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.
• Methods is complete and sound for proving ground clauses.

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 $mother$.

Example $KB$

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

Overview

- Top-down Reasoning Procedure
- Proof Procedures
- Function Symbols
- Top-down Reasoning Procedure with Variables
- Variables
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\}\}

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 \(t_j\).

• Application of a substitution \(\sigma = \{V_1/t_1, ..., V_n/t_n\}\) to expression \(e\) is \(e\sigma\).
  - \(e\sigma\) is an instance of \(e\) if \(e\sigma\) is ground.

Need Alternative

• Number of ground instances of clauses could be huge.

Example

Has 5 variables: Room Now Prev Diff Time

- Has 5 variables: Room Now Prev Diff Time
- Number of ground instances of clauses could be huge.
Overview

- Top-Down Reasoning Procedure
- Proof Procedures
- Function Symbols
- Top-Down Reasoning Procedure

Unifiers

- Substitution $\sigma$ is a unifier of expressions $e_1$ and $e_2$ if $e_1\sigma$ is the same as $e_2\sigma$.
- Expressions have many unifiers.
- Most General Unifier (MGU)

Most General Unifier

- If two expressions can be unified, they will have a MGU.
- If $a$ is a number, $v$ is a variable, and $e$ is any other unifier of them,
  $\{v/a\}$ is a unifier of $e$ and $v$ is an instance of $e$.

Which is Best?

- Example: $\{x/y, y/x\}$
- Expressions have many unifiers.
- $\{v/a\}$ is a unifier of $e$ and $v$ is an instance of $e$.
- $\{x/y\}$ is a unifier of $e$ and $e$ is an instance of $x/y$.
- $\{x/y, y/x\}$

Most General Unifier (MGU)

- If $e_1$ is the same as $e_2$ and $\sigma$ is a substitution of expressions $e_1$ and $e_2$,

Unifiers

- Example: $\{x/y\}$
- Expressions have many unifiers.
- $\{v/a\}$ is a unifier of $e$ and $v$ is an instance of $e$.
- $\{x/y\}$ is a unifier of $e$ and $e$ is an instance of $x/y$.
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 $\text{yes}(t_1, \ldots, t_k) \leftarrow$

Specification of a proof procedure!

Definite Resolution with Variables

- Generalized answer clause
  - $\text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_m$
- Resolution Rule
  - $\text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_i \land a_i+1 \land \ldots \land a_m$

- Where $\theta$ is the most general unifier of $a_i$ and $a_{i+1}$

Definite Clause Resolution for Ground Case

- Start with goal, work toward facts in KB
  - $\text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_m$

Top-down Proof Procedure Recap

• Start with goal, work toward facts in KB
  - $\text{yes}(t_1, \ldots, t_k) \leftarrow a_1 \land \ldots \land a_m$
Overview

• Variables

⇒ Top-Down Reasoning Procedure with Variables

• Function Symbols

• Proof Procedures

Robot Delivery KB

Example: Robot Delivery
Example

Depth-first Search

- Choose a clause in KB whose head matches
- Always select first atom in body
- Choose a clause in KB whose head matches
- Select an atom in body of
- Choice points

Reasoning Procedure

- Good procedure was incomplete
- Search space is large so need to search carefully
- Needs to be done thoughtfully
- Requires the nondeterminism of proof procedure
- Requires procedure (Not in chapter 2)

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 space is too large)
Usefulness of Function Symbols

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

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
  - Functions refer to something in the domain
  - Constants refer to something in the domain
  - Variables refer to something in the domain
  - Predicate symbols used to assert that something is true or false
Semantics of Function Symbols

• $\phi$ used to just map constants to objects in the domain

• $\phi$ also maps $n$-ary function $f$ to $D^n \to D$

- 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

- Notice that its defined so that $\phi(f)(D^n) = D$

- Also maps any function to an object in the domain

- Need to just map constants to objects in the domain

Interpretations no longer finite

- One 1-ary function symbol can name an infinite number of objects

- Functions have to be treated as predicates (multivalued function)

Function Syntax in Datalog

• Function symbol is a token starting with lowercase letter

• Term is either a variable, constant or of the form $f(t_1, \ldots, t_n)$

- Where $f$ is a function symbol and the $t_i$'s are terms

- Terms can only appear inside of predicates (arbitrarily nested)

- Terms cannot appear alone in a KB, as part of a body, as part of a head of a clause

Further Usefullness of Function Symbols

• What about lists or sets of individuals

- We could make up a constant symbol for each list

- 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 $\text{list}(a, b)$ or $\text{list}(c, d, e)$

- Instead use function that lets you specify list one element at a time

- $\text{cons}(\text{Peter}, \text{null})$  $\text{cons}(\text{Tim}, \text{null})$

- $\text{null}$ is an empty list

- $\text{cons}(X, L)$ refers to the list whose first element is $X$ and the rest of the list as $L$
Clauses about Trees

- has leaf \((L, T)\) is true if \(L\) is the label of a leaf in tree \(T\).
- has leaf \((L, l(L))\).
- has leaf \((L, \text{node}(N, LT, RT))\) ← has leaf \((L, LT)\).
- has leaf \((L, \text{node}(N, LT, RT))\) ← has leaf \((L, RT)\).

