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. Could do proof procedure on all ground instances of the clauses must be true in the interpretation for any variable assignment. In order for a clause to be true for an interpretation.

Example

- Method is complete and sound for finite ground spaces
- Only a finite number of variable assignments to stop
- In no constant KB, one (just one) needs to be invented
- Includes all clauses in KB and in query

Example KB

When we add function symbols, only way to express an infinite amount of knowledge without having to specify a lot of extra facts.

Variables in KB useful for expressing knowledge.

Example KB

Variables in Clauses

When we add function symbols, only way to express an infinite amount of knowledge without having to specify a lot of extra facts.

Variables in KB useful for expressing knowledge.

Example KB

Variables in Clauses

Top-Down Reasoning Procedure

Top-Down Reasoning Procedure

Top-Down Reasoning Procedure

Function Symbols

Top-Down Reasoning Procedure

Top-Down Reasoning Procedure

Top-Down Reasoning Procedure with Variables

Overview
Examples

\[
\{a/Z \cdot X/X\} \ (Z \cdot X/X)^{d \rightarrow (X \cdot d)}
\]

\[
\{a/Z \cdot X/X\} \ (Z \cdot X/X)^{d \rightarrow (X \cdot d)}
\]

\[
\{a/Z \cdot X/X\} \ (X/a)^{d \rightarrow (X \cdot a)}
\]

\[
\{a/Z \cdot X/X\} \ (X/a)^{d \rightarrow (X \cdot a)}
\]

\[
\{a/Z \cdot X/X\} \ (X/a)^{d \rightarrow (X \cdot a)}
\]

\[
\{a/Z \cdot X/X\} \ (X/a)^{d \rightarrow (X \cdot a)}
\]

\[
\{a/Z \cdot X/X\} \ (X/a)^{d \rightarrow (X \cdot a)}
\]

Substitution

- If \( e \) is ground then \( e \) is called a ground instance of \( e \).
- A substitution is in normal form if no \( V_i \) appears in any \( t_j \).
- Each \( V_i \) is a distinct variable and each \( t_i \) is a term.
- Substitution is a finite set of the form \( \{V_1/t_1, \ldots, V_n/t_n\} \).

Need Alternative

- Number of ground instances of clauses could be huge.
- If 100 constants in KB & Query, will be \( 100^{5} \) instances.
- Has 5 variables: Room, Now, Prev, Diff, Time.
- Need proof procedure to directly handle clauses with variables.

Example

- Has 5 variables: Room, Now, Prev, Diff, Time.
- Need proof procedure to directly handle clauses with variables.
Overview

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

Most General Unifier

- A 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.

- Example: $\{X \mapsto a, Y \mapsto b\}$ is a unifier of $t(a, Y, c)$ and $t(X, b, c)$.
- If $\sigma$ is a unifier of $e$ and $\tau$ is a substitution of $e$ and $\gamma$ is the same as $\sigma\gamma$, then $\sigma\gamma$ is a unifier of $e$ and $\tau$.

Which is best?

Expressions result from applying unifiers.

- Example: $\{X \mapsto Z, Y \mapsto Z\}$ is an MGU resulting in $p(Z, Z)$.
- $\{Y \mapsto X, Z \mapsto X\}$ is an MGU resulting in $p(X, X)$.
Derivation

• Sequence of $\gamma_0, \gamma_1, ..., \gamma_n$
  • $\gamma_0$ 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 $\text{yes}(t_1, ..., t_k) \leftarrow$.

Definite Resolution with Variables

• Generalized answer clause
  - $\text{yes}(t_1, ..., t_k) \leftarrow a_1 \land ... \land a_m$
• Resolution Rule
  \[
  \text{yes}(t_1, ..., t_k) \leftarrow a_1 \land ... \land a_i \leftarrow b_1 \land ... \land b_p
  \]
  \[
  \text{yes}(t_1, ..., t_k) \leftarrow a_1 \land ... \land a_{i+1} \land ... \land a_m
  \]
  - Where $\theta$ is the most general unifier of $a_i$ and $a_i$

Top-down Proof Procedure Recap

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

Specification of a proof procedure
Overview

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

Robot Delivery KB

Example: Robot Delivery
Example

A: Use two east(E,W) ← east(E,M) ∧ east(M,W) yes(1)

