CSC384
Knowledge Representation
Part 2

Bahar Aameri & Sonya Allin

Summer 2020

Credits

We gratefully acknowledge those who have contributed to these slides, most recently Bahar Aameri, who merged and augmented slides from Yongmei Liu and a CSC384 slide deck historically developed by Craig Boutilier, Fahiem Bacchus, Sheila McIlraith, Sonya Allin, Hojjat Ghaderi, and others. We also acknowledge the use of material written by Michael Winter, and the use of material originating from slides and the book by Ron Brachman and Hector Levesque.

Logical Consequence

Let Φ be a set of sentences and A be a sentence.
A is a **logical consequence** of Φ (denoted by $\Phi \models A$) iff for every structure $\mathcal{M}$,
if $\mathcal{M} \models \Phi$ then $\mathcal{M} \models A$.

If A is a logical consequence of Φ, then there is no $\mathcal{M}$ such that $\mathcal{M} \models \Phi \cup \{\neg A\}$.
In other words, $\Phi \cup \{\neg A\}$ is **unsatisfiable**.

Example:
Assume Φ includes the following sentences:
$\forall x \forall y \forall z[(above(z, y) \wedge above(y, x)) \rightarrow above(z, x)]$
$above(\boldsymbol{c_1}, \boldsymbol{c_2}) \wedge above(\boldsymbol{c_2}, \boldsymbol{c_3})$

Knowledge Base (KB): A collection of sentences that represents what the agent/program believes about the world.

Sentences in the KB are **explicit** knowledge of the agent.
Logical consequences of the KB are **implicit** knowledge of the agent.

Example: Suppose **KB** includes the following sentences:

- The capital of Canada is Ottawa
- The largest province in Canada is Quebec
- The provinces neighbouring Quebec are Ontario, New Brunswick, and Newfoundland

Implicit knowledge of the KB:
Ontario, New Brunswick and Newfoundland are the neighbouring provinces of the largest province in Canada.

Proof Procedures

- To compute implicit knowledge of the KB (i.e., logical consequences) we need a **mechanical** procedure that can be implemented as an **algorithm**.

- This would allow us to **reason** with our **knowledge**:

  - Represent the knowledge as **logical formulas**.

  - Apply the **procedure** for generating logical consequences

- Mechanical proof procedures work by **manipulating formulas**.
  They do not know or care anything about interpretations.
  Nevertheless they **respect the semantics** of interpretations!

Proof Procedures

A proof procedure is **sound** if whenever it **produces** a sentence A by manipulating sentences in a KB, then A is a **logical consequence** of KB (i.e., $KB \models A$).
That is, **all conclusions** arrived at via the proof procedure are **correct**: they are logical consequences.

A proof procedure is **complete** if it can produce **all logical consequences** of KB.
That is, if $KB \models A$, then the procedure can produce A.

We will develop a sound and complete proof procedure for first-order logic called **Resolution**.

Resolution

Resolution works with formulas expressed in clausal form.

A **literal** is an **atomic formula** or the **negation** of an atomic formula.
Example: $dog(\boldsymbol{fido}), \neg cat(\boldsymbol{fido}), P(x), \neg Q(y)$

A **clause** is a **disjunction** of **literals**:
Example:
$P(x) \vee \neg Q(x, y)$
$\neg owns(\boldsymbol{fido}, \boldsymbol{fred}) \vee \neg dog(\boldsymbol{fido}) \vee person(\boldsymbol{fred})$

A **clausal theory** is a **conjunction** of **clauses**.
Example:
$\big(P(x) \vee \neg Q(x, y)\big) \wedge$
$\big(\neg owns(\boldsymbol{fido}, \boldsymbol{fred}) \vee \neg dog(\boldsymbol{fido}) \vee person(\boldsymbol{fred})\big)$

Resolution

The **resolution** proof procedure uses only one **inference rule**:

$\big(Q(x, y) \vee P(\boldsymbol{a})\big)$ and $\big(R(y) \vee \neg P(\boldsymbol{a})\big)$

$\big(Q(x, y) \vee P(\boldsymbol{a})\big)$ and $\neg P(\boldsymbol{a})$

$P(\boldsymbol{a})$ and $\neg P(\boldsymbol{a})$

We denote a contradiction by an empty clause: $()$

Resolution by Refutation

Resolution by Refutation:

- Assume $\neg A$ is true to generate a contradiction. (**Refutation**)
- Convert $\neg A$ and all sentences in KB to a clausal theory C.
- Resolve the clauses in C until an empty clause is obtained.

