Artificial Intelligence

Topic 12

Logical Inference

Reading: Russell and Norvig, Chapter 7, Section 5
Outline

♦ Inference systems

♦ Soundness andCompleteness

♦ Proof methods
  – normal forms
  – forward chaining
  – backward chaining
  – resolution
From Entailment to Inference

• We have answered what it means to say \( \alpha \) follows from from a knowledge base \( KB \):
  \[ KB \models \alpha \]

• We have seen that this can be determined semantically by model checking or by truth table enumeration
  \[ 2^n \] models or rows for \( n \) symbols

• Is there a better way?
  – can we do it from syntax alone?
  – can we automate it?
  – can we even turn it into a programming language?
Inference Systems

Inference system - set of *rules* for deriving new sentences from existing ones

- AKA Proof System, Derivation System, Theorem-Proving System
- rules operate directly on syntax

Example:

$P_1 = \text{Socrates is a man}$
$P_2 = \text{Socrates is mortal}$
$P_1 \Rightarrow P_2$ (If Socrates is a man, then Socrates is mortal)

Assume $KB = \{P_1, P_1 \Rightarrow P_2\}$.

We know $KB \models P_2$. (check)

What about inference rules?
Inference Systems

Modus Ponens

\[
\alpha \quad \alpha \Rightarrow \beta \\
\hline
\beta 
\]

pattern matching — from sentences that match \( \alpha \) and \( \alpha \Rightarrow \beta \), generate a new sentence that matches \( \beta \)

More examples...

And Elimination

\[
\alpha \land \beta \\
\hline
\alpha 
\]

Or Introduction

\[
\alpha \\
\hline
? 
\]
Inference Systems

Modus Tolens

\[ \begin{align*}
\neg \beta & \quad \alpha \Rightarrow \beta \\
\hline \\
\neg \alpha
\end{align*} \]

An inference system may contain one or more inference rules.

Notation:

\[ KB \vdash \alpha = \text{sentence } \alpha \text{ can be derived from } KB \text{ using the rules of the inference system} \]
Soundness and Completeness

In fact we could make up any inference rule we like. How about:

And Introduction

\[
\frac{\alpha \quad \beta}{\alpha \land \beta}
\]

Why wouldn't we want this rule in our inference system?
Soundness and Completeness

We only want rules that “correspond” to logical consequences. Formally…

**Soundness**: an inference system is *sound* if
whenever $KB \vdash \alpha$, it is also true that $KB \models \alpha$

That is, it *only* allows you to generate logical consequences.

**Completeness**: an inference system is *complete* if
whenever $KB \models \alpha$, it is also true that $KB \vdash \alpha$

That is, it allows you to generate *all* logical consequences.

(Which do you think is worse, sound but not complete, or complete but not sound?)

Ideally we would like to use an inference system that is both sound and complete.
Recall our logical agent:

- **Inference engine**
- **Knowledge base**

- **Knowledge base** = set of sentences in a **formal** language ✓
- **TELL** it what it needs to know ✓
  \[(KB \leftarrow KB \cup \{\alpha\})\]
- **ASK** — answers should follow from the KB ✓?
  \[(KB \vdash \alpha)\]

A sound and complete inference system means that if \(\alpha\) follows from \(KB\) then there is a sequence of rule applications that allow you to generate \(\alpha\) starting with \(KB\) — but it doesn’t tell you *how* to get there!
Proof Methods

Consequences of \( KB \) are a haystack; \( \alpha \) is a needle. Entailment = needle in haystack; inference = finding it
Proof Methods

Application of inference rules

– Legitimate (sound) generation of new sentences from old
– Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.
  Typically require translation of sentences into a normal form

Examples

• forward and backward chaining (Horn form)
• resolution (conjunctive normal form)
Horn Form

**Horn Form** (restricted)

\[ \text{KB} = \text{conjunction of Horn clauses} \]

Horn clause =

- proposition symbol or
- \((\text{conjunction of symbols} \Rightarrow \text{symbol})\)

E.g., \( C \land (B \Rightarrow A) \land (C \land D \Rightarrow B) \)

**Modus Ponens** (for Horn Form): complete for Horn KBs

\[
\frac{\alpha_1, \ldots, \alpha_n, \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta}{\beta}
\]

Can be used with **forward chaining** or **backward chaining**. These algorithms are very natural and run in linear time.
Forward chaining

Idea: systematically iterate through knowledge base, fire any rule whose premises are satisfied in the $KB$, add its conclusion to the $KB$, until query is found.

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example
Forward chaining example

Forward chaining example
Forward chaining example

Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Backward chaining

Idea: work backwards from the query $q$:
   to prove $q$ by BC,
       check if $q$ is known already, or
   prove by BC all premises of some rule concluding $q$

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal
   1) has already been proved true, or
   2) has already failed
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example

Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Forward vs. backward chaining

FC is data-driven, cf. automatic, unconscious processing, e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving, e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be much less than linear in size of KB
Resolution

**Conjunctive Normal Form (CNF—universal)**

*conjunction* of *disjunctions* of *literals*

*clauses*

E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)

**Resolution** inference rule (for CNF): complete for propositional logic

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n
\]

where \(\ell_i\) and \(m_j\) are complementary literals. E.g.,

\[
P_{1,3} \lor P_{2,2}, \quad \neg P_{2,2}
\]

\[
P_{1,3}
\]

Resolution is sound and complete for propositional logic
Conversion to CNF

\[ B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \]

1. Eliminate \( \iff \), replacing \( \alpha \iff \beta \) with \( (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) \).

\[ (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}) \]

2. Eliminate \( \Rightarrow \), replacing \( \alpha \Rightarrow \beta \) with \( \neg \alpha \lor \beta \).

\[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \lor P_{2,1}) \lor B_{1,1}) \]

3. Move \( \neg \) inwards using de Morgan’s rules and double-negation:

\[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1}) \]

4. Apply distributivity law (\( \lor \) over \( \land \)) and flatten:

\[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1}) \]

Resolution algorithm

“Proof by contradiction”

Based on the fact that $KB \models \alpha$ iff $KB \cup \{\neg \alpha\}$ is unsatisfiable (prove!)

