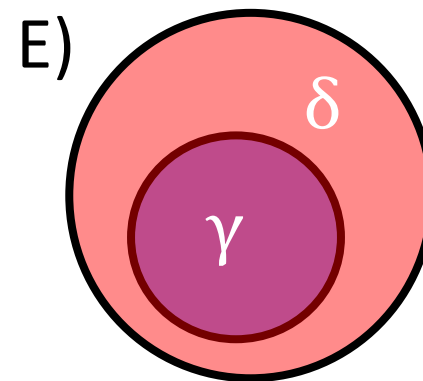
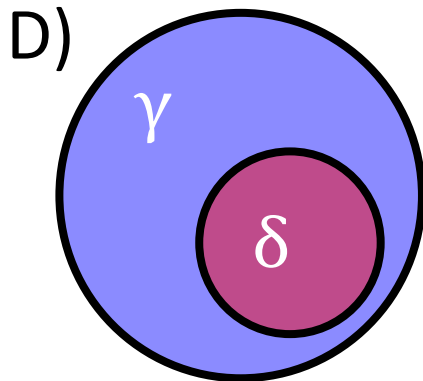
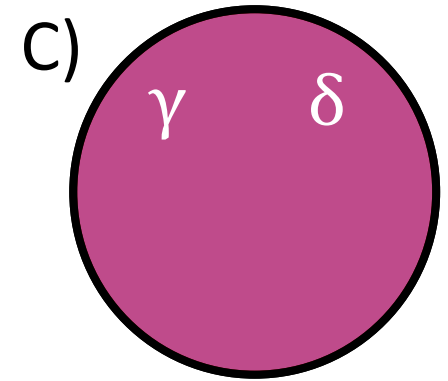
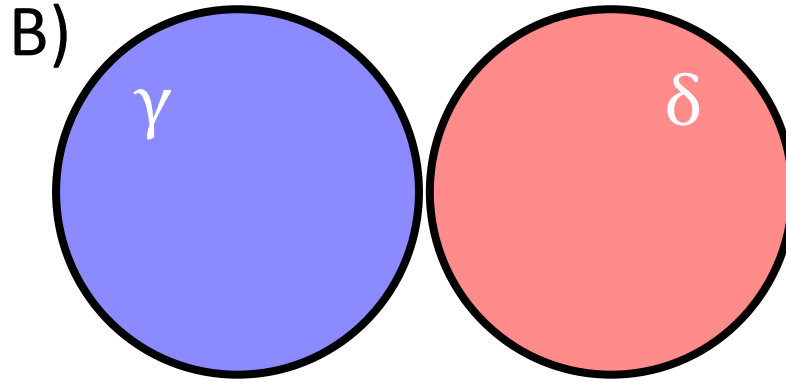
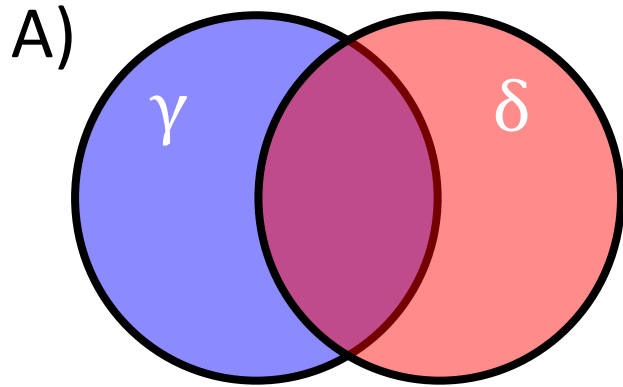


Warm-up

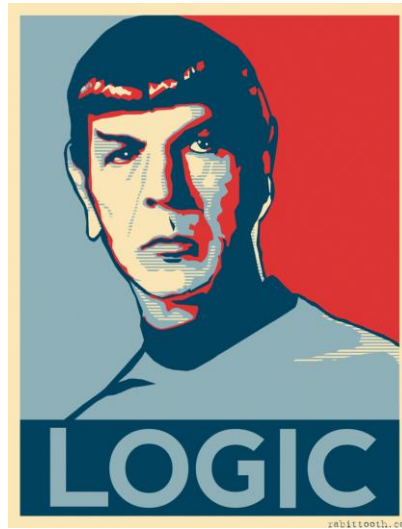
- The regions below visually enclose the set of models that satisfy the respective sentence γ or δ . For which of the following diagrams does γ entail δ . Select all that apply.



AI: Representation and Problem Solving

Boolean Satisfiability Problem (SAT)

& Logical Agents



Instructors: Fei Fang & Pat Virtue

Slide credits: CMU AI, <http://ai.berkeley.edu>

Learning Objectives

- Describe the definition of (Boolean) Satisfiability Problem (SAT)
- Describe the definition of Conjunctive Normal Form (CNF)
- Describe the following algorithms for solving SAT
 - DPLL, CDCL, WalkSAT, GSAT
- Determine whether a sentence is satisfiable
- Describe Successor-State Axiom
- Describe and implement SATPlan (Planning as Satisfiability)
- (Hybrid Agent)

Logical Agent Vocab: Recap

- Symbol: Variable that can be true or false
- Model: Complete assignment of symbols to True/False
- Operators: $\neg A$ (not), $A \wedge B$ (conjunction), $A \vee B$ (disjunction), $A \Rightarrow B$ (implication), $A \Leftrightarrow B$ (biconditional)
- Sentence: A logical statement composed of logic symbols and operators
- KB: Collection of sentences representing facts and rules we know about the world
- Query: Sentence we want to know if it is *provably* True, *provably* False, or *unsure*.

Logical Agent Vocab: Recap

- Entail
 - Does `sentence1 entail sentence2`?
 - Input: `sentence1, sentence2`
 - Output: True if each model that satisfies `sentence1` must also satisfy `sentence2`; False otherwise
 - "If I know 1 holds, then I know 2 holds"
- Satisfy
 - Does `model satisfy sentence`?
 - Input: `model, sentence`
 - Output: True if this `sentence` is true in this `model`; False otherwise
 - "Does this particular state of the world work?"

(Boolean) Satisfiability Problem (SAT)

- Satisfiable
 - Is **sentence** **satisfiable**?
 - Input: **sentence**
 - Output: True if at least one model satisfies **sentence**
 - "Is it possible to make this **sentence** true?"
- SAT problem is the problem of determining the satisfiability of a sentence
 - SAT is a typical problem for logical agents
 - SAT is the first problem proved to be NP-complete
 - If satisfiable, we often want to know what that model is

SAT and Entailment

- A sentence is *satisfiable* if it is true in **at least one** world
- 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
- More generally, to prove a sentence is valid (i.e., true in all models), introduce the negated claim and test for unsatisfiability; also known as *reductio ad absurdum* (*reduction to absurdity*)

SAT and CSPs

- SAT problems are essentially CSPs with Boolean variables
 - Can apply backtracking based algorithms
 - Can apply local search algorithms
- Naïve way to solve SAT: Truth table enumeration
- Efficient SAT solvers operate on *conjunctive normal form*
 - Often based on backtracking and local search

Propositional Logical Vocab: Recap

- Literal
 - Atomic sentence: True , False , Symbol , $\neg\text{Symbol}$
- Clause
 - Disjunction of literals: $A \vee B \vee \neg C$
- Definite clause
 - Disjunction of literals, *exactly one* is positive
 - $\neg A \vee B \vee \neg C$
- Horn clause
 - Disjunction of literals, *at most one* is positive
 - All definite clauses are Horn clauses

Conjunctive Normal Form (CNF)

- Every sentence can be expressed as a **conjunction** of **clauses**
- Each clause is a **disjunction** of **literals**
- Each literal is a symbol or a negated symbol
- We can convert a sentence to CNF through a sequence of standard transformations

Conjunctive Normal Form (CNF)

- Original sentence:
 - $A \Rightarrow (B \Leftrightarrow C)$
- Biconditional Elimination: Replace biconditional by two implications
 - $A \Rightarrow ((B \Rightarrow C) \wedge (C \Rightarrow B))$
- Implication Elimination: Replace $\alpha \Rightarrow \beta$ by $\neg\alpha \vee \beta$
 - $\neg A \vee ((\neg B \vee C) \wedge (\neg C \vee B))$
- Distribution: Distribute $\vee$ over $\wedge$, i.e., replace $\alpha \vee (\beta \wedge \gamma)$ by $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
 - $(\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg C \vee B)$

Conjunctive Normal Form (CNF)

- Original sentence:
 - $(\neg(A \vee B) \vee C) \wedge (\neg C \wedge A)$
- De Morgan's Law: Replace $\neg(\alpha \vee \beta)$ by $\neg\alpha \wedge \neg\beta$, and $\neg(\alpha \wedge \beta)$ by $\neg\alpha \vee \neg\beta$
 - $((\neg A \wedge \neg B) \vee C) \wedge (\neg C \wedge A)$
- Distribution: Distribute $\vee$ over $\wedge$, i.e., replace $\alpha \vee (\beta \wedge \gamma)$ by $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
 - $(\neg A \vee C) \wedge (\neg B \vee C) \wedge (\neg C \wedge A)$

Other Logical Equivalences

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:

- *Early termination*: stop if

- all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by $\{A=\text{true}\}$

SAT solver can stop with partial models; no need to assign all variables (can assign arbitrarily if a complete model is needed).

- any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is falsified by $\{A=\text{false}, B=\text{false}\}$

Stop when a conflict is found. Similar to backtracking algorithm for general CSPs.

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:
 - *Early termination*
 - *Pure symbols*: 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**

Claim: If a sentence has a model to satisfy it, then it has a model in which the pure symbols are assigned values that make their literals true. Why?

W.l.o.g., assume symbol A shows up in all clauses as A . Assume there is a model satisfies the sentence with A =false. Then construct a new model with A =true and everything else the same. Since there are no opposite sign literals, making A =true that could make any clause be false.

