Any questions about previous logic lectures?

**Pacman’s knowledge base**: Transition model

How does each *state variable* at each time gets its value?

- Here we care about location variables, e.g., \( \text{At}_3,3_17 \)

A state variable \( X \) gets its value according to a *successor-state axiom*

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

For Pacman location:

- \( \text{At}_3,3_17 \Leftrightarrow [\text{At}_3,3_16 \land \neg((\neg \text{Wall}_3,4 \land \text{N}_16) \lor (\neg \text{Wall}_4,3 \\
\land \text{E}_16) \lor ...)] \\
\lor [\neg \text{At}_3,3_16 \land ((\text{At}_3,2_16 \land \neg \text{Wall}_3,3 \land \text{N}_16) \lor \\
(\text{At}_2,3_16 \land \neg \text{Wall}_3,3 \land \text{E}_16) \lor ...)]
\]

\( \text{Food}_3,3_17 \Leftrightarrow ?? \)
Reminder: Partially observable Pacman

- Basic question: where am I?
- Variables:
  - Wall_0,0, Wall_0,1, ...
  - Blocked_W_0, Blocked_N_0, ..., Blocked_W_1, ...
  - W_0, N_0, ..., W_1, ...
  - At_0,0_0, At_0,1_0, ..., At_0,0_1, ...
- Sensor model:
  - Blocked_W_0 ⇔ (At_1,1_0 ∧ Wall_0,1) v
    (At_1,2_0 ∧ Wall_0,2) v
    (At_1,3_0 ∧ Wall_0,3) v ...
- Map: where are the walls
- Initial state: Pacman definitely somewhere
- Domain constraints: e.g. only one action per timestep
- Transition model: how state variables change (or don’t)
Localization demo

- Percept
- Action
- Percept
- Action
- Percept
- Percept
Localization demo

- Percept
- Action: WEST
- Percept
- Action
- Percept
- Action
- Percept
Localization demo

- Percept
- Action: *WEST*
- Percept
- Action
- Percept
- Action
- Percept
Localization demo

- Percept
- Action \textit{WEST}
- Percept
- Action \textit{WEST}
- Percept
Localization demo

- Percept
- Action *WEST*
- Percept
- Action *WEST*
- Percept
- Action
- Percept
Localization demo

- Percept
- Action: WEST
- Percept
- Action: WEST
- Percept
- Action: WEST
Localization demo

- Percept
- Action: WEST
- Percept
- Action: WEST
- Percept
- Action: WEST
- Percept
Example: Mapping from a known relative location

- Without loss of generality, call the initial location 0,0
- The percept tells Pacman which actions work, so he always knows where he is
  - “Dead reckoning”
- Initialize the KB with **PacPhysics** for $T$ time steps, starting at 0,0
- Run the Pacman agent for $T$ time steps
  - At each time step
    - Update the KB with previous action and new percept facts
    - For each wall variable $Wall_{x,y}$
      - If $Wall_{x,y}$ is entailed, add to KB
      - If $\neg Wall_{x,y}$ is entailed, add to KB
    - Choose an action
- The wall variables constitute the map
Mapping demo

- Percept
- Action: NORTH
- Percept
- Action: EAST
- Percept
- Action: SOUTH
- Percept
Example: Simultaneous localization and mapping

- Often, dead reckoning won’t work in the real world
  - E.g., sensors just count the *number* of adjacent walls (0,1,2,3 = 2 bits)
- Pacman doesn’t know which actions work, so he’s “lost”
  - So if he doesn’t know where he is, how does he build a map???
- Initialize the KB with *PacPhysics* for $T$ time steps, starting at 0,0
- Run the Pacman agent for $T$ time steps
  - At each time step
    - Update the KB with previous action and new percept facts
    - For each $x,y$, add either $\text{Wall}_{x,y}$ or $\neg\text{Wall}_{x,y}$ to KB, if entailed
    - For each $x,y$, add either $\text{At}_{x,y,t}$ or $\neg\text{At}_{x,y,t}$ to KB, if entailed
    - Choose an action
Resolution (briefly)

