Resolution and Refutation

York University CSE 3401
Vida Movahedi
Overview

• Propositional Logic
  – Resolution
  – Refutation

• Predicate Logic
  – Substitution
  – Unification
  – Resolution
  – Refutation
  – Search space

[ref.: Nilsson- Chap.3]
[also Prof. Zbigniew Stachniak’s notes]
Theorems from Logic

[from Mathematical Logic, George Tourlakis]

• Modus Ponens

\[ A, A \rightarrow B \vdash B \]

• Cut Rule

\[ A \lor B, \neg A \lor C \vdash B \lor C \]

\[ A, \neg A \vdash \bot \]

• Transitivity of \( \rightarrow \)

\[ A \rightarrow B, B \rightarrow C \vdash A \rightarrow C \]

• Proof by Contradiction

\[ \Gamma \vdash A \text{ iff } \Gamma + \neg A \vdash \bot \]
Resolution in Logic

• By A. Robinson (1965)

• Example: Prove

\[ A \rightarrow (B \rightarrow C), A \rightarrow B, A \vdash C \]

• We need to show that the

\[ \{ \neg A \lor \neg B \lor C, \neg A \lor B, A, \neg C \} \]
Resolution in Logic Programming

• Program P (facts and rules in clause form)
• Goal G negated and added to program P
• To prove G, we need to show \( P + \{\neg G\} \) is inconsistent

Complementary literals

\[ \text{q, t} : \neg p, \quad p : \neg s. \]

Parent clauses

Resolving upon p, being on different sides of :-

\[ \text{q, t} : \neg s. \]

Resolvent
Example (1)

- Program \( P = \{ q : -., p : -q. \} \)

- Query :\(-p.\)
  - This is already the negated form of our goal!

\[
\begin{align*}
q : - & \quad p : - q. \quad : - p. \\

&p : - q. \\
&: - \text{ empty clause, inconsistency} \\
&\quad \text{therefore } p \text{ is satisfiable } \rightarrow \text{ true}
\end{align*}
\]
Refutation

• When resolution is used to prove inconsistency, it is called refutation. (refute=disprove)

• The above binary tree, showing resolution and resulting in the empty clause, is called a refutation tree.

• NOTE: To avoid potential mistakes, DO NOT RESOLVE UPON MORE THAN ONE LITERAL SIMULTANEOUSLY.
Example (2)

A1. If Henry has two days off, then if the weather is bad, Henry is not fishing.
A2. If Henry is not fishing and is not drinking in a pub with his friends, then he is watching TV at home.
A3. If Henry is working, then he is neither drinking in a pub with his friends nor watching TV at home.

Q. If Henry is not watching TV at home and he has two days off, then he is drinking in a pub with his friends provided that the weather is bad.
Example (2) (cont.)

• From logical point of view, we want to prove $Q$, given $A_1, A_2, A_3$. 

$$\{ A_1, A_2, A_3 \} \models Q .$$

• By refutation principle, the consistency of 

$$C = \{ A_1, A_2, A_3 \} \cup \{ \neg Q \}$$

is examined.

– Step 1: Represent as propositional formulas
– Step 2: Represent as clauses
– Step 3: Determine the consistency of $C$

  • If $C$ is consistent, answer NO (false)
  • If $C$ is inconsistent, answer YES (true)
A1. If Henry has two days off, then if the weather is bad, Henry is not fishing.

A2. If Henry is not fishing and is not drinking in a pub with his friends, then he is watching TV at home.

A3. If Henry is working, then he is neither drinking in a pub with his friends nor watching TV at home.

Q. If Henry is not watching TV at home and he has two days off, then he is drinking in a pub with his friends provided that the weather is bad.

p: H has two days off
q: weather is bad
r: H is fishing
s: H is drinking in a pub with his friends
t: H is watching TV at home
u: H is working

A1. p -> (q -> ~r)
A2. (~r & ~s) -> t
A3. u -> (~s & ~t)

