Overview

MIDTERM: Friday 10/18 Project: Monday 10/21 In Class

- Substitution
- Unification algorithm
- Unification in LISP
- Factors
- Resolvents

Quantifier Equivalences Revisited

Unification in LISP

Standard Form: A Summary

We followed these three steps to convert first-order logic formulas into a standard form amenable to algorithmic verification:

1. Transform formula into Prenex Normal Form.
2. Transform the matrix into Conjunctive Normal Form.
3. Eliminate existential quantifiers through Skolemization.

⇒ A set of clauses in CNF in which all variables are universally quantified.
Resolution for Predicate Calculus

The resolution step is valid for predicate calculus, when two clauses contain complementary predicates. For example, clause $C_1$ may contain predicate $P(x)$ and clause $C_2$ may contain predicate $\neg P(x)$.

$$C_1 : P(x) \lor Q(x)$$
$$C_2 : \neg P(f(x)) \lor R(x)$$

We could substitute $f(\alpha)$ for $x$ in $C_1$ and $\alpha$ for $x$ in $C_2$, and then resolve to get

$$C_3 : Q(f(\alpha)) \lor R(\alpha)$$

More generally, we could substitute $f(x)$ for $x$ in $C_1$ and resolve to get

$$C_3 : Q(f(x)) \lor R(x)$$

Remaining Issues

Theorem proving steps:

1. Conversion of natural language sentences into first-order logic formulas
2. Conversion to standard form
3. Resolution

Remaining issue: how to substitute variables to resolve two clauses and generate a new clause \( \Rightarrow \) do substitution and unification.

Ground Term

A term (constant, variable, or function of terms) is a **ground term** if no variable appears in the term.

- ground constant
- ground literal
- ground clause
- etc.

Substitution

- A **substitution** is a finite set of the form

$$\{v_1/t_1, \ldots, v_n/t_n\}$$

where each $v_i$ is a variable, each $t_i$ is a term (constant, variable, or function of terms), and no two $v_i$ are identical.

- A substitution in which each $t_i$ is a ground term is called **ground substitution**.

- The **empty substitution** \( \epsilon = \{\} \) contains no elements.

**Why is substitution important**: assists in resolving two clauses by making the two clauses with different variables compatible.
Substitution Applied to a Formula

- Let $\theta = \{v_1/t_1, ..., v_n/t_n\}$ be a substitution and $E$ be an expression. Then $E\theta$ is an expression obtained from $E$ by replacing simultaneously each occurrence of variable $v_i$ ($1 \leq i \leq n$) in $E$ by the term $t_i$.
- $E\theta$ is called an instance of $E$.

In the textbook, $E\theta$ is denoted $\text{SUBST}(\theta, E)$.

Substitution Examples

- $\theta = \{x/a, y/f(b), z/c\}$, $E = P(x, y, z)$
  - $E\theta = P(a, f(b), c)$
- $\theta = \{x/f(x), y/x\}$, $E = P(x, y)$
  - $E\theta = P(f(x), x)$
- $\theta = \{x/\text{Socrates}\}$, $E = \lnot \text{MAN}(x) \lor \text{MORTAL}(x)$
  - $E\theta = \lnot \text{MAN}(\text{Socrates}) \lor \text{MORTAL}(\text{Socrates})$

Composition of Substitutions

Let $\theta = \{x_1/t_1, ..., x_n/t_n\}$ and $\lambda = \{y_1/u_1, ..., y_m/u_m\}$ be substitutions. Then the composition of $\theta$ and $\lambda$, denoted $\theta \circ \lambda$ is the substitution obtained from the set

$$\{x_1/t_1 \lambda, ..., x_n/t_n \lambda, y_1/u_1, ..., y_m/u_m\}$$

by deleting any element $x_j/t_j \lambda$ such that $t_j \lambda = x_j$ (e.g. $x_k/x_k$ is meaningless) and any element $y_k/u_i$ such that $y_i \in \{x_1, ..., x_n\}$ (because $y_i$ is already covered by $\theta$).

Examples: Composition of Substitution

Given

$$\begin{align*}
\theta &= \{x_1/t_1, x_2/t_2\} = \{x/f(y), y/z\} \\
\lambda &= \{y_1/u_1, y_2/u_2, y_3/u_3\} = \{x/a, y/b, z/y\}
\end{align*}$$

$$\begin{align*}
\theta \circ \lambda &= \{x_1/t_1 \lambda, x_2/t_2 \lambda, y_1/u_1, y_2/u_2, y_3/u_3\} \\
&= \{x/f(y)\lambda, y/z\lambda, x/a, y/b, z/y\} \\
&= \{x/f(b), \underbrace{y/y}_\text{identity appeared in $\theta$}, x/a, y/b, z/y\} \\
&= \{x/f(b), z/y\}
\end{align*}$$
**Unification**

- A substitution $\theta$ is called a **unifier** for a set $\{E_1, \ldots, E_k\}$ iff $E_1\theta = E_2\theta = \ldots = E_k\theta$.
- The set $\{E_1, \ldots, E_k\}$ is said to be **unifiable** if there is a unifier for it.
- A unifier $\sigma$ for a set $\{E_1, \ldots, E_k\}$ of expressions is a **Most General Unifier** iff for each unifier $\theta$ for the set there is a substitution $\lambda$ such that $\theta = \sigma \circ \lambda$.
- A Most General Unifier will avoid unnecessary substitution(s).

