331 - Intro to Intelligent Systems
Week07c
Inference
R&N Chapter 9

T.J. Borrelli

Resolution in Propositional Logic

- Deductive reasoning can be used to make proofs in propositional and predicate logic
- It is not clear how this process can be automated because at each stage some initiative is required to choose the right next step
- *Resolution* is a proof method that can be automated because it involves a fixed set of steps
- Resolution requires that the problem be expressed in *conjunctive normal form*

Conjunctive Normal Form

- A well-formed formula (wff) is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctions:
  - $X_1 \wedge X_2 \wedge X_3 \wedge \ldots \wedge X_n$

  where each clause, $X_{i,}$ is of the form:
  - $Y_1 \vee Y_2 \vee Y_3 \vee \ldots \vee Y_n$
    - The Ys are literals
- For example, $A \wedge (B \vee C) \wedge (\neg A \vee \neg B \vee \neg C \vee D)$
- Similarly, a wff is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctions.
- For example, $A \vee (B \wedge C) \vee (\neg A \wedge \neg B \wedge \neg C \wedge D)$

Conversion to CNF

- Any wff can be converted to CNF by using the following equivalences:

(1) $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

(2) $A \rightarrow B \equiv \neg A \vee B$

(3) $\neg(A \wedge B) \equiv \neg A \vee \neg B$

(4) $\neg(A \vee B) \equiv \neg A \wedge \neg B$

(5) $\neg\neg A \equiv A$

(6) $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

- Importantly, this can be converted into an algorithm – this will be useful for *automating resolution*

Example 1 Conversion to CNF

- Convert $(A \rightarrow B) \rightarrow C$ to CNF:

  $\neg(A \rightarrow B) \vee C$

  $\neg(\neg A \vee B) \vee C$

  $(A \wedge \neg B) \vee C$

  $(A \vee C) \wedge (\neg B \vee C)$

- Now express the CNF form as a set of clauses:

  $\{(A, C), (\neg B, C)\}$

Example 2 Conversion to CNF

- Convert $A \leftrightarrow (B \wedge C)$:

  $(A \rightarrow (B \wedge C)) \wedge ((B \wedge C) \rightarrow A)$

  $(\neg A \vee (B \wedge C)) \wedge (\neg(B \wedge C) \vee A)$

  $(\neg A \vee (B \wedge C)) \wedge (\neg B \vee \neg C \vee A)$

  $(\neg A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C \vee A)$

- Now express the CNF form as a set of clauses:

  $\{(\neg A, B), (\neg A, C), (\neg B, \neg C, A)\}$

The Resolution Rule

- The resolution rule is written as follows:

$$\frac{A \vee B \qquad \neg B \vee C}{A \vee C}$$

- This tells us that if we have two clauses that have a literal and its negation, we can combine them by removing that literal
- For example, if we have {(A, B), (¬B, C)} we would apply resolution to get {(A, C)}

The Resolution Rule

- Example: {(A,B,C), D, (¬A,D,E),(¬D,F)} can be resolved to {(B,C,D,E), D, (¬D,F)} which can be further resolved to {(B,C,D,E), F} or {(B,C,E,F), D}
- Note that at the first step, we also had a choice and could have resolved to {(A,B,C),D,(¬A,E,F)} which can be further resolved to {(B,C,E,F), D}

The Resolution Rule

- If wff P resolves to wff Q, we write $P \models Q$
- For example, resolve $(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee C)$:

  $\{(A,B), (\neg A,C), (\neg B,C)\}$

  $\{(B,C), (\neg B,C)\}$

  $\{C\}$
- We can express this as

  $(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee C) \models C$
- The *resolvent* is a logical consequence of the clauses

Resolution Refutation

- Let us resolve: {(¬A,B), (¬A,¬B,C), A, ¬C}
- We begin by resolving the first clause with the second clause, thus eliminating B and ¬B:

  {(¬A,C), A, ¬ C}

  {C, ¬C}
- Now we can resolve both remaining literals, which gives falsum (a contradiction):

  ⊥
- If we reach falsum, we have proved that our initial set of clauses were inconsistent
- This is written:

  {(¬A,B),(¬A,¬B,C),A,¬C} ⊨ ⊥

Proof by Refutation

- If we want to prove that a logical argument is valid, we negate its conclusion, convert it to clause form, and then try to derive falsum using resolution
- If we derive falsum, then our clauses were inconsistent, meaning the original argument was valid, since we negated its conclusion

Example 1 of Proof by Refutation

- Prove the following 3 statements by refutation:

  $(A \wedge \neg B) \rightarrow C$

  $A \wedge \neg B$

  $\therefore C$

- Negate the conclusion and convert to clauses:

  $\{(\neg A, B, C),\ A,\ \neg B,\ \neg C\}$

- Now resolve:

  $\{(B, C),\ \neg B,\ \neg C\}$

  $\{C,\ \neg C\}$

  $\bot$

- We have reached falsum, so our original argument was valid

Example 2 of Proof by Refutation

- Prove or disprove the following:

