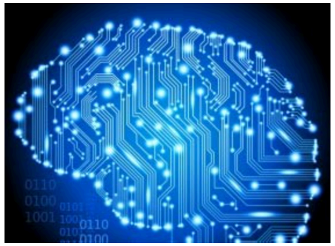
Lecture 9: Knowledge 3

http://cs.nju.edu.cn/yuy/course_ai17.ashx



Previously...

Propositional Logic

PL-Forward chaining
PL-Backward chaining
PL-Resolution

First Order Logic (FOL)

Instantiation
FOL-Forward chaining
FOL-Backward chaining
FOL-Resolution

SAT problems

Propositional logic, CNF

literals: $x_1, x_2, \ldots, x_n$

clauses: $(x_1 \vee x_2 \vee x_5)$ $(\neg x_2 \vee x_3 \vee \neg x_7)$ ...

problem: find an assignment to literals so that the conjunction of the clauses is true, or prove unsatisfiable

$$(x_1 \vee x_2 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_7) \wedge \ldots$$

2SAT: every clause has at most 2 literals
P-solvable

3SAT: every clause has at most 3 literals
NP-hard

SAT solvers

SAT problems have many important applications

many SAT solvers are ready for use

DPLL

WalkSAT

DPLL

Davis–Putnam–Logemann–Loveland algorithm

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

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ 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*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
      DPLL($clauses, rest, model \cup \{P{=}false\}$))

a deep-first search with heuristics

DPLL heuristics

Pure symbol heuristic: A **pure symbol** is a symbol that always appears with the same "sign" in all clauses.

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

A and B is pure, but not C

Unit clause heuristic: A **unit clause** is a clause with just one literal.

$$(A \vee \neg B) \text{ with } A = \text{true}$$

is a unit clause

Other tricks

Component analysis : find disjoint subsets

Variable and value ordering : assign most frequent variable at first

Intelligent backtracking : remember conflicts

Random restart

Clever indexing

WalkSAT

a local search hill-climbing or others.

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, typically around 0.5
    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*

failure ≠ unsatisfiable

The landscape of random SAT problems

Not all SAT instances are hard
under-constraint: a few clauses => easy to enumerate
over-constraint: too many clauses => unsatisfiable

