Logical Agents
Outline

• Knowledge-based agents
• Logic in general - models and entailment
• Propositional (Boolean) logic
• Equivalence, validity, satisfiability
• Inference rules and theorem proving
  – forward chaining
  – backward chaining
  – resolution
Knowledge bases

- Knowledge base = set of *sentences* in a *formal* language
- **Declarative** approach to building an agent (or other system):
  - *Tell* it what it needs to know
- Then it can *Ask* itself what to do - answers should follow from the KB
- Agents can be viewed at the *knowledge level*
  - i.e., what they know, regardless of how implemented
- Or at the *implementation level*
  - i.e., data structures in KB and algorithms that manipulate them
A simple knowledge-based agent

The agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
        t, a counter, initially 0, indicating time
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```
Logic in general

- **Logics** are formal languages for representing information such that conclusions can be drawn.
- **Syntax** defines the sentences in the language.
- **Semantics** define the "meaning" of sentences; i.e., define truth of a sentence in a world.
- E.g., the language of arithmetic:
  - $x+2 \geq y$ is a sentence; $x2+y > \{\}$ is not a sentence.
  - $x+2 \geq y$ is true iff the number $x+2$ is no less than the number $y$.
  - $x+2 \geq y$ is true in a world where $x = 7$, $y = 1$.
  - $x+2 \geq y$ is false in a world where $x = 0$, $y = 6$. 
Entailment

- **Entailment** means that one thing **follows from** another:
- $\text{KB} \models \alpha$
- Knowledge base $\text{KB}$ entails sentence $\alpha$ if and only if $\alpha$ is true in all worlds where $\text{KB}$ is true
  
  - E.g., the KB containing “the Giants won” and “the Reds won” entails “Either the Giants won or the Reds won”
  
  - E.g., $x+y = 4$ entails $4 = x+y$
  
  - Entailment is a relationship between sentences (i.e., syntax) that is based on **semantics**
Models

• Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated.

• We say $m$ is a model of a sentence $\alpha$ if $\alpha$ is true in $m$.

• $M(\alpha)$ is the set of all models of $\alpha$.

• Then $KB \models \alpha$ iff $M(KB) \subseteq M(\alpha)$.
  – E.g. $KB = \text{Giants won and Reds won}$ $\alpha = \text{Giants won}$.
Wumpus World description

• **Performance measure**
  - gold +1000, death -1000
  - -1 per step, -10 for using the arrow

• **Environment**
  - Squares adjacent to wumpus are smelly
  - Squares adjacent to pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square

**Sensors:** Stench, Breeze, Glitter, Bump, Scream

• **Actuators:** Left turn, Right turn, Forward, Grab, Release, Shoot
Entailment in the wumpus world

Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

Consider possible models for $KB$ assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models
Wumpus models

- $KB = \text{wumpus-world rules + observations}$
Wumpus models

- $KB = \text{wumpus-world rules} + \text{observations}
- \alpha_1 = \text{"[1,2] is safe"}$, $KB \models \alpha_1$, proved by model checking
Wumpus models

- $KB = \text{wumpus-world rules} + \text{observations}$
- $\alpha_2 = "[2,2] \text{ is safe}"$, $KB \not\models \alpha_2$
Inference

- $KB \vdash_{i} \alpha = \text{sentence } \alpha \text{ can be derived from } KB \text{ by procedure } i$
- **Soundness:** $i$ is sound if whenever $KB \vdash_{i} \alpha$, it is also true that $KB \models \alpha$
- **Completeness:** $i$ is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_{i} \alpha$
- **Preview:** we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.
- That is, the procedure will answer any question whose answer follows from what is known by the $KB$. 
Propositional logic: Syntax

• Propositional logic is the simplest logic – illustrates basic ideas

• The proposition symbols $P_1$, $P_2$ etc are sentences
  – If $S$ is a sentence, $\neg S$ is a sentence (negation)
  – If $S_1$ and $S_2$ are sentences, $S_1 \land S_2$ is a sentence (conjunction)
  – If $S_1$ and $S_2$ are sentences, $S_1 \lor S_2$ is a sentence (disjunction)
  – If $S_1$ and $S_2$ are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  – If $S_1$ and $S_2$ are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)
Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g. \( P_{1,2} \quad P_{2,2} \quad P_{3,1} \)
false \quad true \quad false

With these symbols, 8 possible models, can be enumerated automatically.

