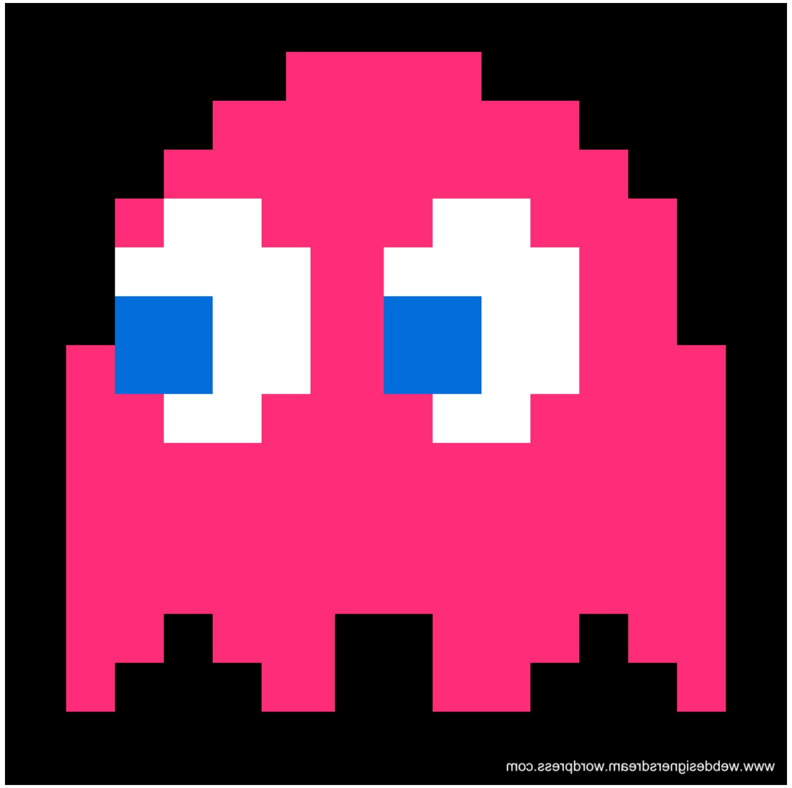
CS 188: Artificial Intelligence

Propositional Logic: Semantics, Inference, Agents



Instructor: Sergey Levine and Stuart Russell

University of California, Berkeley

You can think about deep learning as equivalent to ... our visual cortex or auditory cortex. But, of course, true intelligence is a lot more than just that, you have to recombine it into higher-level thinking and symbolic reasoning, a lot of the things classical AI tried to deal with in the 80s. … We would like to build up to this symbolic level of reasoning — maths, language, and logic. So that’s a big part of our work.

Demis Hassabis, CEO of Google Deepmind

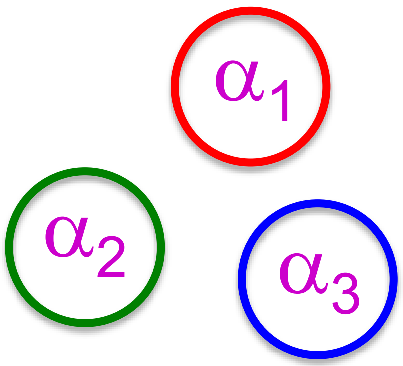
Knowledge

- 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 (or have it ***Learn*** the knowledge)
  - 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
- A single inference algorithm can answer any answerable question
  - Cf. a search algorithm answers only "how to get from A to B" questions

Knowledge base	Domain-specific facts
Inference engine	Generic code

Logic

- ***Syntax***: What sentences are allowed?
- ***Semantics***:
  - What are the ***possible worlds***?
  - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)