Want to prove likes(**clyde**,**peanuts**) from:

1. $elephant(\boldsymbol{clyde}) \vee giraffe(\boldsymbol{clyde})$
2. $\neg elephant(\boldsymbol{clyde}) \vee likes(\boldsymbol{clyde}, \boldsymbol{peanuts})$
3. $\neg giraffe(\boldsymbol{clyde}) \vee likes(\boldsymbol{clyde}, \boldsymbol{leaves})$
4. $\neg likes(\boldsymbol{clyde}, \boldsymbol{leaves})$

Assume: $5. \quad \neg likes(\boldsymbol{clyde}, \boldsymbol{peanuts})$

$\neg likes(\boldsymbol{clyde}, \boldsymbol{peanuts})$ $\qquad$ $\neg elephant(\boldsymbol{clyde}) \vee likes(\boldsymbol{clyde}, \boldsymbol{peanuts})$

$\neg elephant(\boldsymbol{clyde})$ $\qquad$ $elephant(\boldsymbol{clyde}) \vee giraffe(\boldsymbol{clyde})$

$giraffe(\boldsymbol{clyde})$ $\qquad$ $\neg giraffe(\boldsymbol{clyde}) \vee likes(\boldsymbol{clyde}, \boldsymbol{leaves})$

$likes(\boldsymbol{clyde}, \boldsymbol{leaves})$ $\qquad$ $\neg likes(\boldsymbol{clyde}, \boldsymbol{leaves})$

Resolution by Refutation: Example

Want to prove likes(**clyde,peanuts**) from:

1. $elephant(\boldsymbol{clyde}) \vee giraffe(\boldsymbol{clyde})$
2. $\neg elephant(\boldsymbol{clyde}) \vee likes(\boldsymbol{clyde, peanuts})$
3. $\neg giraffe(\boldsymbol{clyde}) \vee likes(\boldsymbol{clyde, leaves})$
4. $\neg likes(\boldsymbol{clyde, leaves})$

Resolution by Refutation Proof:

- $\neg likes(\boldsymbol{clyde, peanuts})[5.]$
- 5&2: $\neg elephant(\boldsymbol{clyde})[6.]$
- 6&1: $giraffe(\boldsymbol{clyde})[7.]$
- 7&3: $likes(\boldsymbol{clyde, leaves})[8.]$
- 8&4: ()

Resolution by Refutation

To develop a complete resolution proof procedure for first-order logic we need :

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

Conversion to Clausal Form

1. Eliminate Implications.
2. Move Negations Inwards (and simplify $\neg\neg$).
3. Standardize Variables.
4. Skolemization.
5. Convert to Prenex Form.
6. Distribute Conjunctions over Disjunctions.
7. Flatten nested Conjunctions and Disjunctions.
8. Convert to Clauses.

Eliminate Implications

Implication Rule: $A \to B$ iff $\neg A \lor B$

$$\forall x \Big[P(x) \to \Big((\forall y [P(y) \to P(f(x,y))]) \land \neg (\forall y [\neg q(x,y) \land P(y)]) \Big) \Big]$$

Eliminate Implication: $\forall x \Big[\neg P(x) \lor \Big((\forall y [\neg P(y) \lor P(f(x,y))]) \land \neg (\forall y [\neg q(x,y) \land P(y)]) \Big) \Big]$

