EECS 391: Introduction to AI

Soumya Ray

Website: http://vorlon.case.edu/~sray/eecs391_sp12/index.html

Email: sray@case.edu

Office: Olin 516

Office hours: Tue 2:30-4:00 or by appointment

Today

- Inference in propositional logic
- First order logic (Chapter 8)

Syntactic Inference: Inference Rules

- Given a KB, syntactic inference algorithms apply rules to derive other formulae
- Sequence of rule applications is called a derivation or "proof"
- Rules are written as:

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\beta_1, \beta_2, \ldots, \beta_m}$$

If the sentences $\alpha_1,\ldots,\alpha_n$ are in the KB, agent can infer $\beta_1,\ldots,\beta_m$ and add them to the KB.

Inference Rules

- Modus Ponens ("Rule of affirming the antecedent")

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

- Modus Tollens ("Rule of denying the consequent")

$$\frac{\alpha \Rightarrow \beta, \quad \neg\beta}{\neg\alpha}$$

- And-Elimination

$$\frac{\alpha \wedge \beta}{\beta}$$

- *Every logical equivalence* can be turned into an inference rule

Resolution (Robinson 1965)

- Unit Resolution: $\dfrac{l_1 \vee l_2 \vee m \ldots \vee l_k,\ \neg m}{l_1 \vee l_2 \ldots \vee l_k}$

- General Resolution:

$$\frac{l_1 \vee l_2 \vee m \ldots \vee l_k,\quad r_1 \vee r_2 \vee \neg m \ldots \vee r_k}{l_1 \vee l_2 \ldots \vee l_k \vee r_1 \vee r_2 \ldots \vee r_k}$$

Factoring: the resulting formula could have duplicate literals. Remove the duplicates.

Resolution

- Special Case:

$$\frac{m,\ \neg m}{\square}$$

Properties of Resolution

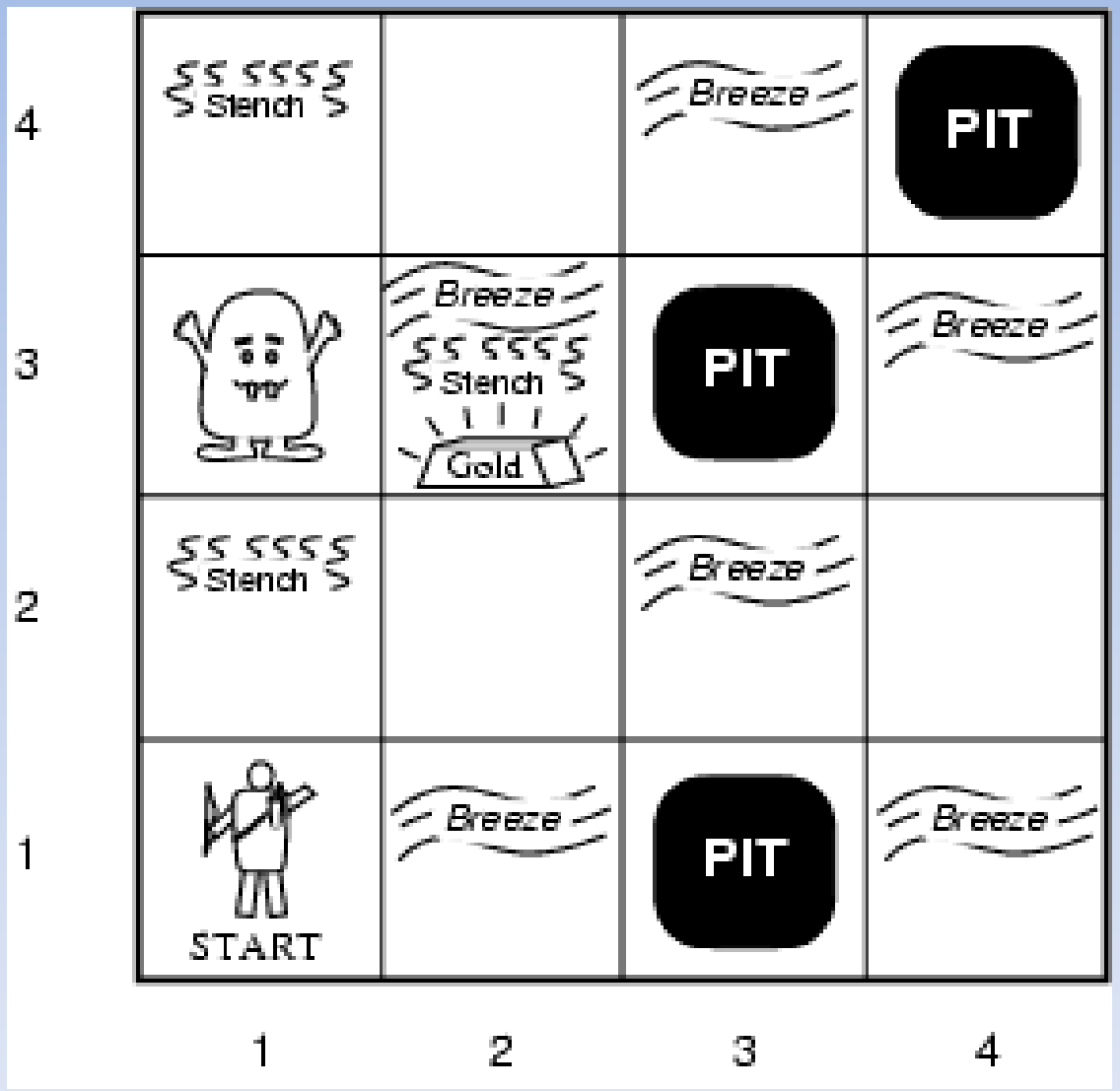
- Sound (why?)