Syntaxland

Semanticsland

Examples

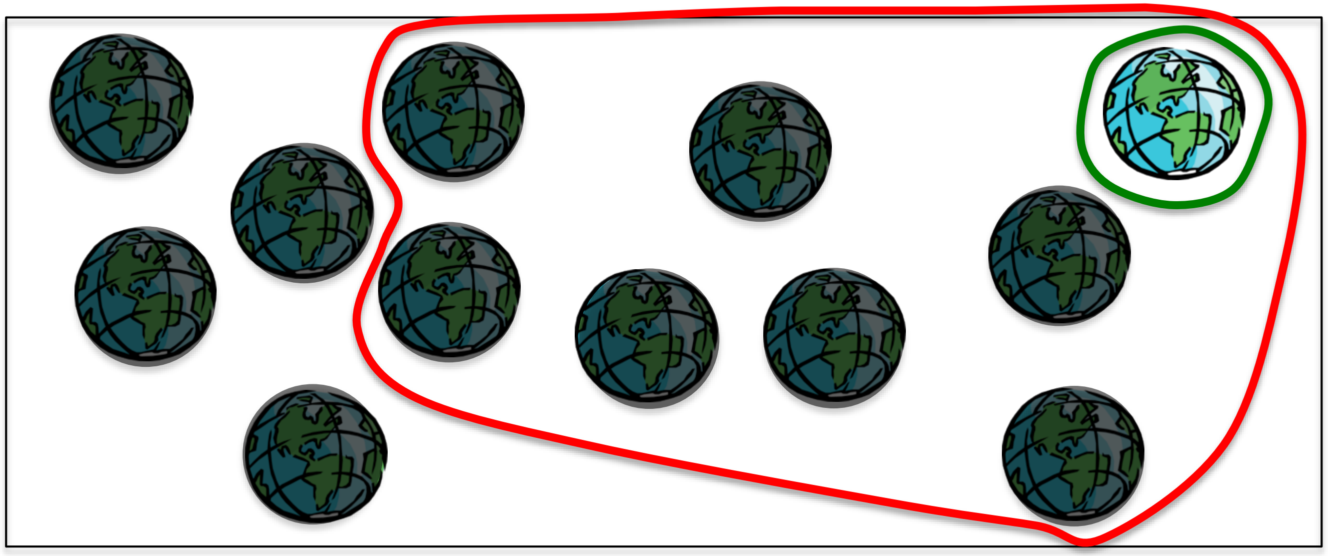
- Propositional logic
  - Syntax: $P \vee (\neg Q \wedge R)$; $X_1 \Leftrightarrow (\text{Raining} \Rightarrow \text{Sunny})$
  - Possible world: {P=true,Q=true,R=false,S=true} or 1101
  - Semantics: $\alpha \wedge \beta$ is true in a world iff α is true and β is true (etc.)
- First-order logic
  - Syntax: $\forall x\ \exists y\ P(x,y) \wedge \neg Q(Joe,f(x)) \Rightarrow f(x)=f(y)$
  - Possible world: Objects o_1, o_2, o_3; P holds for $<o_1,o_2>$; Q holds for $< o_1, o_3>$; $f(o_1)=o_1$; $Joe=o_3$; etc.
  - Semantics: $\phi(\sigma)$ is true in a world if $\sigma=o_j$ and ϕ holds for o_j; etc.

Inference: entailment

- ***Entailment***: $\alpha \models \beta$ ("α entails β" or "β follows from α") iff in every world where α is true, β is also true
  - I.e., the α-worlds are a subset of the β-worlds [***models***(α) $\subseteq$ ***models***(β)]
- In the example, $\alpha_2 \models \alpha_1$
- (Say α_2 is $\neg Q \wedge R \wedge S \wedge W$
  α_1 is $\neg Q$)

α_1

α_2



Inference: proofs

- A proof is a ***demonstration*** of entailment between α and β
- Method 1: ***model-checking***
  - For every possible world, if α is true make sure that is β true too
  - OK for propositional logic (finitely many worlds); not easy for first-order logic
- Method 2: ***theorem-proving***
  - Search for a sequence of proof steps (applications of ***inference rules***) leading from α to β
  - E.g., from $P \wedge (P \Rightarrow Q)$, infer Q by ***Modus Ponens***
- ***Sound*** algorithm: everything it claims to prove is in fact entailed
- ***Complete*** algorithm: every that is entailed can be proved

Quiz

- What's the connection between complete inference algorithms and complete search algorithms?
- Answer 1: they both have the words "complete…algorithm"
- Answer 2: they both solve any solvable problem
- Answer 3: Formulate inference as a search problem
  - Initial state: KB contains α
  - Actions: apply any inference rule that matches KB, add conclusion
  - Goal test: KB contains β

Hence any complete search algorithm (BFS, IDS, …) yields a complete inference algorithm…

provided the inference rules themselves are strong enough

Propositional logic syntax: The gruesome details

- Given: a set of proposition symbols $\{X_1, X_2, \ldots, X_n\}$
  - (we often add True and False for convenience)
- X_i is a sentence
- If α is a sentence then $\neg\alpha$ is a sentence
- If α and β are sentences then $\alpha \wedge \beta$ is a sentence
- If α and β are sentences then $\alpha \vee \beta$ is a sentence
- If α and β are sentences then $\alpha \Rightarrow \beta$ is a sentence
- If α and β are sentences then $\alpha \Leftrightarrow \beta$ is a sentence
- And p.s. there are no other sentences!

