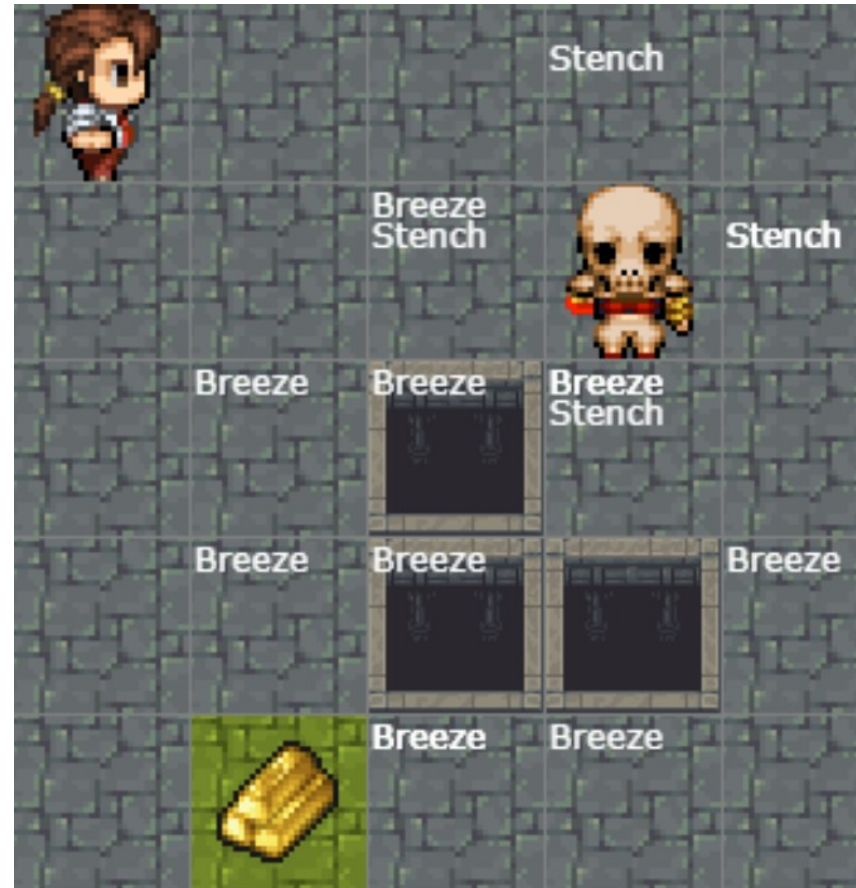


Warm-up:

Play [Minesweeper](#) or [Wumpus World](#)!



Warm-up

Play [Minesweeper](#) (link on website/piazza)

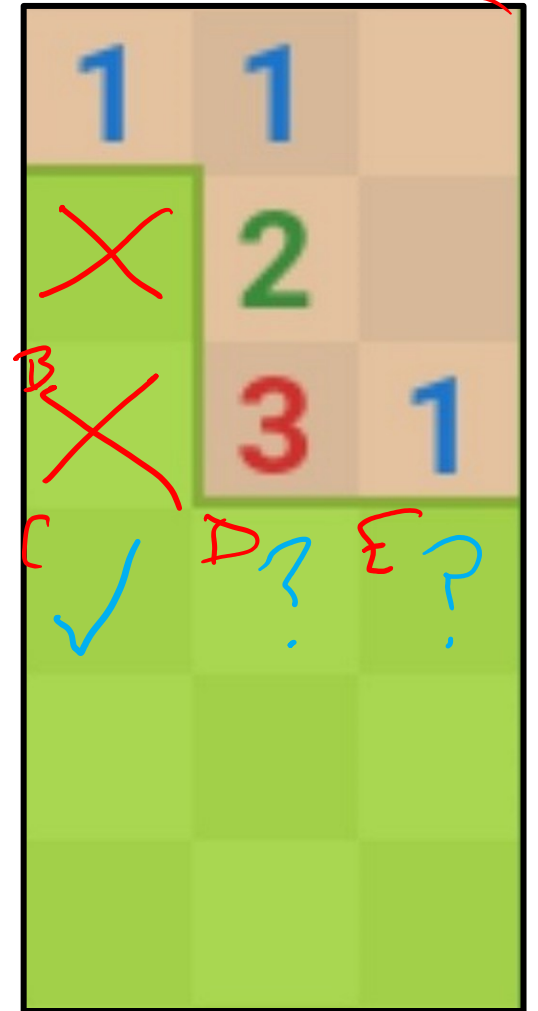
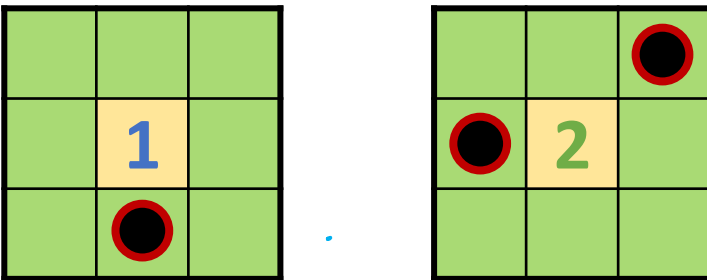
then fill in the green squares with:

