First-order Logic

26 October 2020
Programming Assignment 1: Search

Programming Assignment 2: Multi-agent search

Exam 1

Programming Assignment 3: Reinforcement learning
  Out now!

Programming Assignment 4: Classification

Exam 2

Final paper
  Out now!
Where are we?
One possible agent architecture: *knowledge* and *inference*.

- Logics provide a formal way to encode knowledge.
- Logical inference computes entailment relations among sentences.
A logic is defined by its syntax, set of possible worlds, and truth condition.

The first logic we looked at is *propositional logic*. 
In propositional logic, each variable represents a
*proposition*, which is true or false.

We can directly apply *connectives* to propositions:

- Negation: $\neg p$
- Conjunction: $p \land q$
- Disjunction: $p \lor q$
- Implication: $p \Rightarrow q$
- Biconditional: $p \Leftrightarrow q$

We can see the possible truth values for a statement by checking every truth assignment for the variables.
SAT solvers based on DPLL provide incredibly efficient inference for propositional logic.

Logical agents can construct plans by asking whether there is a future in which the goal is achieved.
Knowledge-based agents
A knowledge-based agent

\[ t = 0 \]

```python
def kb_agent(kb, percept):
    tell(kb, make_percept_sentence(percept, t))
    action = ask(kb, make_action_query(t))
    tell(kb, make_action_sentence(action, t))
    t += 1
    return action
```
Example: Partially observable Pac-Man

Pac-Man has to act given only *local perception* – he can only see the four squares (or walls) around him.

What knowledge does he need and how can we represent it?
Variables

*Pac-Man’s location*, e.g., $A_{1,1}^0$: Pac-Man is at $[1, 1]$ at time 0
Variables

**Pac-Man’s location**, e.g., $A_{1,1}^0$: Pac-Man is at [1, 1] at time 0

**Wall locations**, e.g., $Wall_{0,0}$, $Wall_{0,1}$, etc. (These don’t change with time.)
Variables

*Pac-Man’s location*, e.g., $A_{t,1}^0$: Pac-Man is at $[1, 1]$ at time 0

*Wall locations*, e.g., $Wall_{0,0}$, $Wall_{0,1}$, etc. (These don’t change with time.)

*Percepts*, e.g., $Blocked^0_W$: Blocked by wall to my West at time 0.
Variables

*Pac-Man’s location*, e.g., $A_{1,1}^0$: Pac-Man is at $[1, 1]$ at time 0

*Wall locations*, e.g., $W_{0,0}$, $W_{0,1}$, etc. (These don’t change with time.)

*Percepts*, e.g., $Block_{W}^0$: Blocked by wall to my West at time 0.

*Actions*, e.g, $W^0$: Pac-Man moves West at time 0
Variables

*Pac-Man’s location*, e.g., \(A_{1,1}^0\): Pac-Man is at \([1, 1]\) at time 0

*Wall locations*, e.g., \(W_{0,0}^0, \ W_{0,1}^0\), etc. (These don’t change with time.)

*Percepts*, e.g., \(Blocked_W^0\): Blocked by wall to my West at time 0.

*Actions*, e.g, \(W^0\): Pac-Man moves West at time 0

An \(N \times N\) world for \(T\) time steps leads to \(N^2T + N^2 + 4T + 4T = O(N^2T)\) variables.

That means \(2^{N^2T}\) possible worlds!

If \(N = 10, T = 100\) leads to \(10^{3010}\) worlds, each a “history”. 
Sensor model

Sentences state facts about how Pac-Man’s current percept variables are determined by the current state variables.
Sensor model

Sentences state facts about how Pac-Man’s current percept variables are determined by the current state variables.

For Partially Observable Pac-Man, the state variables are $A_{txy}^t$, e.g., $A_{3,3}^{17}$. 
Sensor model

Sentences state facts about how Pac-Man’s current percept variables are determined by the current state variables.

For Partially Observable Pac-Man, the state variables are \( At_{x,y}^t \), e.g., \( At_{3,3}^1 \).

E.g., Pac-Man perceives a wall to the West at time \( t \) if and only if he is in \([x, y]\) and there is a wall at \([x-1, y]\)…

\[
\text{Blocked}_W^0 \leftrightarrow ((At_{1,1}^0 \land Wall_{0,1}) \lor (At_{1,2}^0 \land Wall_{0,2}) \lor (At_{1,3}^0 \land Wall_{0,3}) \lor \cdots)
\]
Sensor model

Sentences state facts about how Pac-Man’s current percept variables are determined by the current state variables.

