CSE 473

Lecture 11

Chapter 7

Inference in Propositional Logic

Logic: another thing that penguins aren't very good at.

Recall: Propositional Logic Terminology

Terminology:

Literal = proposition symbol or its negation
E.g., A, ¬A, B, ¬B, etc. (positive vs. negative)

Clause = disjunction of literals
E.g., $(B \vee \neg C \vee \neg D)$

Conjunctive Normal Form (CNF):
sentence = conjunction of clauses
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Can think of KB as a conjunction of clauses, i.e. one long sentence

Recall: Satisfiability

- A sentence is *satisfiable* if it is true in *some* model

  e.g., $A \vee B$, C
- A sentence is *unsatisfiable* if it is true in no models

  e.g., $A \wedge \neg A$
- Satisfiability is connected to inference via the following:

  $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is *unsatisfiable* (proof by contradiction)

3

Last Time: Propositional Inference

Two main approaches to inference:

1. Inference by *Model Checking (TT enumeration)*
2. Inference by *Theorem Proving*: Use rules of inference to construct a proof of a sentence

4

Review: Inference by Theorem Proving

Use rules of inference to construct a proof of a sentence

- *Search for proof* based on modus ponens, and-elimination, logical equivalences

  One important equivalence: $A \Rightarrow B \equiv \neg A \vee B$

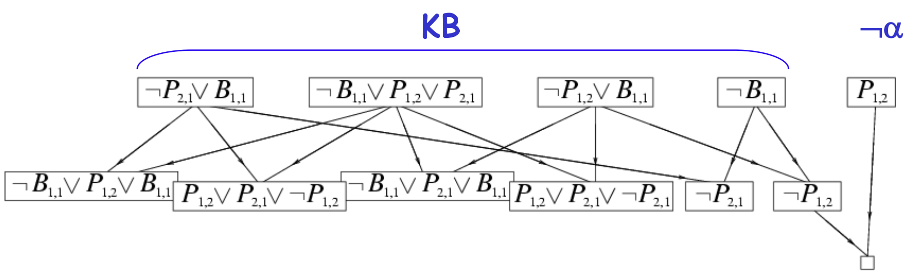
- *Forward and backward chaining* for KBs of Horn clauses (disjunctions of literals, at most 1 positive literal)

  If A and B are true and $A \wedge B \Rightarrow C$, then C true

- *Resolution*: A single complete and sound rule

5

Review: Resolution example



You got a literal and its negation **Empty clause**

What does this (empty clause) mean?

Recall that KB is a *conjunction* of all these clauses
Is $P_{1,2} \wedge \neg P_{1,2}$ satisfiable? No!

Therefore, KB $\wedge \neg \alpha$ is unsatisfiable, i.e., KB $\models \alpha$

Review: Inference by Model Checking

Complete search algorithms

Truth table enumeration: Recursive depth-first enumeration of assignments to all symbols (*TT-entails*)

Heuristic search

DPLL algorithm (Davis, Putnam, Logemann, Loveland): Recursive depth-first enumeration of possible models *with heuristics* (such as early termination)

Incomplete *local search* algorithms

`WalkSAT` algorithm for checking satisfiability

7

Why Satisfiability?

8

Why Satisfiability?

- Recall: $KB \models \alpha$ iff $KB \wedge \neg\alpha$ is unsatisfiable
  - Equivalent to proving sentence α by contradiction
- Thus, algorithms for satisfiability can be used for inference (entailment)
- However, determining if a sentence is satisfiable or not (the SAT problem) is NP-complete

  Finding a fast algorithm for SAT automatically yields fast algorithms for hundreds of difficult (NP-complete) problems

9

Satisfiability Examples

E.g. 2-CNF sentences (2 literals per clause):

$(\neg A \vee \neg B) \wedge (A \vee B) \wedge (A \vee \neg B)$
Satisfiable?
Yes (e.g., A = true, B = false)

$(\neg A \vee \neg B) \wedge (A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$
Satisfiable?
No

10

The WalkSAT algorithm

- Local search algorithm
  - Incomplete: may not always find a satisfying assignment even if one exists
- Evaluation function?
  = Number of satisfied clauses
  - WalkSAT tries to maximize this function
- Balance between greediness and randomness
  - Each iteration:
  - Randomly select a symbol for flipping
  - Or select symbol that maximizes # satisfied clauses

11

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*

Greed   **Randomness**

12

Hard Satisfiability Problems

Consider random 3-CNF sentences. e.g.,
$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee A) \wedge (A \vee \neg D \vee B) \wedge (B \vee D \vee \neg C)$