Rules for evaluating truth with respect to a model \( m \):

\[
\neg S \quad \text{is true iff} \quad S \text{ is false}
\]

\[
S_1 \land S_2 \quad \text{is true iff} \quad S_1 \text{ is true and } S_2 \text{ is true}
\]

\[
S_1 \lor S_2 \quad \text{is true iff} \quad S_1 \text{ is true or } S_2 \text{ is true}
\]

\[
S_1 \Rightarrow S_2 \quad \text{is true iff} \quad S_1 \text{ is false or } S_2 \text{ is true}
\]

i.e., \( S_1 \iff S_2 \) is false iff \( S_1 \text{ is true and } S_2 \text{ is false} \)

\[
S_1 \iff S_2 \quad \text{is true iff} \quad S_1 \Rightarrow S_2 \text{ is true and } S_2 \Rightarrow S_1 \text{ is true}
\]

Simple recursive process evaluates an arbitrary sentence, e.g.,

\[
\neg P_{1,2} \land (P_{2,2} \lor P_{3,1}) = \text{true} \land (\text{true} \lor \text{false}) = \text{true} \land \text{true} = \text{true}
\]
Truth tables for connectives

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$\neg P$</th>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \Rightarrow Q$</th>
<th>$P \leftrightarrow Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

\[
\neg P_{1,1} \\
\neg B_{1,1} \\
B_{2,1}
\]

• "Pits cause breezes in adjacent squares"

\[
B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \\
B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})
\]
Truth tables for inference

<table>
<thead>
<tr>
<th>B_{1,1}</th>
<th>B_{2,1}</th>
<th>P_{1,1}</th>
<th>P_{1,2}</th>
<th>P_{2,1}</th>
<th>P_{2,2}</th>
<th>P_{3,1}</th>
<th>KB</th>
<th>α_{1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Inference by enumeration

• Depth-first enumeration of all models is sound and complete

```plaintext
function TT-ENTAILS?(KB, α) returns true or false
    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [ ])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
    if EMPTY?(symbols) then
        if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
        else return true
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
             TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

• For $n$ symbols, time complexity is $O(2^n)$, space complexity is $O(n)$
Logical equivalence

• Two sentences are **logically equivalent** iff true in same models: \( \alpha \equiv \beta \iff \alpha \models \beta \text{ and } \beta \models \alpha \)

•

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg \alpha) & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \rightarrow \beta) & \equiv (\neg \beta \rightarrow \neg \alpha) \quad \text{contraposition} \\
(\alpha \rightarrow \beta) & \equiv (\neg \alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \leftrightarrow \beta) & \equiv ((\alpha \rightarrow \beta) \land (\beta \rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg \alpha \lor \neg \beta) \quad \text{de Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg \alpha \land \neg \beta) \quad \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]

*inference by rules vs model checking*
Validity and satisfiability

A sentence is **valid** if it is true in **all** models, e.g., True, A ∨¬A, A ⇔ A, (A ∧ (A ⇒ B)) ⇒ B

Validity is connected to inference via the **Deduction Theorem**:

\[ KB \models \alpha \text{ if and only if } (KB \Rightarrow \alpha) \text{ is valid} \]

A sentence is **satisfiable** if it is true in **some** model e.g., A ∨ B, C

A sentence is **unsatisfiable** if it is true in **no** models e.g., A ∧¬A

Satisfiability is connected to inference via the following:

\[ KB \models \alpha \text{ if and only if } (KB \land \neg\alpha) \text{ is unsatisfiable} \]
Proof methods

- Proof methods divide into (roughly) two kinds:
  
  - **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 algorithm
    - Typically require transformation of sentences into a normal form
  
  - **Model checking**
    - truth table enumeration (always exponential in $n$)
    - improved backtracking, e.g., Davis--Putnam-Logemann-Loveland (DPLL)
    - heuristic search in model space (sound but incomplete)
      - e.g., min-conflicts-like hill-climbing algorithms
Resolution

Conjunctive Normal Form (CNF)

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):

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

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

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
Resolution

Soundness of resolution inference rule:

\[ \neg (l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow l_i \]

\[ \neg m_j \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]

\[ \neg (l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]
Conversion to CNF

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

1. Eliminate \( \iff \), replacing \( \alpha \iff \beta \) with \((\alpha \implies \beta) \land (\beta \implies \alpha)\).
\[
(B_{1,1} \implies (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \implies B_{1,1})
\]

2. Eliminate \( \implies \), replacing \( \alpha \implies \beta \) 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 (\( \land \) over \( \lor \)) 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, i.e., show $KB \land \neg \alpha$ unsatisfiable