  $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E \vee F$

  $\therefore A \rightarrow F$

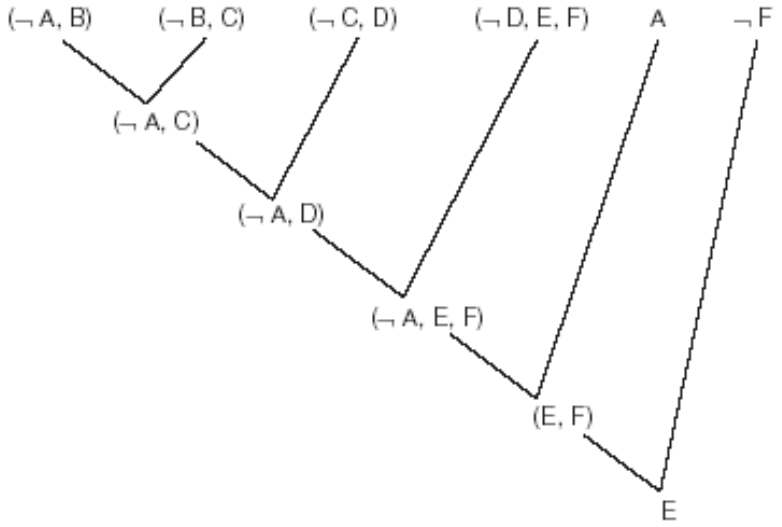
- First, we negate the conclusion to give $\neg(A \rightarrow F)$
- Next we convert to clause form:

  $D \rightarrow E \vee F \equiv \neg D \vee (E \vee F)$ and $\neg(A \rightarrow F) \equiv \neg (\neg A \vee F)$
  $\equiv A \wedge \neg F$

- So our clauses are: $\{(\neg A,B), (\neg B,C), (\neg C,D), (\neg D,E,F), A, \neg F\}$
- This will resolve to $\{E\}$; we cannot reach falsum. Hence, we can conclude that our original conclusion was not valid. (You can prove this for yourself by using a truth table.)

Refutation Proof in Tree Form

- It is often helpful to represent a refutation proof in tree form:



- In this case, the proof has failed, as we are left with {E} instead of falsum

Example: Graph Coloring

- Resolution refutation can be used to determine if a solution exists for a particular combinatorial problem
- For example, for graph coloring, we represent the assignment of colors to the nodes and the constraints regarding edges as propositions, and attempt to prove that the complete set of clauses is consistent
- This does not tell us how to color the graph, simply that it is possible

Example: Graph Coloring

Available colors are r (red), g (green) and b (blue).

A_r means that node A has been colored red.

Each node must have exactly one color:

$A_r \vee A_g \vee A_b$

$\neg A_r \vee \neg A_g \quad (\equiv A_r \rightarrow \neg A_g)$

$\neg A_g \vee \neg A_b$

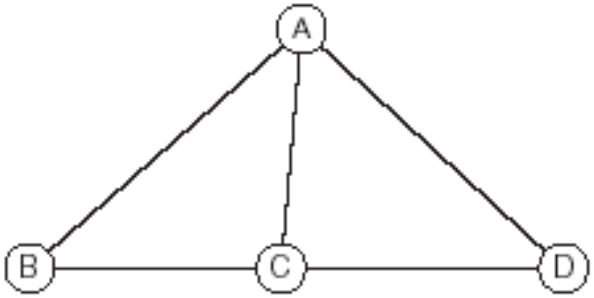
$\neg A_b \vee \neg A_r$

If (A,B) is an edge, then:

$\neg A_r \vee \neg B_r$

$\neg A_b \vee \neg B_b$

$\neg A_g \vee \neg B_g$



Now we construct the complete set of clauses for our graph, and try to see if they are consistent, using resolution

Resolution in Propositional Logic: Truth-Tellers and Liars

Suppose that liars always speak what is false, and truth-tellers always speak what is true. Further, suppose that Amy, Bob, and Cal are each either a liar or a truth-teller. Amy says, “Bob is a liar.” Bob says, “Cal is a liar.” Cal says, “Amy and Bob are liars.” Which, if any, of these people are truth-tellers?

Resolution in Propositional Logic: Truth-Tellers and Liars

The atomic sentences for this problem are:

A, for “Amy is a truth-teller.”

B, for “Bob is a truth-teller.”

C, for “Cal is a truth-teller.”

A literal is an atomic sentence or its negation. For example, A, ¬A, B, ¬B, C, ¬C are all literals.

A sentence is either an atomic sentence or a complex sentence.

