Models

- Let our knowledge base KB, consist of a set of formulas.

- We say that I is a model of KB or that I satisfies KB
  - If, every formula $f \in$ KB is true under I

- We write I $\models$ KB if I satisfies KB, and I$\models f$ if f is true under I.

What’s Special About Models?

- When we write KB, we have an intended interpretation for it – a way that we think “the world” – the scenario we’re modeling -- will be.
- This means that every statement in KB is true in “the world”.
- Note however, that not every thing true in the world need be contained in KB. We might have only incomplete knowledge.

Models support reasoning.

Suppose formula f is not mentioned in KB, but is true in **every** model of KB; i.e.,

For all I, if $I \models KB$ then $I \models f$.

Then we say that f is a logical consequence of KB or that KB entails f

$KB \models f$.

Since “the world” is a model of KB, f must be true in “the world”.

This means that entailment is a way of finding new true facts that were not explicitly mentioned in KB.

QUESTION: If KB doesn’t entail f, is f false in “the world” ?

Axiomatizing the Domain

The more sentences in KB, the fewer models (satisfying interpretations) there are.

The more you write down (as long as it's all true!), the "closer" you get to a complete description of "the world" and the less is left to doubt, because each sentence in KB rules out certain unintended interpretations.

Writing down the description of the world is called

axiomatizing the domain.

We will **construct a knowledge base** by writing down a description of (some of) the world in first-order logic.

Computing logical consequences

We want **procedures for computing logical consequences** that can be implemented in our programs.

This would allow us to reason about the world. In particular we would:

1. Represent aspects of our knowledge of the world as logical formulas
2. Apply procedures for generating logical consequences

Procedures for computing logical consequences are called

proof procedures.

Proof Procedures

- Proof procedures work by simply manipulating formulas. They do not know or care anything about interpretations.
- Nevertheless they respect the semantics of interpretations!
- We will employ a proof procedure for first-order logic called resolution.
  - Resolution is the mechanism used by PROLOG

Properties of Proof Procedures

Before presenting the details of resolution, we want to look at properties we would like to have in a (any) proof procedure.

We write $KB \vdash f$ to indicate that f can be proved from KB.

The proof procedure used is implicit. Many different proof procedures exist. Again, we will be studying *resolution.*

Properties of Proof Procedures

Soundness

If KB $\vdash$ f then KB $\models$ f

i.e., all conclusions arrived at via the proof procedure are correct: they are logical consequences.

Completeness

If KB $\models$ f then KB $\vdash$ f

i.e. every logical consequence can be generated by the proof procedure.

Note proof procedures are computable, but they might have very high complexity in the worst case. So completeness is not necessarily achievable in practice.

Resolution

Resolution is a rule of inference leading to a theorem proving technique in propositional and first-order logic.

Iteratively applying the resolution rule allows us to determine whether a propositional formula is satisfiable or a first-order formula is unsatisfiable.

Resolution Rule

From the two clauses

$P \vee Q$

$\neg P$

We infer the new clause

Q

The original algorithm required ground instances of formulas. *What's the problem with this?*

The unification algorithm allows "instantiation or grounding on demand" to preserve refutation completeness.

Clausal form

Resolution works with formulas expressed in **clausal form**.

- A literal is an atomic formula or the negation of an atomic formula.
  - dog(fido), ¬cat(fido)
- A clause is a disjunction of literals:
  - ¬owns(fido,fred) ∨ ¬dog(fido) ∨ person(fred)
  - We write
    (¬owns(fido,fred), ¬dog(fido), person(fred)) ← Notation
- A ground clause is a clause containing no variables
  e.g., the clause dog(X) ∨ person(fred) is not ground.
  The clause dog(fido) ∨ person(fred) is ground.
- A clausal theory is a conjunction of clauses.

Resolution Rule for Ground Clauses

The resolution proof procedure consists of only **one simple rule**:

From the two clauses

- (P, Q1, Q2, …, Qk)
- (¬P, R1, R2, …, Rn)

We infer the new clause

- (Q1, Q2, …, Qk, R1, R2, …, Rn)

Example:

- (¬largerThan(clyde,cup), ¬fitsIn(clyde,cup))
- (fitsIn(clyde,cup))

Infer ¬largerThan(clyde,cup)

Resolution Proof: Forward chaining

Logical consequences can be generated from the resolution rule in two ways:

1. **Forward Chaining Inference** ("Consequence Finding")*
   - If we have a sequence of clauses C1, C2, …, Ck
   - Such that each Ci is either in KB or is the result of a resolution step involving two prior clauses in the sequence.
   - We then have that KB $\vdash$ Ck.

   Forward chaining is sound so we also have KB $\models$ Ck

** The meat grinder*

Resolution Proof: Refutation proofs

2. **Refutation Proofs.**
   - We determine if KB ⊢ f by showing that a contradiction can be generated from KB ∧ ¬f.
   - In this case a contradiction is an empty clause ().
   - We employ resolution to construct a sequence of clauses C1, C2, …, Cm such that
     - Ci is in KB ∧ ¬f, or is the result of resolving two previous clauses in the sequence.
     - Cm = () i.e. its the empty clause.