For Partially Observable Pac-Man, the state variables are $At_{x,y}^t$, e.g., $At_{3,3}^{17}$.

E.g., Pac-Man perceives a wall to the West at time $t$ if and only if he is in $[x, y]$ and there is a wall at $[x-1, y]$…

$$Blocked_W^0 \iff (At_{1,1}^0 \land Wall_{0,1}) \lor (At_{1,2}^0 \land Wall_{0,2}) \lor (At_{1,3}^0 \land Wall_{0,3}) \lor \cdots$$

How many of these sentences do we need?
Sensor model

Sentences state facts about how Pac-Man’s current percept variables are determined by the current state variables.

For Partially Observable Pac-Man, the state variables are $At_{x,y}^t$, e.g., $At_{3,3}^1$.

E.g., Pac-Man perceives a wall to the West at time $t$ if and only if he is in $[x, y]$ and there is a wall at $[x-1, y]$...

$$Blocked^0_W \Leftrightarrow ((At_{1,1}^0 \land Wall_{0,1}) \lor (At_{1,2}^0 \land Wall_{0,2}) \lor (At_{1,3}^0 \land Wall_{0,3}) \lor \cdots)$$

**How many of these sentences do we need?**

**How big are they?**
Quiz

What’s wrong with sentences like

\[ \text{At}_{1,1}^0 \land \text{Wall}_{0,1} \Rightarrow \text{Blocked}_W^0 \]

“If you’re at [1, 1] at time 0 and there’s a wall in [0, 1], the west percept is blocked.”
Quiz

What’s wrong with sentences like

\[ \text{At}_{1,1}^0 \land \text{Wall}_{0,1} \Rightarrow \text{Blocked}_W^0 \]

“If you’re at [1, 1] at time 0 and there’s a wall in [0, 1], the west percept is blocked.”

True – but incomplete!

They say “under these conditions the percept variable is \textit{true}”. They don’t say when it’s \textit{false}.

In particular, they allow for worlds where the percept is \textit{always} true!
Transition model

Sentences stating how the next-state variables are determined by the current state variables and Pac-Man’s action.
Transition model

Sentences stating how the next-state variables are determined by the current state variables and Pac-Man’s action.

A state variable gets its value according to a successor-state axiom:

\[ X^t \leftrightarrow [X^{t-1} \land \neg(some\ action^{t-1}\ made\ it\ false)] \lor [\neg X^{t-1} \land (some\ action^{t-1}\ made\ it\ true)] \]
Transition model

Sentences stating how the next-state variables are determined by the current state variables and Pac-Man’s action.

A state variable gets its value according to a successor-state axiom:

\[ X_t \iff [X_{t-1} \land \neg (\text{some action}_{t-1} \text{ made it false})] \lor \\
[\neg X_{t-1} \land (\text{some action}_{t-1} \text{ made it true})] \]

For Partially Observable Pac-Man, the state variables are \( At_{xy}^t \), e.g., \( At_{3,3}^{17} \).

\[ At_{3,3}^{17} \iff [At_{3,3}^{16} \land \neg ((\neg Wall_{3,4} \land N^{16}) \lor (\neg Wall_{4,3} \land E^{16}) \lor \cdots)] \lor \\
[\neg At_{3,3}^{16} \land ((At_{3,2}^{16} \land \neg Wall_{3,3} \land N^{16}) \lor (At_{2,3}^{16} \land \neg Wall_{3,3} \land N^{16}) \lor \cdots)] \]
Initial conditions

What Pac-Man knows about the initial state.
Initial conditions

What Pac-Man knows about the initial state.

Pac-Man might know its initial location:

\[ A_{t,1}^0 \]
Initial conditions

What Pac-Man knows about the initial state.

Pac-Man might know its initial location:

\(At_{1,1}^0\)

Or, it might not:

\(At_{1,1}^0 \lor At_{1,2}^0 \lor At_{1,3}^0 \lor \cdots \lor At_{3,3}^0\)
Initial conditions

What Pac-Man knows about the initial state.

Pac-Man might know its initial location:

$$At_{1,1}^0 \land \neg At_{1,2}^0 \land \neg At_{1,3}^0 \ldots$$

Or, it might not:

$$At_{1,1}^0 \lor At_{1,2}^0 \lor At_{1,3}^0 \lor \ldots \lor At_{3,3}^0$$
Domain constraints

What is generally true in the world.