- "Refutation Complete"
  - Everything that can be entailed can be derived, but only by checking if (KB $\wedge \neg\alpha$) derives the empty clause

Algorithm

- We can use resolution in a search algorithm as follows:
  - Convert (KB $\wedge \neg\alpha$) to CNF
  - Generate all possible consequences using resolution as search operator
  - Continue until:
    - No new clauses are generated. Then KB does NOT entail α
    - Two clauses resolve to yield the empty clause. Then KB entails α (Why?)

Example


4
3
2
1
Stench
Breeze
PIT
Stench
Breeze
Gold
PIT
Breeze
Stench
Breeze
START
Breeze
PIT
Breeze
1
2
3
4

Semantic Inference

- So far, we have considered algorithms for syntactic inference
  - These algorithms work by using inference rules to produce derivations
- Now we will look at algorithms that perform semantic inference
  - Inference via models and entailment
  - For propositional logic, this means considering possible truth assignments
  - Also called inference via “model checking”

Satisfiability

- A key question for propositional logic formulae is *satisfiability*
  - Does this formula have *any* model?

- Useful for consistency checking
  - Does a KB entail $FALSE$?
  - If a knowledge base is inconsistent, it is useless

Satisfiability and Inference

- Satisfiability and inference are linked by the "proof-by-refutation" theorem: $\alpha \models \beta$ iff $(\alpha \wedge \neg\beta)$ is unsatisfiable

- Satisfiability checking algorithms use search
  - Systematic search (DPLL)
  - Local search (WalkSAT)

Satisfiability by Systematic Search

- How can we set up the problem?

DPLL

(Davis, Putnam, Logemann, Loveland 1960)

- Input: formula in CNF
- Output: satisfying assignment if any
- Algorithm: Depth first search of space of all models
  - Uses heuristics to guide search (prune search space)
  - Sound and complete

Heuristic 1: Pure literals

- Suppose a CNF formula has a symbol that appears only negated or only unnegated in all clauses
  - This is a "pure" literal
  - If the formula is satisfiable, it is satisfiable with the pure literal set to *true*
  - Example:

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$

Heuristic 2: Unit Propagation

- Suppose a clause has only a single literal (because everything else is assigned *false*)
- Then that literal has to be set to *true*
  - This can cascade
  - Example

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$

$$B = true$$

Heuristic 3: Early Termination

- Since the formula is in CNF, for any assignment to be a satisfying one, every clause has to be true
  - Any (partial) assignment that makes any clause false can be pruned
- Since a clause is a disjunction, once any literal is set to true, that clause does not have to be considered any more

DPLL

- Start with initial list of clauses and a current assignment (initially empty)
- If all clauses true return true
- If any clause false return false
- If :
  - There is an unassigned pure literal, set to true and call DPLL on result
  - Else if there is an unassigned unit clause, set to true and call DPLL on result
  - Else pick a random symbol, return DPLL with symbol set to true OR'ed with DPLL with symbol set to false

Satisfiability by Local Search

- How can we set up the problem?