- Every CNF clause can be written as
  - Conjunction of symbols ⇒ disjunction of symbols
    - \( A \lor B \lor \neg C \lor \neg D = C \land D \Rightarrow A \lor B \)
  - The resolution inference rule takes two such clauses and infers a new one by **resolving** complementary symbols:
    - Example: \( A \land B \land C \Rightarrow U \lor V \)
      - \( D \land E \land U \Rightarrow X \lor Y \)
      - \( A \land B \land C \land D \land E \Rightarrow V \lor X \lor Y \)

- Sentence unsatisfiable iff repeated resolution produces \( () \Rightarrow () \)
- Resolution is complete for propositional logic, but exp-time
Summary

- Logical inference computes entailment relations among sentences
- Theorem provers apply inference rules to sentences
  - Forward chaining applies modus ponens with definite clauses; linear time
  - Resolution is complete for PL but exponential time in the worst case
- SAT solvers based on DPLL provide incredibly efficient inference
- Logical agents can do localization, mapping, SLAM, planning (and many other things) just using one generic inference algorithm on one knowledge base
CS 188: Artificial Intelligence

First-Order Logic

Slides mostly from Stuart Russell
University of California, Berkeley
Spectrum of representations

(a) Atomic
Search, game-playing

(b) Factored
Planning, propositional logic, Bayes nets

(b) Structured
First-order logic, databases, logic programs, probabilistic programs
Expressive power

- **Rules of chess:**
  - 100,000 pages in propositional logic
  - 1 page in first-order logic

- **Rules of Pacman:**
  - $\forall t \text{Alive}(t) \Leftrightarrow$
    
    $$[\text{Alive}(t-1) \land \neg \exists g,x,y [\text{Ghost}(g) \land \text{At}(\text{Pacman},x,y,t-1) \land \text{At}(g,x,y,t-1)]]$$
Possible worlds

- A possible world of five objects:
  - “left leg” unary function (arity is # arguments)
  - “on head” binary relation
  - “brother” binary relation
  - “person” unary relation
  - “king” unary relation
  - “crown” unary relation
  - “John” constant (0-ary function)
  - “Richard” constant (0-ary function)

- If a function/relation/constant is mentioned
- World must have object(s) plus definitions of those functions/relations/constants
A possible world for FOL consists of:

- A non-empty set of objects
- For each k-ary predicate in the language, a set of k-tuples of objects (i.e., the set of tuples of objects that satisfy the predicate in this world)
- For each k-ary function in the language, a mapping from k-tuples of objects to objects
- For each constant symbol, a particular object (can think of constants as 0-ary functions)
Possible worlds

- A possible world for FOL consists of:
  - A non-empty set of objects
  - For each k-ary predicate in the language, a set of k-tuples of objects (i.e., the set of tuples of objects that satisfy the predicate in this world)
  - For each k-ary function in the language, a mapping from k-tuples of objects to objects
  - For each constant symbol, a particular object (can think of constants as 0-ary functions)

Knows(A, BFF(B))
Possible worlds

- A possible world for FOL consists of:
  - A non-empty set of objects
  - For each k-ary predicate in the language, a set of k-tuples of objects (i.e., the set of tuples of objects that satisfy the predicate in this world)
  - For each k-ary function in the language, a mapping from k-tuples of objects to objects
  - For each constant symbol, a particular object (can think of constants as 0-ary functions)

*How many possible worlds?*
Syntax and semantics: Terms

- A term is something that refers to an object; it can be
  - A constant symbol, e.g., A, B, EvilKingJohn
    - The possible world fixes these referents
  - A function symbol with terms as arguments, e.g., BFF(EvilKingJohn)
    - The possible world specifies the value of the function, given the referents of the terms
      - BFF(EvilKingJohn) -> BFF(2) -> 3
  - A logical variable, e.g., x
    - (more later)
Syntax and semantics: Atomic sentences

- An atomic sentence is an elementary proposition (cf symbols in PL)
  - A predicate symbol with terms as arguments, e.g., \( \text{Knows}(A, \text{BFF}(B)) \)
    - \( \text{Knows}(A, \text{BFF}(B)) \rightarrow \text{Knows}(1, \text{BFF}(2)) \rightarrow \text{Knows}(1, 3) \rightarrow F \)
    - True iff the objects referred to by the terms are in the relation referred to by the predicate
  - An equality between terms, e.g., \( \text{BFF}(\text{BFF}(\text{BFF}(B))) = B \)
    - True iff the terms refer to the same objects
    - \( \text{BFF}(\text{BFF}(\text{BFF}(B))) = B \rightarrow \text{BFF}(\text{BFF}(\text{BFF}(2))) = 2 \rightarrow \text{BFF}(\text{BFF}(3)) = 2 \rightarrow \text{BFF}(1) = 2 \rightarrow 2 = 2 \rightarrow T \)
Syntax and semantics: Complex sentences