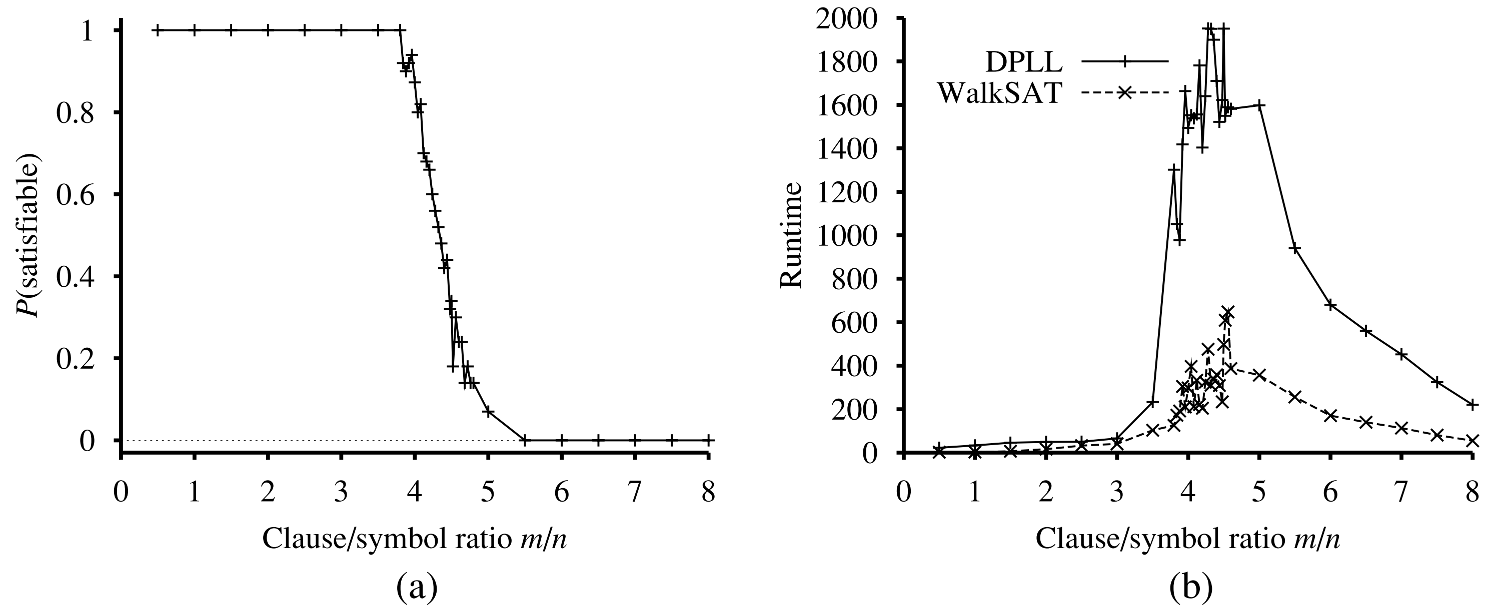


Figure 7.19 (a) Graph showing the probability that a random 3-CNF sentence with $n = 50$ symbols is satisfiable, as a function of the clause/symbol ratio m/n. (b) Graph of the median run time (measured in number of recursive calls to DPLL, a good proxy) on random 3-CNF sentences. The most difficult problems have a clause/symbol ratio of about 4.3.

Planning

Language

There are many languages description the world
Planning Domain Definition Language
1.2, 2.1, 2.2, 3.0, 3.1

state s
Action(s)
Result(s,a)

$$
\begin{aligned}
&Action(Fly(p, from, to),\\
&\quad \text{PRECOND:} At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)\\
&\quad \text{EFFECT:} \neg At(p, from) \wedge At(p, to))
\end{aligned}
$$

$$
\begin{aligned}
&Action(Fly(P_1, SFO, JFK),\\
&\quad \text{PRECOND:} At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)\\
&\quad \text{EFFECT:} \neg At(P_1, SFO) \wedge At(P_1, JFK))
\end{aligned}
$$

Precondition

action a is **applicable** in state s if the preconditions are satisfied by s

$$(a \in \textsc{Actions}(s)) \;\Leftrightarrow\; s \models \textsc{Precond}(a)$$

$$\forall\, p, \mathit{from}, \mathit{to}\;\; (\mathit{Fly}(p, \mathit{from}, \mathit{to}) \in \textsc{Actions}(s)) \;\Leftrightarrow$$
$$s \models (\mathit{At}(p, \mathit{from}) \land \mathit{Plane}(p) \land \mathit{Airport}(\mathit{from}) \land \mathit{Airport}(\mathit{to}))$$

Result

removing the fluents that appear as negative literals in the action's effects (what we call the **delete list** or DEL(a)), and adding the fluents that are positive literals in the action's effects (what we call the **add list** or ADD(a))

$$\textsc{Result}(s, a) = (s - \textsc{Del}(a)) \cup \textsc{Add}(a) \ .$$

$$\begin{aligned}
&Action(Fly(P_1, SFO, JFK), \\
&\quad \textsc{Precond}{:}At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK) \\
&\quad \textsc{Effect}{:}\neg At(P_1, SFO) \land At(P_1, JFK))
\end{aligned}$$

Example

$$
\begin{aligned}
& \mathit{Init}(\mathit{On}(A, \mathit{Table}) \land \mathit{On}(B, \mathit{Table}) \land \mathit{On}(C, A) \\
& \quad \land \mathit{Block}(A) \land \mathit{Block}(B) \land \mathit{Block}(C) \land \mathit{Clear}(B) \land \mathit{Clear}(C)) \\
& \mathit{Goal}(\mathit{On}(A, B) \land \mathit{On}(B, C)) \\
& \mathit{Action}(\mathit{Move}(b, x, y), \\
& \quad \textsc{Precond}: \mathit{On}(b, x) \land \mathit{Clear}(b) \land \mathit{Clear}(y) \land \mathit{Block}(b) \land \mathit{Block}(y) \land \\
& \qquad\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
& \quad \textsc{Effect}: \mathit{On}(b, y) \land \mathit{Clear}(x) \land \neg \mathit{On}(b, x) \land \neg \mathit{Clear}(y)) \\
& \mathit{Action}(\mathit{MoveToTable}(b, x), \\
& \quad \textsc{Precond}: \mathit{On}(b, x) \land \mathit{Clear}(b) \land \mathit{Block}(b) \land (b \neq x), \\
& \quad \textsc{Effect}: \mathit{On}(b, \mathit{Table}) \land \mathit{Clear}(x) \land \neg \mathit{On}(b, x))
\end{aligned}
$$

Figure 10.3 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[\mathit{MoveToTable}(C, A), \mathit{Move}(B, \mathit{Table}, C), \mathit{Move}(A, \mathit{Table}, B)]$.