E.g., Pac-Man must take exactly one action at a time:

\[ \neg(W^0 \land E^0) \land \neg(W^0 \land S^0) \land \cdots \]

\[ \neg(W^1 \land E^1) \land \neg(W^1 \land S^1) \land \cdots \]

\[ \cdots \land (W^0 \lor E^0 \lor N^0 \lor S^0) \land \cdots \]

And Pac-Man must be in exactly one place at a time…
**State estimation** means keeping track of what's true now.

A logical agent can just ask itself!

E.g., ask whether $KB \land \langle actions \rangle \land \langle percepts \rangle \models Wall_{2,2}$

This is “lazy”: it involves reasoning about one’s whole life history at each step!
**State estimation** means keeping track of what's true now.

A logical agent can just ask itself!

E.g., ask whether $KB \land \langle \text{actions} \rangle \land \langle \text{percepts} \rangle \models Wall_{2,2}$

This is “lazy”: it involves reasoning about one’s whole life history at each step!

A more “eager” form of state estimation:

After each action and percept,

For each state variable $X^t$:

- If $X^t$ is entailed, add to KB.
- If $\neg X^t$ is entailed, add to KB.
Planning as satisfiability

Given a hyper-efficient SAT solver, can we use it to make plans?

Yes, for fully observable, deterministic case:

- Planning problem is solvable iff there is some satisfying assignment.
- Solution is obtained from truth values of action variables.

For $T = 1$ to $\infty$:

- Set up the KB as follows:
  - Initial state, domain constraints
  - Transition model sentences up to time $T$
  - Goal is true at time $T$
- Run SAT solver and read off action variables from the solution.
First-order logic
CONCEPT-SCRIPT

A Formal Language of Pure Thought on the Pattern of Arithmetic
Spectrum of representations

Atomic

Factored

Structured

Search, game-playing

CSPs, planning, propositional logic, Bayes nets, neural nets

First-order logic, databases, probabilistic programs
The universe of propositional logic

\[ p \land q \Rightarrow \neg r \lor \neg s \]
The universe of propositional logic

\[ p \land q \Rightarrow \neg r \lor \neg s \]
First-order logic is a logical system for reasoning about properties of objects.
The universe of first-order logic
In first-order logic, each name refers to some object in a set called the domain of discourse.

Some objects may have multiple names.

Some objects may have no name at all.
First-order logic augments the connectives from propositional logic with

**predicates** that describe properties of objects, **functions** that map objects to one another, and **quantifiers** that allow us to reason about multiple objects.
Likes(You, Music) \land Likes(You, Plays) \\
\Rightarrow Likes(You, MusicalTheater) \\
Learns(You, History) \lor \\
ForeverRepeats(You, History)
Likes(You, Music) \land Likes(You, Plays) \\
\Rightarrow Likes(You, MusicalTheater) \\
Learns(You, History) \lor \\
ForeverRepeats(You, History) \\

These are constant symbols. They refer to objects, not propositions.
Likes(You, Music) \land Likes(You, Plays)
\Rightarrow Likes(You, MusicalTheater)

Learns(You, History) \lor
ForeverRepeats(You, History)

These are *predicates*. They take objects as arguments and evaluate to true or false.
Likes\((\text{You, Music}) \land \text{Likes(You, Plays)}\) \\
⇒ Likes\((\text{You, MusicalTheater})\) \\
\text{Learns(You, History) ∨} \\
\text{ForeverRepeats(You, History)}

*What remain are traditional propositional connectives. Because each predicate evaluates to true or false, we can connect the truth values of predicates using normal propositional connectives.*
To reason about objects, first-order logic uses *predicates*, e.g.,

*Cute*(*Quokka*)

*Argue*(*Democrats, Republicans*)

Applying a predicate to arguments produces a proposition, which is either true or false.
Each predicate can take a fixed number of arguments, called its *arity*.

So, in first-order logic, you can’t have both

- `Eat(Garfield, Lasagna)`
- `Eat(Garfield, Lasagna, Home)`

Since they use the same predicate, `Eat`, with different numbers of arguments.
Sentences in first-order logic can be constructed from predicates applied to objects:

\[
\text{Cute}(A) \Rightarrow \text{Bunny}(A) \lor \text{Kitty}(A) \lor \text{Puppy}(A)
\]

\[
\text{Succeeds}(\text{You}) \Leftrightarrow \text{Practices}(\text{You})
\]

\[
x < 8 \Rightarrow x < 137
\]

*The < sign is just another predicate. Binary predicates are sometimes written in infix notation this way.*

*Numbers aren’t “built in” to FOL. They’re constant symbols just like “You” or “A” above.*
First-order logic is equipped with the special predicate $=$ that says whether two objects are equal to one another.