Resolution Proof: Refutation proofs

If we can find a sequence C1, C2, …, Cm=(), we have that

- KB $\vdash$ f

Furthermore, this procedure is **sound** so

- KB $\models$ f

The procedure is also **complete (refutation complete)** so it is capable of finding a proof of any f that is a logical consequence of KB. I.e.

- If KB $\models$ f then we can generate a refutation from KB $\land \neg$f

Resolution Proofs Example

Want to prove likes(clyde,peanuts) from:

(elephant(clyde), giraffe(clyde))	[1]
(¬elephant(clyde), likes(clyde,peanuts))	[2]
(¬giraffe(clyde), likes(clyde,leaves))	[3]
¬likes(clyde,leaves)	[4]

Approach 1: Forward Chaining Proof:

[3&4]	¬giraffe(clyde)	[5]
[5&1]	elephant(clyde)	[6]
[6&2]	likes(clyde,peanuts)	[7] ✓

Resolution Proofs Example

(elephant(clyde), giraffe(clyde))	[1]
(¬elephant(clyde), likes(clyde,peanuts))	[2]
(¬giraffe(clyde), likes(clyde,leaves))	[3]
¬likes(clyde,leaves)	[4]

Approach 2: Refutation Proof:

(negated query)	¬likes(clyde,peanuts)	[5]
[5&2]	¬elephant(clyde)	[6]
[6&1]	giraffe(clyde)	[7]
[7&3]	likes(clyde,leaves)	[8]
[8&4]	() ✓	

Resolution Proofs

Proofs by refutation are generally easier to find.

They're more focused to the particular conclusion we are trying to reach.

To develop a resolution proof procedure for First-Order logic we need :

1. A way of converting KB and f (the query) into clausal form.
2. A way of doing resolution even when we have variables.

   This is called **unification.**

Conversion to Clausal Form

To convert the KB into Clausal form we perform the following 8-step procedure:

1. **Eliminate Implications.**
2. **Move Negations inwards (and simplify ¬¬).**
3. **Standardize Variables.**
4. **Skolemize.**
5. **Convert to Prenex Form.**
6. **Distribute disjunctions over conjunctions.**
7. **Flatten nested conjunctions and disjunctions.**
8. **Convert to Clauses.**

C-T-C-F: Eliminate implications

We use this example to show each step:

$$\forall X.p(X) \rightarrow \Big(\quad \forall Y.p(Y) \rightarrow p(f(X,Y)) \\ \land \neg(\forall Y.\ \neg q(X,Y) \land p(Y))\Big)$$

1. Eliminate implications: $A \rightarrow B$ ➔ $\neg A \lor B$

$$\forall X.\ \neg p(X) \\ \lor \Big(\quad \forall Y.\neg p(Y) \lor p(f(X,Y)) \\ \land \neg(\forall Y.\ \neg q(X,Y) \land p(Y))\ \Big)$$

C-T-C-F: Move ¬ Inwards

$$\forall X.\ \neg p(X) \\ \quad \vee \Big(\quad \forall Y. \neg p(Y) \vee p(f(X,Y)) \\ \qquad \wedge \neg(\forall Y.\ \neg q(X,Y) \wedge p(Y)) \Big)$$

2. Move Negations Inwards (and simplify ¬¬)

$$\forall X.\ \neg p(X) \\ \quad \vee \Big(\quad \forall Y. \neg p(Y) \vee p(f(X,Y)) \\ \qquad \wedge \exists Y.\ q(X,Y) \vee \neg p(Y) \Big)$$

C-T-C-F: : ¬ continue…

Rules for moving negations inwards

- $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$
- $\neg(A \vee B) \rightarrow \neg A \wedge \neg B$
- $\neg\forall X.\ f \rightarrow \exists X.\ \neg f$
- $\neg\exists X.\ f \rightarrow \forall X.\ \neg f$
- $\neg\neg A \rightarrow A$

C-T-C-F: Standardize Variables

$$\forall X.\ \neg p(X)$$
$$\quad \vee \Big(\ \ \forall Y. \neg p(Y) \vee p(f(X,Y))$$
$$\qquad \wedge\ \exists Y. q(X,Y) \vee \neg p(Y)\ \Big)$$

3. Standardize Variables (Rename variables so that each quantified variable is unique)

$$\forall X.\ \neg p(X)$$
$$\quad \vee \Big(\ \ \forall Y. (\neg p(Y) \vee p(f(X,Y))$$
$$\qquad \wedge\ \exists Z. q(X,Z) \vee \neg p(Z)\ \Big)$$

C-T-C-F: Skolemize

$$\forall X.\ \neg p(X)$$
$$\quad \vee \Big(\forall Y. \neg p(Y) \vee p(f(X,Y))$$
$$\qquad \wedge\ \exists Z. q(X,Z) \vee \neg p(Z) \Big)$$

4. Skolemize (Remove existential quantifiers by introducing new function symbols).

$$\forall X.\ \neg p(X)$$
$$\quad \vee \Big(\forall Y. \neg p(Y) \vee p(f(X,Y))$$
$$\qquad \wedge\ q(X,g(X)) \vee \neg p(g(X)) \Big)$$

C-T-C-F: Skolemization continued…

Consider $\exists Y.elephant(Y) \land friendly(Y)$

- This asserts that there is some individual (binding for Y) that is both an elephant and friendly.