```plaintext
function PL-Resolution(KB, \alpha) returns true or false

    clauses \leftarrow the set of clauses in the CNF representation of $KB \land \neg \alpha$
    new \leftarrow \{ \}
    loop do
        for each $C_i, C_j$ in clauses do
            resolvents \leftarrow PL-Resolve($C_i, C_j$)
            if resolvents contains the empty clause then return true
            new \leftarrow new \cup resolvents
        if new \subseteq clauses then return false
        clauses \leftarrow clauses \cup new
```

*empty=contradiction*  
*failure of resolve*
Resolution example

- $KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \alpha = \neg P_{1,2}$
Forward and backward chaining

- **Horn Form** (restricted)
  - KB = *conjunction of Horn clauses*
  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) ⇒ symbol
  - E.g., \( C \land (B \Rightarrow A) \land (C \land D \Rightarrow B) \)
- **Modus Ponens** (for Horn Form): complete for Horn KBs
  
  \[
  \alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta
  \]
  \[\beta\]
  
  Can be used with **forward chaining** or **backward chaining**.
  - These algorithms are very natural and run in **linear time**
Forward chaining

- Idea: 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 algorithm

function PL-FC-ENTAILS?(KB, q) returns true or false

local variables: count, a table, indexed by clause, initially the number of premises
inferred, a table, indexed by symbol, each entry initially false
agenda, a list of symbols, initially the symbols known to be true

while agenda is not empty do
    p ← POP(agenda)
    unless inferred[p] do
        inferred[p] ← true
        for each Horn clause c in whose premise p appears do
            decrement count[c]
            if count[c] = 0 then do
                if HEAD[c] = q then return true
                PUSH(HEAD[c], agenda)
        return false

• Forward chaining is sound and complete for Horn KB
Proof of completeness

• FC derives every atomic sentence that is entailed by $KB$

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final state as a model $m$, assigning true/false to symbols
3. Every clause in the original $KB$ is true in $m$
   \[ a_1 \land \ldots \land a_k \Rightarrow b \]
4. Hence $m$ is a model of $KB$
5. If $KB \vDash q$, $q$ is true in every model of $KB$, including $m$
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
Forward vs. backward chaining

- FC is data-driven, 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
Efficient propositional inference

Two families of efficient algorithms for propositional inference:

Complete backtracking search algorithms
  - DPLL algorithm (Davis, Putnam, Logemann, Loveland)

Incomplete local search algorithms
  - WalkSAT algorithm

*not restricted to Horn clauses, for general SAT
The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. Early termination
   A clause is true if any literal is true.
   A sentence is false if any clause is false.

2. Pure symbol heuristic
   Pure symbol: always appears with the same "sign" in all clauses.
   e.g., In the three clauses (A ∨ ¬B), (¬B ∨ ¬C), (C ∨ A), A and B are pure, C is impure.
   Make a pure symbol literal true.

3. Unit clause heuristic
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.
The DPLL algorithm

backtracking search algorithm

function DPLL-SATISFIABLE?(s) returns true or false
inputs: s, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← 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 ← FIND-PURE-SYMBOL(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])
P, value ← FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])
P ← FIRST(symbols); rest ← REST(symbols)
return DPLL(clauses, rest, [P = true|model]) or
DPLL(clauses, rest, [P = false|model])

*find assignment for making s true
*can use other heuristics that learned in CSP section such as variable/value ordering, component analysis, etc
*choice for backtracking
The WalkSAT algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness
The WalkSAT algorithm

function WalkSAT(clauses, p, max-flips) returns a satisfying model or failure
inputs: clauses, a set of clauses in propositional logic
        p, the probability of choosing to do a “random walk” move
        max-flips, number of flips allowed before giving up

model ← a random assignment of true/false to the symbols in clauses
for i = 1 to max-flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    with probability p flip the value in model of a randomly selected symbol
    from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
return failure
Hard satisfiability problems

• Consider random 3-CNF sentences. e.g.,

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

\[m = \text{number of clauses}\]
\[n = \text{number of symbols}\]

– Hard problems seem to cluster near \(m/n = 4.3\) (critical point)
Hard satisfiability problems
Hard satisfiability problems

- Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$
Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

