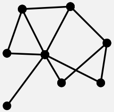

Knowledge

Reasoning

Introduction to Artificial Intelligence

Chandra Gummaluru

University of Toronto

Version W22.1

- The following is based on material developed by many individuals, including (but not limited to):
    - Sheila McIlraith
    - Bahar Aameri
    - Fahiem Bacchus
    - Sonya Allin

- Part of being an intelligent agent involves being able to infer implicit facts based on known or assumed ones.
- To achieve this ability artificially, we need to do things:

  1. Develop a **formal languages** to represent statements (previous chapter).
  2. Develop a **reasoning mechanism** for the formal system (this chapter).

- We developed a representation called first-order logic (FOL). We now develop a reasoning mechanism called **resolution**.

- Resolution assumes the knowledge base is a "clausal theory".
- A **clausal theory** is a conjunction of "clauses":
  - If $c_1, \ldots, c_n$ are clauses, then $c_1 \wedge \cdots \wedge c_n$ is a clausal theory.
- A **clause** is a disjunction of "literals":
  - If $l_1, \ldots, l_n$ are literals, then $l_1 \vee \cdots \vee l_n$ is a clause.
- A **literal** is an atomic formula, f, or its negation, $\neg f$.

- Resolution only uses one inference rule:
  - If $l_1 \vee l$ and $l_2 \vee \neg l$ are two clauses, then $l_1 \vee l_2$.
  - The logic here is that l and $\neg l$ cannot simultaneously hold. Therefore, at least one of l_1 or l_2 must hold.

- Suppose our knowledge base consists of the clauses, $c_1, \ldots, c_n$, and want to check if c is a logical consequence.
- We can add $\neg c$ to our knowledge base and show that we can prove a contradictory statement.
- Since we assume $c_1, \ldots, c_n$ hold, we conclude that $\neg c$ cannot hold, i.e., c holds.
- This is called **resolution by refutation**.

- **Example:** Resolution by Refutation

  Consider the following knowledge base:

  1. Clyde is an elephant or a giraffe: elephant(**Cylde**) $\vee$ giraffe(**Clyde**).
  2. Either Clyde is not an elephant or he likes peanuts: $\neg$elephant(**Clyde**) $\vee$ likes(**peanuts**, **Clyde**).
  3. Either Clyde is not an giraffe or he likes leaves: $\neg$giraffe(**Clyde**) $\vee$ likes(**leaves**, **Clyde**).
  4. Clyde does not like leaves: $\neg$likes(**leaves**, **Clyde**).

  We want to show that Clyde is an elephant. Thus, we assume:

  5. Clyde is not an elephant: $\neg$elephant(**Clyde**)

We perform resolution as follows:

1. elephant(**Clyde**) $\vee$ giraffe(**Clyde**)
2. $\neg$elephant(**Clyde**) $\vee$ likes(**peanuts**, **clyde**)
3. $\neg$giraffe(**Clyde**) $\vee$ likes(**leaves**, **Clyde**)
4. $\neg$likes(**leaves**, **Clyde**)
5. $\neg$elephant(**Clyde**)
6. R[5a,1a] giraffe(**Clyde**)
7. R[6a,3a] likes(**leaves**, **Clyde**)
8. R[7a,4a] $\{\}$

- Often, assertions in our knowledge base are not expressed in clausal form.

  Example: Non-Clausal Assertion

  - If Clyde is an elephant, then he likes peanuts:
    $\text{elephant}(\mathbf{Clyde}) \rightarrow \text{likes}(\mathbf{peanuts}, \mathbf{Clyde})$.

- We need a systemic way to convert any non-clausal assertion into a set of clauses.

- The following procedure can be used:
  1. Eliminate $\rightarrow$
  2. Move $\neg$ inward
  3. Distinguish quantified variables
  4. Eliminate $\exists$
  5. Move $\forall$ outward
  6. Distribute $\vee$ over $\wedge$
  7. Flatten nested $\wedge/\vee$.
  8. Remove $\forall$
  9. Split on $\wedge$

1. Eliminate $\rightarrow$

   - $A \rightarrow B \equiv \neg A \vee B$
     E.g: If Clyde is an elephant, he likes peanuts. Equivalently, either Clyde likes peanuts, or he is not an elephant.