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:
 - *Early termination*
 - *Pure symbols*: 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**

Note: In determining the purity of a symbol, the algorithm can ignore clauses that are already known to be true in the model constructed so far

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:
 - *Early termination*
 - *Pure symbols*
 - *Unit clauses*: A unit clause is a clause in which all **literals** but one are already assigned false by the model (i.e., left with a single literal that can potentially satisfy the clause). Set the remaining symbol of a unit clause to satisfy it.
 - E.g., if **A=false** and the sentence (in CNF) has a clause **(A $\vee$ B)**, then set B **true**

Similar to Generalized Forward Checking (nFC0) for general CSPs

- Unit propagation: Assigning values to the symbol in a unit clause can lead to new unit clauses. Iteratively find unit clauses until no more remains.

Similar to Constraint Propagation for general CSPs

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

Early termination

P, value $\leftarrow$ **FIND-PURE-SYMBOL**(symbols, clauses, model)

if P is non-null then return **DPLL**(clauses, symbols-P, modelU{P=value})

P, value $\leftarrow$ **FIND-UNIT-CLAUSE**(clauses, model)

if P is non-null then return **DPLL**(clauses, symbols-P, modelU{P=value})

P $\leftarrow$ First(symbols)

rest $\leftarrow$ Rest(symbols)

return or(**DPLL**(clauses, rest, modelU{P=true}),
 DPLL(clauses, rest, modelU{P=false}))

Essentially backtracking

DPLL Algorithm

Clauses:

$$\neg a \vee b \vee c$$

$$a \vee c \vee d$$

$$a \vee c \vee \neg d$$

$$a \vee \neg c \vee d$$

$$a \vee \neg c \vee \neg d$$

$$\neg b \vee \neg c \vee d$$

$$\neg a \vee b \vee \neg c$$

$$\neg a \vee \neg b \vee c$$

Assign $a = \text{true}$

DPLL Algorithm

Clauses:

$$\neg a \vee b \vee c$$

$$a \vee c \vee d$$

$$a \vee c \vee \neg d$$

$$a \vee \neg c \vee d$$

$$a \vee \neg c \vee \neg d$$

$$\neg b \vee \neg c \vee d$$

$$\neg a \vee b \vee \neg c$$

$$\neg a \vee \neg b \vee c$$

Assign $a = \text{true}$

Assign $b = \text{true}$

DPLL Algorithm

Clauses:

$$\neg a \vee b \vee c$$

$$a \vee c \vee d$$

$$a \vee c \vee \neg d$$

$$a \vee \neg c \vee d$$

$$a \vee \neg c \vee \neg d$$

$$\neg b \vee \neg c \vee d$$

$$\neg a \vee b \vee \neg c$$

$$\neg a \vee \neg b \vee c$$

Assign $a = \text{true}$

Assign $b = \text{true}$

Find unit clause $\neg a \vee \neg b \vee c$, so $c = \text{true}$

DPLL Algorithm

Clauses:

$$\neg a \vee b \vee c$$

$$a \vee c \vee d$$

$$a \vee c \vee \neg d$$

$$a \vee \neg c \vee d$$

$$a \vee \neg c \vee \neg d$$

$$\neg b \vee \neg c \vee d$$

$$\neg a \vee b \vee \neg c$$

$$\neg a \vee \neg b \vee c$$

Assign $a = \text{true}$

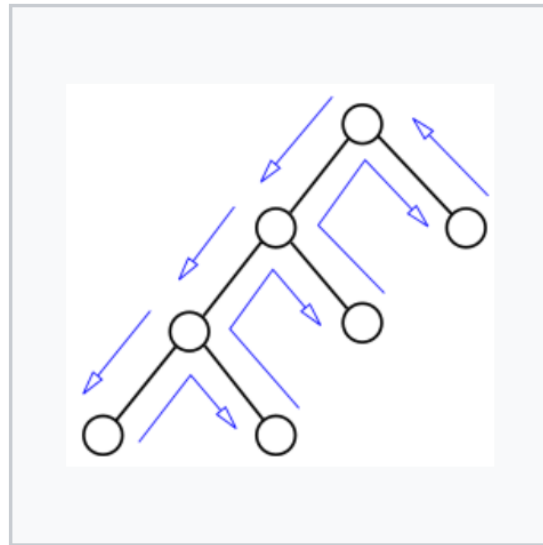
Assign $b = \text{true}$

Find unit clause $\neg a \vee \neg b \vee c$, so $c = \text{true}$

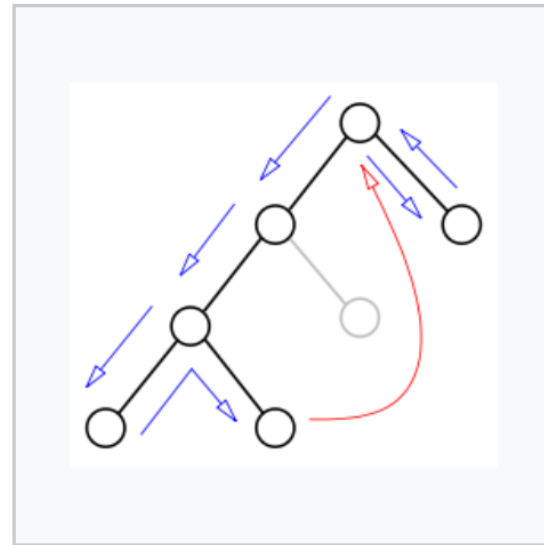
Find unit clause $\neg b \vee \neg c \vee d$, so $d = \text{true}$

Backjumping

- Backjumping is a technique in backtracking algorithms
- Go up more than one level in the search tree when backtrack



A search tree visited by
regular backtracking

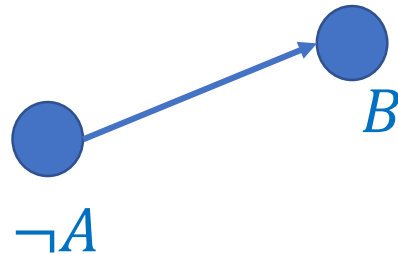


A backjump: the grey
node is not visited

Implication Graph

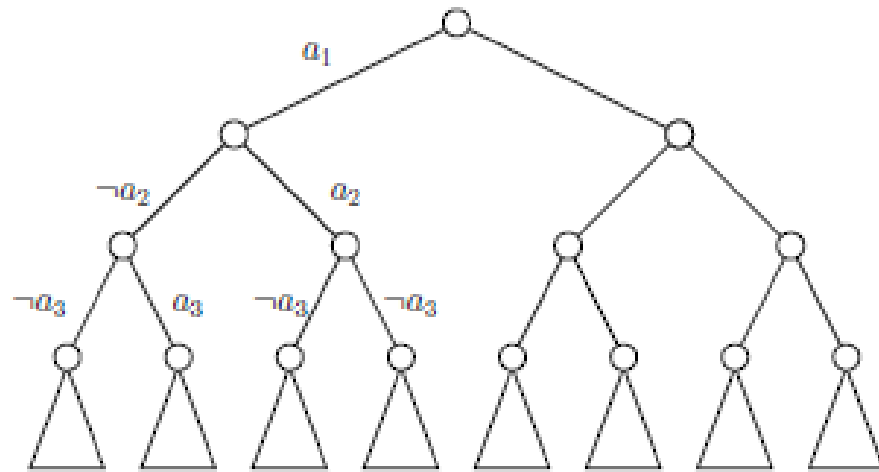
- A directed graph $G = (V, E)$ composed of vertex set V and directed edge set E . Each vertex in V represents the truth status of a Boolean literal, and each directed edge from vertex u to vertex v represents the implication "If the literal u is true then the literal v is also true".

Example: Given a clause $(A \vee B)$, $A=\text{false}$ implies $B=\text{true}$

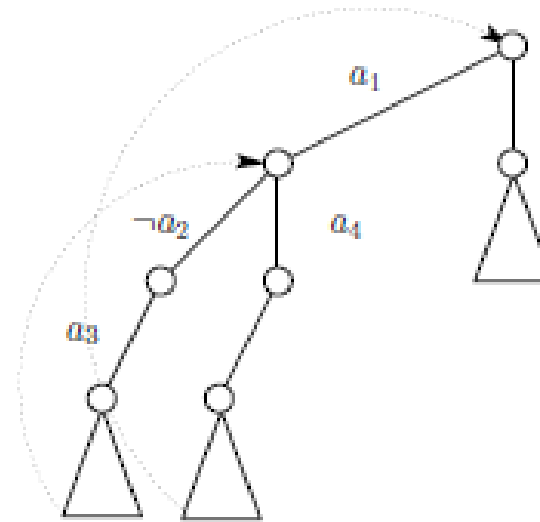


Conflict Driven Clause Learning (CDCL)

- Use implication graph
- Use non-chronological backjumping



DPLL



CDCL

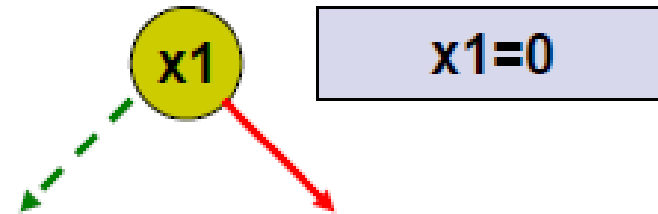
Conflict Driven Clause Learning (CDCL)

1. Select a variable and assign True or False
2. Apply unit propagation to **build the implication graph**
3. If there is any conflict
 - a) **Find the cut** in the implication graph that led to the conflict
 - b) **Derive a new clause** which is the negation of the assignments that led to the conflict
 - c) **Backjump** to the appropriate decision level, where the first-assigned variable involved in the conflict was assigned
4. Otherwise continue from step 1 until all variable values are assigned

Conflict Driven Clause Learning (CDCL)

$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$

Step 1

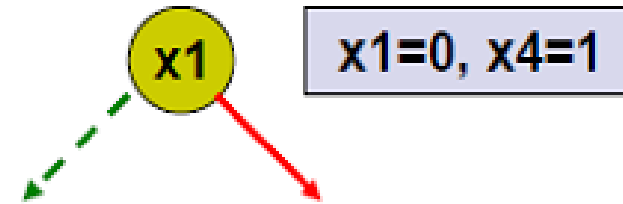
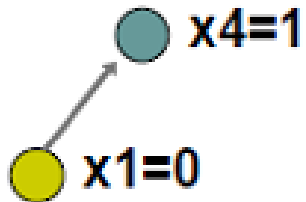


