Logic in AI
Chapter 7

Mausam

(Based on slides of Dan Weld, Stuart Russell, Dieter Fox, Henry Kautz...)

Knowledge Representation

• represent knowledge in a manner that facilitates inferencing (i.e. drawing conclusions) from knowledge.

• Typically based on
  – Logic
  – Probability
  – Logic and Probability
Some KR Languages

• Propositional Logic
• Predicate Calculus
• Frame Systems
• Rules with Certainty Factors
• Bayesian Belief Networks
• Influence Diagrams
• Semantic Networks
• Concept Description Languages
• Non-monotonic Logic
Basic Idea of Logic

• By starting with true assumptions, you can deduce true conclusions.
Truth

• Francis Bacon (1561-1626)
No pleasure is comparable to the standing upon the vantage-ground of truth.

• Thomas Henry Huxley (1825-1895)
Irrationally held truths may be more harmful than reasoned errors.

• John Keats (1795-1821)
Beauty is truth, truth beauty; that is all ye know on earth, and all ye need to know.

• Blaise Pascal (1623-1662)
We know the truth, not only by the reason, but also by the heart.

• François Rabelais (c. 1490-1553)
Speak the truth and shame the Devil.

• Daniel Webster (1782-1852)
There is nothing so powerful as truth, and often nothing so strange.
Components of KR

- **Syntax**: defines the sentences in the language
- **Semantics**: defines the “meaning” to sentences
- **Inference Procedure**
  - Algorithm
  - Sound?
  - Complete?
  - Complexity
- **Knowledge Base**
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
Propositional Logic

• Syntax
  – Atomic sentences: P, Q, ...
  – Connectives: \( \land, \lor, \neg, \Rightarrow \)

• Semantics
  – Truth Tables

• Inference
  – Modus Ponens
  – Resolution
  – DPLL
  – GSAT

• Complexity

© Daniel S. Weld
Propositional Logic: Syntax

• Atoms
  – P, Q, R, ...

• Literals
  – P, \( \neg P \)

• Sentences
  – Any literal is a sentence
  – If S is a sentence
    • Then \( (S \land S) \) is a sentence
    • Then \( (S \lor S) \) is a sentence

• Conveniences
  \( P \rightarrow Q \) same as \( \neg P \lor Q \)
Semantics

- **Syntax**: which arrangements of symbols are *legal*
  - (Def “sentences”)
- **Semantics**: what the symbols *mean* in the world
  - (Mapping between symbols and worlds)
Propositional Logic: **SEMANTICS**

- “Interpretation” (or “possible world”)
  - Assignment to each variable either T or F
  - Assignment of T or F to each connective via defns

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
Satisfiability, Validity, & Entailment

• S is **satisfiable** if it is true in *some* world

• S is **unsatisfiable** if it is false *all* worlds

• S is **valid** if it is true in *all* worlds

• S1 **entails** S2 if *wherever* S1 is true S2 is also true
Examples

\( P \rightarrow Q \)

\( R \rightarrow \neg R \)

\( S \land (W \land \neg S) \)

\( T \lor \neg T \)

\( X \rightarrow X \)
Notation

\[ \implies \]
\[ \cup \]
\[ \downarrow \]
\[ \equiv \]

Implication (syntactic symbol)

\textbf{Proves:} \quad S_1 \vdash S_2 \text{ if } \text{"ie" algorithm says } S_2 \text{ from } S_1

\textbf{Entails:} \quad S_1 \models S_2 \text{ if wherever } S_1 \text{ is true } S_2 \text{ is also true}

• Sound \quad \vdash \rightarrow \models

• Complete \quad \models \rightarrow \vdash
Prop. Logic: Knowledge Engr

1) One of the women is a biology major
2) Lisa is not next to Dave in the ranking
3) Dave is immediately ahead of Jim
4) Jim is immediately ahead of a bio major
5) Mary or Lisa is ranked first

1. Choose Vocabulary
   Universe: Lisa, Dave, Jim, Mary
   LD = “Lisa is immediately ahead of Dave”
   D = “Dave is a Bio Major”