Propositional logic semantics: The unvarnished truth

function PL-TRUE?(α,model) **returns** true or false

if α is a symbol **then return** Lookup(α, model)

if Op(α) = $\neg$ **then return** not(PL-TRUE?(Arg1(α),model))

if Op(α) = $\wedge$ **then return** and(PL-TRUE?(Arg1(α),model),
PL-TRUE?(Arg2(α),model))

if Op(α) = $\Rightarrow$ **then return** or(PL-TRUE?(Arg1(α),model),
not(PL-TRUE?(Arg2(α),model)))

etc. (Sometimes called "recursion over syntax")

PacMan facts

- If Pacman is at 3,3 at time 16 and goes North and there is no wall at 3,4 then Pacman is at 3,4 at time 17:
  - $At\_3,3\_16 \wedge N\_16 \wedge \neg Wall\_3,4 \Rightarrow At\_3,3\_17$
- At time 0 Pacman does one of four actions:
  - $(W\_0 \vee E\_0 \vee N\_0 \vee S\_0)$
  - $\neg(W\_0 \wedge E\_0) \wedge \neg(W\_0 \wedge S\_0) \wedge \ldots$

Simple theorem proving: Forward chaining

- Forward chaining applies Modus Ponens to generate new facts:
  - Given $X_1 \wedge X_2 \wedge \ldots X_n \Rightarrow Y$ and $X_1, X_2, \ldots, X_n$
  - Infer Y
- Forward chaining keeps applying this rule, adding new facts, until nothing more can be added
- Requires KB to contain only ***definite clauses***:
  - (Conjunction of symbols) $\Rightarrow$ symbol; or
  - A single symbol (note that X is equivalent to $True \Rightarrow X$)

Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
   count ← a table, where count[c] is the number of symbols in c's premise
   inferred ← a table, where inferred[s] is initially false for all s
   agenda ← a queue of symbols, initially symbols known to be true in KB
   while agenda is not empty do
         p ← Pop(agenda)
         if p = q then return true
         if inferred[p] = false then
               inferred[p]←true
               for each clause c in KB where p is in c.premise do
                     decrement count[c]
                     if count[c] = 0 then add c.conclusion to agenda
   return false
```

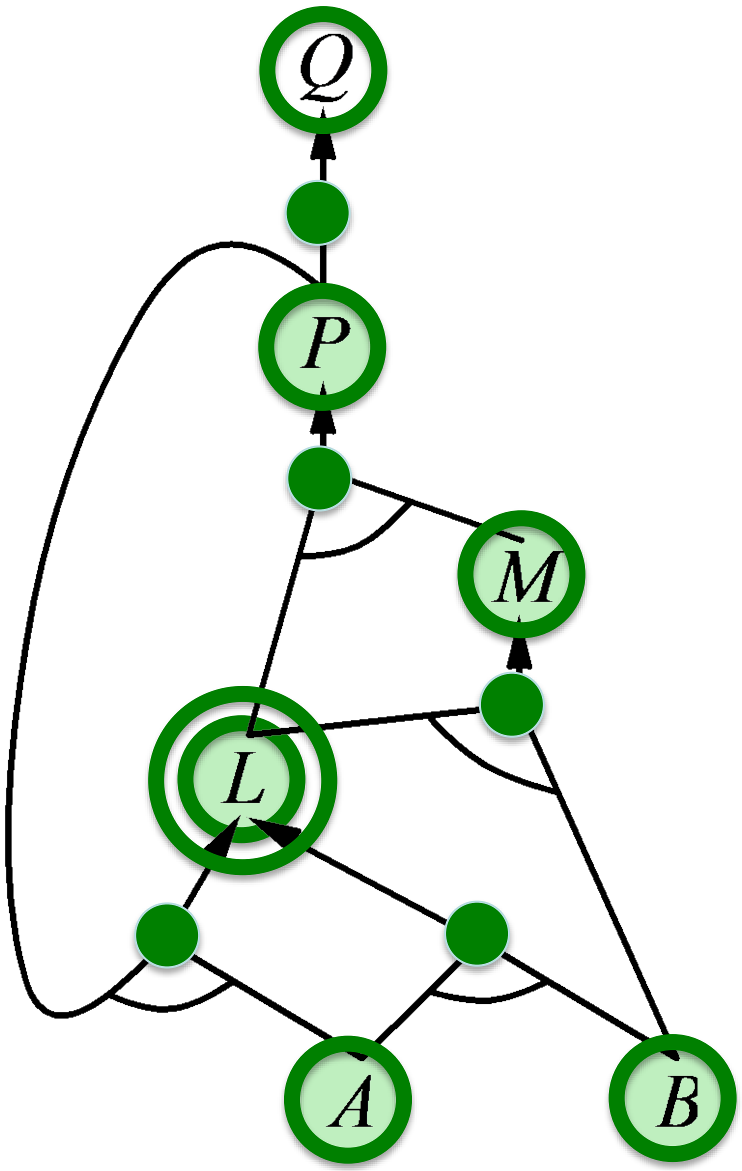
Forward chaining example: Proving Q

CLAUSES	COUNT
$P \Rightarrow Q$	$\cancel{1}$ 0
$L \wedge M \Rightarrow P$	$\cancel{2}$ $\cancel{1}$ 0
$B \wedge L \Rightarrow M$	$\cancel{2}$ $\cancel{1}$ 0
$A \wedge P \Rightarrow L$	$\cancel{2}$ $\cancel{1}$ 0
$A \wedge B \Rightarrow L$	$\cancel{2}$ $\cancel{1}$ 0
A	0
B	0

INFERRED	
A	~~false~~ true
B	~~false~~ true
L	~~false~~ true
M	~~false~~ true
P	~~false~~ true
Q	~~false~~ true

AGENDA

~~A~~ ~~B~~ ~~L~~ ~~M~~ ~~P~~ ~~L~~ ~~Q~~



Properties of forward chaining

- Theorem: FC is sound and complete for definite-clause KBs
- Soundness: follows from soundness of Modus Ponens (easy to check)
- Completeness proof:

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final *inferred* table as a model ***m***, assigning true/false to symbols
3. Every clause in the original KB is true in ***m***

   Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in ***m***
   Then $a_1 \wedge \ldots \wedge a_k$ is true in ***m*** and b is false in ***m***
   Therefore the algorithm has not reached a fixed point!

4. Hence ***m*** is a model of KB
5. If KB |= q, q is true in every model of KB, including ***m***

A ~~false~~ true
B ~~false~~ true
L ~~false~~ true
M ~~false~~ true
P ~~false~~ true
Q ~~false~~ true

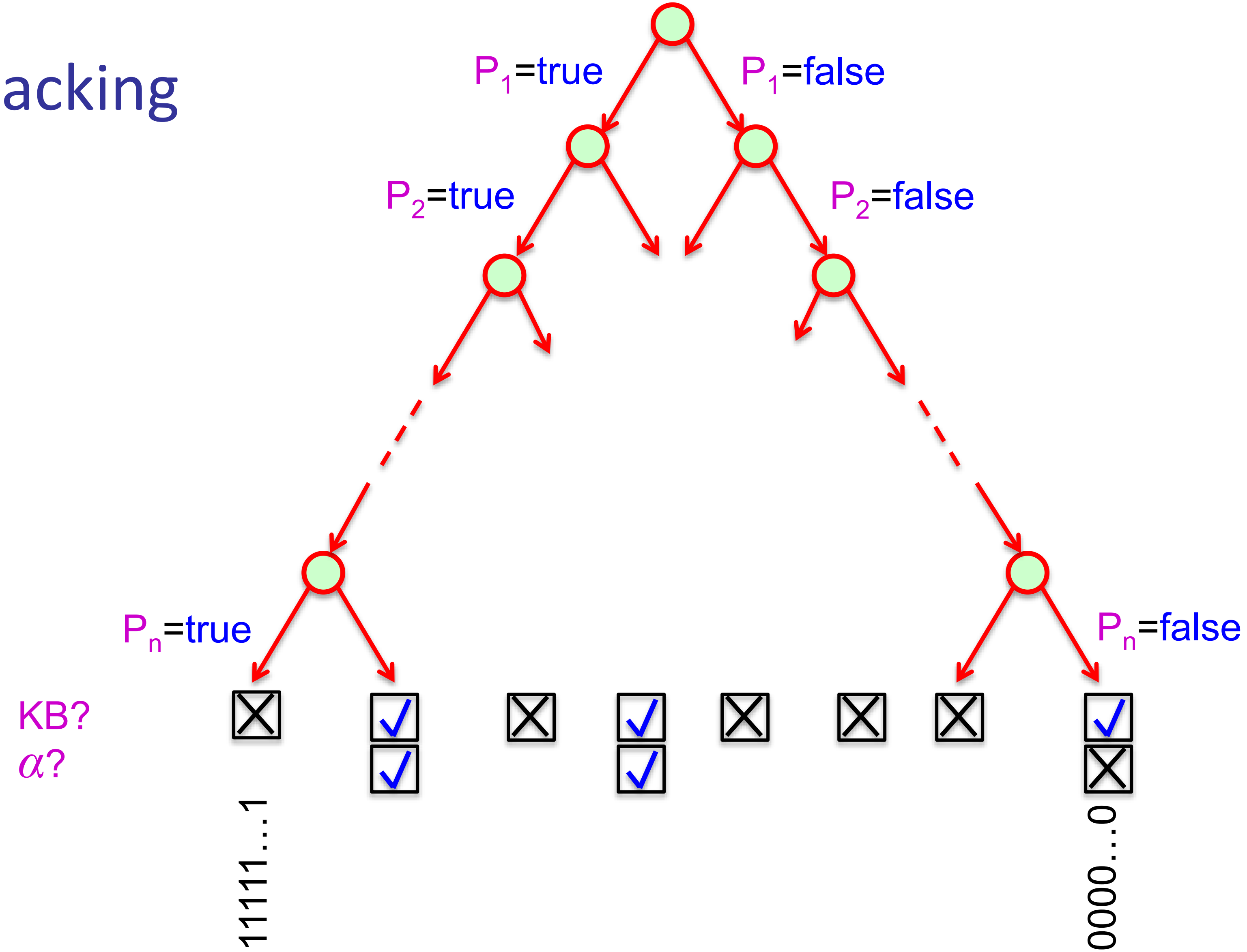
Simple model checking