Q. (~t & p) -> (q -> s)
Example (2) (cont.)

• Conversion to clause form

\[ A1: p \rightarrow (q \rightarrow \neg r) \Rightarrow \neg p \lor \neg q \lor \neg r \Rightarrow C_1 = \neg p, q, r. \]
\[ A2: (\neg r \land \neg s) \rightarrow t \Rightarrow \neg(\neg r \land \neg s) \lor t \Rightarrow r \lor s \lor t \Rightarrow C_2 = r, s, t : \neg. \]
\[ A3: u \rightarrow (\neg s \land \neg t) \Rightarrow \neg u \lor (\neg s \land \neg t) \Rightarrow (\neg u \lor \neg s) \land (\neg u \lor \neg t) \]
\[ \Rightarrow \begin{cases} C_3 = : \neg u, s. \\ C_4 = : \neg u, t. \end{cases} \]
\[ t, u, r \in C_2 \]
\[ \neg Q : \neg ((\neg t \land p) \rightarrow (q \rightarrow s)) \Rightarrow (\neg t \land p) \land \neg(\neg q \lor s) \Rightarrow \neg t \land p \land q \land \neg s \]
\[ \Rightarrow \begin{cases} C_5 = : \neg t. \\ C_6 = p : \neg. \\ C_7 = q : \neg. \\ C_8 = : \neg s. \end{cases} \]
Example (2) (cont.)

- Determining the consistency of \( \{C_1, C_2, ..., C_8\} \)

\[ \neg p, q, r. \quad r, s, t \vdash \neg. \quad \neg u, s. \quad \neg u, t. \quad \neg t. \quad p : \neg. \quad q : \neg. \quad \neg s. \]

```
C_1
C_2
C_3
C_4
C_5
C_6
C_7
C_8
```

```

\[ \neg q, r. \quad r, s : \neg. \quad \neg r. \quad r : \neg. \quad \neg. \]
```
Example (2) (cont.)

• \( C = \{C_1, C_2, \ldots, C_8\} \) is inconsistent (by resolution/refutation)

• Therefore \( Q \) is provable (deducible)

• Answer: YES (true)

• This is how Prolog answers Queries. If the empty string is deduced, Prolog answers YES (or TRUE).
Resolution in Predicate Logic

- A literal in Predicate Logic (PL) is either
  - A positive literal in the form of $p(t_1, ..., t_k)$ where $p$ is a predicate and $t_i$ are terms
  - Or a negative literal in the form of $\neg p(t_1, ..., t_k)$

- Two clauses in PL can be resolved upon two complementary unifiable literals

- Two literals are unifiable if a substitution can make them identical.

- Example:
  - $\text{study\_hard}(X)$ and $\text{study\_hard}(\text{john})$
  - $\text{date}(D, \text{M}, 2001)$ and $\text{date}(D1, \text{may}, Y1)$
**Substitution**

- **Substitution**: is a finite set of pairs of terms denoted as \([X_1/t_1, ..., X_n/t_n]\) where each \(t_i\) is a term and each \(X_i\) is a variable.

- Every variable is mapped to a term; if not explicitly mentioned, it maps to itself.

- For example:
  - `date(D, M, 2001)` and `date(D1, may, Y1)`
Applying substitution to literals

• Example:
  \[ p(X, f(X, 2, Z), 5) \]
  \[ e = \left[ \frac{X}{5}, \frac{Z}{h(a, 2+X)} \right] \]
  \[ e(p(X, f(X, 2, Z), 5)) = p(5, f(5, 2, h(a, 2+X)), 5) \]

• Note:
  – Simultaneous substitution
  – \( X \) in \( h(a, 2+X) \) is not substituted

• Example:
  \[ r(X, Y) \]
  \[ e = \left[ \frac{X}{Y}, \frac{Y}{X} \right] \]
  \[ e(r(X, Y)) = r(Y, X) \]

• Example:
  \[ r(X, f(2, Y)) \]
  \[ e = \left[ \frac{f(2, Y)}{Z} \right] \]
  illegal substitution- only variables can be substituted
Applying substitution to clauses

• Substitution of a clause is defined by applying substitution to each of its literals:
  \[ e(\, p \, :\, -\, q_1, \, ...\, , \, q_k.\,) = e(p) \, :\, -\, e(q_1), \, ...\, , \, e(q_k). \]

• Example:
  C: \, pass_3401(X)\, :\, -\, student(X, \, Y), \, study\_hard(X). \\
  e=[X/\, john, \, Y/\, 3401] \\
  e(C)= \, pass_3401(john)\, :\, -\, student(john, \, 3401), \, study\_hard(john).
Let $p_1$ and $p_2$ be two literals and let $e$ be a substitution. We call $e$ a **unifier** of $p_1$ and $p_2$ if $e(p_1)=e(p_2)$.

