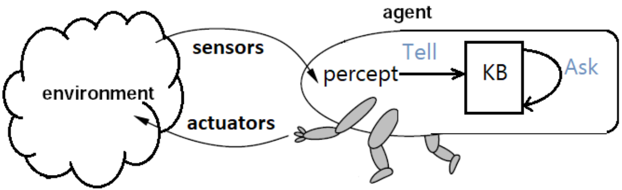
CS761 Artificial Intelligence

Propositional Logic Inference

Recall: Propositional KB and IE

To design a knowledge-based agent, the following questions are important:

1. What knowledge representation language to be used to define the KB?
2. How do we implement an IE using this language?



Two important operations in a knowledge-based agent:

- Tell: Add a sentence to KB.
  - We introduced proposition logic as a knowledge representation language.
  - A clause is a proposition of the form

$$(h_1 \vee h_2 \vee \cdots \vee h_m) \leftarrow (\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_k)$$

  - A propositional knowledge base is a set of clauses.
- Ask: Reason if a sentence is entailed by the KB
  - A model of a propositional knowledge base KB is an interpretation that satisfies KB.
  - A proposition g is called a logical consequence of a knowledge base KB, written as

$$\mathrm{KB} \models g$$

    if g is true in every model of KB.
  - An inference engine decides for any KB, a set of percept atoms Percepts, and proposition g, whether

$$\mathrm{KB} \cup \mathrm{Percepts} \models g$$

Propositional Logic Inference

Propositional Logic Inference Problem

INPUT	A set of propositions α, a proposition β
OUTPUT	Decide if $\alpha \models \beta$

We are now going to investigate ways to solve the inference problem above.

Logic Inference versus Constraint Satisfaction

- A sentence is called satisfiable if there is an interpretation that satisfies the sentence, i.e., evaluates the sentence to true.

  E.g. $(p \vee \neg q) \wedge (\neg p \vee q)$ is satisfiable by $\pi(p) = 1$ and $\pi(q) = 1$.

  $(p \vee \neg q) \wedge (\neg p \vee q) \wedge (\neg p \vee \neg q) \wedge (p \vee q)$ is not satisfiable.

- The satisfiability problem (SAT) is asks for a satisfying interpretation of a proposition.

  E.g. To solve the problem for $(p \wedge q \wedge \neg r) \vee (\neg q \rightarrow (\neg p \wedge r))$ is to find an interpretation π that defines

$$\pi(p), \pi(q), \pi(r) \text{ such that}$$
$$(p \wedge q \wedge \neg r) \vee (\neg q \rightarrow (\neg p \wedge r)) \text{ is true.}$$

Theorem [Equivalence between inference and satisfiability]

For propositions α and β, $\alpha \models \beta$ if and only if $\alpha \land \neg\beta$ is not satisfiable.

Theorem [Equivalence between inference and satisfiability]

For propositions α and β, $\alpha \models \beta$ if and only if $\alpha \wedge \neg\beta$ is not satisfiable.

To show $P \Leftrightarrow Q$, it suffices to show "$P \Rightarrow Q$" and "$Q \Rightarrow P$" ($\neg P \Rightarrow \neg Q$).

Theorem [Equivalence between inference and satisfiability]

For propositions α and β, $\alpha \models \beta$ if and only if $\alpha \wedge \neg\beta$ is not satisfiable.

To show $P \Leftrightarrow Q$, it suffices to show "$P \Rightarrow Q$" and "$Q \Rightarrow P$" ($\neg P \Rightarrow \neg Q$).

Proof. Suppose $\alpha \models \beta$, but $\alpha \wedge \neg\beta$ is satisfiable.

- All models of α satisfies β.
- Take the interpretation μ that satisfies $\alpha \wedge \neg\beta$, i.e., $\mu(\alpha \wedge \neg\beta) = 1$.
- Then $\mu(\alpha) = 1$ and $\mu(\beta) = 0$. Contradiction.
- Thus $\alpha \models \beta$ implies $\alpha \wedge \neg\beta$ is not satisfiable.

Theorem [Equivalence between inference and satisfiability]

For propositions α and β, $\alpha \models \beta$ if and only if $\alpha \wedge \neg\beta$ is not satisfiable.

To show $P \Leftrightarrow Q$, it suffices to show "$P \Rightarrow Q$" and "$Q \Rightarrow P$" ($\neg P \Rightarrow \neg Q$).

Proof. Suppose $\alpha \models \beta$, but $\alpha \wedge \neg\beta$ is satisfiable.