Rules for Simplifying and Moving Negations Inwards

- $\neg\neg A$ iff A
- $\neg(A \wedge B)$ iff $\neg A \vee \neg B$
- $\neg(A \vee B)$ iff $\neg A \wedge \neg B$
- $\neg\forall x A$ iff $\exists x \neg A$
- $\neg\exists x A$ iff $\forall x \neg A$

Simplify and Move Negations Inwards

$$\forall x\Big[\neg P(x) \vee \Big(\big(\forall y[\neg P(y) \vee P(f(x,y))]\big) \wedge \neg\big(\forall y[\neg Q(x,y) \wedge P(y)]\big)\Big)\Big]$$

Move Negations Inwards:

$$\forall x\Big[\neg P(x) \vee \Big(\big(\forall y[\neg P(y) \vee P(f(x,y))]\big) \wedge \big(\exists y[\neg\neg Q(x,y) \vee \neg P(y)]\big)\Big)\Big]$$

Simplify Negations:

$$\forall x\Big[\neg P(x) \vee \Big(\big(\forall y[\neg P(y) \vee P(f(x,y))]\big) \wedge \big(\exists y[Q(x,y) \vee \neg P(y)]\big)\Big)\Big]$$

Standardize Variables

Standardize Variables: Rename variables so that each quantified variable is unique.

$$\forall x \Big[\neg P(x) \vee \Big((\forall y [\neg P(y) \vee P(f(x,y))]) \wedge (\exists y [Q(x,y) \vee \neg P(y)]) \Big) \Big]$$

$$\forall x \Big[\neg P(x) \vee \Big((\forall y [\neg P(y) \vee P(f(x,y))]) \wedge (\exists z [Q(x,z) \vee \neg P(z)]) \Big) \Big]$$

Skolemization: Remove existential quantifiers by introducing new function symbols.

$$\forall x \Big[\neg P(x) \vee \Big((\forall y [\neg P(y) \vee P(f(x,y))]) \wedge (\exists z [Q(x,z) \vee \neg P(z)]) \Big) \Big]$$

Skolemization

- Consider $\exists y(elephant(y) \wedge friendly(y))$

- 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 $\boldsymbol{a}$.
  This "name" must be a new constant symbol (not equal to any previous constant symbols in the vocabulary of the KB):

$$elephant(\boldsymbol{a}) \wedge friendly(\boldsymbol{a})$$

Skolemization

- Consider $\exists y(elephant(y) \wedge friendly(y))$
- 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 $\boldsymbol{a}$.
  This "name" must be a new constant symbol (not equal to any previous constant symbols in the vocabulary of the KB):

$$elephant(\boldsymbol{a}) \wedge friendly(\boldsymbol{a})$$

- The new sentence says the same thing, since we do not know anything about $\boldsymbol{a}$.
- **IMPORTANT:** The introduced symbol $\boldsymbol{a}$ must be **new**.
  Else we might know something else about $\boldsymbol{a}$ in KB.
  - If we did know something else about $\boldsymbol{a}$ we would be asserting more than the existential.
  - In original quantified formula we know nothing about the variable y. Just what was being asserted by the existential formula.

- Now consider

$$\forall x \exists y (loves(x, y))$$

This formula states that for **every** x there is **some** y that x loves (possibly a different y for each x).

- Replacing the existential by a new constant **won't work**

$$\forall x (loves(x, \boldsymbol{a}))$$

This asserts that there is a **particular individual** $\boldsymbol{a}$ loved by **every** x.

Skolemization

- Now consider

$$\forall x \exists y (loves(x, y))$$

  This formula states that for **every** x there is **some** y that x loves (possibly a different y for each x).

- Replacing the existential by a new constant **won't work**

$$\forall x (loves(x, \boldsymbol{a}))$$

  This asserts that there is a **particular individual** $\boldsymbol{a}$ loved by **every** x.

- To properly convert existential quantifiers **scoped by universal** quantifiers we must use **functions**:

  – Use a **new function symbol** that mentions every universally quantified variable that scopes the existential.

$$\forall x (loves(x, g(x)))$$

  where g is a **new** function symbol.
  This formula asserts that for every x there is some individual (denoted by $g(x)$) that x loves.