**Examples: Unification**

$P(x, g(x))$ will unify with:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Necessary Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(x, y)$</td>
<td>${y/g(x)}$</td>
</tr>
<tr>
<td>$P(z, g(z))$</td>
<td>${z/x}$ or ${x/z}$</td>
</tr>
<tr>
<td>$P(Socrates, g(Socrates))$</td>
<td>${x/Socrates}$</td>
</tr>
<tr>
<td>$P(x, g(y))$</td>
<td>${x/y}$ or ${y/x}$</td>
</tr>
<tr>
<td>$P(g(y), z)$</td>
<td>${x/g(y), z/g(g(y))}$</td>
</tr>
</tbody>
</table>

but not with $P(Socrates, f(Socrates))$ or $P(g(y), y)$

**Disagreement Set**

Let $W$ be a nonempty set of expressions $\{E_1, \ldots, E_n\}$. The **disagreement set** $D$ of $W$ is obtained by locating the first symbol (counting from the left) at which not all the expressions in $W$ have exactly the same symbol, and then extracting from each expression $E_k$ in $W$ the subexpression that begins with the symbol occupying that position.

Example:

$W = \{P(x, y, a), f(x)), P(x, y, a), g(x)), P(x, y, a)\}$

Symbols to the right of the vertical bar differ.

$D = \{f(x), g(x), z\}$

**Disagreement Set: More Examples**

**Examples:**

1. $W = \{P(a), P(x)\}$  \hspace{1cm} $D = \{a, x\}$

2. $W = \{P(x, f(y, a)), P(x, a), P(x, g(h(k(x))))\}$

   $D = \{f(y, a), a, g(h(k(x)))\}$

3. $W = \{P(x, f(g(h(y)))), P(x, f(g(\overline{z})))\}$

   $D = \{h(y), z\}$
Unification Algorithm

Let $W = \{E_1, \ldots, E_n\}$ be the set of expressions to be unified.

1. If necessary, rename variables so that no pair $(E_i, E_j)$ from different clauses has any variables in common.
2. Set $k = 0$, $W_k = W$, $\sigma_k = \epsilon$ (empty substitution).
3. If $W_k$ is a singleton (contains only one expr), stop; $\sigma_k$ is a most general unifier for $W$. Otherwise, let $D_k$ be the disagreement set for $W_k$.
4. If there exist elements $v_k$ and $t_k$ in $D_k$ such that $v_k$ is a variable that does not occur in term $t_k$, go to step 5. Otherwise, stop; $W$ is not unifiable.
5. Let $\sigma_k+1 = \sigma_k \circ \{v_k/t_k\}$ and $W_{k+1} = W_k \{v_k/t_k\}$. (Note that $W_{k+1} = W_k \sigma_k+1$)
6. Set $k = k + 1$ and go to step 3.

Unification Theorem

If $W$ is a finite nonempty unifiable set of expressions, then the unification algorithm will always terminate at step 3, and the last $\sigma_k$ is a most general unifier for $W$ (i.e. not unnecessary substitutions).

The algorithm must terminate because each pass through the loop reduces the number of variables by 1, and there are only finitely many of them.

Unification Example

$P(x, f(x), z)$ vs.
$P(g(y), f(g(a)), y)$:

1. $\{x/g(y)\}$:
   $P(g(y), f(g(y)), z)$
   $P(g(y), f(g(a)), y)$
2. $\{y/a\}$:
   $P(g(a), f(g(a)), z)$
   $P(g(a), f(g(a)), a)$
3. $\{z/a\}$:
   $P(g(a), f(g(a)), a)$

Unifier: $\{x/g(a), y/a, z/a\}$

Representation of Predicates and Terms in LISP

- Constants: $a = (A)$, Socrates = (SOCRATES)
- Variables: $x = X$, $y = Y$
- Functions: $f(x) = (F \ X)$, $f(a,y,z) = (F \ (A) \ Y \ Z)$
- Predicates: $P(x) = (P \ X)$, $P(x,b,f(z)) = (P \ X \ (B) \ (F \ Z))$

Note how the representation of the constants can come in handy.
**SUBLIS : substitution in LISP**

(sublis <list-of-alist> <expr>): simultaneous substitution

- **alist**, or association list: (A . B), which is the same as (cons 'A 'B) (note that B is not a list but an atom in this case).
- **<list-of-alist>:** a list of (<pattern> <replace>) pairs.
- **<expr>:** the expression to be worked on.
- Replace every occurrence of <pattern> in <expr> with <replace>.