- To remove the existential, we **invent** a name for this individual, say ***a*****.** This is a new constant symbol not equal to any previous constant symbols to obtain:

  $elephant(\mathbf{a}) \land friendly(\mathbf{a})$

- This is saying the same thing, since we do not know anything about the new constant **a**. Recall that a constant is a 0-ary function symbol.

C-T-C-F: Skolemization continue

- It is essential that the introduced symbol “a” is **new.** Else we might know something else about “a” in KB.
- If we did know something else about “a” we would be asserting more than the existential.
- In the original quantified formula we know nothing about the variable “Y”. Just what was being asserted by the existential formula.

C-T-C-F: Skolemization continue

Now consider $\forall X \exists Y.\ loves(X,Y).$

- This formula claims that for every X there is some Y that X loves (perhaps a different Y for each X).
- Replacing the existential by a new constant won't work
  $\forall X.loves(X,\mathbf{a}).$

  Because this asserts that there is a **particular** individual "a" loved by every X.
- To properly convert existential quantifiers scoped by universal quantifiers we must use **functions** not just constants.

C-T-C-F: Skolemization continue

Recall we are considering the example

$$\forall X \exists Y.\ loves(X,Y)$$

- We must use a function that mentions every universally quantified variable that scopes the existential.
- In this case X scopes Y so we must replace the existential Y by a function of X

$$\forall X.\ loves(X,g(X)).$$

where g is a **new** function symbol.

- This formula asserts that for every X there is some individual (given by g(X)) that X loves. g(X) can be different for each different binding of X.

C-T-C-F: Skolemization Examples

- $\forall XYZ\ \exists W.r(X,Y,Z,W) \rightarrow ?$

- $\forall XY \exists W.r(X,Y,g(W)) \rightarrow ?$

- $\forall XY \exists W \forall Z.r(X,Y,W) \land q(Z,W) \rightarrow ?$

C-T-C-F: Skolemization Examples

- $\forall XYZ\ \exists W.r(X,Y,Z,W) \rightarrow$

  $\forall XYZ.r(X,Y,Z,h1(X,Y,Z))$

- $\forall XY\exists W.r(X,Y,g(W)) \rightarrow$

  $\forall XY.r(X,Y, g(h2(X,Y)))$

- $\forall XY\exists W\forall Z.r(X,Y,W) \land q(Z,W) \rightarrow$

  $\forall XYZ.r(X,Y,h3(X,Y)) \land q(Z,h3(X,Y))$

C-T-C-F: Convert to prenex

$\forall X.\ \neg p(X)$
$\quad \lor (\ \forall Y. \neg p(Y) \lor p(f(X,Y))$
$\qquad \land\ q(X,g(X)) \lor \neg p(g(X))\ )$

5. Convert to prenex form. (Bring all quantifiers to the front—only universals, each with different name).

$\forall X \forall Y.\ \neg p(X)$
$\quad \lor (\neg p(Y) \lor p(f(X,Y))$
$\qquad \land\ q(X,g(X)) \lor \neg p(g(X))\ )$

C-T-C-F: disjunctions over conjunctions

$\forall X \forall Y.\ \neg p(X)$

$\quad \vee \big(\neg p(Y) \vee p(f(X,Y))$

$\qquad \wedge\ q(X,g(X)) \vee \neg p(g(X))\ \big)$

6. Disjunction over Conjunction

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

$\forall XY.\quad \neg p(X) \vee \neg p(Y) \vee p(f(X,Y))$

$\quad \wedge\ \neg p(X) \vee q(X,g(X)) \vee \neg p(g(X))$

C-T-C-F: flatten & convert to clauses

7. Flatten nested conjunctions and disjunctions.

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

8. Convert to Clauses

(remove quantifiers and break apart conjunctions).

$\forall XY.\ \neg p(X) \vee \neg p(Y) \vee p(f(X,Y))$
$\wedge \neg p(X) \vee q(X,g(X)) \vee \neg p(g(X))$

a) $\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))$
b) $\neg p(X) \vee q(X,g(X)) \vee \neg p(g(X))$

Unification

- Ground clauses are clauses with no variables in them. For ground clauses we can use syntactic identity to detect when we have a P and ¬P pair.

- What about variables? Can the clauses

  (P(john), Q(fred), R(X))

  (¬P(Y), R(susan), R(Y))

  be resolved?

Unification.

- Intuitively, once reduced to clausal form, all remaining variables are universally quantified. So, implicitly the clause

  (¬P(Y), R(susan), R(Y))

  represents clauses like

  (¬P(fred), R(susan), R(fred))

  (¬P(john), R(susan), R(john))

  …
- So there is a "specialization" of (¬P(Y), R(susan), R(Y)) that can be resolved w/ (a specialization of) (P(john), Q(fred), R(X)) from the prev page
- In particular,

  (¬P(john), R(susan), R(john)) can be resolved with

  (P(john), Q(fred), R(john)) producing the new clause

  (R(susan),R(john),Q(fred))

Unification.

- We want to be able to match conflicting literals, even when they have variables. This matching process automatically determines whether or not there is a "specialization" that matches.
- But, we don't want to over specialize!

Unification.

Consider the following example

(¬p(X), s(X), q(fred))

(p(Y), r(Y))

We need to make ¬p(X) *and* p(Y) *look the same so we can resolve them away, producing a new clause* (s(?), q(fred), r(??)).