Resolution in Propositional Logic: Truth-Tellers and Liars

The set of sentences that represent our knowledge is called our *knowledge base*. The knowledge base for this problem is:

$\{A \leftrightarrow \neg B, B \leftrightarrow \neg C, C \leftrightarrow (\neg A \wedge \neg B)\}$

(Amy is a truth-teller if and only if Bob is not a truth-teller)

Note that these sentences could all be combined with conjunction into a single sentence which expresses all knowledge of the knowledge base:

$(A \leftrightarrow \neg B) \wedge (B \leftrightarrow \neg C) \wedge (C \leftrightarrow (\neg A \wedge \neg B))$

Resolution in Propositional Logic: Truth-Tellers and Liars

Convert the knowledge base to CNF:

1. Re-write {A ↔ ⌐B, B ↔ ⌐C, C ↔ (⌐A ∧ ⌐B)} as

{(A→⌐B), (⌐B→A), (B→⌐C), (⌐C→B), (C→(⌐A∧⌐B)), ((⌐A∧⌐B)→C)}

2. Eliminate →

{⌐A ∨ ⌐B, ⌐⌐B ∨ A, ⌐B ∨ ⌐C, ⌐⌐C ∨ B, ⌐C ∨ (⌐A∧⌐B), ⌐(⌐A∧⌐B) ∨ C}

3. Apply DeMorgan and eliminate ⌐⌐

{⌐A ∨ ⌐B, B ∨ A, ⌐B ∨ ⌐C, C ∨ B, ⌐C ∨ (⌐A∧⌐B), A ∨ B ∨ C}

4. Distribute ∨ over ∧

{⌐A ∨ ⌐B, B ∨ A, ⌐B ∨ ⌐C, C ∨ B, ⌐C ∨ ⌐A, ⌐C ∨ ⌐B, A ∨ B ∨ C}

Resolution in Propositional Logic: Truth-Tellers and Liars

Thus, the knowledge base consists of:

{(¬A,¬B), (B,A), (¬B,¬C), (C,B), (¬C,¬A), (¬C,¬B), (A,B,C)}

It is important to note that a model in CNF is a truth assignment that makes *at least one literal in each clause true*.

Resolution in Propositional Logic: Truth-Tellers and Liars

Consider the two clauses (B,A) and (¬B,¬C) from the knowledge base. The first reads, “Bob is a truth-teller or Amy is a truth-teller.” The second reads, “Bob is not a truth-teller or Cal is not a truth-teller.” Consider what happens according to Bob’s truthfulness:

- **Bob is a truth-teller.** In this case, the first clause is satisfied. The second clause is satisfied if and only if Cal is not a truth-teller
- **Bob is not a truth-teller.** In this case, the second clause is satisfied. The first clause is satisfied if and only if Amy is a truth-teller

Resolution in Propositional Logic: Truth-Tellers and Liars

- Since one case or the other holds, we know that in any model of *both* clauses Amy is a truth-teller or Cal is not a truth-teller
- In other words, from clauses (B,A) and (¬B,¬C), we can derive the clause (A,¬C) and add it to our knowledge base
- This is a specific application of the Resolution Rule

Resolution in Propositional Logic: Truth-Tellers and Liars

- Note that not all possible resolution rule derivations are useful
- Consider what happens when we apply the resolution rule to the first two clauses of the knowledge base
- From (¬A,¬B) and (B,A) we can derive either (¬A,A) or (¬B,B) depending on which atomic sentence we use for the resolution
- In either case we derive a tautology

Resolution in Propositional Logic: Truth-Tellers and Liars

Let's use proof by contradiction to prove that Cal is a liar (¬C). In addition to the knowledge base, we assume the negation of what we wish to prove (¬¬C, that is, C).

	(¬A, ¬B)		Knowledge base
	(B, A)		
	(¬B, ¬C)		
	(C, B)		
	(¬C, ¬A)		
	(¬C, ¬B)		
	(A, B, C)		
	(C)		Assumed negation
	(¬A)	(5, 8)	Resolution Rule
	(B)	(2, 9)	Resolution Rule
	(¬C)	(3, 10)	Resolution Rule
(12)	⊥	(8, 11)	Contradiction!

Resolution in Propositional Logic: Truth-Tellers and Liars

- Recall that at least one literal of each clause must be true for a truth assignment to be a model. The last clause (the empty clause) has no literals at all and represents a clear contradiction. It cannot be the case that C is true as we had assumed. Thus ¬C logically follows from our knowledge base.
- It should be noted that a contradictory knowledge base can derive an empty clause without any need of assumptions. One can thus prove any sentence using proof by contradiction, starting with a contradictory knowledge base.

Resolution in Propositional Logic: Truth-Tellers and Liars

- When we derive a sentence s_2 from s_1, we denote it as $s_1 \vdash s_2$
- A proof procedure that derives only what is entailed is called sound
- A proof procedure that can derive anything that is entailed is called complete
- Resolution theorem proving is both sound and complete

Improving Performance

- Reasoning performance can be greatly improved by changing the knowledge base
  - A proof is essentially a search through a maze of possible resolutions to a contradiction, so compacting the knowledge base can greatly simplify search
- Reasoning engines often have a pre-processing stage that eliminates unnecessary clauses: *equivalent clauses*, *subsumed clauses*, and *tautology clauses*
- Reasoning engines can also add new knowledge produced through reasoning to speed future reasoning

Improving Performance

(¬A, ¬B)
(B, A)
(¬B, ¬C)
(C, B)
(¬C, ¬A)
(¬C, ¬B)
(A, B, C)

Notice that (3) and (6) are logically equivalent.
Also, note that (2) entails (7). This means that (2) *subsumes* (7), and we can eliminate (7).
Delete (6) and (7) from the knowledge base.

