
Overview

⇒ Variables

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

Variables in Clauses

- Variables in KB useful for expressing facts

- Example KB

<i>father(tim,steve)</i>	<i>father(steve,john)</i>
<i>mother(pam,john)</i>	<i>mother(susan,pam)</i>
<i>mother(helen,steve)</i>	<i>motherpaula,tim</i>
<i>parent(X,Y) ← father(X,Y)</i>	
<i>parent(X,Y) ← mother(X,Y)</i>	
<i>grandparent(X,Y) ← parent(X,Z) ∧ parent(Z,Y)</i>	

- We can derive *parent* and *grandparent* from *father* relations, and hence do not have to duplicate all of the information
- More important, when there are an infinite number of clauses that are true

Room Example

- KB

hasdetector(foyer)

hasdetector parlour)

$room(X) \leftarrow hasdetector(X)$

- Here, don't have to explicitly say which things are rooms

Explained Motion Example

- If we see motion in a room within a certain amount of time of previous motion in that room, it's fine

$explained(Room, Now) \leftarrow hasdetector(Room)$

$\wedge lastmotion(Room, Prev)$

$\wedge subtract(Now, Prev, Diff)$

$\wedge motionlessinroom(Room, Time)$

$\wedge less(Diff, Time)$

- $subtract(X, Y, Z)$ is true for all X, Y, Z where $Z = X - Y$

- Notice that we are representing time as the number of seconds from some fixed point, so we actually have an infinite number of constants, and hence an infinite number of clauses that are true

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 still guaranteed to stop
 - Method is complete and sound for proving ground atoms

- Example

$$\begin{aligned} & q(a). \\ & q(b). \\ & r(a). \\ & s(W) \leftarrow r(W). \\ & p(X, Y) \leftarrow q(X) \wedge s(Y). \end{aligned}$$

Need Alternative

- Number of ground instances of clauses could be huge
- Example
 - $explained(Room, Now) \leftarrow hasdetector(Room)$
 - $\wedge lastmotion(Room, Prev)$
 - $\wedge subtract(Now, Prev, Diff)$
 - $\wedge motionlessinroom(Room, Time)$
 - $\wedge less(Diff, Time)$
- 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 $\{V_1/t_1, \dots, V_n/t_n\}$
 - 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 $\sigma = \{V_1/t_1, \dots, V_n/t_n\}$ 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 σ 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 $p(Z, Z)$

Most General Unifier

- *Most General Unifier (MGU)*
 - If σ 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
- 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

$$\frac{\begin{array}{l} \text{yes} \leftarrow a_1 \wedge \dots \wedge a_n \\ a_i \leftarrow b_1 \wedge \dots \wedge b_p \end{array}}{\text{yes} \leftarrow a_1 \wedge \dots \wedge a_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge a_{i+1} \wedge \dots \wedge a_n}$$

Definite Resolution with Variables

- Generalized answer clause
 - $yes(t_1, \dots, t_k) \leftarrow a_1 \wedge \dots \wedge a_m$

- Resolution Rule

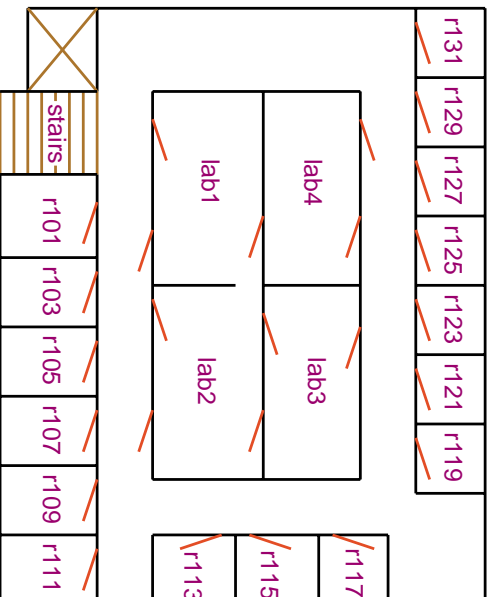
$$\frac{yes(t_1, \dots, t_k) \leftarrow a_1 \wedge \dots \wedge a_m \quad a \leftarrow b_1 \wedge \dots \wedge b_p}{(yes(t_1, \dots, t_k) \leftarrow a_1 \wedge \dots \wedge a_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge a_{i+1} \wedge \dots \wedge a_m) \theta}$$