2. Move $\neg$ inward and simplify:

- $\neg\neg A \equiv A$
  E.g: Clyde isn't not an elephant iff he's an elephant.
- $\neg(A \vee B) \equiv \neg A \wedge \neg B$
  E.g: Clyde is neither a tiger nor a giraffe iff he isn't a tiger and he isn't a giraffe.
- $\neg(A \wedge B) \equiv \neg A \vee \neg B$
  E.g: Clyde doesn't like both leaves and meat iff he dislikes leaves or meat.
- $\neg\forall x A \equiv \exists x \neg A$
  E.g: Not every person likes this class iff some people don't like it.
- $\neg\exists x A \equiv \forall x \neg A$
  E.g: Not one person dislikes this class iff everyone likes this class.

3. Distinguish quantified variables:

   - $\forall x f(x) \equiv \forall y f(y)$
   - $\exists x f(x) \equiv \exists y f(y)$

4. Eliminate $\exists$:

   - $\exists x[f(x)] \equiv f(\mathbf{c})$, for some unique constant, $\mathbf{c}$.
     E.g: If there is a friendly elephant, call him "Clyde". If "Clyde" is a friendly elephant, then there exists a friendly elephant.

   - $\forall x[\exists y f(y)] \equiv \forall x[f(g(x))]$, for some g.
     E.g: Everyone likes someone (usually a different someone). If we use partnerOf($\cdot$) to denote that person, we can say, everyone likes their partner.

5. Move $\forall$ outward:
   - $\forall x[f(x) \wedge \forall y g(y)] \equiv \forall x \forall y[f(x) \wedge g(y)]$

6. Distribute $\vee$ over $\wedge$:

   - $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$.
     E.g: Clyde is either a giraffe, or he is an elephant and likes peanuts iff Clyde is either a giraffe or an elephant, and he is either a giraffe or likes peanuts.

7. Flatten nested $\wedge / \vee$:
   - $(A \wedge (B \wedge C)) \equiv (A \wedge B \wedge C)$
   - $(A \vee (B \vee C)) \equiv (A \vee B \vee C)$
8. Remove $\forall$:
   - $\forall x f(x) \equiv f(x)$
9. Split on $\wedge$:
   - $f \wedge g \equiv f, g$.

- Consider the statement:

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

- We can convert it to clausal form as follows:

1. Eliminate $\to$

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

2. Move $\neg$ inward

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

2. Move $\neg$ inward

$$\forall x\left[\neg P(x) \vee \left(\left(\forall y\left[\neg P(y) \vee P(f(x,y))\right]\right) \wedge \left(\exists y\left[q(x,y) \vee \neg P(y)\right]\right)\right)\right]$$

3. Standardize variables

$$\forall x\left[\neg P(x) \vee \left(\left(\forall y\left[\neg P(y) \vee P(f(x,y))\right]\right) \wedge \left(\exists z\left[q(x,z) \vee \neg P(z)\right]\right)\right)\right]$$

4. Eliminate $\exists$

$$\forall x\left[\neg P(x) \vee \left(\left(\forall y\left[\neg P(y) \vee P(f(x,y))\right]\right) \wedge \left(\left[q(x,g(x)) \vee \neg P(g(x))\right]\right)\right)\right]$$

5. Move $\forall$ outward

$$\forall x \forall y\left[\neg P(x) \vee \left(\left(\neg P(y) \vee P(f(x,y))\right) \wedge \left(\left[q(x,g(x)) \vee \neg P(g(x))\right]\right)\right)\right]$$

5. Move $\forall$ outward

$$\forall x \forall y \left[\neg P(x) \vee \left(\left(\neg P(y) \vee P(f(x,y)) \right) \wedge \left(\left[q(x, g(x)) \vee \neg P(g(x)) \right] \right) \right) \right]$$

6. Distribute $\vee$ over $\wedge$

$$\forall x \forall y \left[\left(\neg P(x) \vee \left(\neg P(y) \vee P(f(x,y)) \right) \right) \wedge \left(\neg P(x) \vee \left[q(x, g(x)) \vee \neg P(g(x)) \right] \right) \right]$$

7. Flatten nested $\wedge/\vee$

$$\forall x \forall y \left[\left(\neg P(x) \vee \neg P(y) \vee P(f(x,y)) \right) \wedge \left(\neg P(x) \vee q(x, g(x)) \vee \neg P(g(x)) \right) \right]$$