WalkSAT (Kautz and Selman, 1993)

- Start with initial complete assignment
- Repeat MAX_FLIPS times:
  - If all clauses true, return current assignment
  - Else, pick random unsatisfied clause
  - With small probability ε, flip the assignment of a random variable in clause
  - Else flip the assignment of the variable in clause that maximizes satisfied clauses

Properties of WalkSAT

- Sound (obviously)
- Not Complete
  - If it returns no satisfying assignment, does not imply formula is unsatisfiable
  - But it turns out that depending on the formula, we can make a really good guess about this

Why is SAT hard?

- The great success of local search in SAT led to a huge amount of research on SAT
  - People knew theoretically that SAT was a hard problem
  - But here was a simple local search procedure which was able to solve extremely large SAT problems extremely fast
  - How to reconcile these facts? Could we isolate the “hard” SAT problems somehow? Are these an interesting subset of SAT problems?

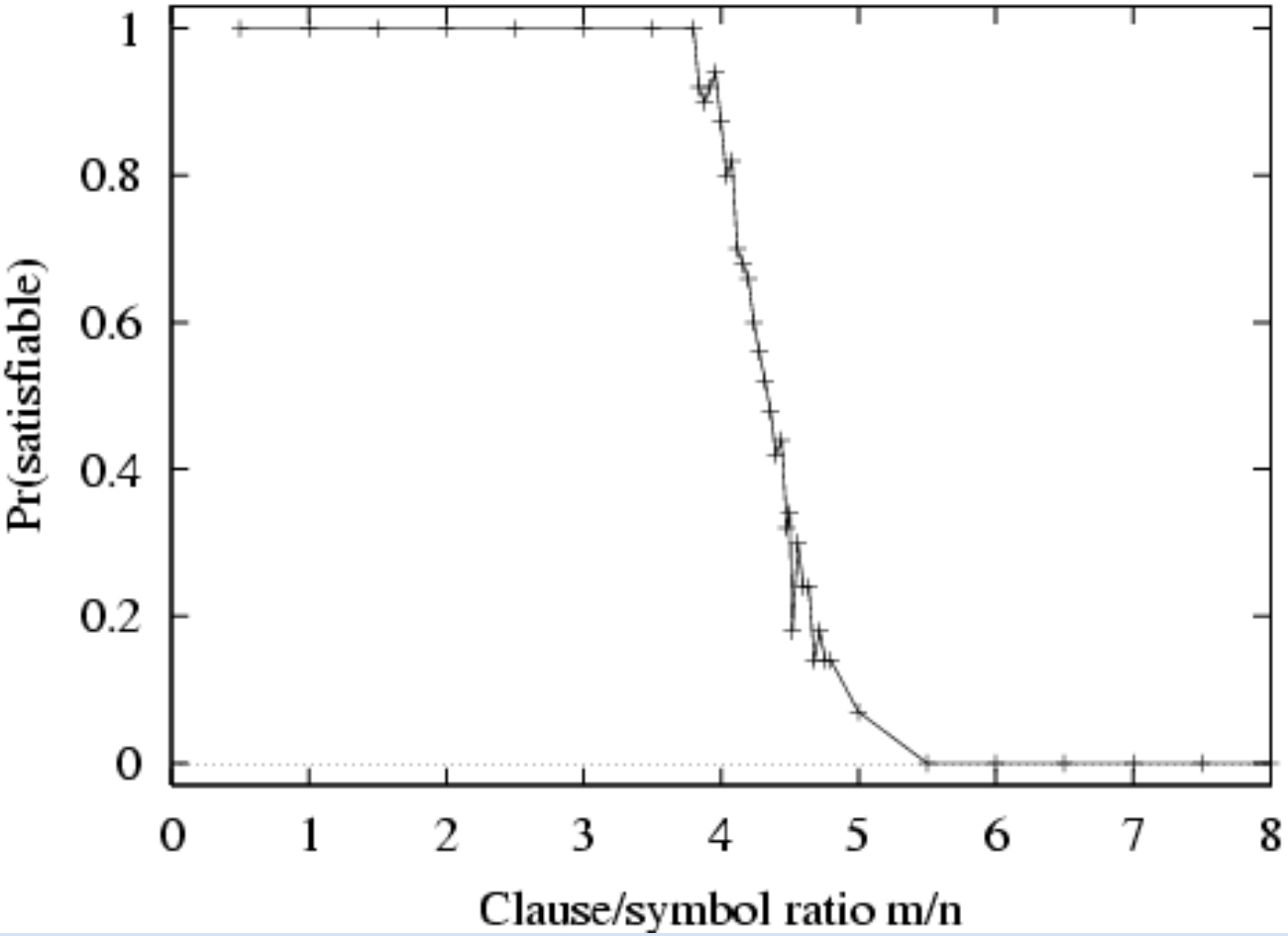
Hard SAT Problems

- Suppose we randomly generate a CNF formula
- There are two key parameters
  - The number of clauses in the formula (M)
  - The number of propositional symbols (N)
- It turns out that the ratio M/N determines the hardness of a SAT problem