 $x_1=0$

Conflict Driven Clause Learning (CDCL)

Step 2

$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$

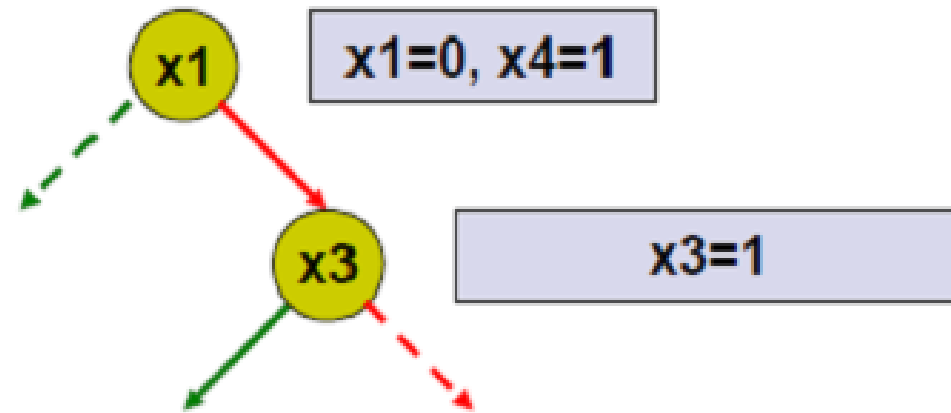
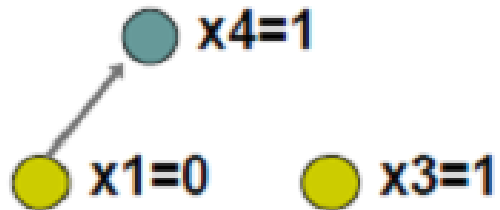


Build the implication graph

Conflict Driven Clause Learning (CDCL)

Step 3

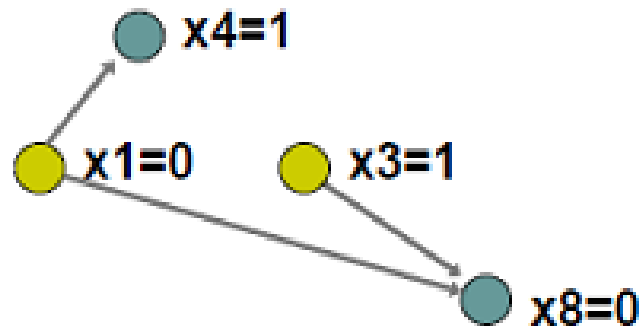
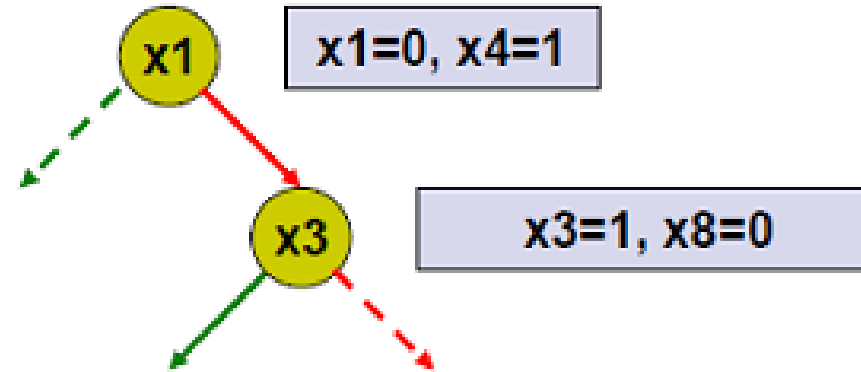
$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$



Conflict Driven Clause Learning (CDCL)

$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$

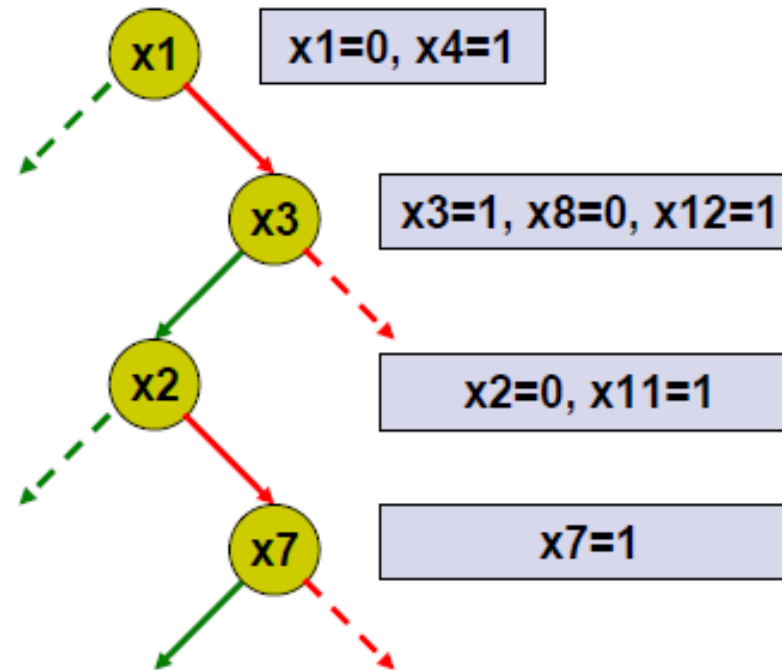
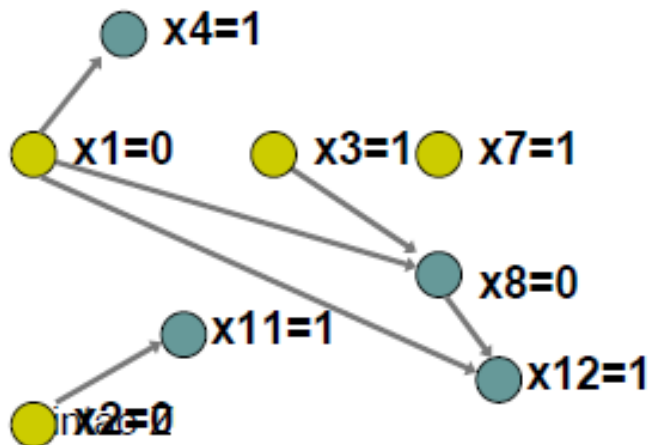
Step 4



Conflict Driven Clause Learning (CDCL)

Step 9

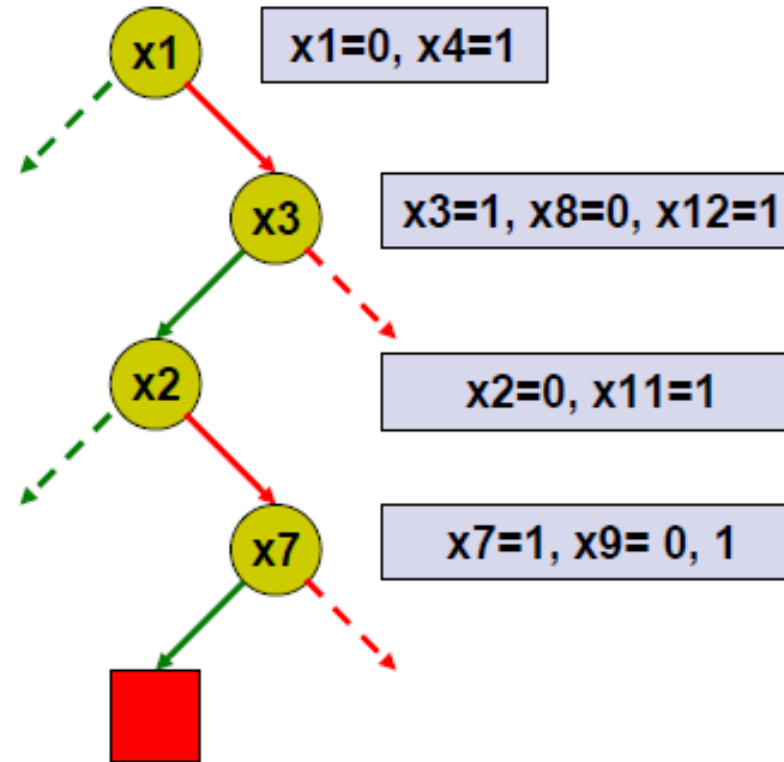
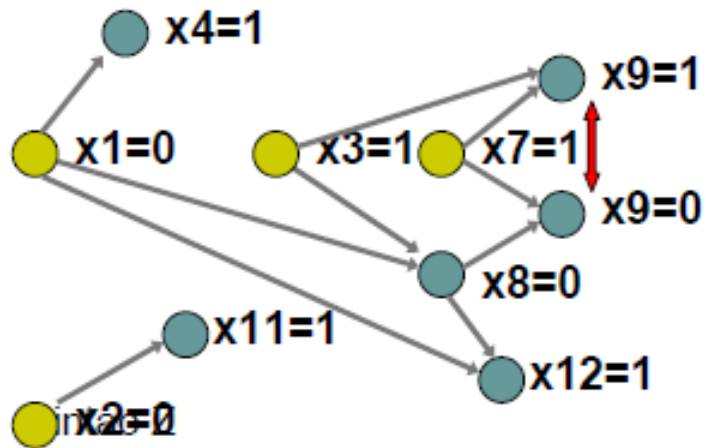
$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$



Conflict Driven Clause Learning (CDCL)