```
function TT-ENTAILS?(KB, α) returns true or false
   return TT-CHECK-ALL(KB,α,symbols(KB) U symbols(α),{})

function TT-CHECK-ALL(KB,α,symbols,model) returns true or false
   if empty?(symbols) then
        if PL-TRUE?(KB,model) then return PL-TRUE?(α,model)
        else return true
   else
        P ← first(symbols)
        rest ← rest(symbols)
        return  and (TT-CHECK-ALL(KB,α,rest,model ∪ {P = true})
                     TT-CHECK-ALL(KB,α,rest,model ∪ {P = false }))
```

Simple model checking, contd.

- Same recursion as backtracking
- $O(2^n)$ time, linear space
- We can do much better!



Satisfiability and entailment

- A sentence is ***satisfiable*** if it is true in at least one world (cf CSPs!)
- Suppose we have a hyper-efficient SAT solver; how can we use it to test entailment?
  - Suppose $\alpha \models \beta$
  - Then $\alpha \Rightarrow \beta$ is true in all worlds
  - Hence $\neg(\alpha \Rightarrow \beta)$ is false in all worlds
  - Hence $\alpha \wedge \neg\beta$ is false in all worlds, i.e., unsatisfiable
- So, add the negated conclusion to what you know, test for (un)satisfiability; also known as `reductio ad absurdum`
- Efficient SAT solvers operate on ***conjunctive normal form***

Conjunctive normal form (CNF)

- Every sentence can be expr … *clauses*
- Each clause is a ***disjunction*** o …
- Each literal is a symbol or a … ga … symb …
- Conversion to CNF by a se … qu … ce of standard … sformations:
  - $At\_1,1\_0 \Rightarrow (Wall\_0,1 \Leftrightarrow Blocked\_W\_0)$
  - $At\_1,1\_0 \Rightarrow ((Wall\_0,1 \Rightarrow Blocked\_W\_0) \wedge (Blocked\_W\_0 \Rightarrow Wall\_0,1))$
  - $\neg At\_1,1\_0 \vee ((\neg Wall\_0,1 \vee Blocked\_W\_0) \wedge (\neg Blocked\_W\_0 \vee Wall\_0,1))$
  - $(\neg At\_1,1\_0 \ \vee \ \neg Wall\_0,1 \ \vee \ Blocked\_W\_0) \wedge$
    $(\neg At\_1,1\_0 \ \vee \ \neg Blocked\_W\_0 \ \vee \ Wall\_0,1)$

Replace biconditional by two implications

Replace $\alpha \Rightarrow \beta$ by $\neg\alpha \vee \beta$

Distribute $\vee$ over $\wedge$

Efficient SAT solvers

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern solvers
- Essentially a backtracking search over models with some extras:
  - ***Early termination***: stop if
    - all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=true}
    - any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=false,B=false}
  - ***Pure literals***: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true
  - ***Unit clauses***: if a clause is left with a single literal, set symbol to satisfy clause
    - E.g., if A=false, $(A \vee B) \wedge (A \vee \neg C)$ becomes $(\text{false} \vee B) \wedge (\text{false} \vee \neg C)$, i.e. $(B) \wedge (\neg C)$
    - Satisfying the unit clauses often leads to further propagation, new unit clauses, etc.

DPLL algorithm