Improving Performance

Using our previous example with the unnecessary clauses eliminated, let's try to prove that Amy is a liar:

	(¬A, ¬B)		Knowledge base
	(B, A)		
	(¬B, ¬C)		
	(C, B)		
	(¬C, ¬A)		
	(A)		Assumed negation
	(¬B)	(1, 6)	Resolution Rule
	(C)	(4, 7)	Resolution Rule
	(¬A)	(5, 8)	Resolution Rule
(10)	⊥	(6, 9)	Contradiction!

Once we have proven the contradiction, we can add ¬A to our knowledge base. Furthermore, we can eliminate all clauses that are subsumed by ¬A.

Improving Performance

After adding ⌐A and deleting subsumed clause (⌐A, ⌐B) and (⌐C, ⌐A), let's try to prove that Bob is a truth-teller:

	(1)	(B, A)		Knowledge base
	(2)	(⌐B, ⌐C)		
	(3)	(C, B)		
	(4)	(⌐A)		
	(5)	(⌐B)		Assumed negation
	(6)	(A)	(1, 5)	Resolution rule
	(7)	⊥	(4, 6)	Contradiction!

Improving Performance

With the new knowledge that Bob is a truth-teller, we can again remove subsumed clauses (B,A) and (C,B) and prove that Cal is a liar:

	(1)	(¬B, ¬C)		Knowledge base
	(2)	(¬A)		
	(3)	(B)		
	(4)	(C)		Assumed negation
	(5)	(¬B)	(1, 4)	Resolution rule
	(6)	⊥	(3, 5)	Contradiction!

Improving Performance

New knowledge that Cal is a liar subsumes clause (1) so our knowledge base is now:

(1)	(¬A)	Knowledge base
(2)	(B)	
(3)	(¬C)	

This is the answer to the original question: Amy is a liar, Bob is a truth-teller, and Cal is a liar.

Improving Performance

Consider this: The number of possible (non-tautology) clauses of length l for *a* atomic sentences is $2^l \binom{a}{l}$

a	= 1	= 2	= 3	= 4	= 5
1	2	0	0	0	0
2	4	4	0	0	0
3	6	12	8	0	0
4	8	24	32	16	0
5	10	40	80	80	32
10	20	180	960	3360	8064
20	40	760	9120	77520	496128
40	80	3120	79040	1462240	21056256
80	150	12640	657260	25305280	769280512

Normal Forms in Predicate Calculus

- To allow inferences to be automated, the logical database must be expressed in an appropriate form
- To apply resolution to first-order predicate logic expressions, we first need to deal with the presence of the quantifiers ∀ and ∃

Normal Forms in Predicate Calculus

- The requirement is that all variables must be universally quantified
  - This allows full freedom in computing substitutions
- The method that is used is to move the quantifiers to the beginning of the expression, resulting in an expression that is in *prenex normal form*

Normal Forms in Predicate Calculus

When converting a wff in predicate calculus to Prenex Normal Form, we use the same rules as we used to convert a wff to CNF:

(1) $a(X) \leftrightarrow b(X) \equiv (a(X) \rightarrow b(X)) \wedge (b(X) \rightarrow a(X))$

(2) $a(X) \rightarrow b(X) \equiv \neg a(X) \vee b(X)$

(3) $\neg(a(X) \wedge b(X)) \equiv \neg a(X) \vee \neg b(X)$

(4) $\neg(a(X) \vee b(X)) \equiv \neg a(X) \wedge \neg b(X)$

(5) $\neg\neg a(X) \equiv a(X)$

(6) $a(X) \vee (b(X) \wedge c(X)) \equiv (a(X) \vee b(X)) \wedge (a(X) \vee c(X))$

Normal Forms in Predicate Calculus

In addition, we use the following rules to move the quantifiers to the front:

$\neg(\forall X)\ a(X) \equiv (\exists X)\ \neg a(X)$

$\neg(\exists X)\ a(X) \equiv (\forall X)\ \neg a(X)$

$(\forall X)\ a(X) \wedge b(Y) \equiv (\forall X)(a(X) \wedge b(Y))$

$(\forall X)\ a(X) \vee b(Y) \equiv (\forall X)\ (a(X) \vee b(Y))$

$(\exists X)\ a(X) \wedge b(Y) \equiv (\exists X)\ (a(X) \wedge b(Y))$

$(\exists X)\ a(X) \vee b(Y) \equiv (\exists X)\ (a(X) \vee b(Y))$

$(\forall X)\ a(X) \wedge (\forall Y)\ b(Y) \equiv (\forall X)\ (\forall Y)\ (a(X) \wedge b(Y))$