2. Choose initial sentences (wffs)
Reasoning Tasks

• Model finding
  
  $KB = \text{background knowledge}$
  
  $S = \text{description of problem}$
  
  Show $(KB \land S)$ is satisfiable
  
  A kind of constraint satisfaction

• Deduction
  
  $S = \text{question}$
  
  Prove that $KB \models S$
  
  Two approaches:
  
  • Rules to derive new formulas from old (inference)
  
  • Show $(KB \land \neg S)$ is unsatisfiable
Special Syntactic Forms

• General Form:
  
  \[ ((q \land \neg r) \rightarrow s)) \land \neg (s \land t) \]

• Conjunction Normal Form (CNF)

  \[ (\neg q \lor r \lor s) \land (\neg s \lor \neg t) \]

  Set notation: \{ (\neg q, r, s), (\neg s, \neg t) \}

  empty clause () = false

• Binary clauses: 1 or 2 literals per clause

  \[ (\neg q \lor r) \quad (\neg s \lor \neg t) \]

• Horn clauses: 0 or 1 positive literal per clause

  \[ (\neg q \lor \neg r \lor s) \quad (\neg s \lor \neg t) \]

  \[ (q \land r) \rightarrow s \quad (s \land t) \rightarrow false \]
Propositional Logic: **Inference**

*A mechanical* process for computing new sentences

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. GSAT
4. Davis Putnam
Inference 1: Forward Chaining

Forward Chaining
Based on rule of *modus ponens*

If know $P_1, \ldots, P_n$ & know $(P_1 \land \ldots \land P_n) \rightarrow Q$
Then can conclude Q

Backward Chaining: search
start from the query and go backwards
Analysis

• Sound?

• Complete?

Can you prove

\{ \} \models Q \lor \neg Q

• If KB has only Horn clauses & query is a single literal
  – Forward Chaining is complete
  – Runs linear in the size of the KB
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]

A

B
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Propositional Logic: Inference

A *mechanical* process for computing new sentences

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. GSAT
4. Davis Putnam
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:

\[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1}) \]
Inference 2: Resolution
[Robinson 1965]

\{ (p \lor \alpha), (\neg p \lor \beta \lor \gamma) \} \vdash_{-R} (\alpha \lor \beta \lor \gamma)

Correctness
If $S_1 \vdash_{-R} S_2$ then $S_1 \models S_2$

Refutation Completeness:
If $S$ is unsatisfiable then $S \vdash_{-R} ()$
Resolution

If the unicorn is mythical, then it is immortal, but if it is not mythical, it is a mammal. If the unicorn is either immortal or a mammal, then it is horned.

Prove: the unicorn is horned.

\( M = \text{mythical} \)

\( I = \text{immortal} \)

\( A = \text{mammal} \)

\( H = \text{horned} \)
Resolution as Search

• States?
• Operators
Model Finding

• Find assignments to variables that makes a formula true

• a CSP
Inference 3: Model Enumeration

for (m in truth assignments) {
    if (m makes $\Phi$ true) then return “Sat!”
}

return “Unsat!”
Inference 4: DPLL
(Enumeration of *Partial* Models)

[Davis, Putnam, Loveland & Logemann 1962]

*Version 1*

dpl1_1(pa) {
    if (pa makes F false) return false;
    if (pa makes F true) return true;
    choose P in F;
    if (dpl1_1(pa ∪ {P=0})) return true;
    return dpl1_1(pa ∪ {P=1});
}

Returns true if F is satisfiable, false otherwise
DPLL Version 1

\[(a \lor b \lor c)\]
\[(a \lor \neg b)\]
\[(a \lor \neg c)\]
\[(\neg a \lor c)\]
(a ∨ b ∨ c)
(a ∨ ¬b)
(a ∨ ¬c)
(¬a ∨ c)
(F ∨ b ∨ c)
(F ∨ ¬b)
(F ∨ ¬c)
(T ∨ c)
DPLL Version 1

(F \lor F \lor c)
(F \lor T)
(F \lor \neg c)
(T \lor c)
(F ∨ F ∨ F)
(F ∨ T)
(F ∨ T)
(T ∨ F)
DPLL Version 1

F
T
T
T

a
b
c

F

© Daniel S. Weld
(a \lor b \lor c)
(a \lor \neg b)
(a \lor \neg c)
(\neg a \lor c)
\[(a \lor b \lor c)\]
\[(a \lor \neg b)\]
\[(a \lor \neg c)\]
\[(\neg a \lor c)\]
DPLL as Search

• Search Space?