```
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 ←FIND-PURE-SYMBOL(symbols,clauses,model)
  if P is non-null then return DPLL(clauses, symbols–P, modelU{P=value})
  P,value ←FIND-UNIT-CLAUSE(clauses,model)
  if P is non-null then return DPLL(clauses, symbols–P, modelU{P=value})
  P ← First(symbols); rest ← Rest(symbols)
  return or(DPLL(clauses,rest,modelU{P=true}),
            DPLL(clauses,rest,modelU{P=false}))
```

Efficiency

- Naïve implementation of DPLL: solve ~100 variables
- Extras:
  - Variable and value ordering (from CSPs)
  - Divide and conquer
  - Caching unsolvable subcases as extra clauses to avoid redoing them
  - Cool indexing and incremental recomputation tricks so that every step of the DPLL algorithm is efficient (typially O(1))
    - Index of clauses in which each variable appears +ve/-ve
    - Keep track number of satisfied clauses, update when variables assigned
    - Keep track of number of remaining literals in each clause
- Real implementation of DPLL: solve ~10000000 variables

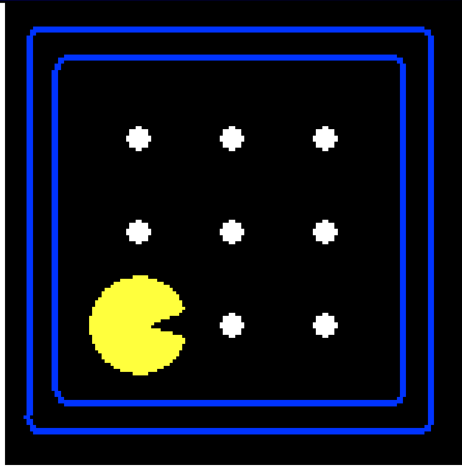
SAT solvers in practice

- Circuit verification: does this VLSI circuit compute the right answer?
- Software verification: does this program compute the right answer?
- Software synthesis: what program computes the right answer?
- Protocol verification: can this security protocol be broken?
- Protocol synthesis: what protocol is secure for this task?
- Planning: ***how can I eat all the dots???***

A knowledge-based agent

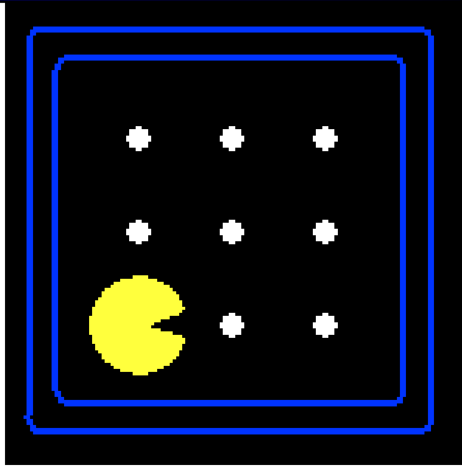
```
function KB-AGENT(percept) returns an action
persistent: KB, a knowledge base
                    t, an integer, initially 0
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t←t+1
    return action
```

Example: Partially observable Pacman



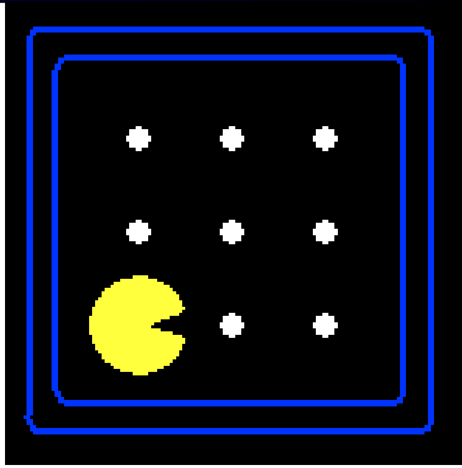
- Pacman has to act given only local perception
  - Four Boolean percept variables for wall in each direction
- What knowledge does he need to begin with?
  - ***Sensor model***: sentences stating how the current percept variables are determined by the current state variables
