L, T)$ as true if $L$ is label of leaf in tree $T$
  
  $\text{has_leaf}(L, l(L))$.
  
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, LT)$.
  
  $\text{has_leaf}(L, n(N, LT, RT)) \leftarrow \text{has_leaf}(L, RT)$.

• Prove $l_4$ is a leaf of $n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$

  $\text{yes} \leftarrow \text{has_leaf}(l_4, n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$.

  1st clause in KB does not unify
  2nd clause in KB fails
  3rd clause in KB unifies  $\text{yes} \leftarrow \text{has_leaf}(l_4, n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$.

  1st clause in KB does not unify
  2nd clause in KB unifies  $\text{yes} \leftarrow \text{has_leaf}(l_4, l(13))$.  

  No clause unifies. Backtrack to $C$. 

© P. Heeman, 2020    38 of 40    CS560 Class 04: Top-Down Reasoning Procedure
Example Proof with Functions

- Defined $\text{has\_leaf}(L, T)$ as true if $L$ is label of leaf in tree $T$
  
  $\text{has\_leaf}(L, l(L))$.
  $\text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, LT)$.
  $\text{has\_leaf}(L, n(N, LT, RT)) \leftarrow \text{has\_leaf}(L, RT)$.

- Prove $l_4$ is a leaf of $n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$

  $\text{yes} \leftarrow \text{has\_leaf}(l_4, n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))).

  1st clause in KB does not unify
  2nd clause in KB fails
  3rd clause in KB unifies

  $\text{yes} \leftarrow \text{has\_leaf}(l_4, n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))$.

  1st clause in KB does not unify
  2nd clause in KB fails

  $\text{yes} \leftarrow \text{has\_leaf}(l_4, l(13))$.  

© P. Heeman, 2020  38 of 40  CS560 Class 04: Top-Down Reasoning Procedure
Example Proof with Functions

• Defined \( \text{has_leaf}(L, T) \) as true if \( L \) is label of leaf in tree \( T \)

\[
\begin{align*}
\text{has_leaf}(L, l(L)). \\
\text{has_leaf}(L, n(N, LT, RT)) & \leftarrow \text{has_leaf}(L, LT). \\
\text{has_leaf}(L, n(N, LT, RT)) & \leftarrow \text{has_leaf}(L, RT).
\end{align*}
\]

• Prove \( l_4 \) is a leaf of \( n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5)))) \)

\[
\text{yes} \leftarrow \text{has_leaf}(l_4, n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))).
\]

1st clause in KB does not unify

2nd clause in KB fails

3rd clause in KB unifies \( \text{yes} \leftarrow \text{has_leaf}(l_4, n(n_3, l(l_3), n(n_4, l(l_4), l(l_5)))) \).

1st clause in KB does not unify

2nd clause in KB fails

3rd clause in KB unifies \( \text{yes} \leftarrow \text{has_leaf}(l_4, l(l_3)) \).

1st clause in KB does not unify.

2nd clause in KB unifies. \( \text{yes} \leftarrow \text{has_leaf}(l_4, l(l_4)) \).

1st clause in KB does. \( \text{yes} \leftarrow . \).
Summary of Proof

\[ yes \leftarrow \text{has_leaf}(l4, n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5))))). \]
1st clause in KB does not unify
2nd clause in KB unifies \[ yes \leftarrow \text{has_leaf}(l4, n(n1, l(l1), l(l2))). \]
   1st clause in KB does not unify
   2nd clause in KB unifies \[ yes \leftarrow \text{has_leaf}(l4, l(l1)). \]
      No clause in KB unifies. Backtrack to B.
   2nd clause in KB fails
   3rd clause in KB unifies \[ yes \leftarrow \text{has_leaf}(l4, l(l2)). \]
      No clause in KB unifies. Backtrack to A.
2nd clause in KB fails
3rd clause in KB unifies \[ yes \leftarrow \text{has_leaf}(l4, n(n3, l(l3), n(n4, l(l4), l(l5))))). \]
   1st clause in KB does not unify
   2nd clause in KB unifies \[ yes \leftarrow \text{has_leaf}(l4, l(l3)). \]
      No clause unifies. Backtrack to C.
   2nd clause in KB fails \[ yes \leftarrow \text{has_leaf}(l4, l(l3)). \]
   3rd clause in KB unifies \[ yes \leftarrow \text{has_leaf}(l4, n(n4, l(l4), l(l5))). \]
      1st clause in KB does not unify.
      2nd clause in KB unifies. \[ yes \leftarrow \text{has_leaf}(l4, l(l4)). \]
         1st clause in KB does. \[ yes \leftarrow . \]
Final Word on Functions

• Functions let you refer to things without having explicit names for them
  - Can refer to any subtree, by describing by functions
    *It is the subtree with node n1 which right branch ... and left branch ...

• Unification does the right thing with functions
  - Just do hierarchal symbol matching
  - Makes it easy to reason about parts of the subtree by symbol matching