• Algorithm?
Improving DPLL

If literal $L_1$ is true, then clause $(L_1 \lor L_2 \lor \ldots)$ is true
If clause $C_1$ is true, then $C_1 \land C_2 \land C_3 \land \ldots$ has the same value as $C_2 \land C_3 \land \ldots$

Therefore: Okay to delete clauses containing true literals!

If literal $L_1$ is false, then clause $(L_1 \lor L_2 \lor L_3 \lor \ldots)$ has the same value as $(L_2 \lor L_3 \lor \ldots)$

Therefore: Okay to delete shorten containing false literals!

If literal $L_1$ is false, then clause $(L_1)$ is false

Therefore: the empty clause means false!
DPLL version 2

dpll_2(F, literal) {
    remove clauses containing literal
    if (F contains no clauses) return true;
    shorten clauses containing \neg literal
    if (F contains empty clause) return false;
    choose V in F;
    if (dpll_2(F, \neg V)) return true;
    return dpll_2(F, V);
}

Partial assignment corresponding to a node is the set of chosen literals on the path from the root to the node
(F ∨ b ∨ c)
(F ∨ ¬b)
(F ∨ ¬c)
(T ∨ c)
DPLL Version 2

\[(b \lor c)\]
\[\neg b\]
\[\neg c\]
DPLL Version 2

\[(F \lor c)\]
\[(T)\]
\[(\neg c)\]
DPLL Version 2

(c)

(¬c)
DPLL Version 2

Empty clause!

(())

a

b

c

a

b

c

© Daniel S. Weld
(F ∨ F ∨ F)
(F ∨ T)
(F ∨ T)
(T ∨ F)
DPLL Version 2
DPLL Version 2

\[(a \lor b \lor c)\]
\[(a \lor \neg b)\]
\[(a \lor \neg c)\]
\[(\neg a \lor c)\]
Benefit

• Can backtrack before getting to leaf
Structure in Clauses

• Unit Literals
  A literal that appears in a singleton clause
  \{\{\neg b \ c\}\{\neg c\}\{a \neg b \ e\}\{d \ b\}\{e \ a \neg c\}\}

  **Might as well set it true! And simplify**
  \{\{\neg b\}\{a \neg b \ e\}\{d \ b\}\}
  \{\{d\}\}

• Pure Literals
  – A symbol that always appears with same sign
  – \{\{a \neg b \ c\}\{\neg c \ d \neg e\}\{\neg a \neg b \ e\}\{d \ b\}\{e \ a \neg c\}\}

  **Might as well set it true! And simplify**
  \{\{a \neg b \ c\}\{\neg a \neg b \ e\}\{e \ a \neg c\}\}
In Other Words

Formula \((L) \land C_2 \land C_3 \land ...\) is only true when literal \(L\) is true
Therefore: Branch immediately on unit literals!

May view this as adding constraint propagation techniques into play
In Other Words

Formula \((L) \land C_2 \land C_3 \land \ldots\) is only true when literal \(L\) is true.

Therefore: Branch immediately on unit literals!

If literal \(L\) does not appear negated in formula \(F\), then setting \(L\) true preserves satisfiability of \(F\).

Therefore: Branch immediately on pure literals!

May view this as adding constraint propagation techniques into play.
DPLL (previous version)
Davis – Putnam – Loveland – Logemann

dpll(F, literal) {
    remove clauses containing literal
    if (F contains no clauses) return true;
    shorten clauses containing \neg literal
    if (F contains empty clause) return false;

    choose V in F;
    if (dpll(F, \neg V)) return true;
    return dpll(F, V);
}
dpll(F, literal) {
    remove clauses containing literal
    if (F contains no clauses) return true;
    shorten clauses containing ¬literal
    if (F contains empty clause) return false;
    if (F contains a unit or pure L) return dpll(F, L);
    choose V in F;
    if (dpll(F, ¬V)) return true;
    return dpll(F, V);
}
DPLL (for real)

\[(a \lor b \lor c)\]
\[(a \lor \neg b)\]
\[(a \lor \neg c)\]
\[(\neg a \lor c)\]
DPLL (for real!)