  - ***Transition model***: sentences stating how the next-state variables are determined by the current state variables and Pacman's action
  - ***Initial conditions***: what Pacman knows about the initial state
  - ***Domain constraints***: what is generally true, e.g., Pacman can do one thing at a time and be in one place at a time

Pacman variables



- Pacman's location
  - At_1,1_0 (Pacman is at [1,1] at time 0)   At_3,3_4 etc
- Wall locations (these do not change with time)
  - Wall_0,0   Wall_0,1 etc
- Percepts
  - Blocked_W_0 (blocked by wall to my West at time 0) etc.
- Actions
  - W_0 (Pacman moves West at time 0), E_0 etc.
- NxN world for T time steps => $N^2T + N^2 + 4T + 4T = O(N^2T)$ variables
- 2^{N^2T} possible worlds! N=10, T=100 => 10^{3010} worlds (each a "history")

Sensor model



- State facts about how Pacman's percepts arise...
- Pacman perceives a wall to the West at time t
  if and only if he is in x,y and there is a wall at x-1,y ....
  - Blocked_W_0 $\Leftrightarrow$

    ((At_1,1_0 $\wedge$ Wall_0,1) v

    (At_1,2_0 $\wedge$ Wall_0,2) v

    (At_1,3_0 $\wedge$ Wall_0,3) v ....)

How many of these sentences?

How big are they?

Quiz

- What is wrong with sentences like
  - $At\_1,1\_0 \wedge Wall\_0,1 \Rightarrow Blocked\_W\_0$
    - If you are at [1,1] at time 0 and there is a wall in [0,1], the west percept is blocked
- True but ***incomplete***!
  - They say "under these conditions the percept variable is true"
  - They don't say when it is false
  - In particular, they allow for worlds where the percept is always true!!
    - ***Unintended or non-standard models***

Transition model

- How does each ***state variable*** or ***fluent*** at each time gets its value?
- State variables for POPacman are At_x,y_t , e.g., At_3,3_17
- A state variable gets its value according to a ***successor-state axiom***
  - $X_t \Leftrightarrow [X_{t-1} \wedge \neg(\text{some action}_{t-1} \text{ made it false})] \vee$
    $[\neg X_{t-1} \wedge (\text{some action}_{t-1} \text{ made it true})]$