$(\forall X)\ a(X) \wedge (\exists Y)\ b(Y) \equiv (\forall X)\ (\exists Y)\ (a(X) \wedge b(Y))$

$(\exists X)\ a(X) \wedge (\forall Y)\ b(Y) \equiv (\exists X)\ (\forall Y)\ (a(X) \wedge b(Y))$

$(\exists X)\ a(X) \wedge (\exists Y)\ b(Y) \equiv (\exists X)\ (\exists Y)\ (a(X) \wedge b(Y))$

*Note that rules 9, 10, 11, and 12 can be used only if X does not occur in b

Example of Converting to Prenex NF

Convert to PNF: $(\forall X)\ (a(X) \rightarrow b(X)) \rightarrow (\exists Y)\ (a(Y) \wedge b(Y))$

$\neg(\forall X)\ (a(X) \rightarrow b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 2 (imp. elim)

$\neg(\forall X)\ (\neg a(X) \vee b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 2 (imp. elim.)

$(\exists X)\ \neg(\neg a(X) \vee b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 7 (DeM Quant.)

$(\exists X)\ (\neg\ \neg a(X) \wedge \neg b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 4 (DeM)

$(\exists X)\ (a(X) \wedge \neg b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 5 (Double Neg)

$(\exists X)\ (\exists Y)\ \ ((a(X) \wedge \neg b(X)) \vee (a(Y) \wedge b(Y)))$ Rule 16 (logical equiv)

$(\exists X)\ (\exists Y)\ ((\mathbf{(a(X) \vee a(Y))} \wedge \mathbf{(a(X) \vee b(Y))} \wedge \mathbf{(\neg b(X) \vee a(Y))} \wedge \mathbf{(\neg b(X) \vee b(Y))}$

Rule 6 (distribute)

The resulting clauses are highlighted in bold

Skolemization

- Before resolution can be carried out on a wff, we need to eliminate all of the existential quantifiers (∃) from the wff
- This is done by replacing a variable that is existentially quantified by a constant:
  - For example, ∃(X) p(X) would be converted to p(c), where c is a constant that has not been used elsewhere in the wff
  - Although p(c) is not logically equivalent to ∃(X) p(X), we can make this substitution because we are only interested in seeing if a solution exists (i.e., assume there exists some X for which p(X) holds)

Skolemization

- Skolemization must proceed slightly differently in the case where the ∃ follows a ∀ as in: (∀X) (∃Y) (p(X,Y))
- In this case, rather than replacing Y with a skolem constant, we must replace it with a skolem function, such as in: (∀X) (p(X, f(X)))
- Note that the skolem function is a function of the variable that is universally quantified (X)
  - Once the existentially quantified variables have been removed from a logical database, unification may be used to match sentences in order to apply inference

Example 1 of Skolemization

Skolemize the following expression:

$(\forall X)\ (\exists Y)\ (\forall Z)\ (p(X) \wedge q(Y,Z))$

Solution:

$(\forall X)\ (\forall Z)\ (p(X) \wedge q(f(X),Z))$

Note that Y has been replaced by f(X) because ∃Y occurred *after* ∀X

Example 2 of Skolemization

Skolemize the following expression:

$(\forall W)\ (\forall X)\ (\exists Y)\ (\forall Z)\ (p(X) \wedge q(W, Y, Z))$

Solution: $(\forall W)\ (\forall X)\ (\forall Z)\ (p(X) \wedge q(W, f(W, X), Z))$

– Note that Y has been replaced by a function of both W and X, f(W, X) because $\exists Y$ occurred after $\forall W$ and $\forall X$

- To proceed with resolution, this wff must now be represented as a set of clauses
- To do this we first drop the universal quantifiers
- Hence, the expression above will be represented as: {(p(X)), (q(W, f(W,X), Z))}

Unification

- To resolve clauses we often need to make substitutions. For example:

  {(p(W,X)), (¬p(Y,Z))}

- To resolve, we need to substitute Y for W, and Z for X, in other words, {W/Y, X/Z}, giving:

  {(p(Y,Z)), (¬p(Y,Z))}

- These resolve to give falsum

Unification

For example, generate legal substitutions (bindings) of the expression “team(X, bob, friend(Y))” using the following unifications:

{X/code_monkeys, Y/Z} unifies the expression to:

team(code_monkeys, bob, friend(Z))

{X/W, Y/jack} unifies the expression to:

team(W, bob, friend(jack))

{X/Z, Y/cousin(Z)} unifies the expression to:

team(Z, bob, friend(cousin(Z)))

Unification

Issues with unification:

A constant is considered a “ground instance” and cannot be replaced

A variable cannot be unified with a term containing the variable

- For example, X cannot be replaced with p(X) as this creates an infinite expression: p(p(p(p(…X)))

Unifying substitutions must be made consistently across all occurrences within the scope of the variable in both expressions being matched