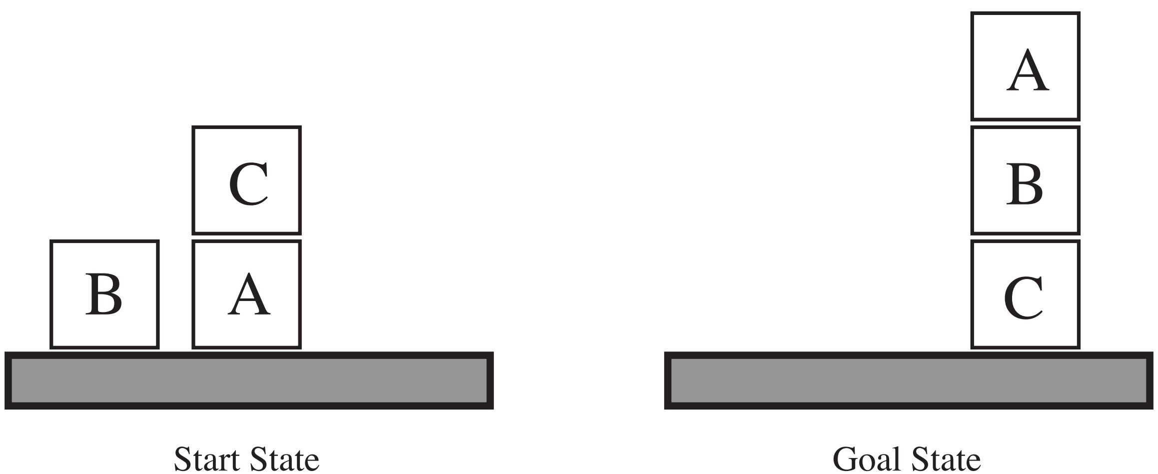
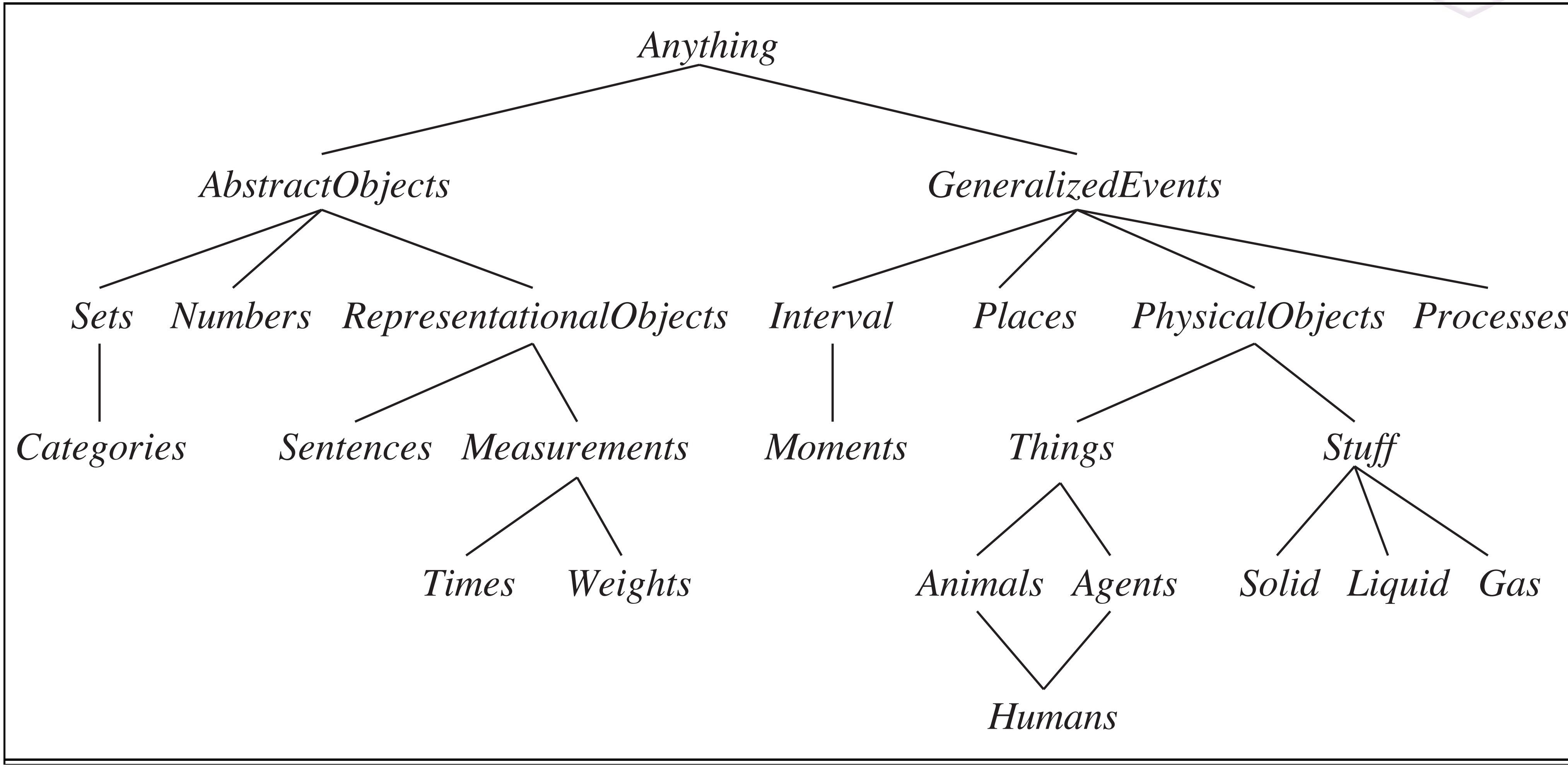