How do we do this?

This says "substitute the constant "john" for the variable Y and the constant "john" for the variable X."

Possible resolvants

- (s(john), q(fred), r(john)) {Y=john, X=john}
- (s(sally), q(fred), r(sally)) {Y=sally, X=sally}
- (s(X), q(fred), r(X)) {Y=X}

🡐 the most-general

- The last resolvant is "most-general", the other two are specializations.
- We want the most general clause for use in future resolution steps.

Unification

Unification is a mechanism for finding a “most general” matching

A key component of unification is substitution.

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

  (V = t)

  where V is a variable and t is a term <u>not containing</u> V. (t might contain other variables).

Substitutions.

- We can apply a substitution σ to a formula f to obtain a new formula $f\sigma$ by simultaneously replacing every variable mentioned in the left hand side of the substitution by the right hand side.

E.g., substituting variable Y for X and f(a) for variable Y:

$$p(X,g(Y,Z))\{X=Y, Y=f(a)\} \rightarrow p(Y,g(f(a),Z))$$

- **IMPORTANT:** Note that the substitutions are not applied sequentially, i.e., the first Y is not subsequently replaced by f(a).

Substitutions: Composition of Substitutions

We can **compose two substitutions** θ and σ to obtain a new substitution $\theta\sigma$. (This is sequential.)

Let $\theta = \{X_1=s_1, X_2=s_2, \ldots, X_m=s_m\}$

$\sigma = \{Y_1=t_1, Y_2=t_2, \ldots, Y_k=t_k\}$

Step 1 to compute the composition $\theta\sigma$

- we apply σ to each <u>right-hand side</u> of θ, and then
- add all of the equations of σ.

Resulting in:

$$S = \{X_1=s_1\underbrace{\sigma}_{\{Y_1=t_1,\ Y_2=t_2,\ \ldots,\ Y_k=t_k\}},\ X_2=s_2\sigma,\ \ldots,\ X_m=s_m\sigma,\ Y_1=t_1,\ Y_2=t_2, \ldots,\ Y_k=t_k\}$$

Substitutions: Composition of Substitutions

Three Steps to compute a composition of substitutions $\theta\sigma$, given

$\theta = \{X_1=s_1, X_2=s_2, \ldots, X_m=s_m\}$

$\sigma = \{Y_1=t_1, Y_2=t_2, \ldots, Y_k=t_k\}$

1. Construct the composition as on the last page

   $S = \{X_1=s_1\sigma, X_2=s_2\sigma, \ldots, X_m=s_m\sigma, Y_1=t_1, Y_2=t_2, \ldots, Y_k=t_k\}$
2. Delete any identities, i.e., equations of the form V=V.
3. Delete any equation $Y_i=s_i$ where Y_i is equal to one of the X_j in θ. Why?

The final set S is the composition $\theta\sigma$.

Composition Example

θ = {X=f(Y), Y=Z}, σ = {X=a, Y=b, Z=Y}

Compute $\theta\sigma$

1. Construct the composition as on the last page
   S = {$X_1=s_1\sigma$, $X_2=s_2\sigma$, …, $X_m=s_m\sigma$, $Y_1=t_1$, $Y_2=t_2$,…, $Y_k=t_k$}
2. Delete any identities, i.e., equations of the form V=V.
3. Delete any equation $Y_i=s_i$ where Y_i is equal to one of the X_j in θ.

Step 1

S = {X=f(Y){X=a, Y=b, Z=Y} , Y=Z{X=a, Y=b, Z=Y} , X=a, Y=b, Z=Y}

= {X=f(b),Y=Y,X=a,Y=b,Z=Y}

Step 2

= {X=f(b),~~Y=Y~~,X=a,Y=b,Z=Y} = {X=f(b),X=a,Y=b,Z=Y}

Step 3

= {X=f(b),~~X=a,Y=b~~,Z=Y} = {X=f(b),Z=Y}

Substitutions.

- The empty substitution $\varepsilon = \{\}$ is also a substitution, and it acts as an identity under composition.

- More importantly substitutions when applied to formulas are associative:

$$(f\theta)\sigma = f(\theta\sigma)$$

- **Composition** is simply a way of converting the sequential application of a series of substitutions to a single simultaneous substitution.

Unifiers

- A unifier of two formulas f and g is a substitution σ that makes f and g syntactically identical.
- Not all formulas can be unified—substitutions only affect variables.

  p(f(X),a)   p(Y,f(w))

- This pair cannot be unified as there is no way of making a = f(w) with a substitution.

MGU

A substitution σ of two formulas f and g is a Most General Unifier (MGU) if

1. σ is a unifier.
2. For every **other** unifier θ of f and g there must exist a third substitution λ such that

$$\theta = \sigma\lambda$$

This says that every other unifier is “more specialized” than σ. The MGU of a pair of formulas f and g is unique up to renaming.

MGU

$p(f(X),Z) \quad p(Y,a)$

1. $\sigma = \{Y = f(a), X=a, Z=a\}$ is a unifier.

   $p(f(X),Z)\sigma =$
   $p(Y,a)\sigma \quad =$

   But it is <u>not</u> an MGU.