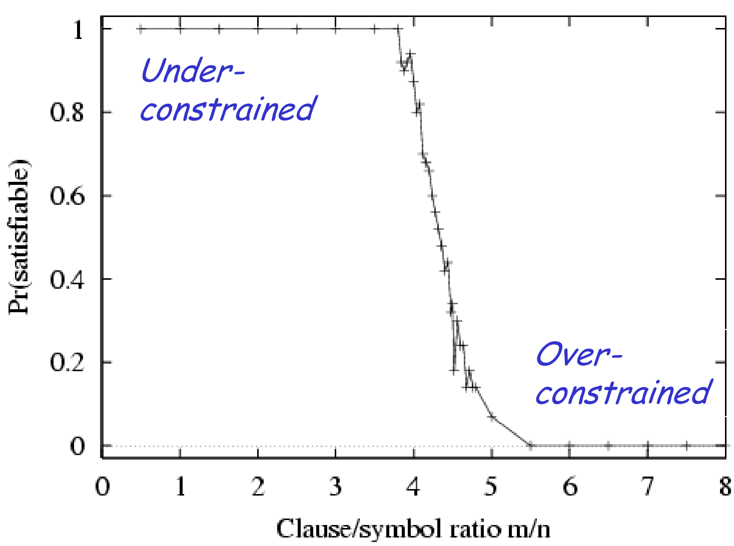
Satisfiable?
(Yes, e.g., A = B = C = *true*)

m = number of clauses (Here 5)
n = number of symbols (Here 4 – A, B, C, D)
m/n = 1.25 (enough symbols, usually satisfiable)

Hard instances of SAT seem to cluster near *m/n* = 4.3 (critical point)

13

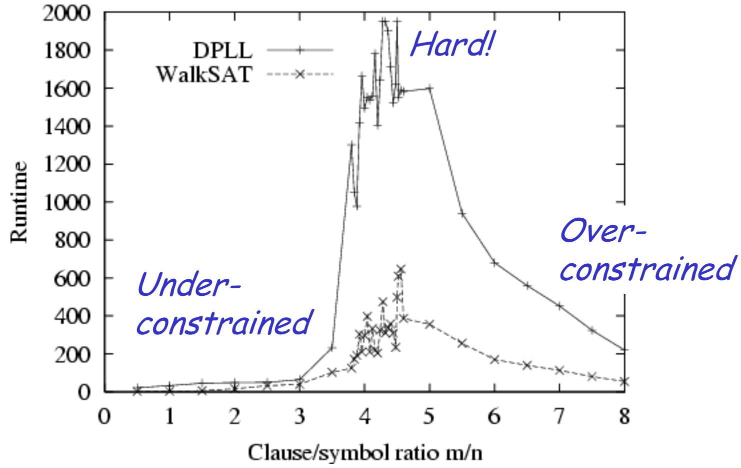
Hard Satisfiability Problems



14

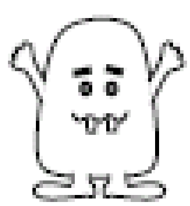
Hard Satisfiability Problems

Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$

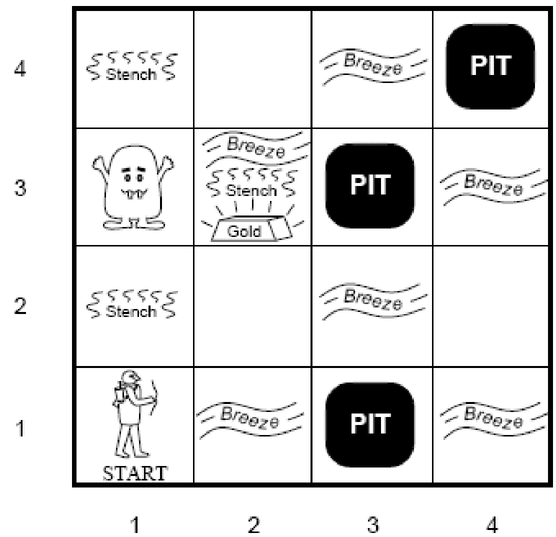


15

What about me?



Wumpus World



17

Putting it all together: Logical Wumpus Agents

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$
$\neg W_{1,1}$
For $x = 1, 2, 3, 4$ and $y = 1, 2, 3, 4$, add (with appropriate boundary conditions):

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$ At least 1 wumpus
$\neg(W_{1,1} \wedge W_{1,2})$
$\neg(W_{1,1} \wedge W_{1,3})$ At most 1 wumpus
...

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences!

18

Limitations of propositional logic

- KB contains "physics" sentences for every single square
- For every time step *t* and every location [*x*,*y*], we need to add to the KB "physics" rules such as:

$$L^{t}_{x,y} \wedge FacingRight^{t} \wedge Forward^{t} \Rightarrow L^{t+1}_{x+1,y}$$

- Rapid proliferation of sentences…

What we'd like is a way to talk about *objects* and *groups* of objects, and to define relationships between them.

Enter: First-order logic
(aka "predicate logic")

Next Time

- First-Order Logic
- To Do:
  - Project #2
  - Read chapter 8