Skolemization: Examples

$\forall x \forall y \forall z \exists w (R(x, y, z, w))$

$\forall x \forall y \exists w (R(x, y, w))$

$\forall x \forall y \exists w \forall z (R(x, y, w) \land Q(z, w))$

Skolemization: Remove existential quantifiers by introducing new function symbols.

$$\forall x \Big[\neg P(x) \vee \Big((\forall y [\neg P(y) \vee P(f(x,y))]) \wedge (\exists z [Q(x,z) \vee \neg P(z)]) \Big) \Big]$$

$$\forall x \Big[\neg P(x) \vee \Big((\forall y [\neg P(y) \vee P(f(x,y))]) \wedge (Q(x,g(x)) \vee \neg P(g(x))) \Big) \Big]$$

Convert to Prenex Form

Convert to Prenex Form: Bring all quantifiers to the front.
We use the following equivalences, where x does not occur free in Q

- $\forall x P \land Q$ iff $Q \land \forall x P$ iff $\forall x (P \land Q)$
- $\forall x P \lor Q$ iff $Q \lor \forall x P$ iff $\forall x (P \lor Q)$

$$\forall x \Big[\neg P(x) \lor \Big((\forall y [\neg P(y) \lor P(f(x,y))]) \land \big(Q(x, g(x)) \lor \neg P(g(x))\big) \Big) \Big]$$

$$\forall x \forall y \Big[\neg P(x) \lor \Big((\neg P(y) \lor P(f(x,y))) \land \big(Q(x, g(x)) \lor \neg P(g(x))\big) \Big) \Big]$$

Distribute Conjunctions over Disjunctions

Conjunctions over Disjunctions: $A \lor (B \land C)$ iff $(A \lor B) \land (A \lor C)$

$$\forall x \forall y \Big[\neg P(x) \lor \Big((\neg P(y) \lor P(f(x,y))) \land (Q(x, g(x)) \lor \neg P(g(x))) \Big) \Big]$$

$$\forall x \forall y \Big[\Big(\neg P(x) \lor (\neg P(y) \lor P(f(x,y))) \Big) \land \Big(\neg P(x) \lor (Q(x, g(x)) \lor \neg P(g(x))) \Big) \Big]$$

Flatten nested Conjunctions and Disjunctions

Flatten nested $\land$ and $\lor$:

- $A \lor (B \lor C)$ to $(A \lor B \lor C)$
- $A \land (B \land C)$ to $(A \land B \land C)$

$$\forall x \forall y \Big[\Big(\neg P(x) \lor (\neg P(y) \lor P(f(x,y)))\Big) \land \Big(\neg P(x) \lor (Q(x,g(x)) \lor \neg P(g(x)))\Big)\Big]$$

$$\forall x \forall y \Big[\Big(\neg P(x) \lor \neg P(y) \lor P(f(x,y))\Big) \land \Big(\neg P(x) \lor Q(x,g(x)) \lor \neg P(g(x))\Big)\Big]$$

Convert to Clauses

Convert to Clauses: Remove universal quantifiers and break apart conjunctions

$$\forall x \forall y \Big[\Big(\neg P(x) \vee \neg P(y) \vee P(f(x,y)) \Big) \wedge \Big(\neg P(x) \vee Q(x, g(x)) \vee \neg P(g(x)) \Big) \Big]$$

- $\neg P(x) \vee \neg P(y) \vee P(f(x,y))$
- $\neg P(x) \vee Q(x, g(x)) \vee \neg P(g(x))$

Unification

- If clauses have no variables syntactic identity can be used to detect if a P and $\neg P$ exists.