Unsatisfiability in CNF is indicated by the empty clause

We repeatedly apply the resolution rule to $\neg \alpha$ and its consequences until we derive the empty clause.

Exercise: What is the complexity of the conversion to CNF?
What is the complexity of the resolution algorithm?
Resolution example

\[ KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \quad \alpha = \neg P_{1,2} \]

Resolution combined with first-order logic is the key mechanism used in logic programming (eg PROLOG).
Efficient Reasoning Algorithms

The satisfiability problem for propositional logic in both infeasible (NP-complete) and very useful (TSP, CSP, planning, etc).

So if an agent is required to perform propositional reasoning, what kind of efficient mechanisms are available?

We will look at two possible algorithms: ♦ The *Davis-Putnam algorithm* is a recursive DFS for satisfying models of a formula, aided by some heuristics; and ♦ *WALKSAT* is a randomized algorithm that performs a local search for a satisfying model.
The DPLL Algorithm (1962)

DPLL is a variation of the Davis-Putnam algorithm. It takes the input as a sentence in CNF, and iterates through potential models using the following heuristics:

- **Early Termination** The algorithm recognizes if a clause is true (one of its literals is true) or if a sentence of false (one of its clauses is false) and therefore does not need to search redundant branches of the search tree.

- **Pure Symbols** A *pure symbol* is a symbol that has the same sign in all (active) clauses. These symbols can be ignored.

- **Unit Clause** A *unit clause* is a clause with just one (active) literal. A unit clause dictates the value of that literal in all the other clauses.
function DPLL-Satisfiable?($s$) returns true or false
    inputs: $s$, a sentence in propositional logic
    $clauses \leftarrow$ the set of clauses in the CNF representation of $s$
    $symbols \leftarrow$ a list of the proposition symbols in $s$
    return DPLL($clauses, symbols, []$)

function DPLL($clauses, symbols, model$) returns true or false
    if every clause in $clauses$ is true in $model$ then return true
    if some clause in $clauses$ is false in $model$ then return false
    $P, value \leftarrow$ Find-Pure-Symbol($symbols, clauses, model$)
    if $P$ is non-null then return DPLL($clauses, symbols-P, [P = value | model]$)
    $P, value \leftarrow$ Find-Unit-Clause($clauses, model$)
    if $P$ is non-null then return DPLL($clauses, symbols-P, [P = value | model]$)
    $P \leftarrow$ First($symbols$); $rest \leftarrow$ Rest($symbols$)
    return DPLL($clauses, rest, [P = true | model]$) or DPLL($clauses, rest, [P = false | model]$)
The WALKSAT algorithm

The WALKSAT algorithm simply performs a random walk over all models hoping to find a model that satisfies a sentence in CNF.

For each step of the walk it flips the value of a symbol (proposition) and tests if the sentence becomes true.

The algorithm nondeterministically chooses either a randomly selected proposition to flip, or chooses the proposition that maximizes the number of satisfied clauses.
The WALKSAT algorithm

function \texttt{WALKSAT}(\texttt{clauses, }p, \texttt{max-flips}) returns a satisfying model or failure

inputs: \texttt{clauses}, a set of clauses in propositional logic

\[ p \text{, the probability of choosing to do a “random walk” move, typically around 0.5} \]

\texttt{max-flips}, number of flips allowed before giving up

\[ \texttt{model} \leftarrow \text{a random assignment of } \texttt{true/false} \text{ to the symbols in } \texttt{clauses} \]

for \( i = 1 \) to \texttt{max-flips} do

if \texttt{model} satisfies \texttt{clauses} then return \texttt{model}

\[ \texttt{clause} \leftarrow \text{a randomly selected clause from } \texttt{clauses} \text{ that is false in } \texttt{model} \]

with probability \( p \) flip the value in \texttt{model} of a randomly selected symbol from \texttt{clause}

else flip whichever symbol in \texttt{clause} maximizes the number of satisfied clauses

return \texttt{failure}
More Logics

Although propositional logic is computationally attractive, it lacks expressive power in practice.

eg. How would you say “All men are mortal” or “All squares adjacent to a pit have a breeze”?

There are many other logics that extend propositional logic, eg:

- first-order logic introduces objects, functions, relations, variables, quantifiers (*for all, there exists*)
- higher order logics allow the logic to refer to its own constructs
- temporal logics introduce specific structures to represent time steps
- modal logics introduce possibility and necessity
- probabilistic logics introduce the probability a statement is true
- fuzzy logics introduce a degree of membership to a class
Summary

Logical agents apply inference to a knowledge base to derive new information and make decisions.

Basic concepts of logic:
- **syntax:** formal structure of sentences
- **semantics:** truth of sentences wrt models
- **entailment:** necessary truth of one sentence given another
- **inference:** deriving sentences from other sentences
- **soundness:** derivations produce only entailed sentences
- **completeness:** derivations can produce all entailed sentences

Forward, backward chaining are linear-time, complete for Horn clauses.

Resolution is complete for propositional logic.

Resolution is the basis of the Prolog programming language.

Uses first-order logic — more expressive power.