Handling Variables

In order for a clause to be true for an interpretation, the 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:

$q(a)\land q(b)\land r(a)\land s(W)\leftarrow r(W).$

Variables in Clauses

Variables in KB useful for expressing knowledge:
- Can derive parent and grandparent from father and mother.
- Variables in KB useful for expressing knowledge:

VARIABLES

Example KB

Top-down Reasoning Procedure

Formal Procedures

Function Symbols

Top-down Procedural with Variables
Examples

\[
\begin{align*}
\{v/Z',X/X\} & \quad \{(Z' \land X' \land Z' \land X) \land \neg \neg (X' \land X)\} \\
\{v/A'/X\} & \quad \{(A' \land X) \land \neg \neg (X' \land X)\} \\
\{X/Z',0/X\} & \quad \{(X' \land 0) \land \neg \neg (X' \land X)\} \\
\{v/X\} & \quad \{(X' \land X) \land \neg \neg (X' \land X)\} \\
\{v/X'\} & \quad \{(X' \land X') \land \neg \neg (X' \land X)\}
\end{align*}
\]

Substitution

- A 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 \(t_j\).
- \(\{v/Z',X/X\}\) is not in normal form, but \(\{v/A'/X\}\) is.
- \(\{v/Z'/0/X\}\) is a substitution of \(X'\) in expression \(X' \land X\).
Overview

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

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 $\sigma'$ giving $e'$, then $e'$ is an instance of $e$.
  - If two expressions can be unified, they will have an MGU.
  - Could be more than one.

- 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 $p(X, X)$

Unifiers

- Substitution $\sigma$ is a unifier of expressions $e_1$ and $e_2$ if $e_1\sigma = e_2\sigma$.
  - Expressions resulting from applying $\sigma$ to each occurrence of each operator.
  - They are both instances of each other.
  - Expression $e_1$ is renaming of $e_2$ if they differ only in names of variables.
  - Could be more than one.

- Example:
  - $\{X/a, Y/b\}$ is a unifier of $t(a, Y, c)$ and $t(X, b, c)$.

Lifters

- Expressions have many unifiers.
  - Example: $(x/\lambda y.x)$ and $(y/\lambda y.y)$ are unifiers of $(z/\lambda y.z)$.
  - If $\sigma$ is a unifier of $e_1$ and $e_2$, then $\sigma$ is in the set of all substitutions that are a unifier of $e_1$ and $e_2$.
Derivation

- \( \gamma_0 \) 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
  - Select an atom in body of \( \gamma_i \)
  - 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
  \[
  \text{yes} \left( t_1, \ldots, t_k \right) \leftarrow .
  \]

Definite Resolution with Variables

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

Top-down Proof Procedure Recap

- \( \text{Start with goal, work toward facts in KB} \)
- \( \text{Definite Clause Resolution for Ground Case} \)
  \[
  \text{yes} \left( t_1, \ldots, t_k \right) \leftarrow a_1 \land \ldots \land a_m
  \]
  \[
  a_i \leftarrow b_1 \land \ldots \land b_p
  \]
  \[
  \text{yes} \left( t_1, \ldots, t_k \right) \leftarrow a_1 \land \ldots \land a_{i-1} b_1 \land \ldots \land b_p a_{i+1} \land \ldots \land a_m
  \]
  \[\theta\]
  Definite Clause Resolution for Ground Case
  \[
  \text{Start with goal, work toward facts in KB} \]
Overview

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

Robot Delivery KB

Example: Robot Delivery
Further Usefulness of Function Symbols

• What about lists or sets of individuals?
  - We could make up a constant symbol for each list:
    - has member(lista, peter)
    - has member(lista, tim)
  - But an 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) list(c, d, e).
  - Instead use function that lets you specify 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 as L.

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 1:15pm.
    - Just need 60 constant symbols rather than 24*60.
  - Can talk about objects in the domain without having a constant symbol.