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 \(l(\text{Name})\).
- Example:
  ```
  node(n1, node(n2, l(l1), l(l2)), node(n3, l(l3), node(n4, l(l4), l(l5))))
  ```

Defining Functions

- Any knowledge about functions must be defined by clauses.
- What knowledge of numbers might we want?
- What knowledge of lists might we want?
Top Down Proof Procedure

Can any substitution be put into normal form?

\{ V/X \} \Leftarrow \{ X/Z'Z/X \}
\{ V/Z'Z/X \} \Leftarrow \{ V/Z'Z/X \}

Also substitutions can be put into normal form

- A substitution is in normal form if no \( \alpha \) appears in any \( \beta \)
- Each \( \alpha \) is a distinct variable and each \( \beta \) is a term
- Substitution is a finite set of the form \{ \( \alpha \) / \( \beta \) \} (1/4)

Need to be careful about normal form

Works with function symbols

Just have to make sure procedure that determines MGU

Bottom-Up Proof Procedure with Variables

Top-Down Proof Procedure

Function Symbols

Top-Down Reasoning Procedure

Top-Down Proof Procedure with Variables

Overview
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
  - Otherwise, if one has constant \( c \), other must as well
  - Otherwise, both are functions, and make sure they unify (recursively)
- Otherwise, one or the other must be well-formed
  - A variable \( A \) should not occur in substitution once (occurs check)
  - Apply \( V/t \) to rest of both expressions and to any terms in substitution
  - If one has \( A \) and other has \( B \), add \( A/B \) to substitution
  - If both terms are same variable, don't need to do anything
- For each term of predicates and functions
  - If one has \( -A \) and other has \( A \), don't need to do anything
  - Otherwise, if one has constant \( c \), other must as well
  - Otherwise, both are functions, and make sure they unify (recursively)
- The two expressions (no variables in common)

Normal Form of Substitutions

- Computes the longest possible normal form
  - For this normal form to be unique, we must have an example of an unsound inference
  - The other normal form, we don't get confused
  - Does follow from \( Y \)
  - (\( X \), \( X \)) \( \rightarrow (\{X\}/Y, \{X\}/Y) \)
  - (\( X \), \( X \)) \( \rightarrow Y \)
  - Consider \( A \)
  - What would this substitution mean?
  - So normal form not cascade
  - 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 l₄ is a leaf of n(n₁, n(n₂, l(l₁), l(l₂)), n(n₃, l(l₃), l(n₄, l(l₄), l(l₅)))), yes ← has leaf(l₄,n(n₃,l(l₃),n(n₄,l(l₄),l(l₅))))).

1st clause in KB does not unify
2nd clause in KB fails
3rd clause in KB unifies yes ← has leaf(l₄,l(15)).

1st clause in KB does not unify
2nd clause in KB unifies
3rd clause in KB unifies yes ← has leaf(l₄,n(n₄,l(l₄),l(15))).

1st clause in KB does not unify.
2nd clause in KB unifies. yes ← has leaf(l₄,l(l₄)).
Final Word on Functions

Functions let you refer to things without having explicit names.

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

Summary of Proof

Yes ← has leaf (l₄, n₁(n₁, l₁, l₂)), n₃(n₄, l₄, l₅)).

1st clause in KB does not unify.
2nd clause in KB unifies.

Yes ← has leaf (l₄, l₁(l₄)).

A 1st clause in KB does not unify.
2nd clause in KB unifies.

Yes ← has leaf (l₄, l₅).

B No clause in KB unifies. Backtrack to B.
2nd clause in KB fails.

3rd clause in KB unifies.

Yes ← has leaf (l₄, l₁(l₄)).

C No clause in KB unifies. Backtrack to C.
2nd clause in KB fails.

1st clause in KB does not unify.
2nd clause in KB unifies.

Yes ← has leaf (l₄, l₅(l₄)).

D 1st clause in KB does.

Yes ← has leaf (l₄, l₄(l₄)).