Two literals are **unifiable** if such a unifier exists.

Example:
- $date(D, M, 2001)$ and $date(D_1, \text{may}, Y_1)$
- $e_1=[D/15, D_1/15, M/\text{may}, Y_1/2001]$
- $e_2=[D_1/D, M/\text{may}, Y_1/2001]$

A unifier $e$ is said to be a **most general unifier (mgu)** of two literals/terms iff $e$ is more general than any other unifier of the terms.
Unification

• Called matching in Prolog

• Rules for matching two terms $S$ and $T$ match [Bratko]:
  
  – If $S$ and $T$ are constants, then $S$ and $T$ match only if they are the same object.
  
  – If $S$ is a variable (and $T$ is anything), then they match and $S$ is substituted by $T$ (*instantiated to* $T$). Conversely, if $T$ is a variable, then $T$ is substituted by $S$.
  
  – If $S$ and $T$ are structures, then they match if
    
    • $S$ and $T$ have the same principal functor
    
    • All their corresponding components match
Unification vs. Matching

• Are \( p(X) \) and \( p(f(X)) \) unifiable?
  
  \[
e = [X/f(X)]
  
  X = f(f(f(f(\ldots \ldots ?!))}
  
• This is not allowed in unification. Proper unification requires the \textit{occurs check}: a variable \( X \) can not be substituted by a term \( t \) if \( X \) occur in \( t \).

• This is not done in Prolog’s matching for efficiency reasons.
  – Therefore it is referred to as ‘matching’ in Prolog, and not ‘unification’.
Examples

Are the following literals unifiable? What is their mgu?

1. \( \text{triangle}(\text{point}(1,2), X, \text{point}(2,4)) \) and \( \text{triangle}(A, \text{point}(5, Y), \text{point}(2, B)) \)
   unifiable: \( \text{mgu} = \{A/\text{point}(1,2), X/\text{point}(5,Y), B/4\} \)

2. \( \text{horizontal}(\text{point}(1,X), Y) \) and \( \text{vertical}(Z,A) \)
   not unifiable: \( \text{horizontal} \neq \text{vertical} \)

3. \( \text{plus}(2,2) \) and \( 4 \)
   not unifiable

4. \( \text{seg}(\text{point}(1,2), \text{point}(3,4)) \) and \( \text{seg}(f(1,2), Y) \)
   not unifiable: \( \text{point} \neq f \)
The resolution rule

• Given two clauses in the form:
  \[ A_0..A_i..A_m:-B_1...B_n. \quad and \quad C_1...C_k:-D_1..D_j..D_l. \]
  
  If \( e \) is a unifier of \( A_i \) and \( D_j \) (i.e. \( e(A_i)=e(D_j) \))
  
  Then the resolvent of the above two clauses is:
  \[ e(A_0)..e(A_{i-1})e(A_{i+1})..e(A_m) e(C_1)..e(C_k):-
  
  e(B_1)..e(B_n) e(D_1)..e(D_{j-1})e(D_{j+1})..e(D_l). \]

• Example:
  
  \( C_1: \quad p(f(1)):-r(X, Y), q(Y, Z). \)
  
  \( C_2: \quad :-p(Y). \)
  
  Unifier of \( p(f(1)) \) and \( p(Y) \): \( e=[Y/f(1)] \)
  
  The resolvent of \( C_1 \) and \( C_2 \): \( :-r(X,f(1)), q(f(1), Z). \)
Example

[Nilsson]

C0:  proud(X) :- parent(X, Y), newborn(Y).
C1:  parent(X, Y) :- father(X, Y).
C2:  parent(X, Y) :- mother(X, Y).
C3:  father(adam, mary).
C4:  newborn(mary).
G0:  :- proud(Z).

Unifier of proud(..) in C0 and G0: e=[X/Z], resolvent:
G1:  :- parent(Z,Y), newborn(Y).

Unifier of parent(..) in C1 and G1: e=[X/Z, Y/Y], resolvent:
G2:  :- father(Z,Y), newborn(Y).

To prevent mistakes, we rename the variables whenever we use a fresh copy of a clause.
Example (cont.)