Unification

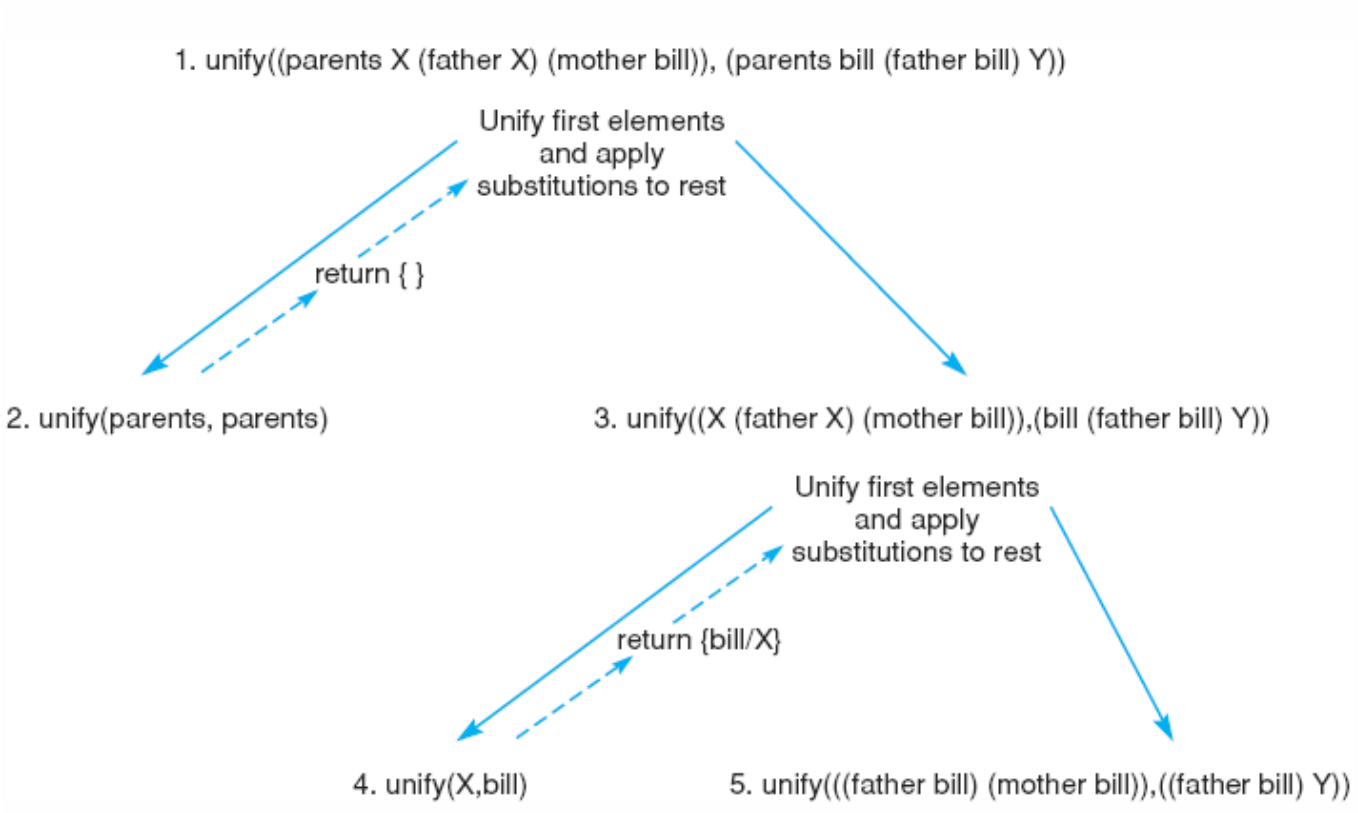
Once a variable is bound to a constant, that variable may not be given a new binding in a future unification

For example, suppose p(adam,X) unifies with the premise of p(Y,Z) → q(Y,Z) using the substitution {Y/adam, Z/X}, then modus ponens lets us infer q(adam,X) under the same substitution.

If we then match this result with the premise of q(W,bob) → r(W,bob), we must infer r(adam,bob).

Example of Unification

First 5 steps in the unification of:
(parents X (father X) (mother bill)) and (parents bill (father bill) Y)



Example of Unification

When unify is first called, because neither argument is an atomic symbol, the function will attempt to recursively unify the first elements of each expression, calling:

unify(parents, parents).

This unification succeeds, returning the empty substitution { }. Applying this to the remainder of the expression creates no change. The algorithm then calls:

unify((X (father X) (mother bill), (bill (father bill) Y)).

Example of Unification

In the second call to unify, neither expression is atomic, so the algorithm separates the expression into its first component and the remainder of the expression. This leads to the call:

unify(X, bill).

This call succeeds because both expressions are atomic and one of them is a variable. The call returns the substitution {bill/X}. This substitution is applied to the remainder of each expression and unify is called on the results: unify(((father bill) (mother bill)), ((father bill) Y)).