- What about variables? Can the following clauses be resolved?
  $(P(\boldsymbol{john}), Q(\boldsymbol{fred}), R(x))$
  $(\neg P(y), R(\boldsymbol{susan}), R(y))$

  - Once reduced to clausal form, all remaining variables are universally quantified. So, implicitly $(\neg P(y), R(\boldsymbol{susan}), R(y))$ represents a whole set of clauses like
    $(\neg P(\boldsymbol{fred}), R(\boldsymbol{susan}), R(\boldsymbol{fred}))$
    $(\neg P(\boldsymbol{john}), R(\boldsymbol{susan}), R(\boldsymbol{john}))$
    ...

  - So there is a specialization of this clause that can be resolved with
    $(P(\boldsymbol{john}), Q(\boldsymbol{fred}), R(x))$

  - In particular
    $(P(\boldsymbol{john}), Q(\boldsymbol{fred}), R(\boldsymbol{john}))$ and $(\neg P(\boldsymbol{john}), R(\boldsymbol{susan}), R(\boldsymbol{john}))$
    can can be resolved, producing
    $(Q(\boldsymbol{fred}), R(\boldsymbol{john}), R(\boldsymbol{susan}))$

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

  - $(\neg P(x), S(x), Q(\boldsymbol{fred}))$
  - $(P(y), R(y))$

  Possible resolvants:

- The last resolvant is **most-general**, the other two are specializations of it.
  We want to keep the most general clause so that we can use it future resolution steps.

Substitution

- **Unification** is a mechanism for finding the 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 not containing V (t might contain other variables).
- We can apply a substitution $\delta = \{V_1 = t_1, ..., V_n = t_n\}$ to a formula A to obtain a new formula $A\delta$ by **simultaneously** replacing every variable V_i by term t_i.

Example: Applying $\delta = \{x = y, y = f(a)\}$ to $P(x, g(y, z))$

Note that the substitutions are NOT applied sequentially, i.e., the first y is not subsequently replaced by $f(a)$.

Composition of Substitutions

- We can compose two substitutions θ and δ to obtain a new substitution $\theta\delta$.

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

  $\theta = \{x_1 = s_1, x_2 = s_2, ..., x_m = s_m\}$
  $\delta = \{y_1 = t_1, y_2 = t_2, ..., y_k = t_k\}$
  To compute $\theta\delta$:

  1. Apply δ to each RHS of θ and then add all of the equations of δ:
     $\theta\delta = \{x_1 = s_1\delta, x_2 = s_2\delta, ..., x_m = s_m\delta, y_1 = t_1, y_2 = t_2, ..., y_k = t_k\}$

  2. Delete any identities, i.e., equations of the form $V = V$ from $\theta\delta$.

  3. Delete any equation $y_i = s_i$ where y_i is equal to one of the x_j in θ.

Example: $\theta = \{x = f(y), y = z\}$, $\delta = \{x = a, y = b, z = y\}$

.

Composition of Substitutions

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

- Substitutions when applied to formulas are **associative**:

$$(f\theta)\delta = f(\theta\delta)$$

Unifiers

A **unifier** of two formulas f and g is a substitution δ that makes f and g syntactically identical.

Not all formulas can be unified since substitutions **only** affect **variables**.

Example:

$$P(f(x), \boldsymbol{a}) \qquad P(y, f(w))$$

This pair cannot be unified as there is no way of making $\boldsymbol{a} = f(w)$ with a substitution.

Most General Unifier (MGU)

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

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

$$\theta = \delta\lambda$$

That is, every other unifier is more specialized than δ.
The MGU of a pair of formulas f and g is unique up to renaming.

The MGU is the “least specialized” way of making clauses with universal variables match.

MGU: Example

$$P(f(x), z) \qquad P(y, \boldsymbol{a})$$

$\delta = \{y = f(\boldsymbol{a}), x = \boldsymbol{a}, z = \boldsymbol{a}\}$ is a **unifier**. But it is **not an MGU**.

$P(f(x), z)\delta =$

$P(y, \boldsymbol{a})\delta =$

$\theta = \{y = f(x), z = \boldsymbol{a}\}$ is an MGU.