B: Use east(E,W) ← west(W,E) yes(3) ← west(4,3) ∧ east(4,r107)

C: Use west(r103,r105) yes(r105) ← east(r105,r107)

D: Use two east(E,W) ← two east(E,W) yes(0) ← two east(0,r107)

Nothing unifies with west(r107,r101). Backtrack to D

Nothing else unifies with east(r105,r107). Backtrack to C

C: Use west(r109,r111) yes(r111) ← east(r109,r107)

D: Use east(E,W) ← west(W,E) yes(r111) ← west(r107,r109)

E: Use west(r107,r109) yes(r111) ← west(r107,r109)

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
  - Needs to be done thoughtfully

Search 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
  - Can talk about objects in the domain without having a constant symbol for them

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
  - Function symbol could also be mapped to Mary
  - Function symbol could also be mapped to John
  - Constant may also be mapped to Mary
  - Variables refer to something in the domain
  - Constants refer to something in the domain

Predicate symbols used to assert that something is true or false

Overview

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

Interpretations no longer finite

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

- Interpreted as a function symbol that can map to elements of \( D \)

- Interpreted as a function symbol that can map to elements of \( D \)

- Interpreted as a function symbol that can map to elements of \( D \)

- Interpreted as a function symbol that can map to elements of \( D \)

Further Usefulness 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 use functions to refer to a list by referring to its elements

- Functions have to have a fixed number of arguments

- So cannot use \( \text{list}(a, b) \), \( \text{list}(c, d, e) \)

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

Further Syntax in Datalog

• Function symbol is a token starting with a lowercase letter

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

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

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

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

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

Function Symbols in Datalog

- Cons is a function symbol for the list whose first element is \( X \) and the rest of the list is \( L \)

- Cons can only take two arguments

- Cons takes two arguments

- Cons takes two arguments

- Cons takes two arguments

When...
Clauses about Trees

- has leaf \((L, T)\) is true if \(L\) is the label of a leaf in \(T\).
- 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

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

Defining Functions

- Can use function symbols to build other data structures.
- What knowledge of lists might we want?
- Any knowledge about functions must be defined by clauses.
Top Down Proof Procedure

- Just have to make sure procedure that determines MGU

Need to be careful about normal form
- A substitution is in normal form if no variable appears in any term
  - Each variable is a distinct variable and each term is a literal
  - A substitution is a finite set of the form
    \[ \{ V_1/X, \ldots, V_n/X \} \]
    - 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?

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
  - But it is countable

But function symbols cause infinite number of terms
- There is a way to enumerate all terms
- But it is countable

Overview

- Top-Down Reasoning Procedure
  - Top-Down Reasoning Procedure
  - Top-Down Reasoning Procedure
  - Top-Down Reasoning Procedure
  - Top-Down Reasoning Procedure
Examples

- Otherwise, both are functions and make sure they unify recursively.
- Otherwise, if one is constant, other must be as well.
- Variables should now only be in substitution once (no free variables).
- Apply $f$ to each expression and add each in substitution the scope of.
- Apply the unification rule (can't change the scope).
- If one has function, other must have same number of function terms.
- If both terms are same variable, don't need to do anything.
- For each term of predicates and functions
- If one has $n$-ary symbol, other must as well.
- If one has a connector, other must have same one.
- Otherwise, if one has constant, other must as well.
- Otherwise, both are functions, and make sure they unify recursively.

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.
  - $V$ variable 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 recursively.

Normal Form of Substitutions

- A substitution is called occurs check.
- Checking for this is called occurs check.
- Does the following from $f$ follow from $B$?
- Does the following from $x$ follow from $B$?
- Does the following from $X$ follow from $B$?
- Does this substitution give same
- Does this make them the same
- Does this give us no new variables we didn't have before?
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 \(n(n_1, n(n_2, l(l_1), l(l_2)), n(n_3, l(l_3), n(n_4, l(l_4), l(l_5))))\).

  - yes ← 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 yes ← 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 yes ← has leaf \((l_4, l(l_5))\).  

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

  1st clause in KB does not unify.
Final Word on Functions

- Makes it easy to reason about parts of the subtree by named matching
- Just do deferred symbol matching
- 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

Summary of Proof

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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