- Sentences with logical connectives
  \( \neg \alpha, \alpha \land \beta, \alpha \lor \beta, \alpha \Rightarrow \beta, \alpha \Leftrightarrow \beta \)

- Sentences with universal or existential quantifiers, e.g.,
  \( \forall x \text{ Knows}(x, \text{BFF}(x)) \)
  - True in world \( w \) iff true in all extensions of \( w \) where \( x \) refers to an object in \( w \)
    - \( x \rightarrow 1: \text{Knows}(1, \text{BFF}(1)) \rightarrow \text{Knows}(1, 2) \rightarrow T \)
    - \( x \rightarrow 2: \text{Knows}(2, \text{BFF}(2)) \rightarrow \text{Knows}(2, 3) \rightarrow T \)
    - \( x \rightarrow 3: \text{Knows}(3, \text{BFF}(3)) \rightarrow \text{Knows}(3, 1) \rightarrow F \)
Syntax and semantics: Complex sentences

- Sentences with logical connectives
  \( \neg \alpha, \alpha \land \beta, \alpha \lor \beta, \alpha \implies \beta, \alpha \iff \beta \)
- Sentences with universal or existential quantifiers, e.g.,
  \( \exists x \text{ Knows}(x, \text{BFF}(x)) \)
    - True in world \( w \) iff true in some extension of \( w \) where \( x \) refers to an object in \( w \)
      - \( x \to 1: \text{Knows}(1, \text{BFF}(1)) \to \text{Knows}(1,2) \to T \)
      - \( x \to 2: \text{Knows}(2, \text{BFF}(2)) \to \text{Knows}(2,3) \to T \)
      - \( x \to 3: \text{Knows}(3, \text{BFF}(3)) \to \text{Knows}(3,1) \to F \)
Fun with sentences

- Everyone knows President Obama
  - $\forall n \text{ Person}(n) \Rightarrow \text{Knows}(n,\text{Obama})$

- There is someone that nobody else knows
  - $\exists s \text{ Person}(s) \land \forall n (\text{Person}(n) \land \neg(n = s)) \Rightarrow \neg\text{Knows}(n,s)$

- Everyone knows someone
  - $\forall x \text{ Person}(x) \Rightarrow \exists y \text{ Person}(y) \land \text{Knows}(x,y)$
  - $\forall x (\text{Person}(x) \Rightarrow \exists y (\text{Person}(y) \land \text{Knows}(x,y)))$
More fun with sentences

- Any two people of the same nationality speak a common language
  - `Nationality(x, n) – x has nationality n`
  - `Speaks(x, l) – x speaks language l`
  - `∀x, y [∃ n Nationality(x, n) ∧ Nationality(y, n)] ⊨ (∃ l Speak(x, l) ∧ Speak(y, l))`
  - `∀t (Alive(t) ↔ [Alive(t-1) ∧ ¬∃ g, x, y [Ghost(g) ∧ At(Pacman, x, y, t-1) ∧ At(g, x, y, t-1)]]))`
Conciseness of first order logic

- Pacman can’t be in two places at once
  - FOL: $\forall x_1, y_1, x_2, y_2, t \, (\text{At}(x_1, y_1, t) \land \text{At}(x_2, y_2, t)) \Rightarrow (x_1 = x_2 \land y_1 = y_2)$
  - PL: $\neg (\text{At}_1,1_0 \land \text{At}_1,2_0) \land \neg (\text{At}_1,1_0 \land \text{At}_1,3_0) \land \neg (\text{At}_1,1_0 \land \text{At}_2,1_0) \land \neg (\text{At}_1,1_0 \land \text{At}_2,2_0) \land \neg (\text{At}_1,1_0 \land \text{At}_2,3_0) \land \neg (\text{At}_1,1_0 \land \text{At}_3,1_0) \land \neg (\text{At}_1,1_0 \land \text{At}_3,2_0) \land \neg (\text{At}_1,1_0 \land \text{At}_3,3_0) \land ...$
  - And that’s just if he’s in the bottom left at the first timestep
Inference in FOL