Step 10

$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$

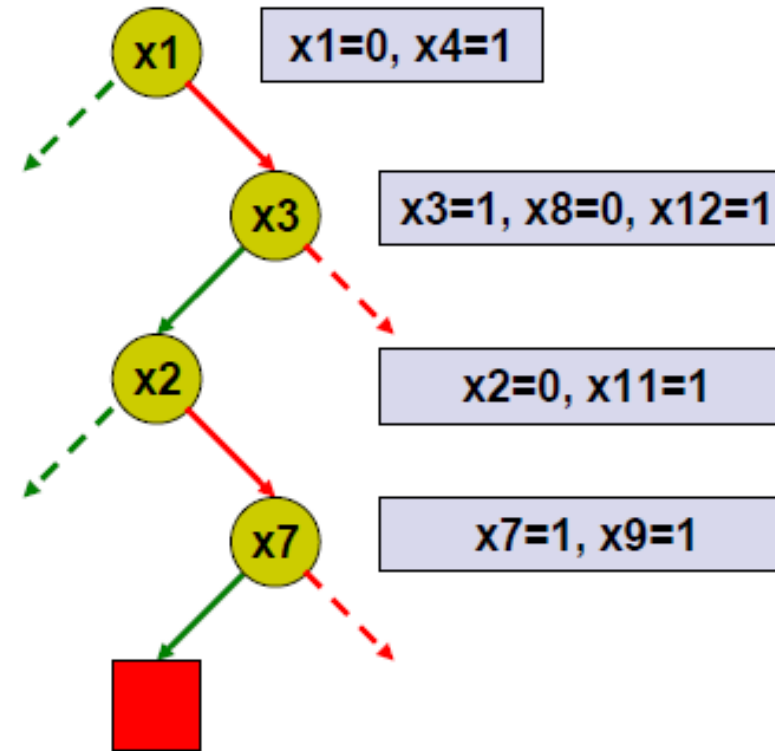
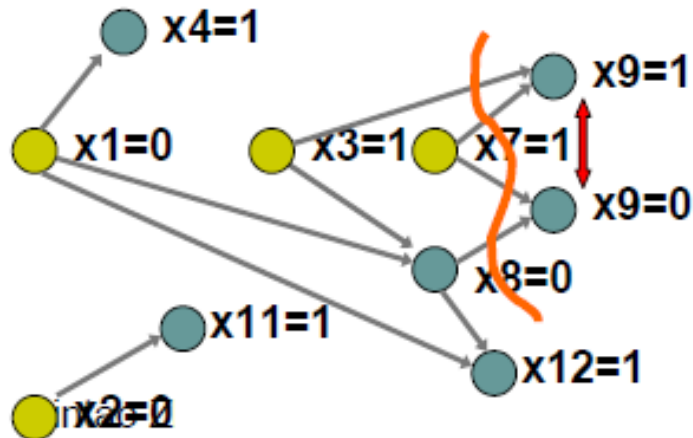


There is a conflict!

Conflict Driven Clause Learning (CDCL)

$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$

Step 11



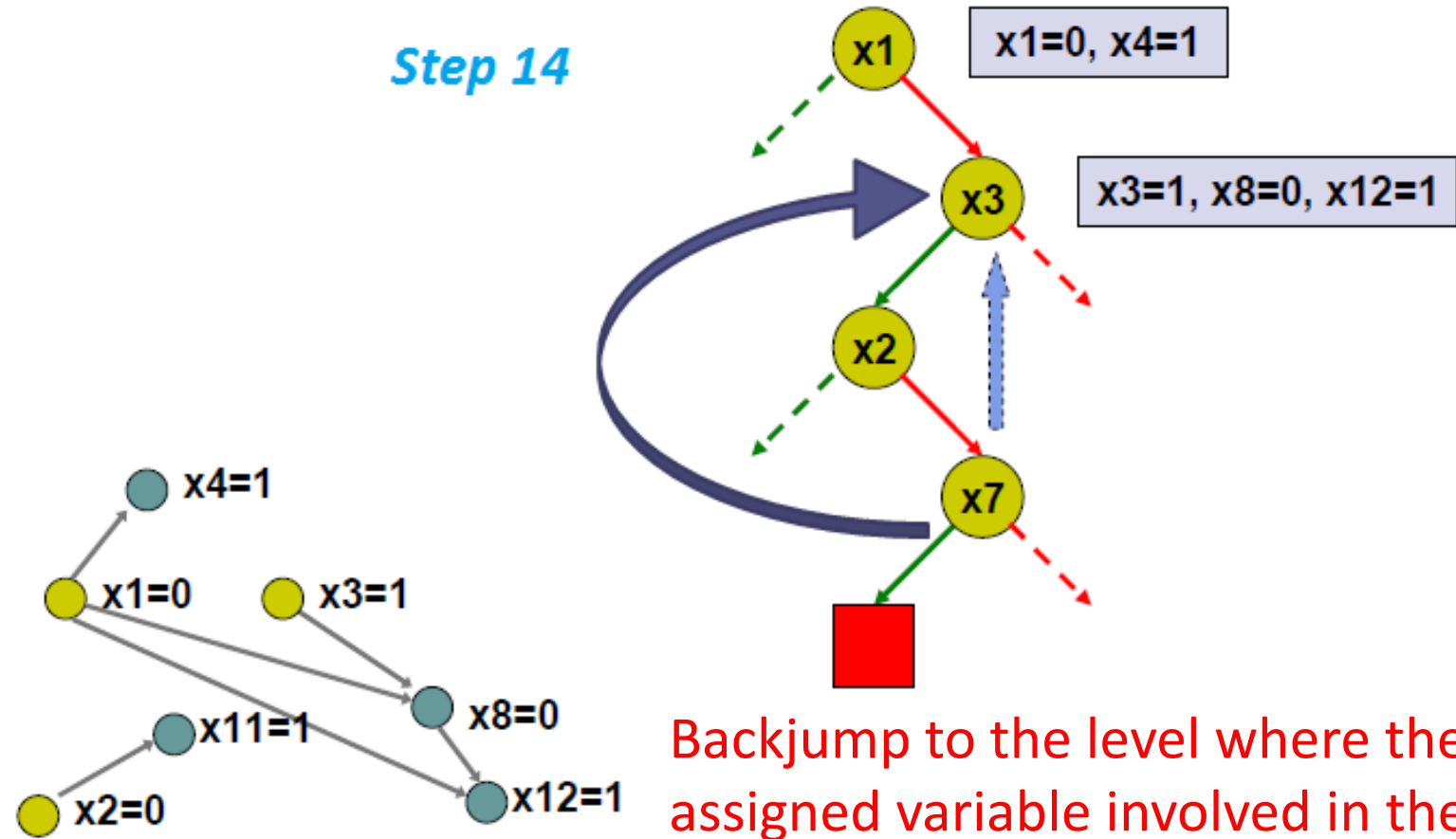
Find the cut and its corresponding literals: $x_3, x_7, \neg x_8$
 Derive a new clause $\neg x_3 \vee \neg x_7 \vee x_8$. Why?

If $x_3 \wedge x_7 \wedge \neg x_8$, then there will be a conflict

Conflict Driven Clause Learning (CDCL)

$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$
 $\neg x_3 \vee \neg x_7 \vee x_8$

Step 14

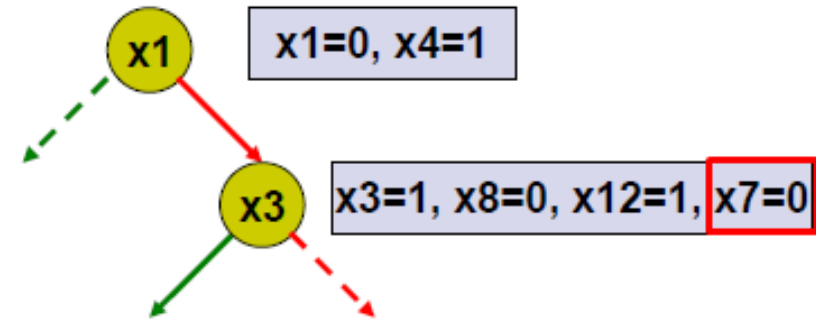


Backjump to the level where the first-assigned variable involved in the conflict was assigned.

Conflict Driven Clause Learning (CDCL)

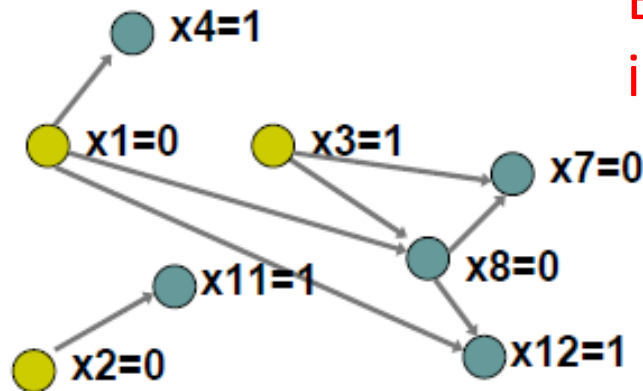
$x_1 \vee x_4$
 $x_1 \vee \neg x_3 \vee \neg x_8$
 $x_1 \vee x_8 \vee x_{12}$
 $x_2 \vee x_{11}$
 $\neg x_7 \vee \neg x_3 \vee x_9$
 $\neg x_7 \vee x_8 \vee \neg x_9$
 $x_7 \vee x_8 \vee \neg x_{10}$
 $x_7 \vee x_{10} \vee \neg x_{12}$
 $\neg x_3 \vee \neg x_7 \vee x_8$

Step 15



Continue Unit propagation.