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.
Defining Functions

Any knowledge about functions must be defined by clauses

Simple Examples

\[ \text{before}(am(H_1, M_1), \text{am}(H_2, M_2)) \rightarrow H_2 < 12 \]
\[ \text{before}(am(H_1, M_1), \text{am}(H_2, M_2)) \rightarrow H_1 < H_2 \land H_2 < 12 \]
\[ \text{before}(am(H, M_1), \text{am}(H, M_2)) \rightarrow M_1 < M_2 \]

\[ \text{before}(pm(12, M_1), \text{pm}(H_2, M_2)) \rightarrow H_2 < 12 \]
\[ \text{before}(pm(H_1, M_1), \text{pm}(H_2, M_2)) \rightarrow H_1 < H_2 \land H_2 < 12 \]
\[ \text{before}(pm(H, M_1), \text{pm}(H, M_2)) \rightarrow M_1 < M_2 \]

Semantics of Function Symbols

\( \varphi \) used to just map constants to objects in the domain
\( \varphi \) also maps n-ary function symbol \( f \) 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

Interpretations no longer finite

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

For example, \( + \) can only be declared in the domain that includes the constant

Notice that it is declared as a predicate, not a constant

\( \varphi \) also maps a function symbol to \( D^n \rightarrow D \)-

\( \cdot \) maps to \( D \)

Terms can only appear inside of predicates (unary only needed)

Where \( f \) is a function symbol then the \( f \) is a term

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

Function Symbols in Datalog

---
Overview

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

Clauses about Trees

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

Building Data Structures

• Can use function symbols to build other data structures

Example:

\[
\text{node}(n_1, \text{node}(n_2, \text{leaf}(l_1), \text{leaf}(l_2)), \text{node}(n_3, \text{leaf}(l_3), \text{node}(n_4, \text{leaf}(l_4), \text{leaf}(l_5))))
\]
Normal Form of Substitutions

• \{X/f(X)\} cannot be put into normal form.
• What is normal form too restrictive?
  • Consider KB = lt(X, s(X))
    lt(X, s(Y)) ← lt(X, Y).
  • Does lt(X, X) follow from KB?
  • Does lt(X, s(X)) unify with lt(X, X)?
  + Note we made up new variables so we don't get confused
  • The unifier \{X_1/X, X/s(X)\}
  + 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 occurs check

Top-Down Proof Procedure
• Just have to make sure procedure that determines MGU
  works with function symbols
  • Need to be careful about normal form
    • Most substitutions can be put into normal form
      \{X/Z, Z/a\} ⇒ \{X/a, Z/a\}
      \{X/Z, Z/X\} ⇒ \{X/Z\}
    • Can any substitution be put into normal form?
      • What about \{X/f(X)\}?

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 have to make sure procedure that determines MGU
Overview

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

Examples

$p(X, Y)$ and $p(Z, Z)$

$p(X, X)$ and $p(f(A, c), B)$

$p(X, X)$ and $p(B, f(A, B))$

```
((g' \vee f' g) d \text{ and } (X' X) d)
```

```
((\neg \vee f' g) d \text{ and } (X' X) d)
```

```
((g' (\neg \vee) f') d \text{ and } (X' X) d)
```

```
(Z' Z') d \text{ and } (A' X') d
```

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, the other must have the same one
- If one has an $n$-ary symbol, the other must as well
- For each term of predicates and functions:
  - If one has a variable, the other must as well
  - If one has a constant, the other must as well
  - If one has a connector, the other must have the same
- For each pair of predicates and functions (in variables in common):

```
\text{let } \mathbf{e}_1 \text{ and } \mathbf{e}_2 \text{ be expressions}
```

- Algorithm for Finding MGU (Not in textbook)
Depth-first Search

- Choose first clause in KB where head matches
- We will error on consistent derivation of goal with that head
- Always select first atom in body
- Choose a clause in KB whose head unifies with the chosen atom
- Select the clause in body of $i-1$
- Choose points

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)

Top-Down Proof Procedure (Repeat)

- 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
- Select 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 $\text{yes}(t_1, ..., t_k) \leftarrow$. Lot's of Choice Points / Nondeterminism
- Functions let you refer to things without 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.

---

**Final Word on Functions**

**Summary of Proof**

- Prove $l_{4}$ is a leaf of $n(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))))).

1. First clause in KB does not unify.
2. Second clause in KB unifies.

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

- First clause in KB does not unify.
- Second clause in KB unifies.

   - $\text{has leaf}(l_4, l(l_1)).$

- Third clause in KB unifies.

   - $\text{has leaf}(l_4, n(n_4, l(l_4), l(15))).$

   - First clause in KB does not unify.
   - Second clause in KB unifies.

   - $\text{has leaf}(l_4, l(l_4)).$

   - First clause in KB does.

---

**Example Proof with Functions**

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

- $\text{has leaf}(L, n(N, LT, RT))$.

- $\text{has leaf}(L, n(N, LT, RT)) \leftarrow \text{has leaf}(L, LT).$

- $\text{has leaf}(L, n(N, LT, RT)) \leftarrow \text{has leaf}(L, RT).$

---

**CSE560 Class 04**

P. Heeman, 2010