Another useful function: (subst <repl> <pattern> <expr>)

**SUBLIS Examples**

Basically, replace (car alist) with (cdr alist) of each element in the <list-of-alist>:

> (sublis '((x . (20))) '(* x 1))
  (* (20) 1)

> (sublis '((x 20)) '(* x 1))
  (* (20) 1)

> (sublis '((x . 20)) '(* x 1))
  (* 20 1)

> (sublis '((x . 20) (y . 10)) '(* x (/ 5 y)))
  (* 20 (/ 5 10))

**Unification in LISP**

(defun unify (u v)
  (let ((*u* (copy-tree u))
         (*v* (copy-tree v)) *subs*)
    (declare (special *u* *v* *subs*)
             (if (unifyb *u* *v*) (or *subs* (list (cons t t)))))
)

(defun unifyb (u v)
  (cond ((eq u v))
         ((symbolp u) (varunify v u))
         ((symbolp v) (varunify u v))
         ((and (consp u) (consp v)
              (eq (car u) (car v))
              (eql (length (cdr u))
                   (length (cdr v))))
              (every #'unifyb (cdr u) (cdr v))))
)

**Unification in LISP (cont’d)**

(defun varunify (term var)
  (declare (special *u* *v* *subs*)
           (unless (occurs var term)
             (dolist (pair *subs*)
               (setf (cdr pair)
                     (subst term var (cdr pair))))
             (nsubst term var *u*)
             (nsubst term var *v*)
             (push (cons var term) *subs*)))

---

<sup>21</sup>

---

<sup>22</sup>

---

<sup>23</sup>

---

<sup>24</sup>

---

<sup>a</sup> Code in this and the previous page by Gordon Novak, http://www.cs.utexas.edu/users/novak. Also downloadable at http://www.cs.tamu.edu/faculty/choe/courses/02spring/src/sunify.lisp
UNIFY : examples

(uni fy ' (p x) ' (p (a)))
(uni fy ' (p (a)) ' (p x))
(uni fy ' (p x (g x) (g (b))) ' (p (f y) z y))
(uni fy ' (p (g x) (h w) w) ' (p y (h y) (g (a))))
(uni fy ' (p (f x) (g (f (a))) x) ' (p y (g y) (b)))
(uni fy ' (p x) ' (p (a) (b)))
(uni fy ' (p x (f x)) ' (p (f y) y))

Resolution in Predicate Calculus

- Factors
- Binary resolvent
- Properties of resolution

Factor of a Clause

Definition: If two or more literals of a clause $C$ (with the same sign) have a most general unifier $\sigma$, then $C\sigma$ is called a Factor of $C$. If $C\sigma$ is a unit clause, it is called a Unit Factor of $C$.

Example: $C = P(x) \lor P(f(y)) \lor \neg Q(x)$.

- The first two literals have a unifier $\sigma = \{x/ f(y)\}$, so $C$ has a factor $C\sigma = P(f(y)) \lor \neg Q(f(y))$.

Note: Factors of a clause are much succinct and when two clauses $C_1$ and $C_2$ cannot be resolved directly, their factors (let’s call them $C'_1$ and $C'_2$ can be resolved.

Resolving Two Clauses

Definition: Let $C_1$ and $C_2$ be two clauses (called parent clauses) with no variables in common, and with complementary literals $L_1$ and $L_2$ such that $L_1$ and $\neg L_2$ have a most general unifier $\sigma$. Then the clause $(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$ is called a binary resolvent of $C_1$ and $C_2$. The literals $L_1$ and $L_2$ are called the literals resolved upon.

Note: A clause can be treated as a set of literals.

Example: Resolve the following (hint: $\sigma = \{x/a\}$)

$C_1 = P(x) \lor Q(x)$ and $C_2 = \neg P(a) \lor R(y)$. 
Resolvent

Definition: A resolvent of parent clauses $C_1$ and $C_2$ is one of the following binary resolvents:

1. a binary resolvent of $C_1$ and $C_2$
2. a binary resolvent of $C_1$ and a factor of $C_2$
3. a binary resolvent of a factor of $C_1$ and $C_2$
4. a binary resolvent of a factor of $C_1$ and a factor of $C_2$

Example: resolve the two clauses
1. $C_1 = P(x) \lor P(f(y)) \lor R(g(y))$ and
2. $C_2 = \neg P(f(g(a))) \lor Q(b)$.
(hint: resolve the factor of $C_1$ and clause $C_2$)

Property of Resolution for First-Order Logic

- **Complete**: If a set of clauses $S$ is unsatisfiable, resolution will *eventually* derive $\mathbf{F}$.
  - *Everything that is true can be proved (eventually).*

- **Sound**: If $\mathbf{F}$ is derived by resolution, then the original set of clauses $S$ is unsatisfiable.
  - *Everything that is proved is true.*

Weakness of Resolution

Basically, resolution tries to derive

$$\text{Axioms} \land \neg \text{Theorem} = \mathbf{F}$$

- Is there a $\mathbf{F}$ in the axioms? If there is, the whole formula will always be unsatisfiable no matter what.
- Can we tell whether axioms alone can derive $\mathbf{F}$? (generally, this is not the case)

Key Points

- substitution and unification: why are these necessary and how to do them.
- unification algorithm
- factors: definition, and how to derive, why factors are important
- resolvent: definition, and how to derive
Next Time

- Resolution: a full example
- Automating resolution: various strategies
- Uncertainty: chapter 14