- $\text{MorningStar} = \text{EveningStar}$
- $\text{TomMarvoloRiddle} = \text{LordVoldemort}$
Equality is a part of FOL, just as $\Rightarrow$ and $\neg$ are.

For notational simplicity, we can define $\neq$ as

$$x \neq y \equiv \neg(x = y).$$

Equality can only be applied to objects; to see if propositions are equal, use $\Leftrightarrow$. 
favoriteMovieOf(You) ≠ favoriteMovieOf(Date) ∧
starOf(favoriteMovieOf(You)) = starOf(favoriteMovieOf(Date))
\[ \text{favoriteMovieOf}(\text{You}) \neq \text{favoriteMovieOf}(\text{Date}) \land \\
\text{starOf}(\text{favoriteMovieOf}(\text{You})) = \text{starOf}(\text{favoriteMovieOf}(\text{Date})) \]
favoriteMovieOf(You) ≠ favoriteMovieOf(Date) ∧
starOf(favoriteMovieOf(You)) = starOf(favoriteMovieOf(Date))

*These are functions. Functions take objects as input and produce objects as output.*
favoriteMovieOf(You) ≠ favoriteMovieOf(Date) ∧
starOf(favoriteMovieOf(You)) = starOf(favoriteMovieOf(Date))
favoriteMovieOf(You) ≠ favoriteMovieOf(Date) ∧
starOf(favoriteMovieOf(You)) = starOf(favoriteMovieOf(Date))
First-order logic allows *functions*, which return objects associated with other functions.

Examples:

```
colorOf(Money)
```
```
medianOf(X, Y, Z)
```
```
X + Y
```

As with predicates, functions can have any fixed arity, but they always return a single value.

Functions evaluate to objects, not propositions.
When working in first-order logic, be careful to keep *objects* (things) and *propositions* (true or false) separate.

You cannot apply connectives to objects:

\[
\text{Venus} \Rightarrow \text{TheSun}
\]

You cannot apply functions to propositions:

\[
\text{starOf}(\text{IsRed(Sun)} \land \text{IsGreen(Mars)})
\]
## Type-checking table

<table>
<thead>
<tr>
<th></th>
<th>Operate on</th>
<th>Produce</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connectives</strong></td>
<td>propositions</td>
<td>a proposition</td>
</tr>
<tr>
<td>like $\Rightarrow$ and $\wedge$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Predicates</strong></td>
<td>objects</td>
<td>a proposition</td>
</tr>
<tr>
<td>like $=$ and Loves</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>objects</td>
<td>an object</td>
</tr>
<tr>
<td>like ageOf and length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
One last (major) change
Some muggle is intelligent.
Some muggle is intelligent.

$\exists m . (\text{Muggle}(m) \land \text{Intelligent}(m))$
Some muggle is intelligent.

\[ \exists m . (\text{Muggle}(m) \land \text{Intelligent}(m)) \]

\( \exists \) is the **existential quantifier** and says “for some choice of \( m \), the following is true.”
The existential quantifier

A statement of the form

$$\exists x . \varphi$$

is true if, for some choice of $x$, the statement $\varphi$ is true when that $x$ is plugged into it.

Examples:

$$\exists x . (\text{Even}(x) \land \text{Prime}(x))$$
$$\exists x . (\text{TallerThan}(x, \text{Me}) \land \text{LighterThan}(x, \text{Me}))$$
$$(\exists w . \text{Will}(w)) \Rightarrow (\exists x . \text{Way}(x))$$
For any natural number $n$, $n$ is even if and only if $n^2$ is even.
For any natural number $n$, 
$n$ is even if and only if $n^2$ is even.

$$\forall n . \ (n \in \mathbb{N} \Rightarrow (\text{Even}(n) \iff \text{Even}(n^2)))$$
For any natural number $n$, $n$ is even if and only if $n^2$ is even.

$\forall n . (n \in \mathbb{N} \Rightarrow (\text{Even}(n) \leftrightarrow \text{Even}(n^2)))$

\(\forall\) is the universal quantifier and says “for any choice of \(n\), the following is true.”
The universal quantifier

A statement of the form

\[ \forall x . \varphi \]

is true if, for every choice of \( x \), the statement \( \varphi \) is true when that \( x \) is plugged into it.