Example of Unification

The result of this call is to unify (father bill) with (father bill). This leads to the calls:

unify(father, father)
unify(bill, bill)
unify(( ), ( )).

All of these succeed, returning the empty set of substitutions. Unify is then called on the remainder of the expressions:
unify(((mother bill)), (Y)).

Example of Unification

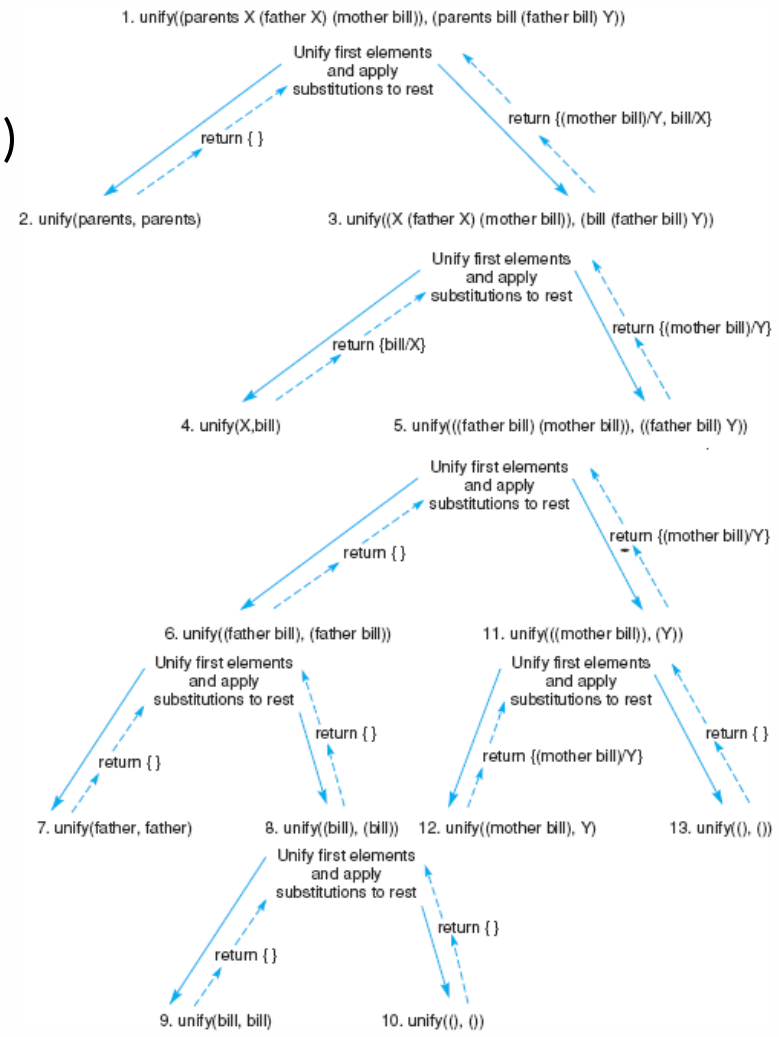
This in turn leads to calls:

unify((mother bill), Y) and
unify(( ), ( )).

In the first of these, (mother bill) unifies with Y. Notice that unification substitutes the whole structure (mother bill) for Y. Thus, unification succeeds and returns the substitution {(mother bill)/Y}. The call unify(( ), ( )) returns { }. All the substitutions are composed as each recursive call terminates, to return the answer {bill/X (mother bill)/Y}.

Example of Unification

Final trace of the unification of
(parents X (father X) (mother bill))
and
(parents bill (father bill) Y)



The Resolution Algorithm

First, negate the conclusion and add it to the list of assumptions

Now convert the assumptions into Prenex Normal Form (PNF)

Next, skolemize the resulting expression

Now convert the expression into a set of clauses

Now resolve the clauses using suitable unifiers

This algorithm means we can write programs that automatically prove theorems using resolution!

Resolution in Predicate Logic

Consider the following set of premises:

- Some children like any food
- No children like food that is green
- All children like food made by Cadbury's

We now wish to prove that the following conclusion follows from these premises:

- "No food made by Cadbury's is green"

Resolution in Predicate Logic

First, we need to represent the premises and the conclusion in predicate calculus:

c(X) means “X is a child”

f(X) means “X is food”

l(X, Y) means “X likes Y”

g(X) means “X is green”

m(X, Y) means “X makes Y”

c means “Cadbury’s”

Resolution in Predicate Logic

The premises can be represented as:

$(\exists X)\ (c(X) \wedge (\forall Y)\ (f(Y) \rightarrow l(X, Y)))$

$(\forall X)\ (c(X) \rightarrow (\forall Y)\ ((f(Y) \wedge g(Y)) \rightarrow \neg l(X, Y)))$

$(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow (\forall Y)\ (c(Y) \rightarrow l(Y, X)))$

Represent the conclusion as:

$(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow \neg g(X))$

Resolution in Predicate Logic

First, we must negate the conclusion and add it to the set of premises, which means we must now prove that the following expression cannot be satisfied:

$$(\exists X)\,(c(X) \wedge (\forall Y)\,(f(Y) \rightarrow l(X, Y))) \wedge$$
$$(\forall X)\,(c(X) \rightarrow (\forall Y)\,((f(Y) \wedge g(Y)) \rightarrow \neg l(X, Y))) \wedge$$
$$(\forall X)\,((f(X) \wedge m(c, X))) \rightarrow (\forall Y)\,(c(Y) \rightarrow l(Y, X))) \wedge$$
$$\neg((\forall X)\,((f(X) \wedge m(c, X)) \rightarrow \neg g(X)))$$

Next, convert this expression into a set of clauses.