Backtrack to the level where the “problem” is instead of merely trying to fix the “symptom”



Conflict Driven Clause Learning (CDCL)

1. Select a variable and assign True or False
2. Apply unit propagation to **build the implication graph**
3. If there is any conflict
 - a) **Find the cut** in the implication graph that led to the conflict
 - b) **Derive a new clause** which is the negation of the assignments that led to the conflict
 - c) **Backjump** to the appropriate decision level, where the first-assigned variable involved in the conflict was assigned
4. Otherwise continue from step 1 until all variable values are assigned

Similar ideas can be applied to general CSPs

Local Search Algorithms for SAT

- WALK-SAT
 - Randomly choose an unsatisfied clause
 - With probability p , flip a randomly selected symbol in the clause
 - Otherwise, flip a symbol in the clause that maximizes the # of satisfied clauses

WALKSAT

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a model or *failure*

inputs: *clauses*, a set of clauses

p, the probability of choosing to do a random walk, typically around 0.5

max_flips, number of flips allowed before giving up

model $\leftarrow$ 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 $\leftarrow$ 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 # of satisfied clauses

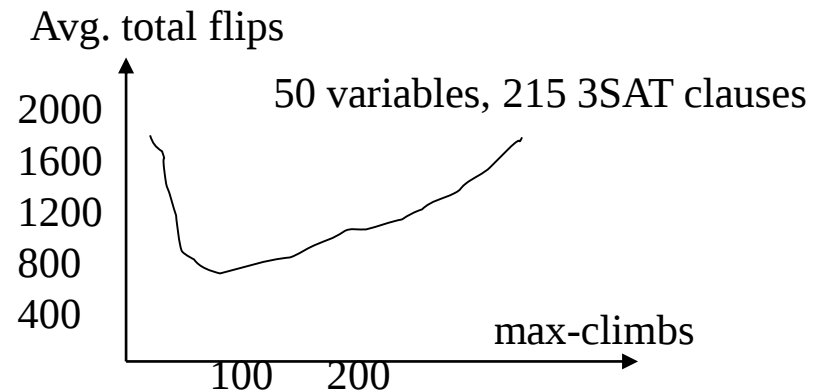
return *failure*

Local Search Algorithms for SAT

- WALK-SAT
 - Randomly choose an unsatisfied clause
 - With probability p , flip a randomly selected symbol in the clause
 - Otherwise, flip a symbol in the clause that maximizes the # of satisfied clauses
- GSAT [Selman, Levesque, Mitchell AAAI-92]
 - Similar to hill climbing but with random restarts and allows for downhill/sideway moves if no better moves available

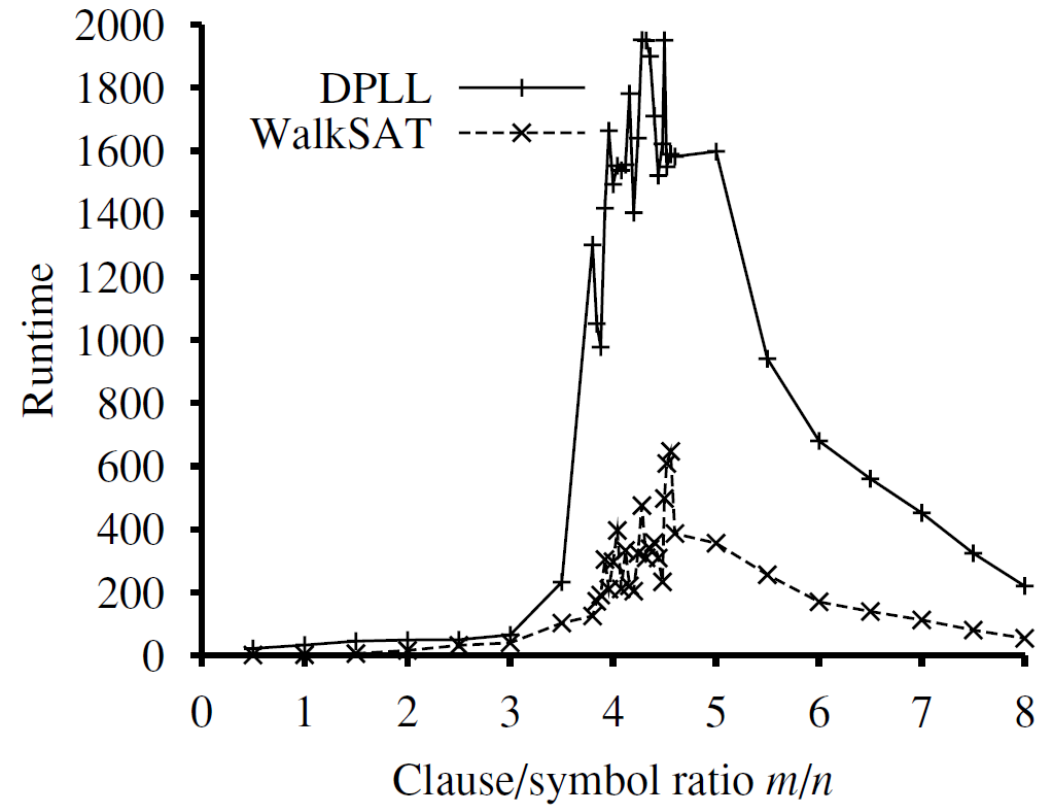
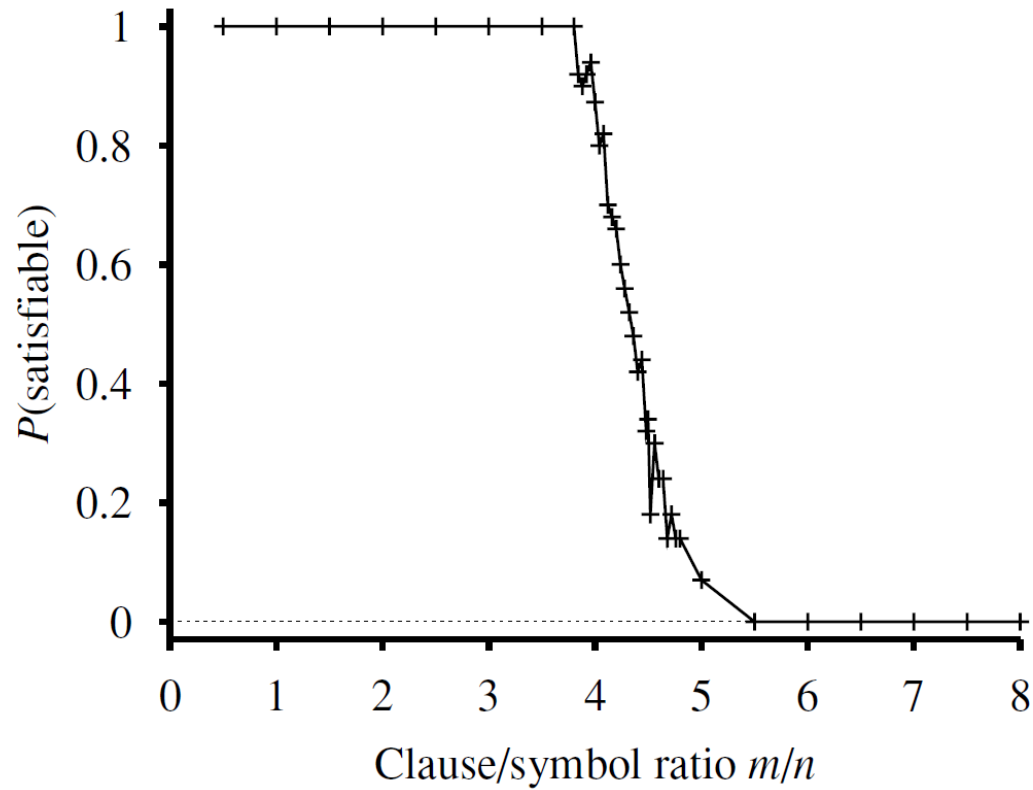
GSAT

```
function GSAT(sentence, max_restarts, max_climbs) returns a model or failure  
  for i = 1 to max_restarts do  
    model  $\leftarrow$  a random assignment of true/false to the symbols in clauses  
    for j = 1 to max_climbs do  
      if model satisfies sentence then return model  
      model  $\leftarrow$  randomly choose one of the best successors  
  return failure
```

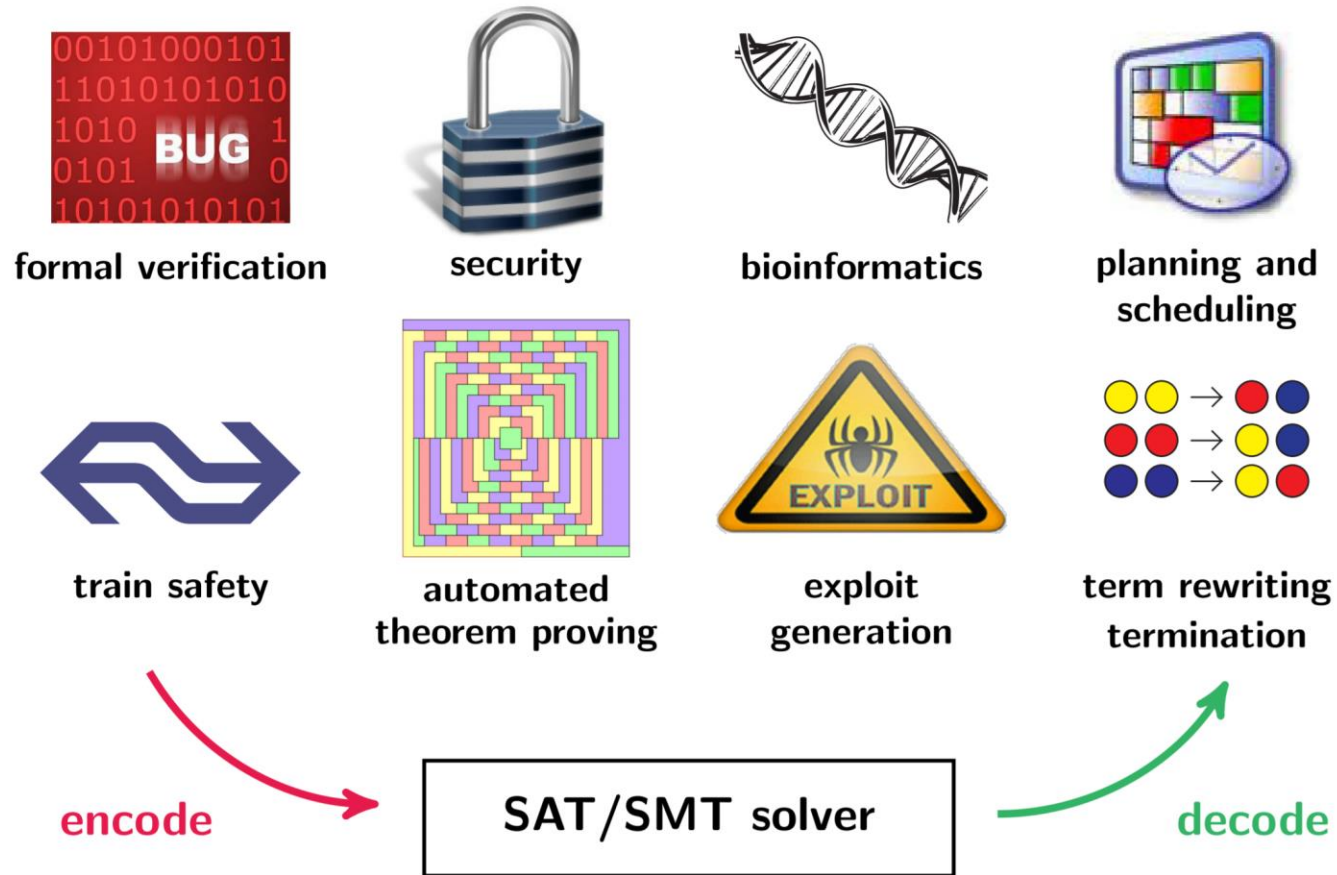


Greediness is not essential as long as climbs and sideways moves are preferred over downward moves.