$P(f(x), z)\theta =$

$P(y, \boldsymbol{a})\theta =$

$\delta = \theta\lambda$, where $\lambda = \{x = \boldsymbol{a}\}$

Computing MGUs: Intuition

- We line up the two formulas and find the first **sub-expression** where they **disagree**.
- The pair of sub-expressions where they first disagree is called the **disagreement set**.
- The algorithm works by **successively fixing disagreement sets** until the two formulas become **syntactically identical**.

Most General Unifier

To find the MGU of two formulas f and g.

1. $k = 0; \quad \delta_0 = \{\}; \quad S_0 = \{f, g\}$.
2. REPEAT UNTIL no more disagreement:
3. Find disagreement set $D_k = \{e_1, e_2\}$.
4. IF $e_1 = V$, where V is a variable,
   and $e_2 = t$, where t is a term not containing V,
   or vice-versa then:
   - $\delta_{k+1} = \delta_k\{V = t\}$ # **Compose** the additional substitution
   - $S_{k+1} = S_k\{V = t\}$ # **Apply** the additional substitution
   - $k = k + 1$
5. ELSE unification is not possible.

Resolution of Clauses with Variables

Consider two clauses:
$(L, Q_1, Q_2, ..., Q_k)$
$(\neg M, R_1, R_2, ..., R_n)$
where there exists an **MGU** δ for L and M.

We apply δ to both clauses, resolve $L\delta$ and $\neg M\delta$, and infer the new clause

$(Q_1\delta, Q_2\delta, ..., Q_k\delta, R_1\delta, R_2\delta, ..., R_n\delta)$

Resolution of Clauses with Variables: Example

$(P(x), Q(g(x)))$
$(R(\boldsymbol{a}), Q(z), \neg P(\boldsymbol{a}))$

$L = P(x), M = P(\boldsymbol{a})$
$\delta = \{x = \boldsymbol{a}\}$

$R[1a, 2c]\{x = \boldsymbol{a}\}(Q(g(\boldsymbol{a})), R(\boldsymbol{a}), Q(z))$

Resolution of Clauses with Variables: Example

$(P(x), Q(g(x)))$
$(R(\boldsymbol{a}), Q(z), \neg P(\boldsymbol{a}))$

$L = P(x), M = P(\boldsymbol{a})$
$\delta = \{x = \boldsymbol{a}\}$

$R[1a, 2c]\{x = \boldsymbol{a}\}(Q(g(\boldsymbol{a})), R(\boldsymbol{a}), Q(z))$

The notation is **important**. You will need to use this notation on the **exam**!

- R: resolution step.
- 1a: the first (a-th) literal in the first clause; i.e. $P(x)$.
- 2c: the third (c-th) literal in the second clause; i.e., $\neg P(\boldsymbol{a})$.
  - 1a and 2c are the **clashing** literals.
- $\{x = a\}$: the **substitution** applied to make the clashing literals identical.

Resolution Proof: Example

Some patients like all doctors.
No patient likes any quack.
Prove: No doctor is a quack.

Step 1: Pick a vocabulary for representing these assertions.

Resolution Proof: Example

Some patients like all doctors.
No patient likes any quack.
Prove: No doctor is a quack.

Step 1: Pick a vocabulary for representing 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

Some patients like all doctors.
No patient likes any quack.
Prove: No doctor is a quack.

Step 2: Convert each assertion to a first-order formula.

Resolution Proof: Example

Some patients like all doctors.
No patient likes any quack.
Prove: No doctor is a quack.

Step 2: Convert each assertion to a first-order formula.

$F_1 : \exists x[P(x) \wedge \forall y[D(y) \rightarrow L(x,y))]]$

Resolution Proof: Example

Some patients like all doctors.
No patient likes any quack.
Prove: No doctor is a quack.

Step 2: Convert each assertion to a first-order formula.

$F_1 : \exists x[P(x) \land \forall y[D(y) \rightarrow L(x, y))]]$

$F_2 : \forall x \forall y[(P(x) \land Q(y)) \rightarrow \neg L(x, y)]$

Resolution Proof: Example

Some patients like all doctors.
No patient likes any quack.
Prove: No doctor is a quack.