Figure 10.4 Diagram of the blocks-world problem in Figure 10.3.

Ontology

Up ontology


Anything
AbstractObjects
GeneralizedEvents
Sets
Numbers
RepresentationalObjects
Interval
Places
PhysicalObjects
Processes
Categories
Sentences
Measurements
Moments
Things
Stuff
Times
Weights
Animals
Agents
Solid
Liquid
Gas
Humans

Domain ontology

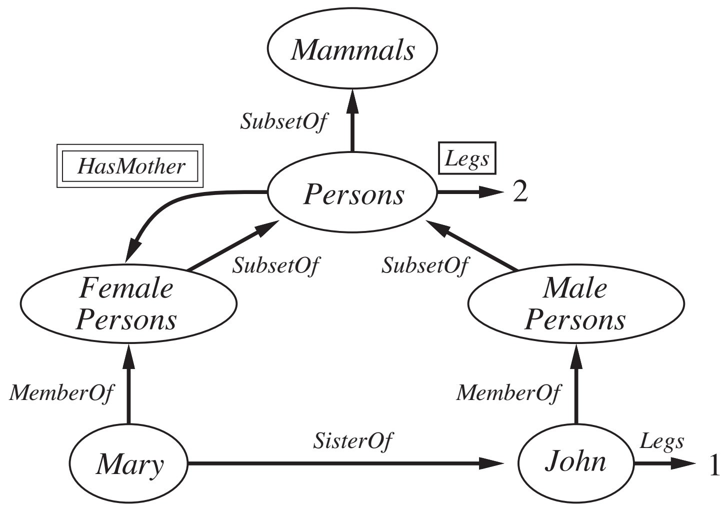


Figure 12.5 A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

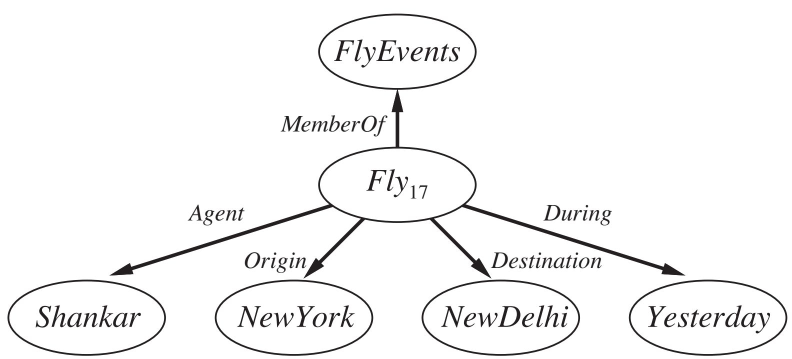


Figure 12.6 A fragment of a semantic network showing the representation of the logical assertion $Fly(Shankar, NewYork, NewDelhi, Yesterday)$.

Example: Wordnet

Hamburger

- Hamburger (an inhabitant of Hamburg)
  - direct hypernym:
    - German (a person of German nationality)
  - sister term
    - German (a person of German nationality)
      - East German (a native/inhabitant of the former GDR)
      - Bavarian (a native/inhabitant of Bavaria)
  - derivationally related form
    - Hamburg (a port city in northern Germany on the Elbe River that was founded by Chalemagne in the…)

[from wikipedia]

Example application