- All models of α satisfies β.
- Take the interpretation μ that satisfies $\alpha \wedge \neg\beta$, i.e., $\mu(\alpha \wedge \neg\beta) = 1$.
- Then $\mu(\alpha) = 1$ and $\mu(\beta) = 0$. Contradiction.
- Thus $\alpha \models \beta$ implies $\alpha \wedge \neg\beta$ is not satisfiable.

Suppose $\alpha \not\models \beta$.

- This means that there is a model of α that does not satisfy β.
- Call this model μ.
- Then $\mu(\alpha) = 1$ and $\mu(\neg\beta) = 1$. This implies $\mu(\alpha \wedge \neg\beta) = 1$.
- Thus if $\alpha \wedge \neg\beta$ is not satisfiable, we have $\alpha \models \beta$. $\square$

Solving the propositional logic inference problem $\alpha \models \beta$ is equivalent to solving the SAT for $\alpha \wedge \neg\beta$. Thus techniques for constraint satisfaction problems can be used for SAT[1].

Efficient implementations of SAT solvers:

- **DPLL algorithm** (Davis, Putnam, Logemann, Loveland, 1960s): A backtracking-based search algorithm that enumerates possible models, with the following tricks:
    - Early termination
    - Pure symbol heuristic
    - Unit clause heuristic
- **Local search algorithms**: Use the number of unsatisfied clauses as the evaluation function.
    - Greedy descent
    - Simulated annealing
    - WalkSAT

[1] SAT is a well-known NP-complete problem

- Solving general propositional logic inference is a hard problem.
- We are going to study the inference engine for a special type of knowledge bases and queries, namely, definite clauses.
- This special case allows very efficient inference.

Definite Clause

Definition

A definite clause is of the form

$$H \leftarrow (A_1 \wedge \cdots \wedge A_m)^a$$

where $m \geq 0$, H and every A_i is an atom.

[a] We often write $H \leftarrow (A_1 \wedge \cdots \wedge A_m)$ as $H \leftarrow A_1 \wedge \cdots \wedge A_m$.

E.g.

- $S_{1,2} \leftarrow W_{2,2} \wedge W_{1,3}$ and $A_{1,1}$ are definite clauses
- $W_{2,2} \leftarrow \neg S_{1,2}$ and $(S_{2,2} \wedge S_{1,3}) \leftarrow W_{1,2}$ are not definite clauses

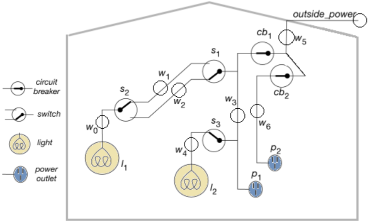
A definite clause knowledge base is a knowledge base that contains only definite clauses.
The definite clause inference engine would need to handle queries of the form

$$\textsf{ask}\ \ b$$

where b is an atom.

Example. Consider the following electrical environment in a house.

- Atoms: ok_ℓ_1, ok_ℓ_2, ok_cb_1, ok_cb_2, $live\_outside$, etc.
- KB is given below.



$live\_l_1 \leftarrow live\_w_0.$

$live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2.$

$live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2.$

$live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1.$

$live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1.$

$live\_l_2 \leftarrow live\_w_4.$

$live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3.$

$live\_p_1 \leftarrow live\_w_3.$

$live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1.$

$live\_p_2 \leftarrow live\_w_6.$

$live\_w_6 \leftarrow live\_w_5 \wedge ok\_cb_2.$

$live\_w_5 \leftarrow live\_outside.$

$lit\_l_1 \leftarrow live\_l_1 \wedge ok\_l_1.$

$lit\_l_2 \leftarrow live\_l_2 \wedge ok\_l_2.$

Percepts: $down\_s_1$, up_s_2, up_s_3, ok_cb_1, $live\_outside$, ok_ℓ_1, ok_ℓ_2.

The inference engine would return true or false for queries e.g.

ask lit_ℓ_2 and ask lit_ℓ_1

Question How to implement a definite clause inference engine?

- An inference engine produces a proof, i.e., a mechanically derivable demonstration that a proposition g logically follows from a set of sentences S.
- The algorithm that generates a proof is called a proof procedure. If there is a proof of g from S, we write $S \vdash g$.
- A proof procedure is sound if every proposition g that is derived from S is a logical consequence, i.e., $S \vdash g$ implies $S \models g$.
- A proof procedure is complete if there is a proof of each logical consequence of S, i.e., $S \models g$ implies $S \vdash g$.

We next introduce two proof procedures that are both sound and complete.