- Where θ is the most general unifier of a and a_i

Derivation

- Sequence of $\gamma_0, \gamma_1, \dots, \gamma_n$
- γ_0 is answer clause corresponding to original query
- γ_i obtained by
 - Select an atom in body of γ_{i-1}
 - Choose a clause in KB whose head with unify with the chosen atom
 - Make a copy of it with variables that have not been used before
 - + This ensures the clause does not use the same variables as already used in γ_{i-1}
 - Resolve γ_{i-1} with copy of clause
- γ_n is an answer, and so is of the form $yes(t_1, \dots, t_k) \leftarrow$.

Example: Robot Delivery



Robot Delivery KB

```

imm_west(r101, r103).
imm_west(r103, r105).
imm_west(r105, r107).
imm_west(r107, r109).
imm_west(r109, r111).
imm_west(r131, r129).
imm_west(r129, r127).
imm_west(r127, r125).
imm_east(E, W) ← imm_west(W, E).
next_door(E, W) ← imm_east(E, W).
next_door(W, E) ← imm_west(W, E).
two_doors_east(E, W) ← imm_east(E, M) ∧ imm_east(M, W).
west(W, E) ← imm_west(W, E).
west(W, E) ← imm_west(W, M) ∧ west(M, E).
?two_doors_east(R, r107)

```

Overview

- Variables
- Top-down Proof Procedure with Variables
 - ⇒ 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

Usefulness of Function Symbols

- Can talk about 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
- Might want to reason about lists or sets of individuals
 - With no functions, need to create constant symbol for each list
 - has_member(lista, peter)*
 - has_member(lista, tim)*
 - With functions, can refer to it by referring to its elements
 - cons(peter, cons(tim, null))*
 - + *null* is an empty list
 - + *cons(X;L)* refers to the list with first element *X* and the rest of the list as *L*

Function Syntax in Datalog

- *Function symbol* is a token starting with lowercase letter
- *Term* is either a variable, constant or of the form $f(t_1, \dots, t_n)$
 - 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

- ϕ used to just map constants to objects in the domain
- ϕ also maps n-ary function f to $D^n \rightarrow D$
 - Notice that it is defined as mapping D^n to D , not constantsⁿ
 - 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: $0, s(0), s(s(0)), s(s(s(0))), \dots$

Defining Functions

- Any knowledge about functions must be defined by clauses
- Time
 - $before(am(H1,M1),pm(H2,M2))$
 - $before(am(I2,M1),am(H2,M2)) \leftarrow H2 < I2$
 - $before(am(H1,M1),am(H2,M2)) \leftarrow H1 < H2 \wedge H2 < I2$
 - $before(am(H,M1),am(H,M2)) \leftarrow M1 < M2$
 - $before(pm(I2,M1),pm(H2,M2)) \leftarrow H2 < I2$
 - $before(pm(H1,M1),pm(H2,M2)) \leftarrow H1 < H2 \wedge H2 < I2$
 - $before(pm(H,M1),pm(H,M2)) \leftarrow M1 < M2$

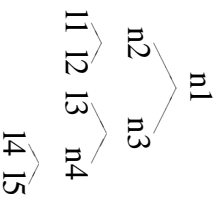
Building Data Structures

- Can use function symbols to build data structures
- Tree data structure:

- A labeled tree is either a node $node(Name, LeftTree, RightTree)$ or a leaf $l(Name)$

- Example:

$node(n1, node(n2, l(l1), l(l2)), node(n3, l(l3), node(n4, l(l4), l(l5))))$



Clauses about Trees

- $has_leaf(L, T)$ is true if L is the label of a leaf in tree T
 $has_leaf(L, l(L))$.
 $has_leaf(L, node(N, LT, RT)) \leftarrow has_leaf(L, LT)$.
 $has_leaf(L, node(N, LT, RT)) \leftarrow has_leaf(L, RT)$.

Overview

- Variables
- Top-down Proof Procedure with Variables
- 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 *fairly* introduces ground instances

~~1/1~~ ~~2/1~~ ~~3/1~~ 4/1 ...

~~1/2~~ ~~2/2~~ 3/2 4/2 ...

