Logic in AI
Chapter 7

Dan Weld
(With some slides from Mausam, 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” of 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: ∧, ∨, ¬, →

• **Semantics**
  – Truth Tables

• **Inference**
  – Modus Ponens
  – Resolution
  – DPLL
  – GSAT

• **Complexity**
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**
  
  \[ P \rightarrow Q \text{ same as } \neg P \lor Q \]
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>P</th>
<th>T</th>
<th>F</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>P ( \land ) Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P ( \lor ) Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<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

\[ \iff \]
\[ \implies \]
\[ \models \]

}\}

Implication (syntactic symbol)

Proves: \( S_1 \vdash_i S_2 \) if inference algo, i, says `S2' from S1

Entails: \( S_1 \models S_2 \) if wherever S1 is true S2 is also true

• Sound \( \vdash \rightarrow \models \)

• Complete \( \models \rightarrow \vdash \)

© Daniel S. Weld
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
  
  \( KB = \) background knowledge
  
  \( S = \) description of problem
  
  Show \((KB \land S)\) is satisfiable
  
  A kind of constraint satisfaction

• Deduction
  
  \( S = \) 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 () = \text{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
  \((\neg q \lor \neg r \lor s) \quad (\neg s \lor \neg t)\)
  \((q \land r) \rightarrow s \quad (s \land t) \rightarrow \text{false}\)
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
$P \Rightarrow Q$
$L \land M \Rightarrow P$
$B \land L \Rightarrow M$
$A \land P \Rightarrow L$
$A \land B \Rightarrow L$
$A$
$B$
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

\begin{align*}
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 & 
\end{align*}
Example

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

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

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 \( (\lor \text{ over } \land) \) 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.

\[ \neg (A \lor H) \land (\neg H) \land (\neg I \lor H) \]

\[ (M \lor A) \land (\neg A) \land (\neg I) \land (\neg M \lor I) \]

\[ (M) \land (\neg M) \land () \]

\[ M = \text{mythical} \]
\[ I = \text{immortal} \]
\[ A = \text{mammal} \]
\[ H = \text{horned} \]
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

\[
dpll_1(pa)\{
    \text{if (pa makes F false) return false;}
    \text{if (pa makes F true) return true;}
    \text{choose P in F;}
    \text{if (dpll_1(pa } \cup \{P=0\}) \text{) return true;}
    \text{return } dpll_1(pa \cup \{P=1\});
\}
\]

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

\((a \lor b \lor c)\)

\((a \lor \neg b)\)

\((a \lor \neg c)\)

\((\neg a \lor c)\)
DPLL Version 1

\[(a \lor b \lor c)\]
\[(a \lor \neg b)\]
\[(a \lor \neg c)\]
\[(\neg a \lor c)\]
DPLL Version 1

(\text{F} \lor b \lor c)

(\text{F} \lor \neg b)

(\text{F} \lor \neg c)

(\text{T} \lor c)
(F ∨ F ∨ c)
(F ∨ T)
(F ∨ ¬c)
(T ∨ c)
DPLL Version 1

((F ∨ F ∨ F))
((F ∨ T))
((F ∨ T))
((T ∨ F))
DPLL Version 1
(a ∨ b ∨ c)
(a ∨ ¬b)
(a ∨ ¬c)
(¬a ∨ c)
(a ∨ b ∨ c)
(a ∨ ¬b)
(a ∨ ¬c)
(¬a ∨ c)
DPLL as Search

• Search Space?

• Algorithm?
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!
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!
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
DPLL Version 2

\[(F \lor b \lor c)\]
\[(F \lor \neg b)\]
\[(F \lor \neg c)\]
\[(T \lor c)\]
$$(b \lor c)$$

$$(\neg b)$$

$$(\neg c)$$
(F ∨ c)
(T)
(¬c)
DPLL Version 2

\((c)\)

\((\neg c)\)
DPLL Version 2

(F)

(T)
Empty clause!

( )

DPLL Version 2
\((F \lor F \lor F)\)
\((F \lor T)\)
\((F \lor T)\)
\((T \lor F)\)
DPLLL Version 2

F
T
T
T
T
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*
  \[
  \{\neg b\} \quad \{a \neg b \ e\} \{d \ b\}\]
  \[
  \{\{d\}\}
  \]

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

  *Might as well set it true! And simplify*
  \[
  \{a \neg b \ c\} \quad \{\neg a \neg b \ e\} \quad \{e \ a \neg c\}\]
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!

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(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 (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);
}
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);
}
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 | 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 | x is in clause };
        else
            var := arg x min { break[x] | 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