\[-P_{1,1}\]
\[-W_{1,1}\]

\[B_{x,y} \iff (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y})\]
\[S_{x,y} \iff (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y})\]

\[W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,4}\]
\[\neg W_{1,1} \lor \neg W_{1,2}\]
\[\neg W_{1,1} \lor \neg W_{1,3}\]

\[\ldots\]

\[\Rightarrow 64 \text{ distinct proposition symbols, 155 sentences}\]
function PL-WUMPUS-AGENT( percept) returns an action
  inputs: percept, a list, [stench, breeze, glitter]
static: KB, initially containing the “physics” of the wumpus world
  x, y, orientation, the agent’s position (init. [1,1]) and orient. (init. right)
  visited, an array indicating which squares have been visited, initially false
  action, the agent’s most recent action, initially null
  plan, an action sequence, initially empty
update x, y, orientation, visited based on action
if stench then TELL(KB, S_{x,y}) else TELL(KB, \neg S_{x,y})
if breeze then TELL(KB, B_{x,y}) else TELL(KB, \neg B_{x,y})
if glitter then action \leftarrow grab
else if plan is nonempty then action \leftarrow POP(plan)
else if for some fringe square [i,j], ASK(KB, (\neg P_{i,j} \land \neg W_{i,j})) is true or
  for some fringe square [i,j], ASK(KB, (P_{i,j} \lor W_{i,j})) is false then do
    plan \leftarrow A*-GRAPH-SEARCH(ROUTE-PB([x,y], orientation, [i,j], visited))
    action \leftarrow POP(plan)
else action \leftarrow a randomly chosen move
return action
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

• Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

• Resolution is complete for propositional logic
  Forward, backward chaining are linear-time, complete for Horn clauses

• Propositional logic lacks expressive power
First-Order Logic
Outline

• Why FOL?
• Syntax and semantics of FOL
• Using FOL
• Wumpus world in FOL
• Knowledge engineering in FOL
Pros and cons of propositional logic

😊 Propositional logic is **declarative**

😊 Propositional logic allows partial/disjunctive/negated information
  - (unlike most data structures and databases)

😊 Propositional logic is **compositional**:
  - meaning of $B_{1,1} \land P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$

😊 Meaning in propositional logic is **context-independent**
  - (unlike natural language, where meaning depends on context)

😢 Propositional logic has very limited expressive power
  - (unlike natural language)
  - E.g., cannot say "pits cause breezes in adjacent squares"
    - except by writing one sentence for each square
First-order logic

- Whereas propositional logic assumes the world contains **facts**, first-order logic (like natural language) assumes the world contains
  - **Objects**: people, houses, numbers, colors, baseball games, wars, …
  - **Relations**: red, round, prime, brother of, bigger than, part of, comes between, …
  - **Functions**: father of, best friend, one more than, plus, …
Syntax of FOL: Basic elements

- **Constants**: KingJohn, 2, NUS,...
- **Predicates**: Brother, >,...
- **Functions**: Sqrt, LeftLegOf,...
- **Variables**: x, y, a, b,...
- **Connectives**: ¬, ⇒, ∧, ∨, ⇔
- **Equality**: =
- **Quantifiers**: ∀, ∃
Atomic sentences

Atomic sentence = \textit{predicate (term}_{1},...,\textit{term}_{n})
\text{ or } \textit{term}_{1} = \textit{term}_{2}

Term = \textit{function (term}_{1},...,\textit{term}_{n})
\text{ or constant or variable}

• E.g., \textit{Brother(KingJohn,RichardTheLionheart)}
> (\textit{Length(LeftLegOf(Richard))},
\textit{Length(LeftLegOf(KingJohn))})
Complex sentences

- Complex sentences are made from atomic sentences using connectives:
  - $\neg S$, $S_1 \land S_2$, $S_1 \lor S_2$, $S_1 \Rightarrow S_2$, $S_1 \Leftrightarrow S_2$,

E.g. $\text{Sibling}(\text{KingJohn}, \text{Richard}) \Rightarrow \text{Sibling}(\text{Richard}, \text{KingJohn})$

$$\Rightarrow (1,2) \lor \leq (1,2)$$

$$\Rightarrow (1,2) \land \neg \Rightarrow (1,2)$$
Truth in first-order logic

- Sentences are true with respect to a model and an interpretation.

- Model contains objects (domain elements) and relations among them.

- Interpretation specifies referents for:
  - Constant symbols $\rightarrow$ objects
  - Predicate symbols $\rightarrow$ relations
  - Function symbols $\rightarrow$ functional relations

- An atomic sentence $\text{predicate}(\text{term}_1,...,\text{term}_n)$ is true iff the objects referred to by $\text{term}_1,...,\text{term}_n$ are in the relation referred to by $\text{predicate}$.
Models for FOL: Example
Universal quantification

• \( \forall \langle \text{variables} \rangle \langle \text{sentence} \rangle \)

Everyone at NUS is smart:
\( \forall x \text{At}(x, \text{NUS}) \Rightarrow \text{Smart}(x) \)

• \( \forall x \ P \) is true in a model \( m \) iff \( P \) is true with \( x \) being each possible object in the model

• Roughly speaking, equivalent to the conjunction of instantiations of \( P \)

\( \text{At}(\text{KingJohn}, \text{NUS}) \Rightarrow \text{Smart}(\text{KingJohn}) \)