Intuition

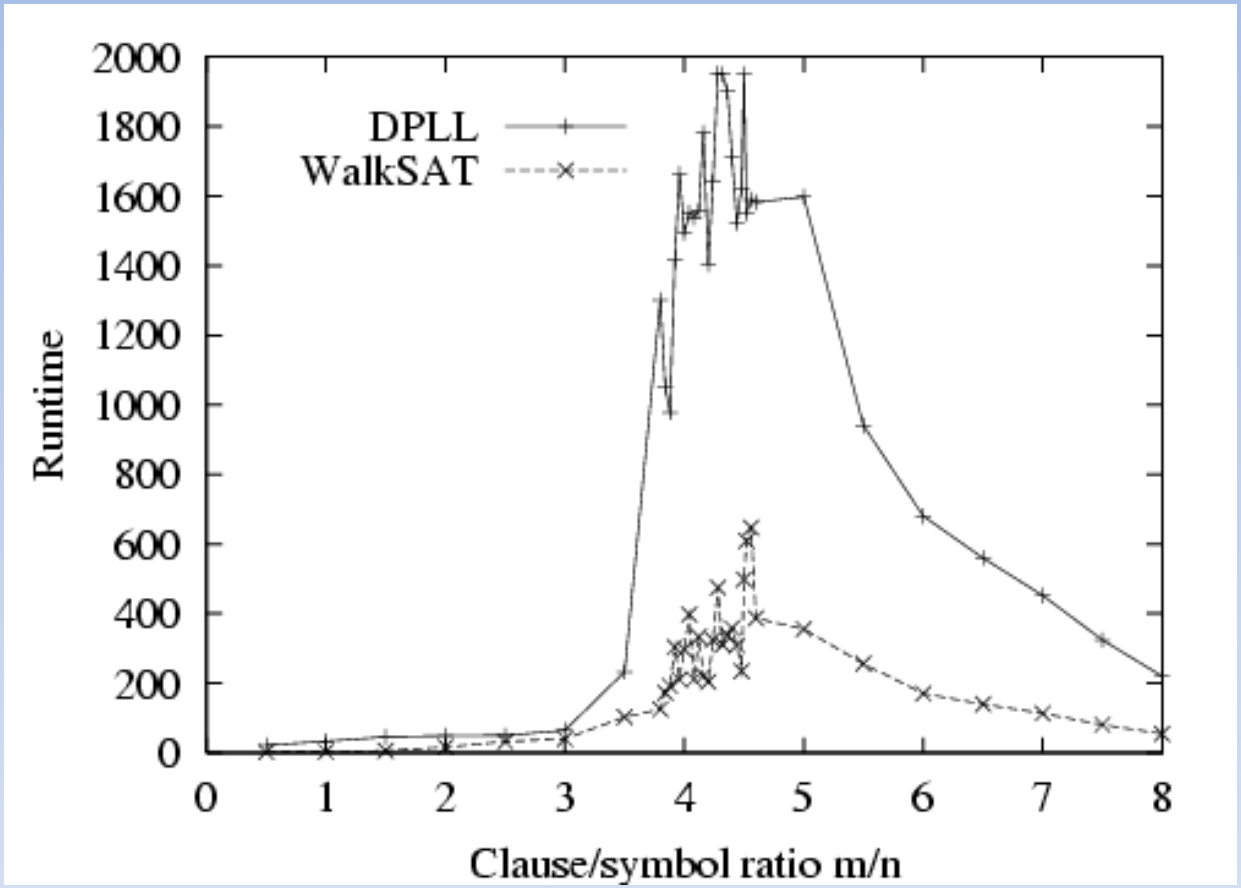
- If there are many symbols and few clauses, the problem is *underconstrained*: the probability of a random assignment being satisfying is close to 1
- If there are few symbols and many clauses, the problem is *overconstrained*: the probability of a random assignment being satisfying is close to 0
- For a critical value of M/N, the probability of a random assignment being satisfying is close to 0.5
  - These are the hardest SAT problems

Phase Transition



If a problem domain has this sort of characteristic, it is said to have a "phase transition." Typically in such cases, problems near the transition are hard to solve.