7. Flatten nested $\wedge/\vee$

$$\forall x \forall y \left[\left(\neg P(x) \vee \neg P(y) \vee P(f(x,y)) \right) \wedge \left(\neg P(x) \vee q(x, g(x)) \vee \neg P(g(x)) \right) \right]$$

8. Remove $\forall$

$$\left(\neg P(x) \vee \neg P(y) \vee P(f(x,y)) \right) \wedge \left(\neg P(x) \vee q(x, g(x)) \vee \neg P(g(x)) \right)$$

9. Split on $\wedge$

$$\begin{cases} \neg P(x) \vee \neg P(y) \vee P(f(x,y)) \\ \neg P(x) \vee q(x, g(x)) \vee \neg P(g(x)) \end{cases}$$

- In the previous example of resolution by refutation, none of the conflicting clauses had variables.
- Suppose instead that we had two conflicting clauses but at least one involves some variables such as $(p(x) \vee p'(y) \dots)$ and $(\neg p(\mathbf{a}) \vee \dots)$.
  - Since x can be any constant, the clause $(p(x) \vee p'(y) \vee \dots)$ actually represents a family of clauses

$$(p(\mathbf{a}) \vee p'(y) \vee \dots)$$
$$\vdots$$
$$(p(\mathbf{z}) \vee p'(y) \vee \dots)$$

  - Thus, we can still resolve the clauses by substituting $x = \mathbf{a}$, yielding $(p'(y) \vee \dots)$.
  - We could have also made additional substitutions, like $y = \mathbf{b}$, however, this would reduce the generality of the resulting clause.

- 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.
  - Applying the substitution, $\delta = \{V_1 = t_1, \ldots, V_n = t_n\}$, to the formula f, is done by simultaneously replacing V_i with t_i.
  - The resulting formula is denoted $f\delta$.
  - **E.g:** If $f = P(x, g(y, z))$, and $\delta = \{x = y, y = f(a)\}$, then $f\delta = P(y, g(f(a), z))$.

- We can compose two substitutions, θ, δ to obtain a new substitution, $\theta\delta$.
  - Let $\theta = \{x_1 = s_1, \ldots x_n = s_n\}$ and $\delta = \{y_1 = t_1, \ldots, y_m = t_m\}$.
  - We compute $\theta\delta$ as follows:
    1. For each equation, $x_i = s_i$ in θ, apply δ to its right side to yield $x_i = s_i\delta$.
    2. If $x_i = s_i\delta$ is a not a tautology (e.g., $V = V$), include it in $\theta\delta$.
    3. If $y_i \neq x_j$ for any j, then include $y_i = t_i$ in $\theta\delta$.
  - Defining composition in this way means that applying $\theta\delta$ to a formula is equivalent to first applying θ, and then applying δ, i.e., $(f\theta)\delta = f(\theta\delta)$.

- **Example:** Composing Substitutions
  - Let $\theta = \{x = f(y), y = z\}$ and $\delta = \{x = a, y = b, z = y\}$.
    1. Applying δ to the right side of $x = f(y)$ and $y = z$ yields $x = f(b)$ and $y = y$, respectively.
    2. Since $y = y$ is a tautology, we do not include it.
    3. We also include $z = y$ from δ.
    4. It follows that $\theta\delta = \{x = f(b), z = y\}$.

- A **unifier** of two formulae, f and g, is a substitution, δ, that makes f and g syntactically identical.
- A unifier, δ, is **most general** iff for every other unifier, θ, there exists a third substitution, λ, such that

$$\theta = \delta\lambda.$$

  In other words, every other unifier is more specialized.

- **Procedure**: Computing the Most General Unifier
  - Let f, g be two formulae we wish to unify.
  - Let δ denote the most general unifier, and $S = \{f, g\}\,\delta$.

1. Start with the empty substitution, $\delta_0 = \{\}$, and $S_0 = \{f, g\}$
2. In each iteration, k, find a disagreement set, $D_k = \{V, t\}$. If one does not exist, then $\delta = \delta_k$ is the most general unifier, and $S = S_k$.
3. If D_k is unifiable (i.e., V is a variable, and t is a term not containing V):
   a. Let $\delta_{k+1} = \delta_k\,\{V = t\}$.
   b. Let $S_{k+1} = S_k\,\{V = t\}$.
4. If D_k is not unifiable, then f and g cannot be unified.