2. $\theta = \{Y=f(X), Z=a\}$ is an MGU.
   $p(f(X),Z)\,\theta =$
   $p(Y,a)\,\theta \quad =$

MGU

p(f(X),Z)   p(Y,a)

1. σ = {Y = f(a), X=a, Z=a} is a unifier.

   p(f(X), Z)σ = p(f(a),a)
   p(Y , a)σ = p(f(a),a)

   But it is not an MGU.

2. θ = {Y=f(X), Z=a} is an MGU.
   p(f(X), Z) θ =
   p(Y , a) θ =

MGU

p(f(X),Z)   p(Y,a)

1. σ = {Y = f(a), X=a, Z=a} is a unifier.

   p(f(X),Z)σ = p(f(a),a)
   p(Y,a)σ = p(f(a),a)

   But it is <u>not</u> an MGU.

2. θ = {Y=f(X), Z=a} is an MGU.
   p(f(X), Z) θ = p(f(X),a)
   p(Y , a) θ = p(f(X),a)

MGU

p(f(X),Z)   p(Y,a)

From the previous page we noted that

θ = {Y=f(X), Z=a} is an MGU.

σ = {Y = f(a), X=a, Z=a} is a unifier but <u>not</u> an MGU.

We can show that unifier σ is not an MGU by observing that σ can be constructed by composing MGU θ with another substitution λ i.e., $\sigma = \theta\lambda$, where λ={X=a}

θ = {Y=f(X), Z=a}
λ = {X=a}

Recall θ = {Y=f(X), Z=a}

Thus $\theta\lambda = \sigma =$ {Y = f(a), X=a, Z=a}

MGU

- The MGU is the “least specialized” way of making clauses with universal variables match.
- We can compute MGUs.
- Intuitively we line up the two formulas and find the first sub-expression where they disagree. The pair of subexpressions where they first disagree is called the disagreement set.
- The algorithm works by successively fixing disagreement sets until the two formulas become syntactically identical.

MGU

To find the MGU of two formulas f and g.

1. $k = 0$; $\sigma_0 = \{\}$; $S_0 = \{f,g\}$
2. If S_k contains an identical pair of formulas stop, and return σ_k as the MGU of f and g.
3. Else find the disagreement set $D_k=\{e_1,e_2\}$ of S_k
4. If $e_1 = V$ a variable, and $e_2 = t$ a term not containing V (**or vice-versa**) then let
   $\sigma_{k+1} = \sigma_k \{V=t\}$ (Compose** the additional substitution)
   $S_{k+1} = S_k\{V=t\}$ (Apply the additional substitution)
   $k = k+1$
   GOTO 2
5. Else stop, f and g cannot be unified.

*** Note that this is compose not conjoin!*

MGU Example 1

S_0 = {p(f(a), g(X)) ; p(Y,Y)}

MGU Example 2

$S_0 = \{p(a,X,h(g(Z))) \;\; ; \;\; p(Z,h(Y),h(Y))\}$

MGU Example 3

$S_0 = \{p(X,X) \;\; ; \;\; p(Y,f(Y))\}$

Non-Ground Resolution

- Resolution of non-ground clauses. From the two clauses

  (L, Q1, Q2, …, Qk)

  (¬M, R1, R2, …, Rn)

  Where there exists σ a MGU for L and M.

  We infer the new clause

  (Q1σ, …, Qkσ, R1σ, …, Rnσ)

Non-Ground Resolution Example

1. (p(X), q(g(X)))
2. (r(a), q(Z), ¬p(a))

   L=p(X); M=p(a)
   σ = {X=a}

3. R[1a,2c]{X=a} (q(g(a)), r(a), q(Z))

The notation is important.

- "R" means resolution step.
- "1a" means the first (a-th) literal in the first clause i.e. p(X).
- "2c" means the third (c-th) literal in the second clause, ¬p(a).
  - 1a and 2c are the "clashing" literals.
- {X=a} is the substitution applied to make the clashing literals identical.

Resolution Proof Example

“Some patients like all doctors. No patient likes any quack. Therefore no doctor is a quack.”

Step 1: English -> FOL-> Clausal form

1A) Pick symbols to represent these assertions.

p(X): X is a patient

d(X): X is a doctor

q(X): X is a quack

l(X,Y): X likes Y

Resolution Proof Example

Step 1: English -> FOL-> Clausal form

1B) Convert each assertion to a first-order formula.

Some patients like all doctors.

???? [F1]

Resolution Proof Example

Step 1: English -> FOL-> Clausal form

1B) Convert each assertion to a first-order formula.

Some patients like all doctors.

$\exists X.(p(X) \land \forall Y.(d(Y) \rightarrow l(X,Y)))$ [F1]

Resolution Proof Example

Step 1: English -> FOL-> Clausal form

1B) Convert each assertion to a first-order formula.

Some patients like all doctors.

$\exists X.(p(X) \land \forall Y.(d(Y) \rightarrow l(X,Y)))$ [F1]

No patient likes any quack

???? [F2]

Therefore no doctor is a quack.

???? [Query]

Resolution Proof Example

Step 1: English -> FOL-> Clausal form

1B) Convert each assertion to a first-order formula.

Some patients like all doctors.

