Handling Variables

- In order for a clause to be true for an interpretation, all variables 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 clauses.
- Can use in $KB$ and in query.
- Could do proof procedure on all ground instances of the clauses.
- If true in the interpretation for any variable assignment, must be true in all interpretations.

Example:

$q(a)$.
$q(b)$.
$r(a)$.
$s(W) ← r(W)$.

Variables in Clauses

- Variables in $KB$ useful for expressing knowledge.
- Only way to express an infinite amount of knowledge.
- Without having to specify a lot of extra facts.
- Can derive $parent$ and $grandparent$ from $father$ and $mother$.
- Only way to express an infinite amount of knowledge.

Example KB:

$grandparent(X,Y) ← parent(X,Z) ∧ parent(Z,Y)$.
$parent(X,Y) ← father(X,Y)$.
$parent(X,Y) ← mother(X,Y)$.
$mother(pam,john)$.
$mother(susan,pam)$.
$mother(helen,steve)$.
$mother(tim,steve)$.
$father(tim,steve)$.

Overview

- Top-down Reasoning Procedure
- Proof Procedures
- Function Symbols
- Variables
- Variables in Clauses
- Variables in KB
Substitution

- 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 \(t_j\).
- \{X/Y, Y/a\} is not in normal form, but \{X/a, Y/a\} is.

Application of a substitution \(\sigma = \{V_1/t_1, \ldots, V_n/t_n\}\) to expression \(e\) 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.
Overview

- Variables
  - Top-down Proof Procedure with Variables
  - Top-down Reasoning Procedure
  - Function Symbols

Most General Unifier

- Most General Unifier (MGU)
  - If two expressions can be unified, they will have a MGU
  - Which is best?

Unifiers

- Substitution only of variables and expressions
- Expressions have many unifiers
- Example: $p(A, B)$ and $p(B, A)$
- Example: $f(x)$ and $f(y)$
- Example: $\{x/y, y/x\}$
- Example: $\{x/z, y/z\}$
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:

- $\gamma_i$ is an answer and so is of the form $\forall e \in \gamma_i$.
  - Resolve $\gamma_i$ with clause
  - Choose a clause in KB whose head will unify with the chosen atom
  - Choose an atom in body of $\gamma_i$
  - Choose a clause in KB whose head will unify with the chosen atom
    - Ensures $\gamma_i$ does not have any variables in common with anything in KB
    - Gives $\gamma_i$ fresh variables
  - Obtain $\gamma_i$ as answer clause corresponding to original query

Definition

Generalized answer clause

- $yes(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_m$

Resolution Rule

- $yes(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_m$
- $a_i \leftarrow b_1 \land \ldots \land b_p$
- $\exists e \in \gamma_i$ (where $e$ is the most general unifier of $a_i$ and $a$)

Resolvent:

- Resolve $\gamma_{i-1}$ with clause
  - $\gamma_{i-1} \leftarrow a_{i-1} \land b_1 \land \ldots \land b_p \land a_i+1 \land \ldots \land a_m$

Definition Resolution with Variables

Top-down Proof Procedure Recap

- $\forall e \in \gamma_i$
- $\exists e \in \gamma_{i-1}$
- Choose clause in KB whose head will unify with the chosen atom
- Resolve $\gamma_{i-1}$ with clause
- $\exists e \in \gamma_{i-1}$
Overview

- Variables
- Top-down Proof Procedure with Variables

Robot Delivery KB

Example: Robot Delivery
Example

next one - If fails to produce an answer, backtrack to most recent choice and pick
- Hill with this as long as possible
- Choose first clause in KB whose head matches
- We will have to consider each atom eventually, so just start with the first
- Always select first atom in body
- Choose a clause in KB whose head matches with the chosen atom
- Select an atom in body of $\gamma_i$...
- Choice point

Depth-first Search

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 space is large so need to search carefully
  + Search for the set of choices that reasoning procedure would have picked
    - Needs to be done through search
    + Resolves the nondeterminism of proof procedure
- Reasoning Procedure (Not in chapter 2)
Usefulness of Function Symbols

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

Predicate 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 refer to something in the domain.

Function Symbols

- Function symbols used to assert that something is true or false.
- Function symbols could also be mapped to objects in the domain.
- Constants could also be mapped to objects in the domain.
- Constants refer to something in the domain.
- Variables refer to something in the domain.

Overview

- Top-down reasoning procedure.
- Proof procedures.
- Function symbols.
- Top-down reasoning procedure.
Semantics of Function Symbols

- φ used to just map constants to objects in the domain.
- φ also maps n-ary function symbol f to D^n → D.
- Notice that it is defined as mapping D^n to D, not constants.
- Hence, there can be objects in the domain that might not have a constant.
- There are no constraints on how many function symbols can be named.

Interpretations no longer finite.

- One 1-ary function symbol can name an infinite number of objects.
- Example: +
  - Constant 0
  - Successor function s: D → D
  - Can specify all of the natural numbers: 0, s(0), s(s(0)), ...

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 are terms.
- Terms can only appear inside of predicates (arbitrarily nested).
- Function symbol is a token starting with a lowercase letter.

Further Usefulness of Function Symbols

- What about lists or sets of individuals?
- We could make a constant symbol for each list.
  - has_member(lista, peter)
  - has_member(lista, tim)
- But infinite number of lists even when there is just a single constant.
  - Can't make up a name for every possible list.
- Can use functions to refer to a list by referring to its elements.
  - Functions have to have a fixed number of arguments.
  - So cannot use list(a, b)
  - Instead use function that lets you specify a list one element at a time.
  - cons(peter, cons(tim, null))
  - null is an empty list.
  - cons(X, L) refers to the list whose first element is X and the rest of the list is L.
Clauses about Trees

- has_leaf(L, tree) is true if L is the label of a leaf in tree.

Building Data Structures

- Can use function symbols to build other data structures.

Defining Functions

- Any knowledge about functions must be defined by clauses.
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
    - $\{X/Z, Z/a\} \Rightarrow \{X/a, Z/a\}$
    - $\{X/Z, Z/X\} \Rightarrow \{X/Z\}$

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
  - There is a way to enumerate all terms
    - Just as there is a way to enumerate all rational numbers
    - There is a way to enumerate all terms with function symbols with all ground instances
  - But it is countable
• Preceding's had bottom-up proof procedure replace classes with

Overiew

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

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
  - V variable should now only be in substitution on left hand side
- Otherwise, both are functions and make sure they unify recursively

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)) \rightarrow lt(X, Y)\}.
  - Does lt(X, X) follow from KB?
  - Does (X/X)\{lt(X, s(X)), lt(X, s(Y)) \rightarrow lt(X, Y)\} follow from (X/X)\{lt(X, s(X)), lt(X, s(Y)) \rightarrow lt(X, Y)\}?
  - Consider (X/X)\{lt(X, s(X)), lt(X, s(Y)) \rightarrow lt(X, Y)\} = (X/X)\{lt(X, s(X)), lt(X, s(Y)) \rightarrow lt(X, Y)\}
  - Where would this substitution even mean?
  - So normal form is too restrictive
  - Each cannot be put into normal form.
Example Proof with Functions

• Defined has leaf \((L, T)\) as true if \(L\) is label of leaf in tree \(T\).

• has leaf \((L, n(L, LT, RT))\) ← has leaf \((L, LT)\).

• has leaf \((L, n(L, LT, RT))\) ← has leaf \((L, RT)\).

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

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

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

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

1st clause in KB unifies. \(\text{yes} \leftarrow .\)
Final Word on Functions

- Functions let you refer to things without having explicit names
  - 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