Davis – Putnam – Loveland – Logemann

dpll(F, literal){
    remove clauses containing literal
    if (F contains no clauses) return true;
    shorten clauses containing \(\neg\)literal
    if (F contains empty clause)
        return false;
    if (F contains a unit or pure L)
        return dpll(F, L);
    choose V in F;
    if (dpll(F, \(\neg\)V)) return true;
    return dpll(F, V);
}

Where could we use a heuristic to further improve performance?
Heuristic Search in DPLL

• Heuristics are used in DPLL to select a (non-unit, non-pure) proposition for branching

• Idea: identify a most constrained variable
  – Likely to create many unit clauses

• MOM’s heuristic:
  – Most occurrences in clauses of minimum length
Success of DPLL

- 1962 – DPLL invented
- 1992 – 300 propositions
- 1997 – 600 propositions (satz)
- Additional techniques:
  - Learning conflict clauses at backtrack points
  - Randomized restarts
  - 2002 (zChaff) 1,000,000 propositions – encodings of hardware verification problems
WalkSat (Take 1)

- **Local** search (Hill Climbing + Random Walk) over space of **complete** truth assignments
  - With prob $p$: flip any variable in any unsatisfied clause
  - With prob $(1-p)$: flip **best** variable in any unsat clause
    - best = one which minimizes #unsatisfied clauses

- SAT encodings of N-Queens, scheduling

- Best algorithm for random K-SAT
  - Best DPLL: 700 variables
  - Walksat: 100,000 variables
Refining Greedy Random Walk

• Each flip
  – makes some false clauses become true
  – breaks some true clauses, that become false

• Suppose \( s_1 \rightarrow s_2 \) by flipping \( x \). Then:
  \[
  \#\text{unsat}(s_2) = \#\text{unsat}(s_1) - \text{make}(s_1, x) + \text{break}(s_1, x)
  \]

• Idea 1: if a choice breaks nothing, it is very likely to be a good move

• Idea 2: near the solution, only the break count matters
  – the make count is usually 1
Walksat (Take 2)

state = random truth assignment;
while ! GoalTest(state) do
    clause := random member \{ C \mid C is false in state \};
    for each x in clause do compute break[x];
    if exists x with break[x]=0 then var := x;
    else
        with probability p do
            var := random member \{ x \mid x is in clause \};
        else
            var := arg x min \{ break[x] \mid x is in clause \};
        endif
    state[var] := 1 – state[var];
end
return state;

Put everything inside of a restart loop.
Parameters: p, max_flips, max_runs
Random 3-SAT

- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - $n$ variables, $l$ clauses

- Which are the hard instances?
  - around $l/n = 4.3$
Random 3-SAT

- Varying problem size, $n$

- Complexity peak appears to be largely invariant of algorithm
  - backtracking algorithms like Davis-Putnam
  - local search procedures like GSAT

- What’s so special about 4.3?
Random 3-SAT

• Complexity peak coincides with solubility transition

  – $l/n < 4.3$ problems under-constrained and SAT
  – $l/n > 4.3$ problems over-constrained and UNSAT
  – $l/n=4.3$, problems on “knife-edge” between SAT and UNSAT
Assignment 2: Graph Subset Mapping

• Given two directed graphs G and G’
  – Check if G is a subset mapping to G’

• I.e. construct a one-one mapping (M) from all nodes of G to some nodes of G’ s.t.
  – (n1,n2) in G → (M(n1), M(n2)) in G’
  – (n1,n2) not in G → (M(n1), M(n2)) not in G’
No, because the directionality of edges doesn’t match.

No, because there is no edge between A and C in $G$ whereas there is one between P and R in $G'$. 

Yes. A mapping is: $M(A) = S$, $M(B) = Q$, $M(C) = R$

The edges from P to other nodes don’t matter since no node in $G$ got mapped to P.
SAT Model for Graph Subset Mapping

• If a mapping exists then SAT formula is satisfiable
• Else unsatisfiable

• The satisfying assignment suggests the mapping M