$\exists X.(p(X) \land \forall Y.(d(Y) \rightarrow l(X,Y)))$ [F1]

No patient likes any quack

$\forall X.\left(p(X) \rightarrow \forall Y.\left(q(Y) \rightarrow \neg l(X,Y) \right) \right)$ [F2]

Therefore no doctor is a quack.

$\neg \exists X.(d(X) \land q(X))$ [Query]

$\exists X.(d(X) \land q(X))$ [Negated Query—for refutation proof)]

Conversion to Clausal Form

To convert the KB into Clausal form we perform the following 8-step procedure:

1. **Eliminate Implications.**
2. **Move Negations inwards (and simplify ¬¬).**
3. **Standardize Variables.**
4. **Skolemize.**
5. **Convert to Prenex Form.**
6. **Distribute disjunctions over conjunctions.**
7. **Flatten nested conjunctions and disjunctions.**
8. **Convert to Clauses.**

Resolution Proof Example

Step 2. Resolution Refuation Proof from the clauses. (try to derive () – the empty clause)

1. p(a)
2. (¬d(Y), l(a,Y))
3. (¬p(Z), ¬q(R), ¬l(Z,R))
4. d(b)
5. q(b)

Answer Extraction

- The previous example shows how we can answer true-false questions. With a bit more effort we can also answer **“fill-in-the-blanks”** questions (e.g., what is wrong with the car?).

- For those who know Prolog, the strategy is similar. We use free variables in the query where we want to fill in the blanks. We simply need to keep track of the <u>binding</u> that these variables received in proving the query.
  - parent(art, jon) – is art one of jon’s parents?
  - parent(X, jon) – who is one of jon’s parents?

Answer Extraction.

A simple bookkeeping device is to use a predicate symbol answer(X,Y,…) to keep track of the bindings automatically.

To answer the query parent(X,jon) and extract the name of jon’s parent, i.e., to answer the question “who is the parent of jon”

1. construct the clause that contains the negation of the query together with the answer predicate over the variables we wish to know the binding of:
   (¬ parent(X,jon), answer(X))
2. perform resolution until we obtain a clause consisting of **only of answer literals** (previously we stopped at empty clauses).

Answer Extraction: Example 1

1. father(art, jon)
2. father(bob,kim)
3. (¬father(Y,Z), parent(Y,Z)) i.e. *all fathers are parents*
4. (¬ parent(X,jon), answer(X)) i.e. the query is: who is parent of jon?

Here is a resolution proof:*

5. R[4,3b]{Y=X,Z=jon}
   (¬father(X,jon), answer(X))
6. R[5,1]{X=art} answer(art)

so art is parent of jon

* Remember a good strategy is to start a refutation proof by resolving the query since the source of the inconsistency you're searching for is derived from the the negated query.

Answer Extraction: Example 2

1. (father(art, jon), father(bob,jon) *// either bob or art is parent of jon*
2. father(bob,kim)
3. (¬father(Y,Z), parent(Y,Z)) *// all fathers are parents*
4. (¬ parent(X,jon), answer(X)) *// query is parent(X,jon)*

Here is a resolution proof:

5. R[4,3b]{Y=X,Z=jon} (¬father(X,jon), answer(X))
6. R[5,1a]{X=art} (father(bob,jon), answer(art))
7. R[6,3a] {Y=bob,Z=jon}
   (parent(bob,jon), answer(art))
8. R[7,4] {X=bob} (answer(bob), answer(art))

A disjunctive answer: either bob or art is parent of jon.

Factoring

1. (p(X), p(Y)) // i.e., ∀ X.∀ Y. ¬p(X) → p(Y)
2. (¬p(V), ¬p(W)) // i.e., ∀ V.∀W. p(V) → ¬p(W)

- These clauses are **intuitively contradictory**, but following the strict rules of resolution only we obtain:

3. R[1a,2a](X=V) (p(Y), ¬p(W))

   Renaming variables: (p(Q), ¬p(Z))

4. R[3b,1a](X=Z) (p(Y), p(Q))

Problem: No way of generating empty clause!

Important: <u>Factoring is needed to make resolution **complete**</u>, without it resolution is incomplete!

Factoring (dealing with duplicate literals in a clause)

If two or more literals from the same clause C have an MGU θ, then Cθ **with all duplicate literals removed** is called a **factor** of C.

Example:

C = (p(X), p(f(Y)), ¬q(X))
θ = {X=f(Y)}

Cθ = (p(f(Y)),p(f(Y)),¬q(f(Y))) so (p(f(Y)),¬q(f(Y)) is a factor of C

Adding a factor of a clause can be a step of proof:

```
1.  (p(X), p(Y))              // potential duplicate literals
2.  (¬p(V), ¬p(W))            // potential duplicate literals
3.  f[1ab]{X=Y} p(Y)          // clause 3 is a factor of clause 1
4.  f[2ab]{V=W} ¬p(W)         // clause 4 is a factor of clause 2
5.  R[3,4]{Y=W} ().           // resolve the factors to complete the proof
```

Prolog*

Prolog search mechanism *(without not and cut)* is simply an instance of resolution, except

1. Clauses are Horn (only one positive literal)
2. Prolog uses a specific depth first strategy when searching for a proof. (Rules are used first mentioned first used, literals are resolved away left to right).