[Nilsson]

G0: :- proud(Z).

(cop[y of] C0: proud(X₁) :- parent(X₁, Y₁), newborn(Y₁).
Resolve with G0: e=[X₁/Z]

G1: :- parent(Z,Y₁), newborn(Y₁).

(cop[y of] C1: parent(X₂,Y₂) :- father (X₂, Y₂).
Resolve with G1: e=[X₂/Z, Y₂/Y₁]

G2: :- father(Z, Y₁), newborn(Y₁).

(cop[y of] C3: father(adam, mary).
Resolve with G2: e=[Z/adam, Y₁/mary]

G3: :-newborn(mary).

(cop[y of] C4: newborn(mary).
Resolve with G3: e=[]

G4: :-

Empty clause → answer to query: true and Z=adam
Example (cont.)

[Nilsson]

Just a different notation for :-

\[
\begin{align*}
\leftarrow & \quad \text{proud}(Z). \\
\quad & \quad \text{proud}(X_1) \leftarrow \text{parent}(X_1, Y_1), \text{newborn}(Y_1). \\
\leftarrow & \quad \text{parent}(Z, Y_1), \text{newborn}(Y_1). \\
\quad & \quad \text{parent}(X_2, Y_2) \leftarrow \text{father}(X_2, Y_2). \\
\leftarrow & \quad \text{father}(Z, Y_1), \text{newborn}(Y_1). \\
\quad & \quad \text{father}(\text{adam}, \text{mary}). \\
\leftarrow & \quad \text{newborn}(\text{mary}). \\
\quad & \quad \text{newborn}(\text{mary}).
\end{align*}
\]

Refutation Tree for G0
Linear Refutation

- We can resolve with different clauses and keep adding new clauses forever!

- To prevent this, Linear Refutation always starts with a goal (as the example showed previously).

- Prolog’s computation rule:
  Always selects the leftmost subgoal, although logically there is no order for the subgoals.

  Example: When resolving \texttt{G1: :- parent(Z,Y_1), newborn(Y_1)., parent(..) was selected to resolve upon.}

  Prolog also starts from the top of knowledge base and goes down the list of facts and rules.
Search Space

• Based on linear refutation and Prolog’s computation rule, we know the search tree of Prolog.

• **Search tree:**
  The root in the search tree is the main goal $G_0$. A child node is a new goal $G_i$ obtained through resolution. A link is labelled with the clause resolved with and the substitution.

• **Example:**
  
  \[
  \begin{align*}
  C0: & \quad \text{grandfather}(X,Z) :- \text{father}(X,Y), \text{parent}(Y,Z).
  
  C1: & \quad \text{parent}(X,Y) :- \text{father}(X,Y).
  
  C2: & \quad \text{parent}(X,Y) :- \text{mother}(X,Y).
  
  C3: & \quad \text{father}(a,b):-.
  
  C4: & \quad \text{mother}(b,c):-.
  
  C5: & \quad \text{mother}(b,d):-.
  
  G0: & \quad :- \text{grandfather}(a,X).
  \end{align*}
  \]
Search Space (example)

\[ \leftarrow \text{grandfather}(a, X). \]
\[ \leftarrow \text{father}(a, Y_0), \text{parent}(Y_0, X). \]
\[ \leftarrow \text{parent}(b, X). \]

C0: \( \text{grandfather}(X,Z) \leftarrow \text{father}(X,Y), \text{parent}(Y,Z). \)
C1: \( \text{parent}(X,Y) \leftarrow \text{father}(X,Y). \)
C2: \( \text{parent}(X,Y) \leftarrow \text{mother}(X,Y). \)
C3: \( \text{father}(a,b) \leftarrow. \)
C4: \( \text{mother}(b,c) \leftarrow. \)
C5: \( \text{mother}(b,d) \leftarrow. \)
G0: \( \leftarrow \text{grandfather}(a,X). \)

Nothing to resolve with, backtrack!

\( \square \ \text{TRUE, } X=c \); (backtrack!)
\( \square \ \text{TRUE, } X=d \)

York University- CSE 3401- V. Movahedi 04_Resolution 28
• What is the search strategy used by Prolog for searching the tree?

Depth First Search  or  Breadth First Search

Prolog uses DFS