Step 2: Convert each assertion to a first-order formula.

$F_1 : \exists x[P(x) \wedge \forall y[D(y) \rightarrow L(x,y))]]$

$F_2 : \forall x \forall y[(P(x) \wedge Q(y)) \rightarrow \neg L(x,y)]$

Query: $\forall x[D(x) \rightarrow \neg Q(x)]$

Step 3: Convert to Clausal form.

$F_1 : \exists x[P(x) \land \forall y[D(y) \rightarrow L(x, y)]]$

Resolution Proof: Example

$F_2 : \forall x \forall y[(P(x) \wedge Q(y)) \rightarrow \neg L(x, y)]$

Negation of Query:

$\neg(\forall x[D(x) \rightarrow \neg Q(x)])$

Resolution Proof: Example

Step 4: Resolution Proof from the Clauses.

1. $P(\boldsymbol{a})$

2. $(\neg D(y), L(\boldsymbol{a}, y))$

3. $(\neg P(x), \neg Q(y), \neg L(x, y))$

4. $D(\boldsymbol{b})$

5. $Q(\boldsymbol{b})$

Resolution Proof: Example

Step 4: Resolution Proof from the Clauses.

1. $P(\boldsymbol{a})$
2. $(\neg D(y), L(\boldsymbol{a}, y))$
3. $(\neg P(x), \neg Q(y), \neg L(x, y))$
4. $D(\boldsymbol{b})$
5. $Q(\boldsymbol{b})$

6. $R[3b, 5]\{y = \boldsymbol{b}\} \quad (\neg P(x), \neg L(x, \boldsymbol{b}))$

Resolution Proof: Example

Step 4: Resolution Proof from the Clauses.

1. $P(\boldsymbol{a})$
2. $(\neg D(y), L(\boldsymbol{a}, y))$
3. $(\neg P(x), \neg Q(y), \neg L(x, y))$
4. $D(\boldsymbol{b})$
5. $Q(\boldsymbol{b})$
6. $R[3b, 5]\{y = \boldsymbol{b}\} \quad (\neg P(x), \neg L(x, \boldsymbol{b}))$
7. $R[6a, 1]\{x = \boldsymbol{a}\} \quad \neg L(\boldsymbol{a, b})$

Resolution Proof: Example

Step 4: Resolution Proof from the Clauses.

1. $P(\boldsymbol{a})$
2. $(\neg D(y), L(\boldsymbol{a}, y))$
3. $(\neg P(x), \neg Q(y), \neg L(x, y))$
4. $D(\boldsymbol{b})$
5. $Q(\boldsymbol{b})$

6. $R[3b, 5]\{y = \boldsymbol{b}\} \quad (\neg P(x), \neg L(x, \boldsymbol{b}))$
7. $R[6a, 1]\{x = \boldsymbol{a}\} \quad \neg L(\boldsymbol{a}, \boldsymbol{b})$
8. $R[7, 2b]\{y = \boldsymbol{b}\} \quad \neg D(\boldsymbol{b})$

Resolution Proof: Example

Step 4: Resolution Proof from the Clauses.

1. $P(\boldsymbol{a})$
2. $(\neg D(y), L(\boldsymbol{a}, y))$
3. $(\neg P(x), \neg Q(y), \neg L(x, y))$
4. $D(\boldsymbol{b})$
5. $Q(\boldsymbol{b})$

6. $R[3b, 5]\{y = \boldsymbol{b}\} \quad (\neg P(x), \neg L(x, \boldsymbol{b}))$
7. $R[6a, 1]\{x = \boldsymbol{a}\} \quad \neg L(\boldsymbol{a, b})$
8. $R[7, 2b]\{y = \boldsymbol{b}\} \quad \neg D(\boldsymbol{b})$
9. $R[8, 4] \quad ()$