* You will not be tested on this material

Prolog* (An aside for those who know Prolog)

The Prolog Predicate "Append":

1. append([], Z, Z)
2. append([E1 | R1], Y, [E1 | Rest]) :-
   append(R1, Y, Rest)

Note:

- 2 is actually the clause
  (append([E1|R1], Y, [E1|Rest]) , ¬append(R1,Y,Rest))
- [ ] is a constant (the empty list)
- [X | Y] is cons(X,Y).
- So [a,b,c] is short hand for cons(a,cons(b,cons(c,[])))

* You will not be tested on this material

Prolog*: Example of proof

Try to prove:

append([a,b], [c,d], [a,b,c,d]):

1. append([], Z, Z)
2. (append([E1|R1], Y, [E1|Rest]),
   ¬append(R1,Y,Rest))
3. ¬append([a,b], [c,d], [a,b,c,d])

4. R[3,2a]{E1=a, R1=[b], Y=[c,d], Rest=[b,c,d]}
   ¬append([b], [c,d], [b,c,d])
5. R[4,2a]{E1=b, R1=[], Y=[c,d], Rest=[c,d]}
   ¬append([], [c,d], [c,d])
6. R[5,1]{Z=[c,d]} ()

* You will not be tested on this material

Review: One Last Example!

Consider the following English description

- Whoever can read is literate.
- Dolphins are not literate.
- Flipper is an intelligent dolphin.

- Who is intelligent but cannot read.

Example:

Step 1: English -> FOL-> Clausal form
1A) Pick symbols to represent these assertions.
1B) Convert each assertion to a first-order formula.
1C) Negate Query & Convert to Clausal Form (the 8 steps)

- Whoever can read is literate.
  $\forall$ X. read(X) $\rightarrow$ lit(X)
- Dolphins are not literate.
  $\forall$ X. dolp(X) $\rightarrow \neg$ lit(X)
- Flipper is an intelligent dolphin
  dolp(flipper) $\wedge$ intell(flipper)

- Who is intelligent but cannot read?
  $\exists$ X. intell(X) $\wedge \neg$ read(X).

Example: convert to clausal form

Step 1: English -> FOL-> Clausal form
1A) Pick symbols to represent these assertions.
1B) Convert each assertion to a first-order formula.
1C) Negate Query & Convert to Clausal Form (the 8 steps)

- Whoever can read is literate.
  $\forall X.\ read(X) \rightarrow lit(X)$
  $(\neg read(X), lit(X))$
- Dolphins are not literate.
  $\forall X.\ dolp(X) \rightarrow \neg\ lit(X)$
  $(\neg dolp(X), \neg lit(X))$
- Flipper is an intelligent dolphin.
  $dolp(flipper)$
  $intell(flipper)$
- Who are intelligent but cannot read?
  $\exists\ X.\ intell(X) \wedge \neg read(X).$
  Negated Query ➔ $\forall\ X.\ \neg\ intell(X) \vee read(X)$
  Clausal Form➔ $(\neg intell(X), read(X), answer(X))$

Example: do the resolution proof

Step 1: English -> FOL-> Clausal form
1A) Pick symbols to represent these assertions.
1B) Convert each assertion to a first-order formula.
1C) Negate Query & Convert to Clausal Form (the 8 steps)
2) Do the resolution proof to derive ()

1. (¬read(X), lit(X))
2. (¬dolp(X), ¬lit(X))
3. dolp(flip)
4. intell(flip)
5. (¬intell(X), read(X),answer(X))

6. R[5a,4] X=flip. (read(flip), answer(flip))
7. R[6,1a] X=flip. (lit(flip), answer(flip))
8. R[7,2b] X=flip. (¬dolp(flip), answer(flip))
9. R[8,3] answer(flip)

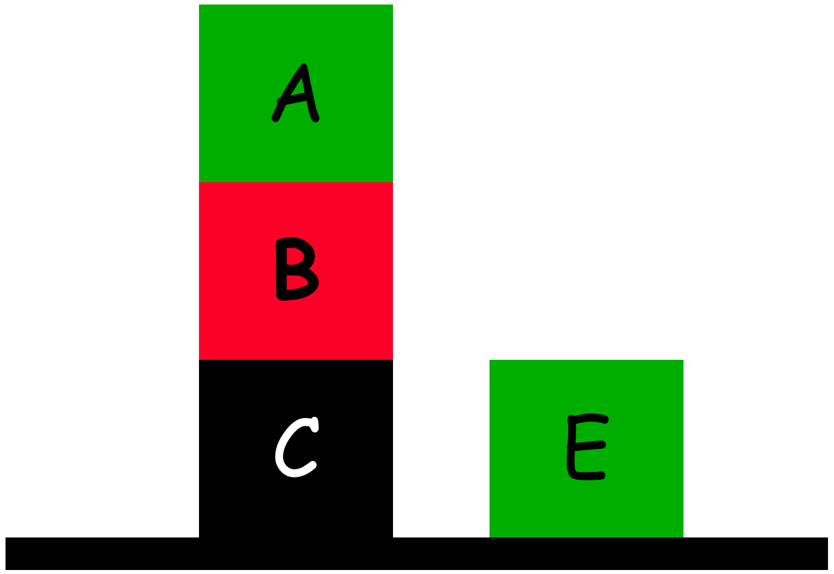
so flip is intelligent but cannot read!