Resolution in Predicate Logic

Expression 1: $(\exists X)\ (c(X) \wedge (\forall Y)\ (f(y) \rightarrow l(X, Y)))$

Eliminate $\rightarrow$: $(\exists X)\ (c(X) \wedge (\forall Y)\ (\neg f(Y) \vee l(X, Y)))$

Bring the quantifiers to front: $(\exists X)(\forall Y)\ (c(X) \wedge (\neg f(Y) \vee l(X, Y)))$

Skolemize: $(\forall Y)\ (c(a) \wedge (\neg f(Y) \vee l(a, Y)))$

Create set of clauses: $\{c(a), (\neg f(Y), l(a, Y))\}$

Resolution in Predicate Logic

Expression 2: $(\forall X)\,(c(X) \rightarrow (\forall Y)\,((f(Y) \wedge g(Y)) \rightarrow \neg l(X,Y)))$

Eliminate $\rightarrow$: $(\forall X)(\neg c(X) \vee (\forall Y)\,(\neg(f(Y) \wedge g(Y)) \vee \neg l(X, Y)))$

DeMorgan's law: $(\forall X)\,(\neg c(X) \vee (\forall Y)\,(\neg f(Y) \vee \neg g(Y) \vee \neg l(X,Y)))$

Quantifiers to front: $(\forall X)\,(\forall Y)(\neg c(X) \vee \neg f(Y) \vee \neg g(Y) \vee \neg l(X,Y))$

Create (single) clause: $\{(\neg c(X), \neg f(Y), \neg g(Y), \neg l(X,Y))\}$

Resolution in Predicate Logic

Expression 3: $(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow (\forall Y)\ (c(Y) \rightarrow l(Y, X)))$

Eliminate $\rightarrow$: $(\forall X)\ (\neg(f(X) \wedge m(c, X)) \vee (\forall Y)\ (\neg c(Y) \vee l(Y, X)))$

DeMorgan's law: $(\forall X)(\neg f(X) \vee \neg m(c, X) \vee (\forall Y)(\neg c(Y) \vee l(Y, X)))$

Quantifiers: $(\forall X)(\forall Y)(\neg f(X) \vee \neg m(c, X) \vee \neg c(Y) \vee l(Y, X))$

Single clause: $\{(\neg f(X), \neg m(c, X), \neg c(Y), l(Y, X))\}$

Resolution in Predicate Logic

Conclusion (which has been negated):

$\neg(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow \neg g(X))$

Eliminate $\rightarrow$: $\neg(\forall X)\ (\neg(f(X) \wedge m(c, X)) \vee \neg g(X))$

Move $\neg$ from front: $(\exists X)\ \neg(\neg(f(X) \wedge m(c, X)) \vee \neg g(X))$

DeMorgan's law: $(\exists X)\ (\neg\neg(f(X) \wedge m(c, X)) \wedge \neg\neg g(X))$

Remove $\neg\neg$: $(\exists X)\ (f(X) \wedge m(c, X) \wedge g(X))$

Skolemize: $f(b) \wedge m(c, b) \wedge g(b)$

Create set of clauses: $\{f(b), m(c, b), g(b)\}$

Now we have arrived at a set of clauses, upon which resolution can be applied

Resolution in Predicate Logic

The clauses we have are the following:

c(a)

(¬f(Y), l(a, Y))

(¬c(X), ¬f(Y), ¬g(Y), ¬l(X, Y))

(¬f(X), ¬m(c, X), ¬c(Y), l(Y, X))

f(b)

m(c, b)

g(b)

Resolution in Predicate Logic

We now apply resolution as follows:

First we unify clauses 1 and 3 using {X/a} and resolve to give:

(¬f(Y), ¬g(Y), ¬l(a, Y))

Similarly, the unifier {Y/b} can be applied to resolve clauses 2 and 5:

l(a, b)

Now we apply {Y/b} to resolve clause 5 with 8 to give:

(¬g(b), ¬l(a, b))

Next, clauses 9 and 10 can be resolved to give:

¬g(b)

Clause 11 can be resolved with 7 to give the empty set, or:

falsum

$\square$

Resolution in Predicate Logic

- We have proven that the set of clauses derived from the premises 1, 2, and 3, and the negation of the conclusion, are unsatisfiable
- Thus we have successfully proved that the conclusion does indeed follow from the premises, and so the argument is a valid one