Answer Extraction

- The previous example shows how we can answer Yes-No questions.
- With a bit more effort we can also answer "fill-in-the-blanks" questions:
  - Use **free variables** in the **query** where we want the fill in the blanks.
  - Keep track of the **binding** that these variables received in **proving the query**.
    $parent(\boldsymbol{art}, \boldsymbol{jon})$ – is art one of jon's parents?
    $parent(x, \boldsymbol{jon})$ - who is one of jon's parents?
  - A simple bookkeeping device is to use a predicate symbol $answer(x, y, \ldots)$ to keep track of the bindings automatically.
    Example: To answer $parent(x, \boldsymbol{jon})$, construct the clause:

    $$(\neg parent(x, \boldsymbol{jon}), answer(x))$$

    Then perform resolution **until** obtain a clause consisting of **only** $answer$ **literals** (previously we stopped at empty clauses).

Answer Extraction: Example 1

1. $father(\boldsymbol{art, jon})$
2. $father(\boldsymbol{bob, kim})$
3. $(\neg father(y, z), parent(y, z))$ (all fathers are parents)
4. $(\neg parent(x, \boldsymbol{jon}), answer(x))$ (who is parent of jon?)

Answer Extraction: Example 1

1. $father(\boldsymbol{art, jon})$
2. $father(\boldsymbol{bob, kim})$
3. $(\neg father(y, z), parent(y, z))$ (all fathers are parents)
4. $(\neg parent(x, \boldsymbol{jon}), answer(x))$ (who is parent of jon?)
5. $R[4, 3b]\ \{y = x, z = \boldsymbol{jon}\} \quad (\neg father(x, \boldsymbol{jon}), answer(x))$
6. $R[5, 1]\ \{x = \boldsymbol{art}\} \quad answer(\boldsymbol{art})$

Answer Extraction: Exercise

Answer the following query (Sentence 4) using the information provided by Sentences 1-3.

1. Either bob or art is father of jon.

2. bob is father of kim.

3. All fathers are parents.

4. Who is parent of jon?

Answer Extraction: Example 2

Answer the following query (Sentence 4) using the information provided by Sentences 1-3.

1. Whoever can read is literate.
2. Dolphins are not literate.
3. Flipper is an intelligent dolphin.
4. Who is intelligent but cannot read?

Answer Extraction: Example 2

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. $dolp(\boldsymbol{flip}) \wedge intell(\boldsymbol{flip})$

Who is intelligent but cannot read?

Answer Extraction: Example 2

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. $dolp(\boldsymbol{flip}) \wedge intell(\boldsymbol{flip})$

Who is intelligent but cannot read?

Whoever that is intelligent but cannot read is the answer

Answer Extraction: Example 2

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. $dolp(\boldsymbol{flip}) \wedge intell(\boldsymbol{flip})$

Who is intelligent but cannot read?

Whoever that is intelligent but cannot read is the answer

$\forall x[(intell(x) \wedge \neg read(x)) \rightarrow answer(x)]$

Answer Extraction: Example 2

1. $(\neg read(x), lit(x))$
2. $(\neg dolp(x), \neg lit(x))$
3. $dolp(\boldsymbol{flip})$
4. $intell(\boldsymbol{flip})$
5. $(\neg intell(x), read(x), answer(x))$

Answer Extraction: Example 2

1. $(\neg read(x), lit(x))$
2. $(\neg dolp(x), \neg lit(x))$
3. $dolp(\boldsymbol{flip})$
4. $intell(\boldsymbol{flip})$
5. $(\neg intell(x), read(x), answer(x))$

6. $R[5a, 4]\ \{x = \boldsymbol{flip}\} \quad (read(\boldsymbol{flip}), answer(\boldsymbol{flip}))$
7. $R[6, 1a]\ \{x = \boldsymbol{flip}\} \quad (lit(\boldsymbol{flip}), answer(\boldsymbol{flip}))$
8. $R[7, 2b]\ \{x = \boldsymbol{flip}\} \quad (\neg dolp(\boldsymbol{flip}), answer(\boldsymbol{flip}))$
9. $R[8, 3] \quad answer(\boldsymbol{flip})$