- Let us find the most general unifier of $P(\mathbf{a}, x, h(g(z)))$ and $P(z, h(y), h(y))$.

Itr.	D_k	S_k	δ_k
0	$\{\mathbf{a}, z\}$	$\left\{P\left(\mathbf{a}, x, h(g(z))\right), P\left(z, h(y), h(y)\right)\right\}$	$\{\}$
1	$\{x, h(y)\}$	$\left\{P\left(\mathbf{a}, x, h(g(\mathbf{a}))\right), P\left(\mathbf{a}, h(y), h(y)\right)\right\}$	$\{z = \mathbf{a}\}$
2	$\{g(\mathbf{a}), y\}$	$\left\{P\left(\mathbf{a}, h(y), h(g(\mathbf{a}))\right), P\left(\mathbf{a}, h(y), h(y)\right)\right\}$	$\{z = \mathbf{a}, x = h(y)\}$
3	$\{\}$	$\left\{P\left(\mathbf{a}, h(g(\mathbf{a})), h(g(\mathbf{a}))\right)\right\}$	$\{z = \mathbf{a}, x = h(g(\mathbf{a})), y = g(\mathbf{a})\}$

- The most general unifier is $\delta = \{z = \mathbf{a}, x = h(g(\mathbf{a})), y = g(\mathbf{a})\}$.

- Let us find the most general unifier of $P(f(\mathbf{a}), g(x))$ and $P(y, y)$.

Itr.	D_k	S_k	δ_k
0	$\{f(\mathbf{a}), y\}$	$\left\{P\big(f(\mathbf{a}), g(x)\big), P\big(y, y\big)\right\}$	$\{\}$
1	$\{g(x), f(\mathbf{a})\}$	$\left\{P\big(f(\mathbf{a}), g(x)\big), P\big(f(\mathbf{a}), f(\mathbf{a})\big)\right\}$	$\{y = f(\mathbf{a})\}$

- The disagreement set, D_1 is not unifiable. Thus, there is no unifier.

- Just as we can sometimes unify multiple clauses, we can also sometimes unify multiple literals of the same clause.
- In the examples seen thus far, we either had clauses with a single literal, or multiple non-unifiable literals.
- If two or more literals from the same clause are unifiable, then it may not be possible to resolve it with a conflicting clause.
- **Example:** Conflicting Non-Resolvable Clauses
  - Consider the clauses: $c_1 : p(x) \vee p(y)$ and $c_2 : \neg p(u) \vee \neg p(v)$.
  - Each one consists of two unifiable literals.
  - Intuitively, c_1 and c_2 are conflicting, but it is impossible to generate an empty clause using the strict rules of resolution.

- If the most general unifier of two or more literals from the same clause, c, is θ, then $c\theta$ with all duplicates removed is called a **factor** of c.

- **Example**: Factor of a Clause
  - Consider the clause $c : p(x) \vee p(y)$.
  - Its two literals, $p(x)$ and $p(y)$ can be unified with $\theta = \{x = y\}$.
  - We have $c\theta = p(y)$, and so $p(y)$ is a factor of c.

- During resolution, if we encounter a clause in which two more more literals can be unified, we can do so. We refer to this step as factoring.
- **Example**: Resolution with Factoring

  1. $p(x) \vee p(y)$
  2. $\neg p(u) \vee \neg p(v)$
  3. $f[1ab]\,\{x = y\}\,p(y)$
  4. $f[2ab]\,\{u = v\}\,\neg p(v)$
  5. $R[3a, 4a]\,\{y = v\}\,\{\}$

- Let us now consider a full example.
- At U of T, it is reasonable to assume that:
    1. Some courses have exams.
    2. No student likes anything with an exam.
- Can we show that no student likes all courses?

1. Pick a vocabulary to represent the assertions:
    - isStudent(x), x is a student
    - isCourse(x), x is a course
    - hasExam(x), x has an exam
    - likes(x, y), y likes x

2. Convert each assertion (including the negation of the query) to a first-order formula:

   i. Some courses have exams:

   $$\exists x[\text{isCourse}(x) \land \text{hasExam}(x)]$$

   ii. No student likes anything that has an exam:

   $$\forall x \forall y[\text{isStudent}(x) \land \text{hasExam}(y) \to \neg\text{likes}(y, x)]$$

   iii. Some student likes all courses:

   $$\exists x[\text{isStudent}(x) \land \forall y[\text{isCourse}(y) \to \text{likes}(y, x)]]$$