- Entailment is defined exactly as for propositional logic:
  - \( \alpha \models \beta \) ("\( \alpha \) entails \( \beta \)"") iff in every world where \( \alpha \) is true, \( \beta \) is also true
  - E.g., \( \forall x \text{Knows}(x,\text{Obama}) \) entails \( \exists y \forall x \text{Knows}(x,y) \)

- In FOL, we can go beyond just answering “yes” or “no”; given an existentially quantified query, return a substitution (or binding) for the variable(s) such that the resulting sentence is entailed:
  - KB = \( \forall x \text{Knows}(x,\text{Obama}) \)
  - Query = \( \exists y \forall x \text{Knows}(x,y) \)
  - Answer = Yes, \( \sigma = \{y/\text{Obama}\} \)
  - Notation: \( \alpha \sigma \) means applying substitution \( \sigma \) to sentence \( \alpha \)
    - E.g., if \( \alpha = \forall x \text{Knows}(x,y) \) and \( \sigma = \{y/\text{Obama}\} \), then \( \alpha \sigma = \forall x \text{Knows}(x,\text{Obama}) \)
Inference in FOL: Propositionalization

- Convert \((\text{KB} \land \neg \alpha)\) to PL, use a PL SAT solver to check (un)satisfiability
  - Trick: replace variables with ground terms, convert atomic sentences to symbols
    - \(\exists x \text{ Knows}(x,\text{Obama})\)
      - \(\text{Knows}(X_1,\text{Obama})\)
      - \(\text{Knows}_{X1}\_\text{Obama}\)
    - \(\forall x \text{ Knows}(x,\text{Obama})\) and Democrat(Feinstein)
      - \(\text{Knows}(\text{Obama},\text{Obama})\) and Knows(Feinstein,Obama) and Democrat(Feinstein)
      - \(\text{Knows}_{\text{Obama}\_\text{Obama}} \land \text{Knows}_{\text{Feinstein}\_\text{Obama}} \land \text{Democrat}_{\text{Feinstein}}\)
    - \(\forall x \text{ Knows}(\text{Mother}(x),x)\)
      - \(\text{Knows}(\text{Mother}(\text{Obama}),\text{Obama})\) and Knows(Mother(Mother(Obama)),Mother(Obama)) .......
  - Real trick: for \(k = 1\) to infinity:
    - Get a set of terms: constants, functions of constants, funcs of funcs of constants, ... up to depth \(k\)
    - Propositionalize as if those are all the terms that exist
    - If a contradiction is found, halt; otherwise, continue
  - If FOL sentence is unsatisfiable, will find a contradiction for some finite \(k\) (Herbrand); if not, may continue for ever; semidecidable
Inference in FOL: Lifted inference

- Apply inference rules directly to first-order sentences, e.g.,
  - KB = Person(Socrates), \( \forall x \) Person(x) \( \Rightarrow \) Mortal(x)
  - conclude Mortal(Socrates)
- The general rule is a version of Modus Ponens:
  - Given \( \alpha \Rightarrow \beta \) and \( \alpha' \), where \( \alpha \sigma = \alpha' \sigma \) for some substitution \( \sigma \), conclude \( \beta \sigma \)
    - \( \sigma \) is \( \{x/\text{Socrates}\} \)
  - Given \( \forall x \) Knows(x,Obama) and \( \forall y, z \) Knows(y,z) \( \Rightarrow \) Likes(y,z)
    - \( \sigma \) is \( \{y/x, z/\text{Obama}\} \), conclude Likes(x,Obama)
- Examples: Prolog (backward chaining), Datalog (forward chaining), production rule systems (forward chaining), resolution theorem provers
FOL is a very expressive formal language

Many domains of common-sense and technical knowledge can be written in FOL (see AIMA Ch. 10)

- circuits, software, planning, law, taxes, network and security protocols, product descriptions, ecommerce transactions, geographical information systems, Google Knowledge Graph, Semantic Web, etc.

Inference is semidecidable in general; many problems are efficiently solvable in practice

Inference technology for logic programming is especially efficient (see AIMA Ch. 9)