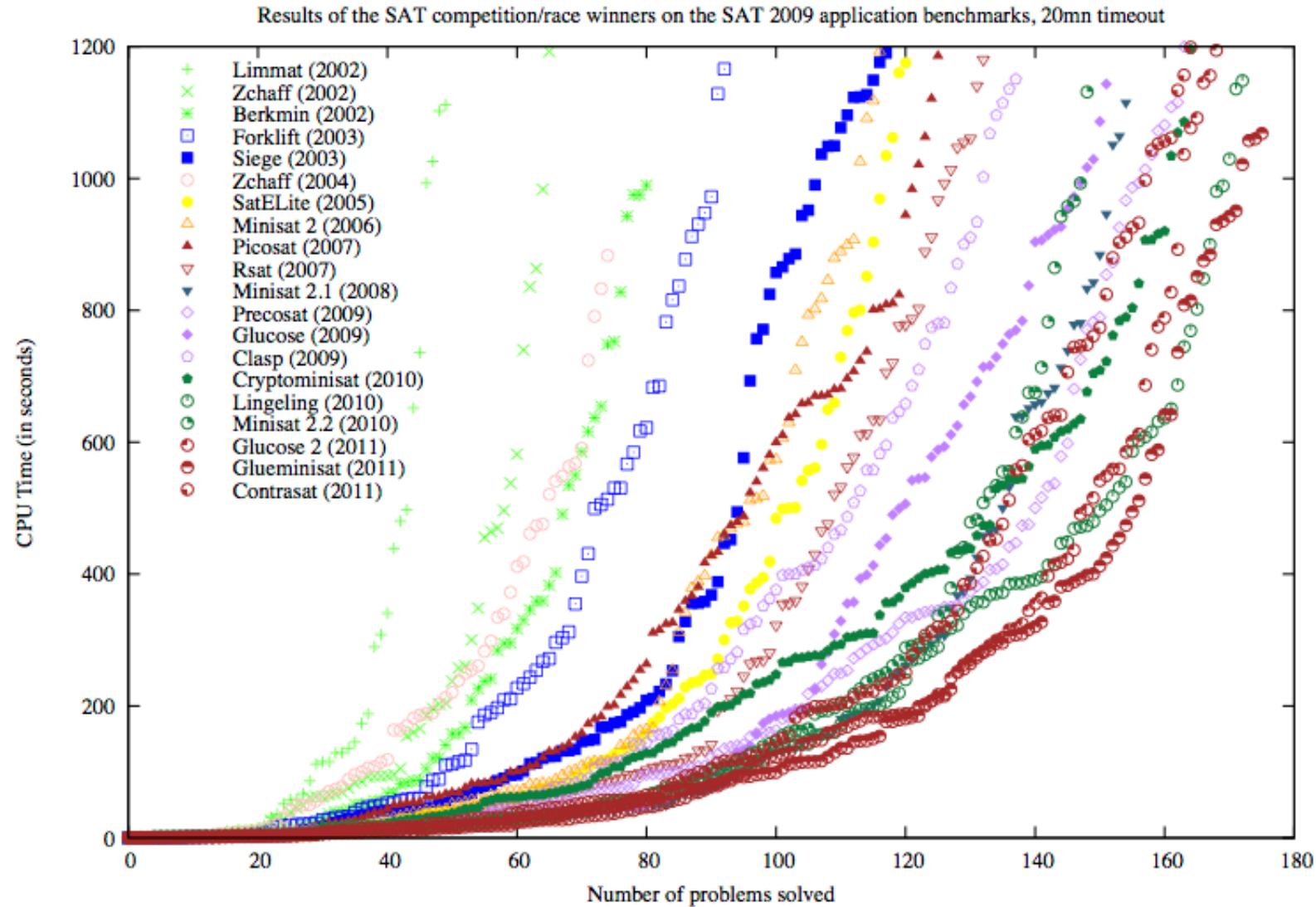
Phase Transition of SAT



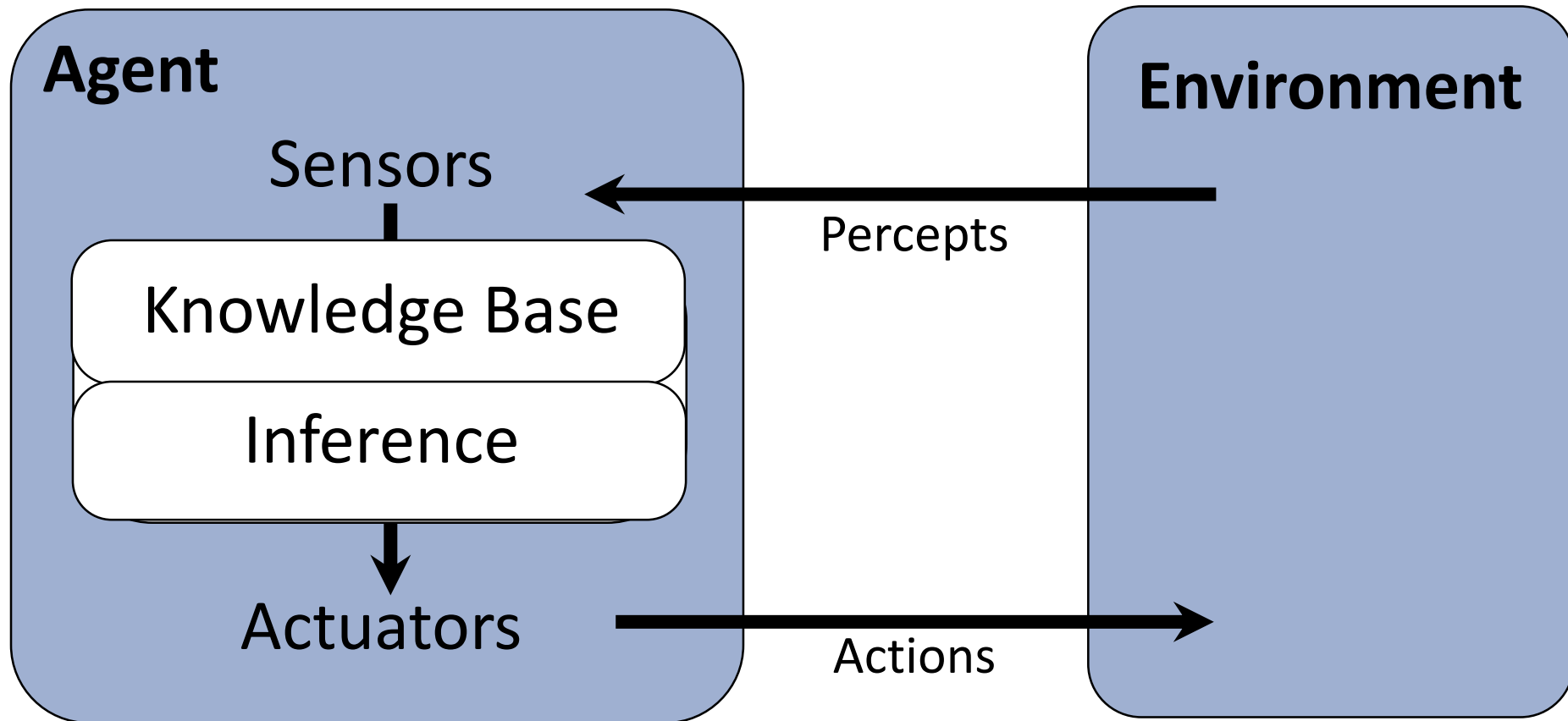
SAT Applications



Evolution of SAT Solvers

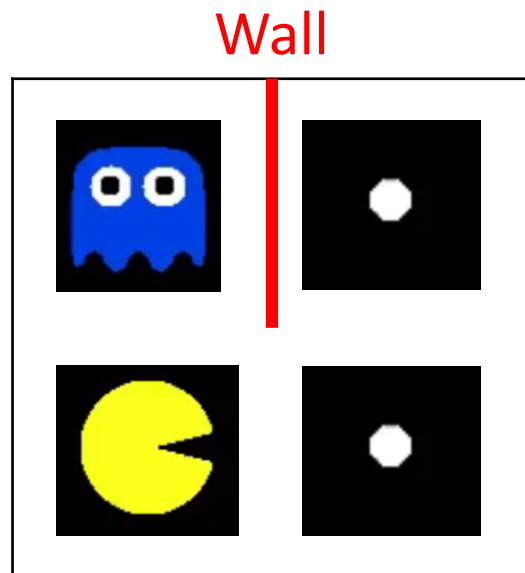


Agent based on Propositional Logic



Planning as Satisfiability (SATPlan)

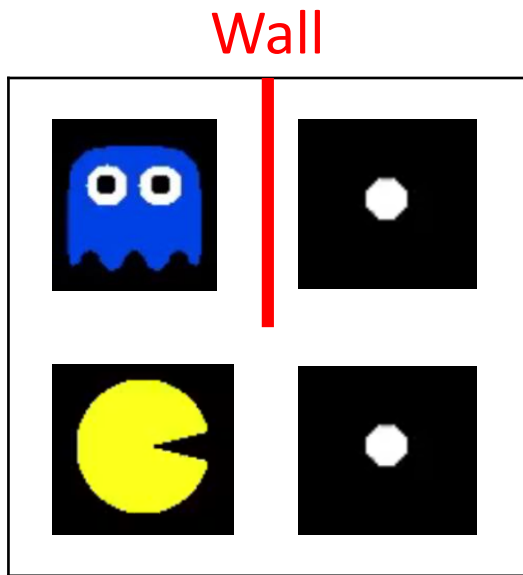
- Given a hyper-efficient SAT solver, can we use it to make plans for an agent so that it is guaranteed to achieve certain goals?
- For fully observable, deterministic case: Yes, planning problem is solvable iff there is some satisfying assignment for actions etc. (No sensor needed due to full observability; KB does not grow)



How can Pacman eat all food given that the ghost will move South, then E, then N, then stop there?

