Artificial Intelligence

Topic 12

Logical Inference

Reading: Russell and Norvig, Chapter 7, Section 5
Extra reading: Russell and Norvig, Sections 8.1, 8.2, 9.4 (not examinable)

Outline

♦ Inference systems
♦ Soundness and Completeness
♦ Proof methods
  – normal forms
  – forward chaining
  – backward chaining
  – resolution
♦ More logics
♦ Logic Programming

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
  – $2^\text{n}$ 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
\[ \alpha \alpha \Rightarrow \beta \]
\[ \beta \]
pattern matching — from sentences that match \( \alpha \) and \( \alpha \Rightarrow \beta \), generate a new sentence that matches \( \beta \)

More examples...

And Elimination
\[ \alpha \land \beta \]
\[ \alpha \]

Or Introduction
\[ \alpha \]
\[ \beta \]

Soundness and Completeness

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

And Introduction
\[ \alpha \]
\[ \alpha \land \beta \]

Why wouldn’t we want this rule in our inference system?

An inference system may contain one or more inference rules.

Notation:

\[ KB \vdash \alpha \] = sentence \( \alpha \) can be derived from \( KB \) using the rules of the 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.
Review of Progress

Recall our logical agent:

<table>
<thead>
<tr>
<th>Inference engine</th>
<th>domain-independent algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge base</td>
<td>domain-specific content</td>
</tr>
</tbody>
</table>

- **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)

- **KB** = conjunction of **Horn clauses**
- **Horn clause** =
  - \( \diamond \) proposition symbol or
  - \( \diamond \) (conjunction of symbols \( \Rightarrow \) symbol)

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

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

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

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 diagram](image)

### 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 diagram](image)
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)

\[ (A \lor \neg B) \land (B \lor \neg C \lor \neg D) \]

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

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

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

\[
\begin{align*}
P_{1,3} \lor P_{2,2}, & \quad \neg P_{2,2} \\
P_{1,3} & \\
\end{align*}
\]

Resolution is sound and complete for propositional logic.

Conversion to CNF

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

1. Eliminate \( \leftrightarrow \), replacing \( \alpha \leftrightarrow \beta \) 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

(Consider the number of models satisfying

(A ∨ B ∨ C ∨ · · ·)
(A ∨ B ∨ C)
(A ∨ B)
(A)
()

Vacuously false)

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

Resolution example

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

Logic Programming

PROLOG

- most common logic programming language (recall Japanese 5th gen.)
- resolution inference rule
- first-order logic
- goal-directed, depth-first search (no occurs check)

Others include:

- GÖDEL — second-order logic
- GOLOG — agent programming

Inductive Logic Programming (ILP) — combining inductive learning with
logic programming
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.