\( \land \ \text{At}(\text{Richard}, \text{NUS}) \Rightarrow \text{Smart}(\text{Richard}) \)
\( \land \ \text{At}(\text{NUS}, \text{NUS}) \Rightarrow \text{Smart}(\text{NUS}) \)
\( \land \ \ldots \)
A common mistake to avoid

• Typically, $\Rightarrow$ is the main connective with $\forall$

• Common mistake: using $\land$ as the main connective with $\forall$:
  
  $\forall x \text{At}(x, \text{NUS}) \land \text{Smart}(x)$

  means “Everyone is at NUS and everyone is smart”
Existential quantification

- $\exists \langle \text{variables} \rangle \langle \text{sentence} \rangle$

- Someone at NUS is smart:
  - $\exists x \ \text{At}(x, \text{NUS}) \land \text{Smart}(x)$

- $\exists x \ P$ is true in a model $m$ iff $P$ is true with $x$ being some possible object in the model

- Roughly speaking, equivalent to the disjunction of instantiations of $P$
  - $\\text{At}(\text{KingJohn, NUS}) \land \text{Smart}(\text{KingJohn})$
  - $\lor \ \text{At}(\text{Richard, NUS}) \land \text{Smart}(\text{Richard})$
  - $\lor \ \text{At}(\text{NUS, NUS}) \land \text{Smart}(\text{NUS})$
  - $\lor \ldots$
Another common mistake to avoid

- Typically, $\wedge$ is the main connective with $\exists$

- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

  $$\exists x \text{ At}(x, \text{NUS}) \Rightarrow \text{Smart}(x)$$

is true if there is anyone who is not at NUS!
Properties of quantifiers

- $\forall x \ \forall y$ is the same as $\forall y \ \forall x$
- $\exists x \ \exists y$ is the same as $\exists y \ \exists x$
- $\exists x \ \forall y$ is not the same as $\forall y \ \exists x$
- $\exists x \ \forall y \ \text{Loves}(x,y)$
  - “There is a person who loves everyone in the world”
- $\forall y \ \exists x \ \text{Loves}(x,y)$
  - “Everyone in the world is loved by at least one person”

- **Quantifier duality:** each can be expressed using the other

- $\forall x \ \text{Likes}(x,\text{IceCream}) \quad \neg \exists x \ \neg \text{Likes}(x,\text{IceCream})$
- $\exists x \ \text{Likes}(x,\text{Broccoli}) \quad \neg \forall x \ \neg \text{Likes}(x,\text{Broccoli})$
Equality

• $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object.

• E.g., definition of $Sibling$ in terms of $Parent$:

$$\forall x,y \: Sibling(x,y) \iff \neg (x = y) \land \exists m,f \: \neg (m = f) \land Parent(m,x) \land Parent(f,x) \land Parent(m,y) \land Parent(f,y)$$
Using FOL

The kinship domain:

- Brothers are siblings
  \[ \forall x,y \, \text{Brother}(x,y) \iff \text{Sibling}(x,y) \]

- One's mother is one's female parent
  \[ \forall m,c \, \text{Mother}(c) = m \iff (\text{Female}(m) \land \text{Parent}(m,c)) \]

- “Sibling” is symmetric
  \[ \forall x,y \, \text{Sibling}(x,y) \iff \text{Sibling}(y,x) \]
Using FOL

The set domain:

• \( \forall s \ Set(s) \Leftrightarrow (s = \{\} \lor (\exists x, s_2 \ Set(s_2) \land s = \{x|s_2\}) \)
• \( \neg \exists x, s \ \{x|s\} = \{\} \)
• \( \forall x, s \ \ x \in s \Leftrightarrow s = \{x|s\} \)
• \( \forall x, s \ x \in s \Leftrightarrow [ \exists y, s_2 \ (s = \{y|s_2\} \land (x = y \lor x \in s_2))] \)
• \( \forall s_1, s_2 \ s_1 \subseteq s_2 \Leftrightarrow (\forall x \ x \in s_1 \Rightarrow x \in s_2) \)
• \( \forall s_1, s_2 \ (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \land s_2 \subseteq s_1) \)
• \( \forall x, s_1, s_2 \ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \land x \in s_2) \)
• \( \forall x, s_1, s_2 \ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \lor x \in s_2) \)

*The only sets are the empty set and the set to which something is added

*No element is added to the empty set

*Adding an already existing element to a set has no effect:
Interacting with FOL KBs

• Suppose a wumpus-world agent is using an FOL KB and perceives a smell and a breeze (but no glitter) at t=5:

\[
\text{Tell}(\text{KB}, \text{Percept}([\text{Smell}, \text{Breeze}, \text{None}], 5))
\]

\[
\text{Ask}(\text{KB}, \exists a \text{ BestAction}(a,5))
\]

• I.e., does the KB entail some best action at t=5?

• Answer: Yes, \{a/Shoot\} ← substitution (binding list)

• Given a sentence \(S\) and a substitution \(\sigma\),
• \(S\sigma\) denotes the result of plugging \(\sigma\) into \(S\); e.g.,
  \[
  S = \text{Smarter}(x,y)
  \sigma = \{x/\text{Hillary}, y/\text{Bill}\}
  S\sigma = \text{Smarter}(\text{Hillary}, \text{Bill})
  \]

• \(\text{Ask}(\text{KB}, S)\) returns some/all \(\sigma\) such that \(\text{KB} \vDash S\sigma\)
Knowledge base for the wumpus world

- **Perception**
  - $\forall t, s, b \text{ Percept}([s, b, \text{Glitter}], t) \Rightarrow \text{Glitter}(t)$

- **Reflex**
  - $\forall t \text{ Glitter}(t) \Rightarrow \text{BestAction(Grab}, t)$
Deducing hidden properties

• \( \forall x,y,a,b \ Adjacent([x,y],[a,b]) \iff [a,b] \in \{[x+1,y], [x-1,y],[x,y+1],[x,y-1]\} \)

Properties of squares:

• \( \forall s,t \ At(Agent,s,t) \land Breeze(t) \implies Breezy(s) \)

Squares are breezy near a pit:

– **Diagnostic** rule---infer cause from effect

\[
\forall s \ Breezy(s) \implies \exists r \ Adjacent(r,s) \land Pit(r)