Planning as Satisfiability (SATPlan)

How can Pacman eat all food given that the ghost will move South, then E, then N, then stop there?



Use symbols to represent the problem, including aspects of the world that do not change over time (called “**atemporal variables**”), e.g., $Wall_{ij}$, and aspects that change over time (called as “**fluent**”, or “**state variables**”), e.g., location L_{ij}^t and action $N^t, S^t, E^t, W^t, \forall t = 1, 2, \dots, T$

1. Set up KB: Write down all the sentences in KB
2. Solve SAT: Find a model that satisfy all these sentences

What should be the value of T ?

Recall Iterative Deepening. Gradually increase T if a small value returns no solution

Planning as Satisfiability (SATPlan)

T_{max} : Max length of planning horizon

function SATPLAN(*init*, *transition*, *goal*, T_{max}) **returns** solution or failure

inputs: *init*, *transition*, *goal*, constitute a description of the problem

T_{max} , an upper limit for plan length

for $T = 0$ **to** T_{max} **do** T is the length of planning horizon. Gradually increase.

$cnf \leftarrow \text{TRANSLATE-TO-SAT}(init, transition, goal, T)$ Set up the KB

$model \leftarrow \text{SAT-SOLVER}(cnf)$ Run SAT solver

if $model$ is not null **then**

return EXTRACT-SOLUTION($model$)

return *failure*

Planning as Satisfiability (SATPlan)

- How to set up the KB? KB often includes sentences describing

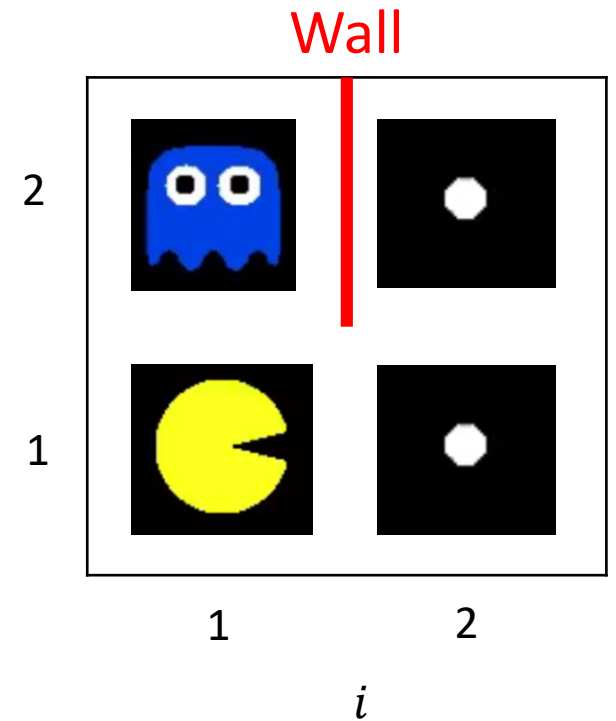
- Initial state

e.g., $L_{11}^0, Food_{21}^0, Ghost_{12}^0, GhostMoveS^0, \neg Wall_{12}^S, Wall_{12}^E, \dots$

- Domain constraints

e.g., Pacman cannot be at two locations at the same time

$\neg(L_{11}^1 \wedge L_{12}^1) \wedge \neg(L_{11}^1 \wedge L_{21}^1) \wedge \neg(L_{11}^1 \wedge L_{212}^1) \wedge \neg(L_{12}^1 \wedge L_{21}^1) \dots$



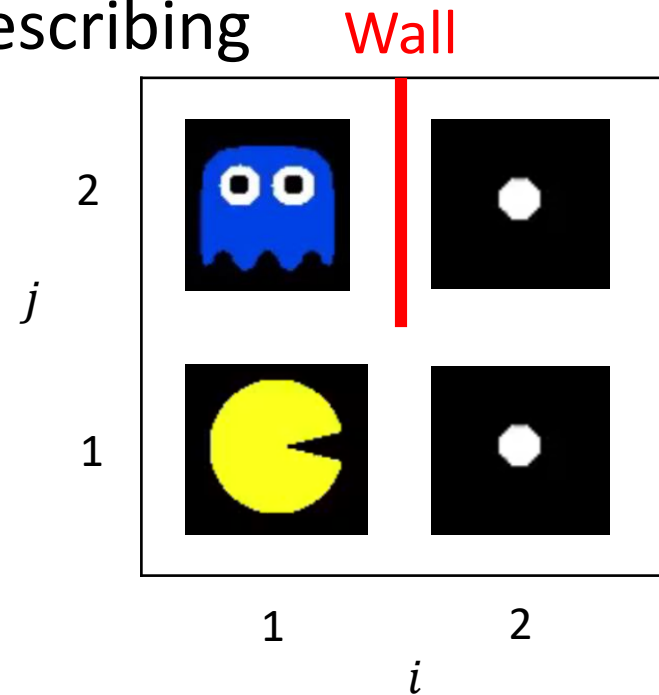
Planning as Satisfiability (SATPlan)

- How to set up the KB? KB often includes sentences describing

- Transition model sentences up to time T

Write down how each *fluent* at each time gets its value based on **successor-state axiom**:

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$$



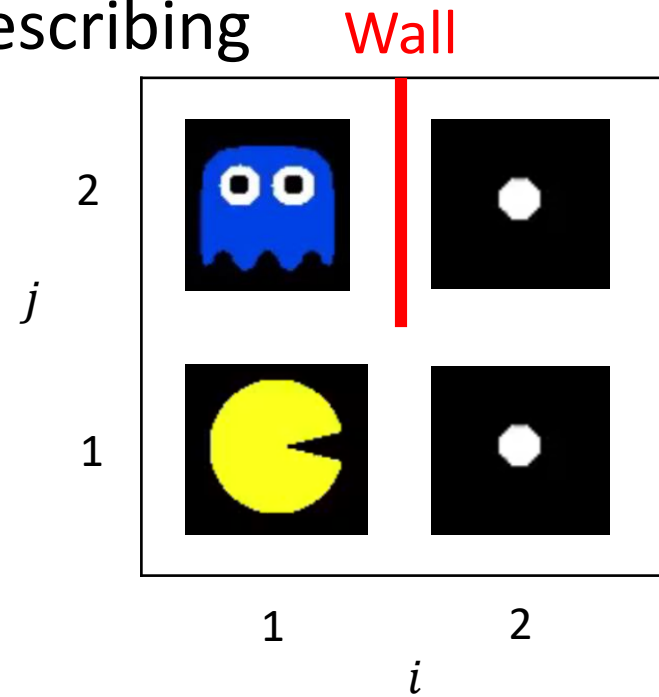
$$\begin{aligned} \text{e.g., } L_{12}^1 \Leftrightarrow & \left(\left(L_{11}^0 \wedge N^0 \wedge \neg Wall_{12}^S \wedge \neg \left(Ghost_{12}^0 \wedge (GhostMoveS^0 \vee GhostStay^0) \right) \right) \vee \dots \right) \\ & \vee \left(L_{12}^0 \wedge \neg \left((S^0 \wedge \neg Wall_{12}^S) \vee (Ghost_{22}^0 \wedge GhostMoveW^0 \wedge \neg Wall_{12}^E) \vee \dots \right) \right) \end{aligned}$$