1. Forward chaining
2. SLD resolution

Forward Chaining

- Start from clauses in $\text{KB} \cup \text{Percepts}$ and generate new logical consequences.
- Each derivation is built on the clauses in $\text{KB} \cup \text{Percepts}$ or the clauses that have already been generated.
- Use a rule of derivation for inference.

Modus Ponens

Modus ponens (MP) is the inference rule:

$$\frac{h \leftarrow a_1 \wedge \ldots \wedge a_m, \quad a_1, \ldots, a_m}{h}$$

Forward chaining applies MP iteratively to the current knowledge to generate new clause, which are then added to knowledge.

Example. Suppose KB contains: (1) $a \leftarrow b \wedge c$, (2) $b \leftarrow d \wedge e$, (3) $b \leftarrow g \wedge e$, (4) $c \leftarrow e$, (5) $f \leftarrow a \wedge g$.

Observation Percepts contains (6) d, (7) e.

Suppose the query is ask a.

Forward chaining would develop the following proof:

$$\begin{array}{rr} (8)\ c & MP(4),(7) \\ (9)\ b & MP(2),(6),(7) \\ (10)\ a & MP(1),(9),(8) \end{array}$$

Thus we can answer KB $\cup$ Percepts $\vdash a$.

ForwardChain(KB, Percepts, g)

INPUT: Definite clause knowledge base KB, Observation Percepts. Query of the form ask g.
OUTPUT: true if $\mathrm{KB} \cup \mathrm{Percepts} \vdash g$; false if $\mathrm{KB} \cup \mathrm{Percepts} \nvdash g$.

  Create an empty set $C \leftarrow \emptyset$
  repeat
    Select $h \leftarrow a_1 \wedge \cdots \wedge a_m$ in $\mathrm{KB} \cup \mathrm{Percepts}$
                where $a_i \in C$ for all $1 \leq i \leq m$ & $h \notin C$
    if $h = g$ **then**
      return true
    end if
    $C \leftarrow C \cup \{h\}$ ▷ Apply MP
  until C doe not change any more
  return false

Theorem [Soundness of Forward Chaining]

For any definite clause KB, Percepts and query g, $\mathrm{KB} \cup \mathrm{Percepts} \vdash g$ implies that $\mathrm{KB} \cup \mathrm{Percepts} \models g$.

Proof. We show that every atom a that is added to C by the algorithm is a logical consequence of $\mathrm{KB} \cup \mathrm{Percepts}$.

- Suppose there is an atom $h \in C$ that is not a logical consequence. Let h be the first ever such atom to be added in C.
- There must be some clause in $\mathrm{KB} \cup \mathrm{Percepts}$, in the form

$$h \leftarrow a_1 \wedge \cdots \wedge a_m$$

  such that $a_1, \ldots, a_m$ are all in C.
- By assumption, $\mathrm{KB} \cup \mathrm{Percepts} \models a_i$ for all $1 \leq i \leq m$.
- Then it must be that $\mathrm{KB} \cup \mathrm{Percepts} \models h$. Contradiction.

□

Theorem [Completeness of Forward Chaining]

For any KB, Percepts and query g, $\mathrm{KB} \cup \mathrm{Percepts} \models g$ implies $\mathrm{KB} \cup \mathrm{Percepts} \vdash g$.

Proof. Again we only discuss the case when g is an atom. Suppose $\mathrm{KB} \cup \mathrm{Percepts} \nvdash g$, and consider the resulting set C after running forward chaining.

- Define an interpretation I such that for any atom a, $I(a) = \text{true}$ if and only if $a \in C$.
- Suppose $h \leftarrow a_1 \wedge \cdots \wedge a_m$ in $\mathrm{KB} \cup \mathrm{Percepts}$ is false in I.
- Then it must be that $a_1, \ldots, a_m \in C$ but $h \notin C$.
- But this is impossible as the algorithm would then apply MP on $h \leftarrow a_1 \wedge \cdots \wedge a_m$ and adds h into C.
- Thus I is a model of $\mathrm{KB} \cup \mathrm{Percepts}$.
- Now suppose $\mathrm{KB} \cup \mathrm{Percepts} \models g$. By definition, g must be true in every model of $\mathrm{KB} \cup \mathrm{Percepts}$.
- In particular, g must be true in I.
- The only way this may happen is $g \in C$.
- This means $\mathrm{KB} \cup \mathrm{Percepts} \vdash g$. Contradiction.