Examples:

\[ \forall k . (\text{Kitten}(k) \Rightarrow \text{Cute}(k)) \]
\[ \text{Richest(JeffBezos)} \Rightarrow \]
\[ \forall x . (x \neq \text{JeffBezos} \Rightarrow \text{RicherThan(JeffBezos, x)}) \]
Expressive power

Rules of chess:

Propositional logic: 100 000 pages
First-order logic: 1 page
Expressive power

Rules of chess:

- Propositional logic: 100 000 pages
- First-order logic: 1 page

Rules of Pac-Man:

\[
\forall x, y, t \ At(x, y, t) \iff [At(x, y, t - 1) \land \neg \exists u, v \ \text{Reachable}(x, y, u, v, \text{action}(t - 1))] \lor \\
[\exists u, v \ At(u, v, t - 1) \land \text{Reachable}(x, y, u, v, \text{action}(t - 1))]
\]
Inference in FOL
**Entailment** is defined the same as for propositional logic:

\[ \varphi \models \psi \text{ ("\varphi \text{ entails } \psi \" or "\psi \text{ follows from } \varphi \") iff in every world where } \varphi \text{ is true, } \psi \text{ is also true.} \]

E.g., \( \forall x \, \text{Knows}(x, \text{Obama}) \) entails \( \exists y \, \forall x \, \text{Knows}(x, y) \)

If asked “Do you know what time it is?”, it’s rude to say “Yes”.

Similarly, given an existentially quantified query, it’s polite to provide an answer in the form of a substitution (or binding) for the variable(s):

\[
\begin{align*}
\text{KB} &= \{ \forall x \, \text{Knows}(x, \text{Obama}) \} \\
\text{Query} &= \exists y \forall x \, \text{Knows}(x, y) \\
\text{Answer} &= \text{Yes, } \{ y/\text{Obama} \}
\end{align*}
\]

Applying the substitution should produce a sentence that is entailed by KB.
Propositionalization

Convert \((KB \land \neg a)\) to propositional logic, and use a PL SAT solver to check (un)satisfiability.

Replace variables with ground terms and convert atomic sentences to symbols:

\[\forall x \text{Knows}(x, Obama) \land \text{Democrat}(Biden)\]
\[\rightarrow \text{Knows}(Obama, Obama) \land \text{Knows}(Biden, Obama) \land \text{Democrat}(Biden)\]
\[\rightarrow K OO \land K BO \land DB\]
Propositionalization

Convert \((KB \land \neg a)\) to propositional logic, and use a PL SAT solver to check (un)satisfiability.

Replace variables with ground terms and convert atomic sentences to symbols:

- \(\forall x \text{Knows}(x, \text{Obama}) \) and \(\forall x \text{Knows}(\text{mother}(x), x)\)
- \(\rightarrow \text{Knows(Obama, Obama)} \) and \(\text{Knows(mother(Obama), Obama)}\) and \(\text{Knows(mother(mother(Obama)), mother(Obama))}\) and …

For \(k = 1\) to \(\infty\), use terms of function nesting depth \(k\):

- If entailed, will find a contradiction for some finite \(k\).
- If not, may continue for ever – semidecidable!
Lifted inference
Lifted inference

Apply inference rules directly to first-order sentences, e.g.,

\[ KB = \{ \text{Person}(Socrates), \ \forall x \ \text{Person}(x) \Rightarrow \text{Mortal}(x) \} \]

conclude \text{Mortal}(Socrates)
Lifted inference

Apply inference rules directly to first-order sentences, e.g.,

\[ KB = \{ \text{Person}(Socrates), \forall x \ \text{Person}(x) \Rightarrow \text{Mortal}(x) \} \]

conclude \( \text{Mortal}(Socrates) \)

The general rule is a version of *modus ponens*:

Given \( a[x] \Rightarrow \beta[x] \) and \( a' \), where \( a'\sigma = a[x]\sigma \) for some substitution \( \sigma \) conclude \( \beta[x]\sigma \)

\( \sigma \) is \( \{x/\text{Socrates}\} \)

Given \( \text{Knows}(x, \text{Obama}) \) and \( \text{Knows}(y, z) \Rightarrow \text{Likes}(y, z) \)

\( \sigma \) is \( \{y/x, z/\text{Obama}\} \), conclude \( \text{Likes}(x, \text{Obama}) \)
Lifted inference

Apply inference rules directly to first-order sentences, e.g.,

\[ KB = \{ \text{Person}(Socrates), \forall x \text{ Person}(x) \Rightarrow \text{Mortal}(x) \} \]

conclude \text{Mortal}(Socrates)