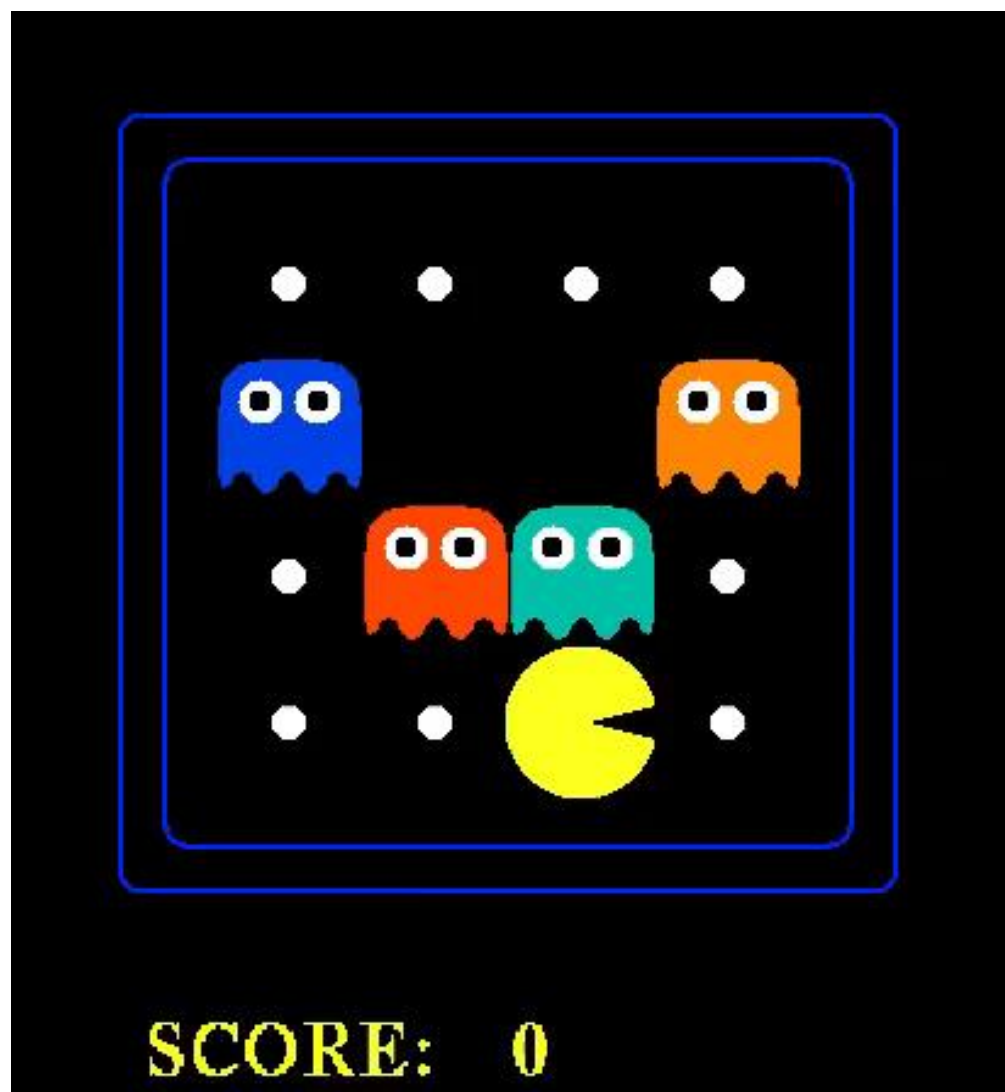
Planning as Satisfiability (SATPlan)

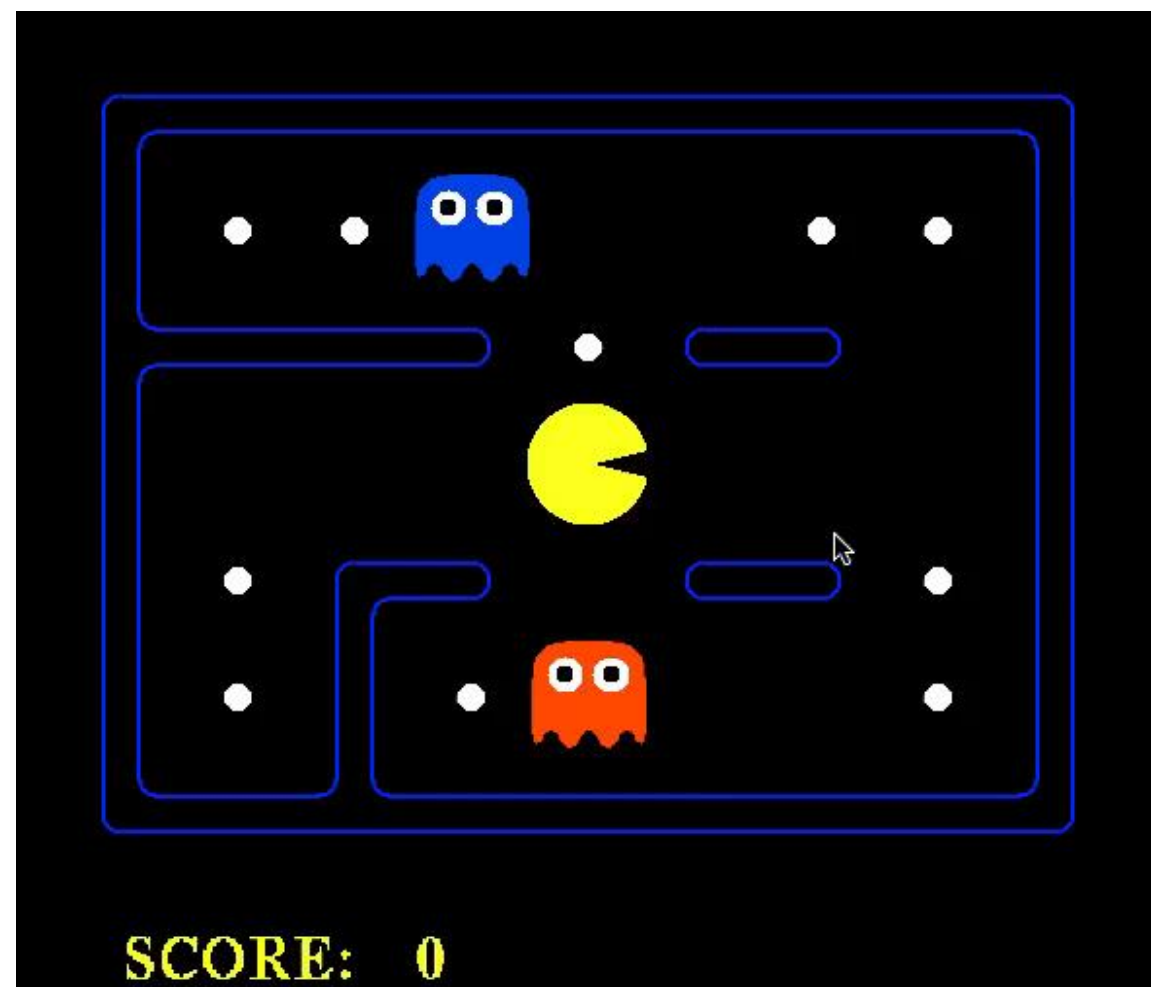
- How to set up the KB? KB often includes sentences describing

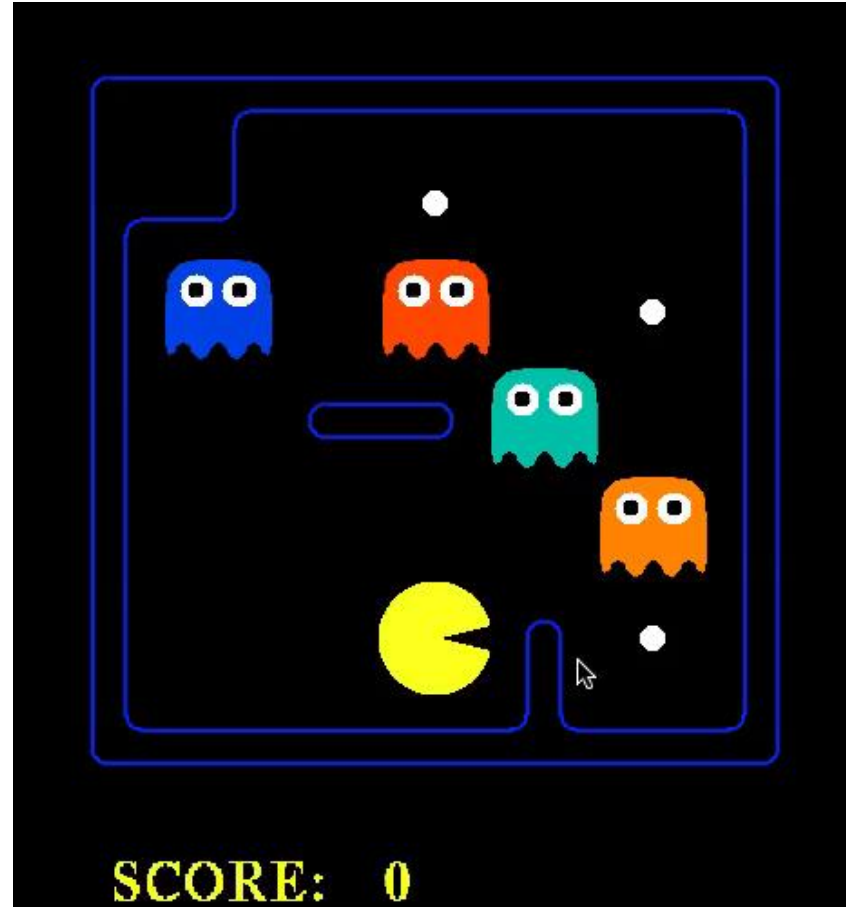
- Goal is achieved at time T

e.g., $\neg Food_{21}^T \wedge \neg Food_{22}^T$



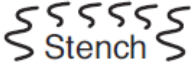
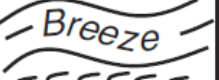
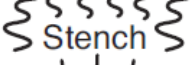
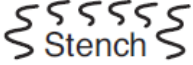






Wumpus World

- The world is not fully observable from the beginning
- KB consists of
 - Facts
 - Rules
 - Percept and Actions
- Keep adding sentences to the KB with new percepts and actions
- At any time step, we can Ask the KB about the current state, e.g., whether a square is safe

4				
3		  		
2				
1	 START			
	1	2	3	4

B_{ij} = breeze felt; S_{ij} = stench smelt

P_{ij} = pit here; W_{ij} = wumpus here; G = gold

Hybrid Agent

- Plan actions by combining search and logical inference
- Maintain and update a KB as well as a current plan
- Construct a plan based on a decreasing priority of goals
- In Wumpus world
 - Ask KB to work out which squares are safe and which have yet to be visited
 - If there is glitter, construct a plan to grab the gold and go back safely
 - If there is no current plan, use A* search to plan a route that only goes through safe squares to the closest unvisited safe square
 - If no such safe squares to explore, ask questions to determine whether to shoot at one of the possible wumpus locations

Summary

- Many problems can be reduced to SAT
- Efficient SAT solvers operate on CNF and use ideas in solving CSPs such as backtracking and local search
- Can frame a planning problem as a satisfiability problem

Learning Objectives

- Describe the definition of (Boolean) Satisfiability Problem (SAT)
- Describe the definition of Conjunctive Normal Form (CNF)
- Describe the following algorithms for solving SAT
 - DPLL, CDCL, WalkSAT, GSAT
- Determine whether a sentence is satisfiable
- Describe Successor-State Axiom
- Describe and implement SATPlan (Planning as Satisfiability)
- (Hybrid Agent)