□

Selective Linear Definite Clause (SLD) Resolution

- Start from the query g, and treat it as a goal.
- Represent the query as

$$yes \leftarrow g$$

  where *yes* is a special atom.
- Infer backwards. Every step derives a clause

$$yes \leftarrow g_1 \wedge \cdots \wedge g_s$$

- Answer true if and only if $yes \leftarrow$ is derived.

Resolution

Resolution is the inference rule:

$$\frac{h \leftarrow a_1 \wedge \cdots \wedge a_m, \quad a_m \leftarrow b_1 \wedge \cdots \wedge b_\ell}{h \leftarrow a_1 \wedge \cdots \wedge a_{m-1} \wedge b_1 \wedge \cdots \wedge b_\ell}$$

In the above, a_m is called a subgoal.

An SLD derivation of a query ask g from KB ∪ Percepts is a sequence of definite clauses $\gamma_0, \ldots, \gamma_n$:

- The head of each γ_i is *yes*
- γ_0 is yes $\leftarrow g$
- For $i > 0$, γ_i is obtained by resolution from γ_{i-1} with a definite clause in KB ∪ Percepts:

$$\frac{\gamma_{i-1}, \qquad \text{a clause in KB} \cup \text{Percepts}}{\gamma_i}$$

- γ_n is *yes* $\leftarrow$

Example. Suppose KB contains: (1) $a \leftarrow b \wedge c$, (2) $b \leftarrow d \wedge e$, (3) $b \leftarrow g \wedge e$, (4) $c \leftarrow e$, (5) $f \leftarrow a \wedge g$
Observation Percepts contains (6) d, (7) e.
Suppose the query is ask a.

An SLD derivation is

$$
\begin{array}{lr}
yes \leftarrow a & \textit{Goal} \\
yes \leftarrow b \wedge c & \textit{Res}.(1) \\
yes \leftarrow c \wedge d \wedge e & \textit{Res}.(2) \\
yes \leftarrow d \wedge e & \textit{Res}.(4) \\
yes \leftarrow e & \textit{Res}.(6) \\
yes \leftarrow & \textit{Res}.(7)
\end{array}
$$

SLD_Resolution(KB, Percepts, g)

INPUT: Definite clause knowledge base KB, query ask g
OUTPUT: true if KB $\cup$ Percepts $\vdash g$; false if KB $\cup$ Percepts $\nvdash g$.

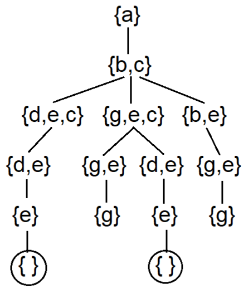
Create a set $G \leftarrow \{g\}$
repeat
  if KB does not contain a clause with head a for any $a \in G$ **then**
    return false
  end if
  Select an atom a in G
  Choose a definite clause $a \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB $\cup$ Percepts with a as head
  $B \leftarrow \{b_1, b_2, \ldots, b_m\}$
  $G \leftarrow B \cup (G \setminus \{a\})$
until $G = \emptyset$
return true

Note. The algorithm described above may lead to a wrong path.

E.g. Suppose KB contains: (1) $a \leftarrow b \wedge c$, (2) $b \leftarrow d \wedge e$, (3) $b \leftarrow g \wedge e$, (4) $c \leftarrow e$, (5) $f \leftarrow a \wedge g$. Percepts contains (6) d, (7) e. A possible execution is

$yes \leftarrow a$	*Goal*
$yes \leftarrow b \wedge c$	*Res.*(1)
$yes \leftarrow g \wedge e \wedge c$	*Res.*(3)

The SLD resolution implies a search tree:



- Thus SLD resolution can be performed using a search algorithm, introduced in previous lectures.
- **Soundness of SLD Resolution:** If the search procedure has derived the goal, the rules used can be used by forward chaining to infer the query.
- **Completeness of SLD Resolution:** If forward chaining can derive an atom, then the rules used can be used to construct an SLD derivation [2].

[2] For completeness of SLD resolution, we need to use a complete search method, e.g., BFS, ID, etc. that will not go into an infinite path

Summary of The Topic

The following are the main knowledge points covered:

- **Propositional Logic Inference Problem:** Decide if $\alpha \models \beta$
- Equivalence between LIP and CSP: $\alpha \models \beta$ iff $\alpha \land \neg\beta$ is not satisfiable.
- **Definite clause inference engine:** `ask` b
- Proof, Proof procedure.
- Two desirable properties of a proof procedure
  - soundness
  - completeness
- **Forward chaining for definite clauses:** Modus Ponens.
- **SLD resolution for definite clauses:** Resolution.