The general rule is a version of \textit{modus ponens}:

Given \( a[x] \Rightarrow \beta[x] \) and \( a' \), where \( a'\sigma = a[x]\sigma \) for some substitution \( \sigma \) conclude \( \beta[x]\sigma \)

\( \sigma \) is {\( x \)/Socrates}

Given \( \text{Knows}(x, \text{Obama}) \) and \( \text{Knows}(y, z) \Rightarrow \text{Likes}(y, z) \)

\( \sigma \) is {\( y/x, z/\text{Obama} \)}, conclude \( \text{Likes}(x, \text{Obama}) \)

Examples: Prolog (backward chaining), Datalog (forward chaining), production rule systems (forward chaining), resolution theorem provers
Knowledge-based agents, revisited
“Programs with Common Sense”, 1958

Proposes the *Advice Taker* for solving problems by symbolic reasoning.

Uses a formal language – logic – both to describe what the program knows and how it reasons (the “heuristics”).
“The main advantages we expect the advice taker to have is that *its behavior will be improvable merely by making statements to it*, telling it about its symbolic environment and what is wanted from it.”
“One will be able to assume that the advice taker will have available to it a fairly wide class of immediate logical consequences of anything it is told and its previous knowledge.”

That is, use inference to draw conclusions.
“…a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows.”
“I have three pets.”

“Are they cats? Dogs?”
“There was a car crash.”

“Was anyone hurt?”
“It was the **anatomical oddity** of US gymnast George Eyser who won a gold medal on the parallel bars in 1904.”

“What is leg?”
“The computer wouldn’t know that a missing leg is odder than anything else.”

David Ferrucci, 2011
This kind of reasoning requires computers to have the same kind of knowledge about the world that people have.

This is the *knowledge acquisition bottleneck*. 
“Our ultimate objective is to make programs that learn from their experience as effectively as humans do. *It may not be realized how far we are presently from this objective.*”
“For example, [Arthur] Samuel has included in his checker program facilities for improving the weights the machine assigns to various factors in evaluating positions. He has also included a scheme whereby the machine remembers games it has played previously and deviates from its previous play when it finds a position which it previously lost.”

Reinforcement learning was already a thing.
“If one wants a machine to be able to discover an abstraction, it seems most likely that the machine must be able to represent this abstraction in some relatively simple way.”

Propositional logic was simple, but it was terrible for abstract rules!
“We base ourselves on the idea that: In order for a program to be capable of learning something it must first be capable of being told it.

“...Once this is achieved, we may be able to tell the advice taker how to learn from experience.”
“A machine is instructed mainly in the form of a sequence of *imperative sentences*; while a human is instructed mainly in *declarative sentences* describing the situation in which action is required together with a few imperatives that say what is wanted.”
Prolog

PROgramming in LOGic (Colmerauer, 1970s)

General-purpose AI programming language
Based on first-order logic
Declarative rather than imperative
Use centered in Europe and Japan
Some parts of Watson (pattern matching over natural language)
Often used as a component of a system
DENDRAL and MYCIN

“Expert systems” – knowledge based

DENDRAL (Feigenbaum et al., c. 1965)

Identify unknown organic molecules
Eliminate most “chemically implausible” hypotheses

MYCIN (Shortliffe et al., 1970s)

Identify bacteria causing severe infections

“research indicated that it proposed an accepted therapy in about 69% of cases, which was better than the performance of infectious disease experts.”
Summary, pointers

See AIMA ch. 12 (3e) or 10 (4e).

See AIMA ch. 9.
Summary, pointers

First-order logic is a very expressive formal language.

Many domains of commonsense and technical knowledge can be written in FOL.

See AIMA ch. 12 (3e) or 10 (4e).

See AIMA ch. 9.
Summary, pointers

First-order logic is a very expressive formal language.

Many domains of commonsense and technical knowledge can be written in FOL.

Inference is semidecidable in general, but many problems are efficiently solvable in practice

Inference technology for logic programming is especially efficient.

See AIMA ch. 12 (3e) or 10 (4e).

See AIMA ch. 9.
Acknowledgments

The lecture incorporates material from:

Ronald Brachman, Cornell University
George Konidaris, Brown University
Hector Levesque, University of Toronto
Peter Norvig, Google
Stuart Russell, University of California, Berkeley; ai.berkeley.edu
Keith Schwarz, Stanford University