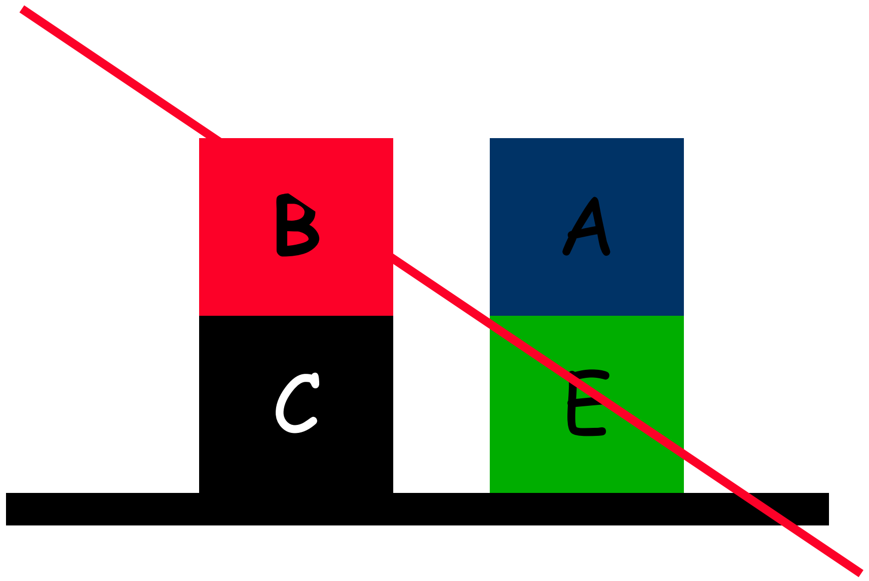
Review

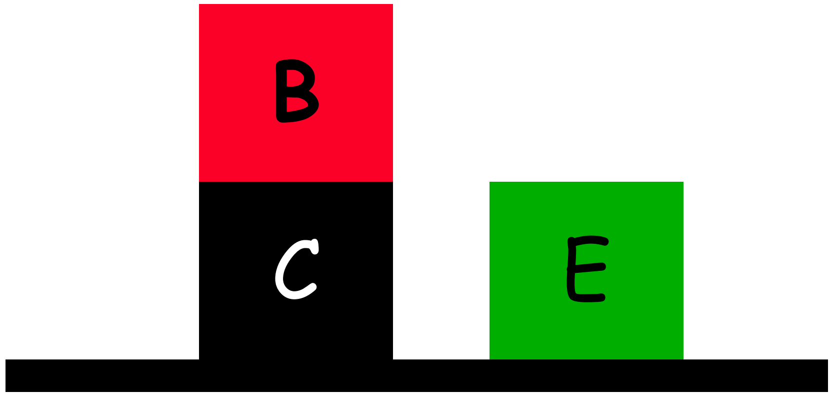
KB—many models

KB

1. on(b,c)
2. clear(e)







Desired Properties of Reasoning

Soundness

If KB $\vdash$ f   then   KB $\models$ f

- i.e all conclusions arrived at via the proof procedure are correct -- they are logical consequences of the KB.

Completeness

If KB $\models$ f   then KB $\vdash$ f

- i.e. every logical consequence of the KB can be generated by the proof procedure.

Note proof procedures are computable, but they might have very high complexity in the worst case. So completeness is not necessarily achievable in practice.

Resolution Proof: Forward chaining

Logical consequences can be generated from the resolution rule in two ways:

1. Forward chaining inference ("Consequence Finding")
   - If we have a sequence of clauses C1, C2, …, Ck
   - Such that each Ci is either in KB or is the result of a resolution step involving two prior clauses in the sequence.
   - We then have that KB $\vdash$ Ck.

   Forward chaining is sound so we also have KB $\models$ Ck

Resolution Proof: Refutation proofs

2. Refutation proofs
   - We determine if KB ⊢ f by showing that a contradiction can be generated from KB ∧ ¬f.
   - In this case a contradiction is an empty clause ().
   - We employ resolution to construct a sequence of clauses C1, C2, …, Cm such that
     - Ci is in KB ∧ ¬f, or is the result of resolving two previous clauses in the sequence.
     - Cm = () i.e. its the empty clause.

Conversion to Clausal Form

To convert the KB into Clausal form we perform the following 8-step procedure:

1. **Eliminate Implications.**
2. **Move Negations inwards (and simplify ¬¬).**
3. **Standardize Variables.**
4. **Skolemize.**
5. **Convert to Prenex Form.**
6. **Distribute disjunctions over conjunctions.**
7. **Flatten nested conjunctions and disjunctions.**
8. **Convert to Clauses.**

C-T-C-F: Skolemization Examples

- $\forall XYZ\ \exists W.r(X,Y,Z,W) \rightarrow \forall XYZ.r(X,Y,Z,h1(X,Y,Z))$
- $\forall XY \exists W.r(X,Y,g(W)) \rightarrow \forall XY.r(X,Y,g(h2(X,Y)))$
- $\forall XY \exists W \forall Z.r(X,Y,W) \land q(Z,W)$

  $\rightarrow \quad \forall XYZ.r(X,Y,h3(X,Y)) \land q(Z,h3(X,Y))$