\]

– **Causal** rule---infer effect from cause

\[
\forall r \ Pit(r) \implies [\forall s \ Adjacent(r,s) \implies Breezy(s)]
\]
Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base
The electronic circuits domain

One-bit full adder

1bit A
1bit B
Carry_in

A + B
xor
and
or
carry_out
The electronic circuits domain

1. Identify the task
   - Does the circuit actually add properly? (circuit verification)

2. Assemble the relevant knowledge
   - Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   - Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary
   - Alternatives:
     Type($X_1$) = XOR

     Type($X_1$, XOR)

     XOR($X_1$)
The electronic circuits domain

4. Encode general knowledge of the domain
   - $\forall t_1, t_2$ Connected$(t_1, t_2) \Rightarrow \text{Signal}(t_1) = \text{Signal}(t_2)$
   - $\forall t$ Signal$(t) = 1 \lor \text{Signal}(t) = 0$
   - $1 \neq 0$
   - $\forall t_1, t_2$ Connected$(t_1, t_2) \Rightarrow \text{Connected}(t_2, t_1)$
   - $\forall g$ Type$(g) = \text{OR} \Rightarrow \text{Signal(Out}(1,g)) = 1 \iff \exists n \\text{Signal(In}(n,g)) = 1$
   - $\forall g$ Type$(g) = \text{AND} \Rightarrow \text{Signal(Out}(1,g)) = 0 \iff \exists n \\text{Signal(In}(n,g)) = 0$
   - $\forall g$ Type$(g) = \text{XOR} \Rightarrow \text{Signal(Out}(1,g)) = 1 \iff \text{Signal(In}(1,g)) \neq \text{Signal(In}(2,g))$
   - $\forall g$ Type$(g) = \text{NOT} \Rightarrow \text{Signal(Out}(1,g)) \neq \text{Signal(In}(1,g))$
The electronic circuits domain

5. Encode the specific problem instance

Type($X_1$) = XOR
Type($X_2$) = XOR
Type($A_1$) = AND
Type($A_2$) = AND
Type($O_1$) = OR

Connected($Out(1,X_1),In(1,X_2)$)    Connected($In(1,C_1),In(1,X_1)$)
Connected($Out(1,X_1),In(2,A_2)$)    Connected($In(1,C_1),In(1,A_1)$)
Connected($Out(1,A_2),In(1,O_1)$)    Connected($In(2,C_1),In(2,X_1)$)
Connected($Out(1,A_1),In(2,O_1)$)    Connected($In(2,C_1),In(2,A_1)$)
Connected($Out(1,X_2),Out(1,C_1)$)    Connected($In(3,C_1),In(2,X_2)$)
Connected($Out(1,O_1),Out(2,C_1)$)    Connected($In(3,C_1),In(1,A_2)$)
The electronic circuits domain

6. Pose queries to the inference procedure
   What are the possible sets of values of all the terminals for the adder circuit?
   \[ \exists i_1, i_2, i_3, o_1, o_2 \quad \text{Signal(In}(1, C_{1})) = i_1 \wedge \text{Signal(In}(2, C_{1})) = i_2 \wedge \text{Signal(In}(3, C_{1})) = i_3 \wedge \text{Signal(Out}(1, C_{1})) = o_1 \wedge \text{Signal(Out}(2, C_{1})) = o_2 \]

7. Debug the knowledge base
   May have omitted assertions like \( 1 \neq 0 \)
Summary

• First-order logic:
  – objects and relations are semantic primitives
  – syntax: constants, functions, predicates, equality, quantifiers

• Increased expressive power: sufficient to define wumpus world
Outline

- Ontological engineering
- Categories and objects
- Actions, situations and events
- Mental events and mental objects
- The internet shopping world
- Reasoning systems for categories
- Reasoning with default information
- Truth maintenance systems
Ontological engineering

- How to create more general and flexible representations.
  - Concepts like actions, time, physical object and beliefs
  - Operates on a bigger scale than K.E.

- Define general framework of concepts
  - Upper ontology

- Limitations of logic representation
  - Red, green and yellow tomatoes: exceptions and uncertainty
The upper ontology of the world
Difference with special-purpose ontologies

- A general-purpose ontology should be applicable in more or less any special-purpose domain.
  - Add domain-specific axioms

- In any sufficiently demanding domain different areas of knowledge need to be unified.
  - Reasoning and problem solving could involve several areas simultaneously

- What do we need to express?

  Categories, Measures, Composite objects, Time, Space, Change, Events, Processes, Physical Objects, Substances, Mental Objects, Beliefs
Categories and objects

- KR requires the organisation of objects into categories
  - Interaction at the level of the object
  - Reasoning at the level of categories

- Categories play a role in predictions about objects
  - Based on perceived properties

- Categories can be represented in two ways by FOL
  - Predicates: apple(x)
  - Reification of categories into objects: apples

- Category = set of its members
Category organization

- Relation = inheritance:
  - All instance of food are edible, fruit is a subclass of food and apples is a subclass of fruit then an apple is edible.

- Defines a taxonomy
FOL and categories

- An object is a member of a category
  - \( \text{MemberOf}(BB_{12}, \text{Basketballs}) \)

- A category is a subclass of another category
  - \( \text{SubsetOf}(\text{Basketballs}, \text{Balls}) \)

- All members of a category have some properties
  - \( \forall x \ (\text{MemberOf}(x, \text{Basketballs}) \Rightarrow \text{Round}(x)) \)

- All members of a category can be recognized by some properties