3. Convert each first-order formula to clausal form:

   i. $\exists x[\text{isCourse}(x) \wedge \text{hasExam}(x)]$:
      - CIF.4: $\text{isCourse}(\mathbf{c}) \wedge \text{hasExam}(\mathbf{c})$
      - CIF.8: $\text{isCourse}(\mathbf{c})$, $\text{hasExam}(\mathbf{c})$

   ii. $\forall x \forall y[\text{isStudent}(x) \wedge \text{hasExam}(y) \rightarrow \neg\text{likes}(x, y)]$:
      - CIF.2: $\forall x \forall y\, [\neg\, (\text{isStudent}(x) \wedge \text{hasExam}(y)) \vee \neg\text{likes}(x, y)]$
      - CIF.3: $\forall x \forall y\, [\neg\text{isStudent}(x) \vee \neg\text{hasExam}(y) \vee \neg\text{likes}(x, y)]$
      - CIF.8: $\neg\text{isStudent}(x) \vee \neg\text{hasExam}(y) \vee \neg\text{likes}(x, y)$

   iii. $\exists x[\text{isCourse}(x) \wedge \forall y[\text{isStudent}(y) \rightarrow \text{likes}(x, y)]]$:
      - CIF.2: $\exists x[\text{isStudent}(x) \wedge \forall y[\neg\text{isCourse}(x) \vee \text{likes}(x, y)]]$
      - CIF.4: $\text{isStudent}(\mathbf{s}) \wedge \forall y[\neg\text{isCourse}(y) \vee \text{likes}(\mathbf{s}, y)]$
      - CIF.5: $\forall y[\text{isStudent}(\mathbf{s}) \wedge [\neg\text{isCourse}(y) \vee \text{likes}(\mathbf{s}, y)]]$
      - CIF.8: $\text{isStudent}(\mathbf{s}), (\neg\text{isCourse}(y) \vee \text{likes}(\mathbf{s}, y))$

4. Apply the inference rule of resolution to the clauses:

   1. isCourse($\mathbf{c}$)
   2. hasExam($\mathbf{c}$)
   3. $(\neg \text{isStudent}(x) \vee \neg \text{hasExam}(y) \vee \neg \text{likes}(x, y))$
   4. isStudent($\mathbf{s}$)
   5. $\neg \text{isCourse}(z), \text{likes}(\mathbf{s}, z)$

   6. R[1a, 5a]$\{z = \mathbf{c}\}$ likes($\mathbf{s}, \mathbf{c}$)
   7. R[6a, 3c]$\{x = \mathbf{s}, y = \mathbf{c}\}$ $\neg \text{isStudent}(\mathbf{s}) \vee \neg \text{hasExam}(\mathbf{c})$
   8. R[7a, 4a] $\neg \text{hasExam}(\mathbf{c})$
   9. R[8a, 2a] $\{\}$

- So far, we can answer any query where the answer is yes/no.
- Queries often are more nuanced.
  - **E.g:** Harry and Ron are friends and Harry and Ginny are married. Assuming all married couples are friends, who are Harry's friends?
- To answer such a query, we can assume the answer, which we denote using an answer clause, answer(x).
  - **E.g**:Either x is the answer or x and Harry and x are not friends:

$$\neg \text{areFriends}(\textbf{Harry}, x) \vee \text{answer}(x)$$

- We then perform resolution until we obtain a clause of only answer literals.

- We could perform resolution as follows:
  1. $\text{areFriends}(\textbf{Harry}, \textbf{Ron})$
  2. $\text{areMarried}(\textbf{Harry}, \textbf{Ginny})$
  3. $\neg\text{areMarried}(x, y) \vee \text{areFriends}(x, y)$
  4. $\neg\text{areFriends}(\textbf{Harry}, x) \vee \text{answer}(x)$
  5. $R[1a, 4a]\,\{x = \textbf{Ron}\}\,\text{answer}(\textbf{Ron})$
  6. $R[2a, 3a]\,\{x = \textbf{Harry}, y = \textbf{Ginny}\}\,\text{areFriends}(\textbf{Harry}, \textbf{Ginny})$
  7. $R[6a, 4a]\,\{x = \textbf{Ginny}\}\,\text{answer}(\textbf{Ginny})$
- Ron and Ginny are Harry's friends.