~~1/3~~ 2/3 3/3 4/3 ...

1/4 2/4 3/4 4/4 ...

... ..

Top Down Proof Procedure

- Just have to make sure procedure to find MGU works with function symbols
- Need to be careful about normal form
 - *Substitution* is a finite set of the form $\{V_1/t_1, \dots, V_n/t_n\}$
 - 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?
- 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
 - The unifier $\{X_1/X, X/s(X)\}$ 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 *occurs check*

Algorithm for Finding MGU (Not in textbook)

- Take two expressions
 - Should not have any 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 the 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
 - + Recursive definition

Examples

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

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

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

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

Overview

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

Top-Down Proof Procedure (Repeat)

- Sequence of $\gamma_0, \gamma_1, \dots, \gamma_n$
 - γ_0 is answer clause corresponding to original query
 - γ_i obtained by
 - **Select** an atom in body of γ_{i-1}
 - **Choose** clause in KB whose head will unify with the chosen atom
 - + Ensures the clause does not use the same variables as already used in γ_{i-1}
 - Resolve γ_{i-1} with copy of clause
 - γ_n is an answer, and so is of the form $yes(t_1, \dots, t_k) \leftarrow$.
- ⇒ Lots of Choice Points / Nondeterminism

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 γ_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 anyways, 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

Example Proof with Functions

- Defined $has_leaf(L, T)$ as true if L is label of leaf in tree T
 - $has_leaf(L, l(L))$.
 - $has_leaf(L, n(N, LT, RT)) \leftarrow has_leaf(L, LT)$.
 - $has_leaf(L, n(N, LT, RT)) \leftarrow has_leaf(L, RT)$.
- Prove $l4$ is a leaf of $n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5))))$
 - $yes \leftarrow has_leaf(l4, n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5))))$.
 - $yes \leftarrow has_leaf(l4, n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5))))$.
 - $Ist\ clause\ in\ KB\ does\ not\ unify$
 - $2nd\ clause\ in\ KB\ fails$ A
 - $3rd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, n(n3, l(l3), n(n4, l(l4), l(l5))))$.
 - $Ist\ clause\ in\ KB\ does\ not\ unify$
 - $2nd\ clause\ in\ KB\ fails\ yes \leftarrow has_leaf(l4, l(l3))$ C
 - $3rd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, n(n4, l(l4), l(l5)))$.
 - $Ist\ clause\ in\ KB\ does\ not\ unify$.
 - $2nd\ clause\ in\ KB\ unifies:\ yes \leftarrow has_leaf(l4, l(l4))$ D
 - $Ist\ clause\ in\ KB\ does:\ yes \leftarrow$.

Summary of Proof

- $yes \leftarrow has_leaf(l4, n(n1, n(n2, l(l1), l(l2)), n(n3, l(l3), n(n4, l(l4), l(l5))))$.
- $Ist\ clause\ in\ KB\ does\ not\ unify$
- $2nd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, n(n1, l(l1), l(l2)))$ A
- $Ist\ clause\ in\ KB\ does\ not\ unify$
- $2nd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, l(l1))$ B
- $No\ clause\ in\ KB\ unifies:\ Backtrack\ to\ B$.
- $2nd\ clause\ in\ KB\ fails$ B
- $3rd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, l(l2))$ B
- $No\ clause\ in\ KB\ unifies:\ Backtrack\ to\ A$.
- $2nd\ clause\ in\ KB\ fails$ A
- $3rd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, n(n3, l(l3), n(n4, l(l4), l(l5))))$.
- $Ist\ clause\ in\ KB\ does\ not\ unify$
- $2nd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, l(l3))$ C
- $No\ clause\ unifies:\ Backtrack\ to\ C$.
- $2nd\ clause\ in\ KB\ fails\ yes \leftarrow has_leaf(l4, l(l3))$ C
- $3rd\ clause\ in\ KB\ unifies\ yes \leftarrow has_leaf(l4, n(n4, l(l4), l(l5)))$.
- $Ist\ clause\ in\ KB\ does\ not\ unify$.
- $2nd\ clause\ in\ KB\ unifies:\ yes \leftarrow has_leaf(l4, l(l4))$ D
- $Ist\ clause\ in\ KB\ does:\ yes \leftarrow$ D

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