- For Pacman location:
  - At_3,3_17 ⇔ [At_3,3_16 ∧ ¬((¬Wall_3,4 ∧ N_16) v (¬Wall_4,3 ∧ E_16) v …)]
    v [¬At_3,3_16 ∧ ((At_3,2_16 ∧ ¬Wall_3,3 ∧ N_16) v
    (At_2,3_16 ∧ ¬Wall_3,3 ∧ N_16) v …)]

Initial state

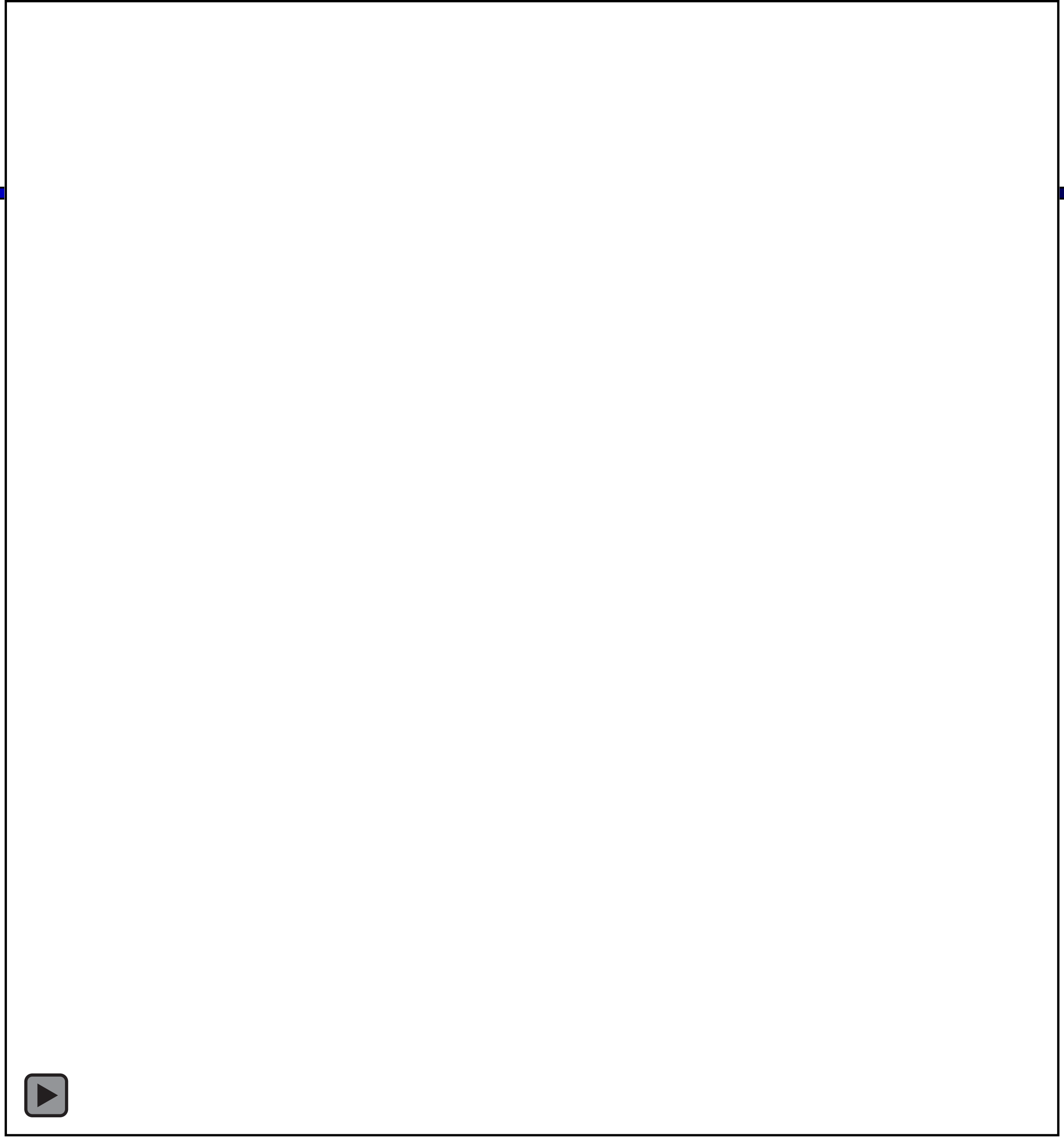
- Pacman may know its initial location:
  - At_1,1_0 ∧ ¬At_1,2_0 ∧ ¬At_1,3_0 ….
- Or, it may not:
  - At_1,1_0 v At_1,2_0 v At_1,3_0 v … v At_3,3_0
- We also need a ***domain constraint*** – exactly one thing at a time
  - ¬(W_0 ∧ E_0) ∧ ¬(W_0 ∧ S_0) ∧ …
  - ¬(W_1 ∧ E_1) ∧ ¬(W_1 ∧ S_1) ∧ …
  - … ∧ (W_0 v E_0 v N_0 v S_0) ∧ …

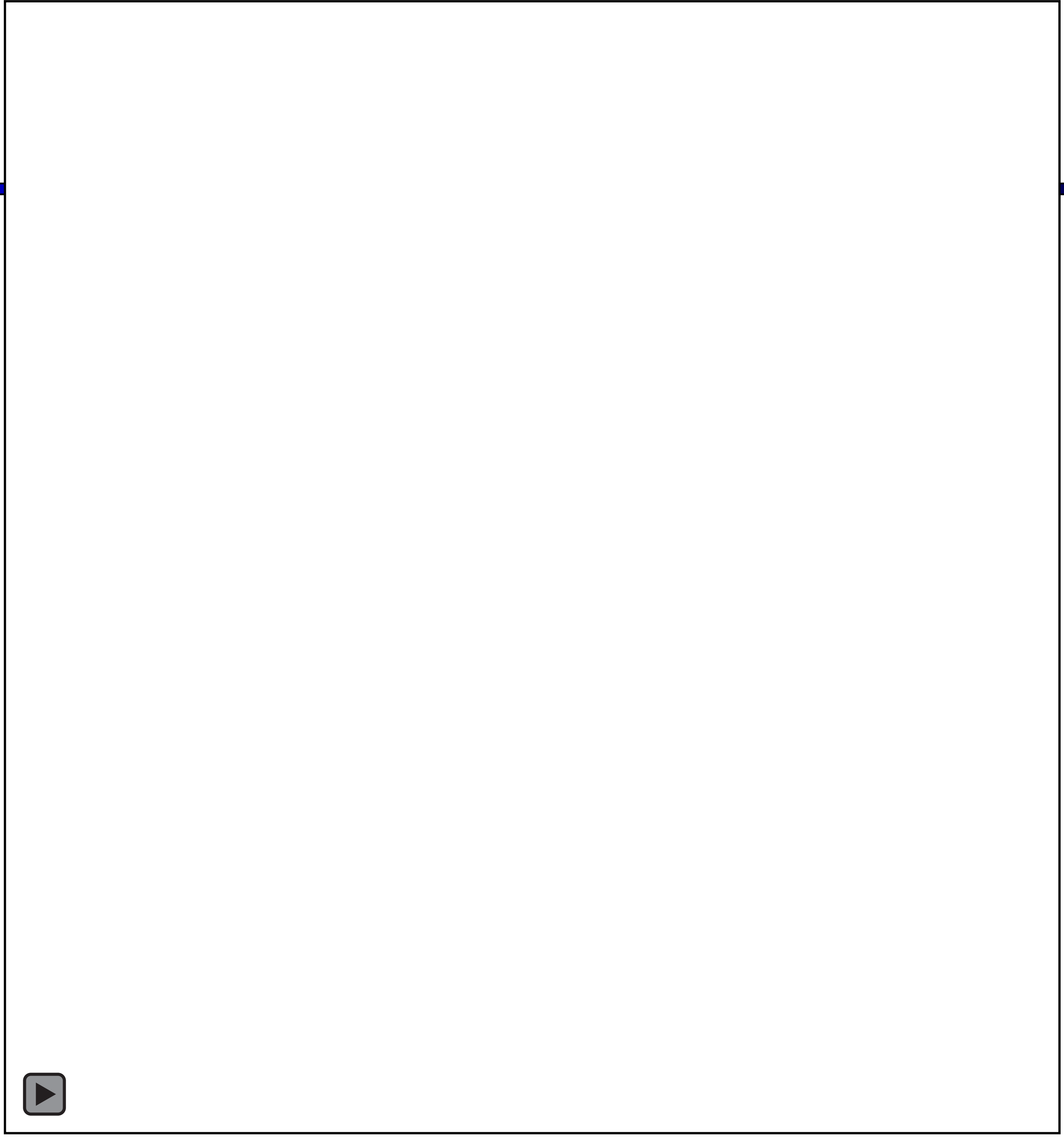
State estimation

- ***State estimation*** means keeping track of what's true now
- A logical agent can just ask itself!
  - E.g., ask whether KB $\wedge$ <actions> $\wedge$ <percepts> |= Wall_2,2
  - This is "lazy": it involves reasoning about one's whole life history at each step!
- A more "eager" form of state estimation:
  - After each action and percept
    - For each state variable X_t
      - If X_t is entailed, add to KB
      - If $\neg X_t$ is entailed, add to KB

Planning as satisfiability

- Given a hyper-efficient SAT solver, can we use it to make plans?
- Yes, for fully observable, deterministic case:
  - planning problem is solvable iff there is some satisfying assignment
  - solution obtained from truth values of action variables
- For T = 1 to infinity, set up the KB as follows and run SAT solver:
  - Initial state, domain constraints
  - Transition model sentences up to time T
  - Goal is true at time T
- Read off action variables from solution





Summary

- One possible agent architecture: knowledge + inference
- Logics provide a formal way to encode knowledge
  - A logic is defined by: syntax, set of possible worlds, truth condition
- Logical inference computes entailment relations among sentences
- SAT solvers based on DPLL provide incredibly efficient inference
- Logical agents can construct plans by asking whether there is a future in which the goal is achieved