Runtime Characteristics



Median runtime for 100 satisfiable random 3CNF formulae, N = 50

First-Order Logic

Issues with Propositional Logic

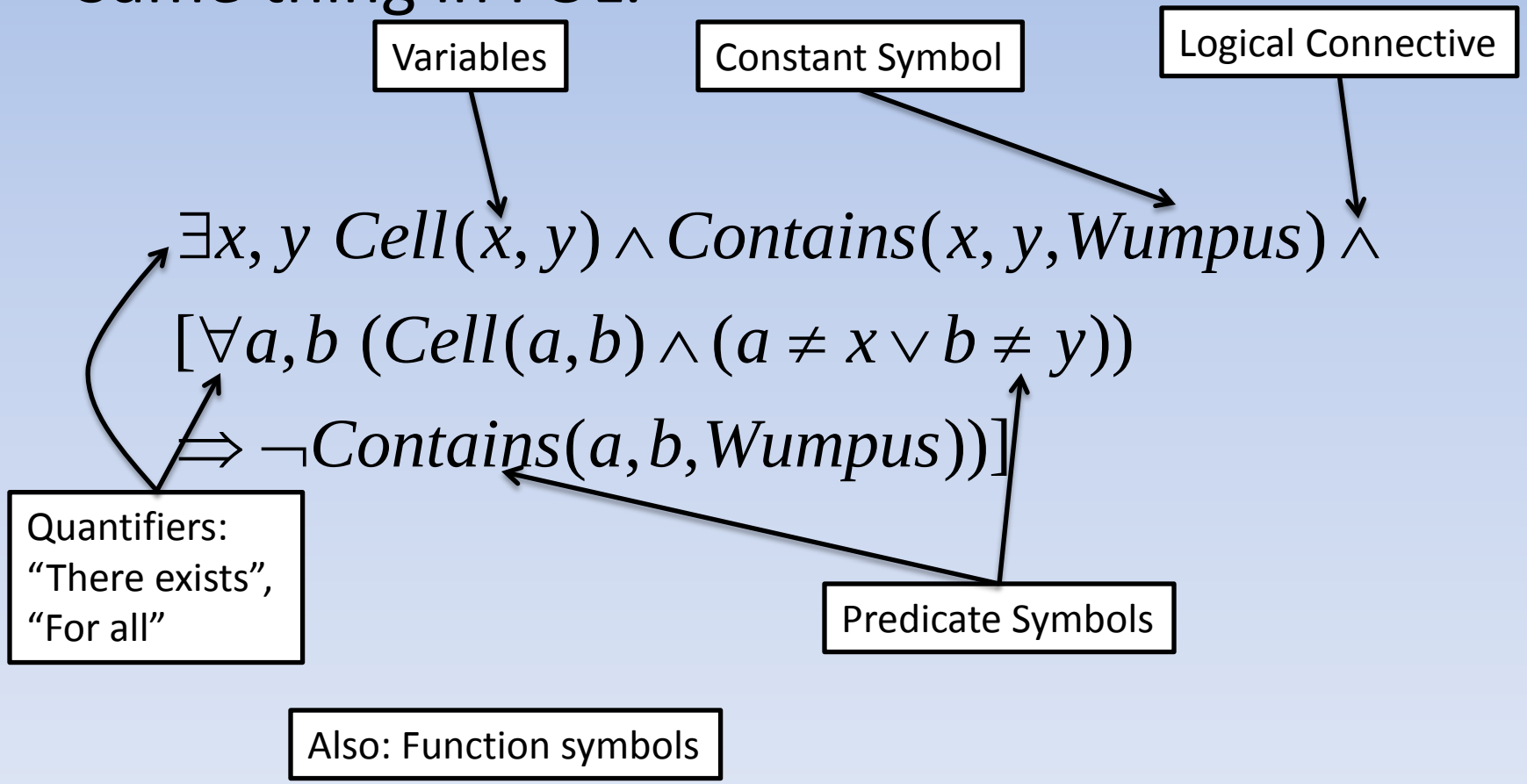
- Want to say: “Somewhere in the grid there is one wumpus.”

$$(W_{1,1} \wedge \neg W_{1,2} \wedge \ldots \wedge \neg W_{m,n}) \vee$$

$$(\neg W_{1,1} \wedge W_{1,2} \wedge \ldots \wedge \neg W_{m,n}) \vee \ldots$$

First Order Logic

- Same thing in FOL:



Key Elements of FOL

- FOL introduces:
  - Objects
  - *Relations* between objects (predicates)
  - *Functions* that map objects to other objects
  - *Variables* that act as placeholders for objects
  - *Quantifiers* that allow us to talk about (infinite) collections of objects