  - \( \forall x \ (\text{Orange}(x) \land \text{Round}(x) \land \text{Diameter}(x)=9.5\text{in} \land \text{MemberOf}(x, \text{Balls}) \Rightarrow \text{MemberOf}(x, \text{BasketBalls})) \)

- A category as a whole has some properties
  - \( \text{MemberOf}(\text{Dogs}, \text{DomesticatedSpecies}) \)
Relations between categories

- Two or more categories are *disjoint* if they have no members in common:
  - **Disjoint(s)**\(\Leftrightarrow (\forall c_1, c_2 \in s \land c_1 \neq c_2 \Rightarrow \text{Intersection}(c_1, c_2) = \emptyset)\)
  - Example: Disjoint({animals, vegetables})

- A set of categories \(s\) constitutes an *exhaustive decomposition* of a category \(c\) if all members of the set \(c\) are covered by categories in \(s\):
  - **E.D.\((s,c)\)**\(\Leftrightarrow (\forall i \in c \Rightarrow \exists c_2 \in s \land i \in c_2)\)
  - Example: ExhaustiveDecomposition({Americans, Canadian, Mexicans}, NorthAmericans).
Relations between categories

- A *partition* is a disjoint exhaustive decomposition:
  - \( \text{Partition}(s,c) \iff \text{Disjoint}(s) \land \text{E.D.}(s,c) \)
  - Example: \( \text{Partition}([\text{Males}, \text{Females}], \text{Persons}) \).

- Is \( ([\text{Americans, Canadian, Mexicans}], \text{North Americans}) \) a partition?

- Categories can be defined by providing necessary and sufficient conditions for membership
  - \( \forall \, x \, \text{Bachelor}(x) \iff \text{Male}(x) \land \text{Adult}(x) \land \text{Unmarried}(x) \)
Natural kinds

- Many categories have no clear-cut definitions (chair, bush, book).
- Tomatoes: sometimes green, red, yellow, black. Mostly round.
- One solution: category Typical(Tomatoes).
  \[ \forall x, x \in \text{Typical(Tomatoes)} \Rightarrow \text{Red}(x) \land \text{Spherical}(x). \]
  - We can write down useful facts about categories without providing exact definitions.
- What about “bachelor”? Quine challenged the utility of the notion of \textit{strict definition}. We might question a statement such as “the Pope is a bachelor”.
Physical composition

- One object may be part of another:
  - `PartOf(Bucharest, Romania)`
  - `PartOf(Romania, EasternEurope)`
  - `PartOf(EasternEurope, Europe)`

- The `PartOf` predicate is transitive (and irreflexive), so we can infer that `PartOf(Bucharest, Europe)`

- More generally:
  - `∀ x PartOf(x, x)`
  - `∀ x, y, z PartOf(x, y) ∧ PartOf(y, z) ⇒ PartOf(x, z)`

- Often characterized by structural relations among parts.
  - **E.g. Biped(a)**

\[
(∃ l_1, l_2, b)(\text{Leg}(l_1) ∧ \text{Leg}(l_2) ∧ \text{Body}(b) ∧ \\
\text{PartOf}(l_1, a) ∧ \text{PartOf}(l_2, a) ∧ \text{PartOf}(b, a) ∧ \\
\text{Attached}(l_1, b) ∧ \text{Attached}(l_2, b) ∧ \\
l_1 \neq l_2 ∧ (∀ l_3)(\text{Leg}(l_3) ⇒ (l_3 = l_1 ∨ l_3 = l_2)))
\]
Measurements

- Objects have height, mass, cost, ....
  Values that we assign to these are measures
- Combine Unit functions with a number: Length(L₁) = Inches(1.5) = Centimeters(3.81).
- Conversion between units:
  \( \forall \ i \text{ Centimeters}(2.54 \times i) = \text{Inches}(i). \)
- Some measures have no scale: Beauty, Difficulty, etc.
  - Most important aspect of measures: is that they are orderable.
  - Don't care about the actual numbers. (An apple can have deliciousness .9 or .1.)
Actions, events and situations

• Reasoning about outcome of actions is central to KB-agent.
• How can we keep track of location in FOL?
  • **Remember the multiple copies in PL.**
• Representing time by situations (states resulting from the execution of actions).
  • **Situation calculus**
Actions, events and situations

- **Situation calculus:**
  - **Actions are logical terms**
  - **Situations are logical terms consisting of**
    - The initial situation $I$
    - All situations resulting from the action on $I$ ($=\text{Result}(a,s)$)
  - **Fluent are functions and predicates that vary from one situation to the next.**
    - E.g. $\neg\text{Holding}(G_1, S_0)$
  - **Eternal predicates are also allowed**
    - E.g. $\text{Gold}(G_1)$
Actions, events and situations

- Results of action sequences are determined by the individual actions.
- *Projection task*: an SC agent should be able to deduce the outcome of a sequence of actions.
- *Planning task*: find a sequence that achieves a desirable effect
Actions, events and situations
Describing change

- Simples Situation calculus requires two axioms to describe change:
  - **Possibility axiom:** when is it possible to do the action
    \[ \text{At}(\text{Agent},x,s) \land \text{Adjacent}(x,y) \Rightarrow \text{Poss}(\text{Go}(x,y),s) \]
  - **Effect axiom:** describe changes due to action
    \[ \text{Poss}(\text{Go}(x,y),s) \Rightarrow \text{At}(\text{Agent},y,\text{Result}(\text{Go}(x,y),s)) \]

- What stays the same?
  - **Frame problem:** how to represent all things that stay the same?
  - **Frame axiom:** describe non-changes due to actions
    \[ \text{At}(o,x,s) \land (o \neq \text{Agent}) \land \neg \text{Holding}(o,s) \Rightarrow \text{At}(o,x,\text{Result}(\text{Go}(y,z),s)) \]
Representational frame problem

- If there are $F$ fluents and $A$ actions then we need $AF$ frame axioms to describe other objects are stationary unless they are held.
  - We write down the effect of each actions

Solution; describe how each fluent changes over time

- Successor-state axiom:

  *possible $\text{Poss}(a,s) \Rightarrow (\text{At}(\text{Agent},y,\text{Result}(a,s)) \iff (a = \text{Go}(x,y)) \lor$
  
  $(\text{At}(\text{Agent},y,s) \land a \neq \text{Go}(y,z))$ for the fluent AT, the action makes the condition true; or the condition was previously true and the action does not make it false.

- Note that next state is completely specified by current state.
- Each action effect is mentioned only once.
Other problems

- How to deal with secondary (implicit) effects?
  - If the agent is carrying the gold and the agent moves then the gold moves too.
  - Ramification problem

- How to decide EFFICIENTLY whether fluents hold in the future?
  - Inferential frame problem.

- Extensions:
  - Event calculus (when actions have a duration)
  - Process categories
Mental events and objects

- So far, KB agents can have beliefs and deduce new beliefs
- What about knowledge about beliefs? What about knowledge about the inference process?
  - Requires a model of the mental objects in someone’s head and the processes that manipulate these objects.
- Relationships between agents and mental objects: believes, knows, wants, ...
  - Believes(Lois, Flies(Superman)) with Flies(Superman) being a function ... a candidate for a mental object (reification).
  - Agent can now reason about the beliefs of agents.
The internet shopping world

- A Knowledge Engineering example
- An agent that helps a buyer to find product offers on the internet.
  - **IN** = product description (precise or ¬precise)
  - **OUT** = list of webpages that offer the product for sale.
- **Environment** = WWW
- **Percepts** = web pages (character strings)
  - Extracting useful information required.
The internet shopping world

- Find relevant product offers
  \[ \text{RelevantOffer}(\text{page}, \text{url}, \text{query}) \iff \text{Relevant}(\text{page}, \text{url}, \text{query}) \land \text{Offer}(\text{page}) \]
  - Write axioms to define Offer(x)
  - Find relevant pages: Relevant(x,y,z) ?
    - Start from an initial set of stores.
    - What is a relevant category?
    - What are relevant connected pages?
  - Require rich category vocabulary.
    - Synonymy and ambiguity
  - How to retrieve pages: GetPage(url)?
    - Procedural attachment

- Compare offers (information extraction).
Reasoning systems for categories

How to organise and reason with categories?

- **Semantic networks**
  - Visualize knowledge-base
  - Efficient algorithms for category membership inference

- **Description logics**
  - Formal language for constructing and combining category definitions
  - Efficient algorithms to decide subset and superset relationships between categories.
Semantic Networks

- Logic vs. semantic networks
- Many variations
  - All represent individual objects, categories of objects and relationships among objects.
- Allows for inheritance reasoning
  - Female persons inherit all properties from person.
  - Cfr. OO programming.
- Inference of inverse links
  - SisterOf vs. HasSister
Semantic network example

[Diagram of a semantic network showing relationships between Mammals, Persons, Female Persons, Male Persons, Mary, John, HasMother, Legs, SubsetOf, MemberOf, SiblingOf]
Semantic networks

- **Drawbacks**
  - Links can only assert binary relations
  - Can be resolved by reification of the proposition as an event

- **Representation of default values**
  - Enforced by the inheritance mechanism.
Description logics

- Are designed to describe definitions and properties about categories
  - A formalization of semantic networks

- Principal inference task is
  - Subsumption: checking if one category is the subset of another by comparing their definitions
  - Classification: checking whether an object belongs to a category.
  - Consistency: whether the category membership criteria are logically satisfiable.
Reasoning with Default Information

*overridden by specific info: nonmonotonic logic

- “The following courses are offered: CS101, CS102, CS106, EE101”
  - Four (db)
    - Assume that this information is complete (not asserted ground atomic sentences are false)
      = CLOSED WORLD ASSUMPTION
    - Assume that distinct names refer to distinct objects
      = UNIQUE NAMES ASSUMPTION
  - Between one and infinity (logic)
    - Does not make these assumptions
    - Requires completion.
Truth maintenance systems

Many of the inferences have default status rather than being absolutely certain

- Inferred facts can be wrong and need to be retracted = BELIEF REVISION.

- Assume KB contains sentence P and we want to execute TELL(KB, ¬P)
  
  \[ \text{To avoid contradiction: RETRACT(KB,P)} \]

  \[ \text{But what about sentences inferred from P?} \]

Truth maintenance systems are designed to handle these complications.