✓ if SAFE or

X if UNSAFE or

? if UNSURE

Numbers indicate how many mines are in the 8 adjacent cells



Monty Python Inference

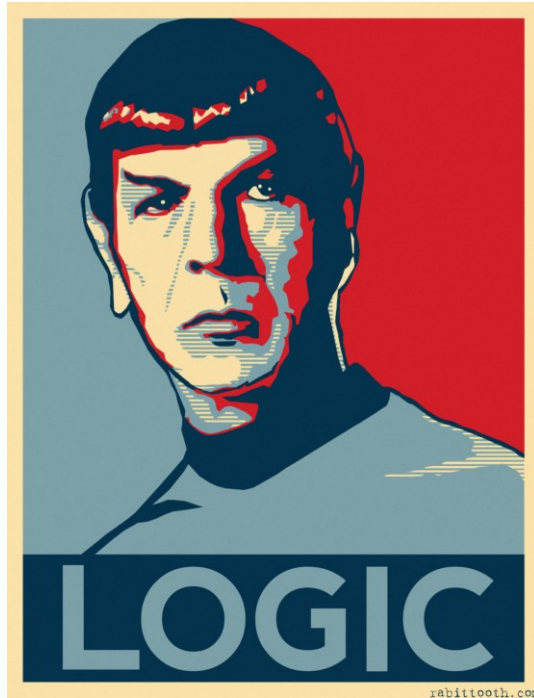
There are ways of telling whether she is a witch



<https://www.youtube.com/watch?v=rf71YotfykQ&t=52>

07-380 AI & ML II

Logical Agents



Instructors: Pat Virtue & Mohammad Salameh

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

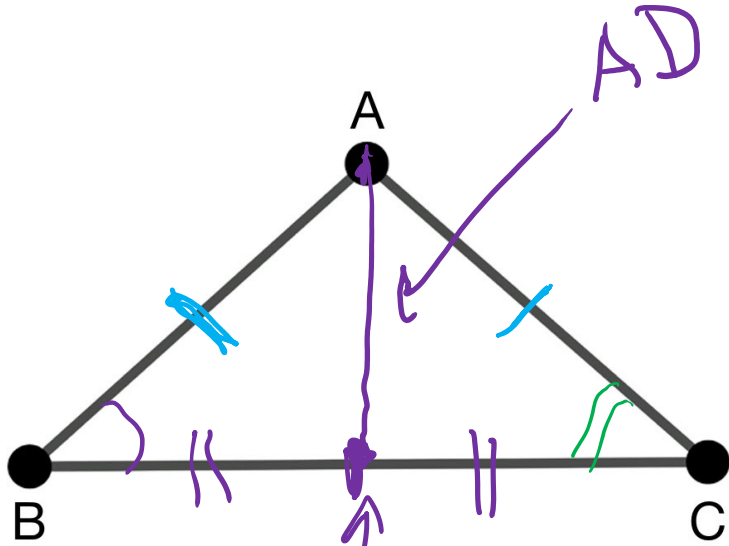
Warm-up

Write a proof for the following:

Given a triangle ABC with $AB=AC$,
 $\angle ABC = \angle ACB$

$$\underline{AB} + \underline{AC} < \underline{BC}$$

$$BD = DC$$



Let D be midpoint of BC

AlphaGeometry

<https://www.nature.com/articles/d41586-025-00406-7>

<https://www.nature.com/articles/s41586-023-06747-5>

nature

Explore content ▾

About the journal ▾

Publish with us ▾

Subscribe

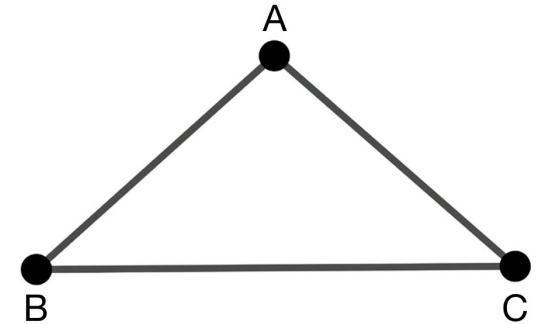
[nature](#) > [news](#) > article

NEWS | 07 February 2025

DeepMind AI crushes tough maths problems on par with top human solvers

The company's AlphaGeometry 2 reaches the level of gold-medal students in the International Mathematical Olympiad.

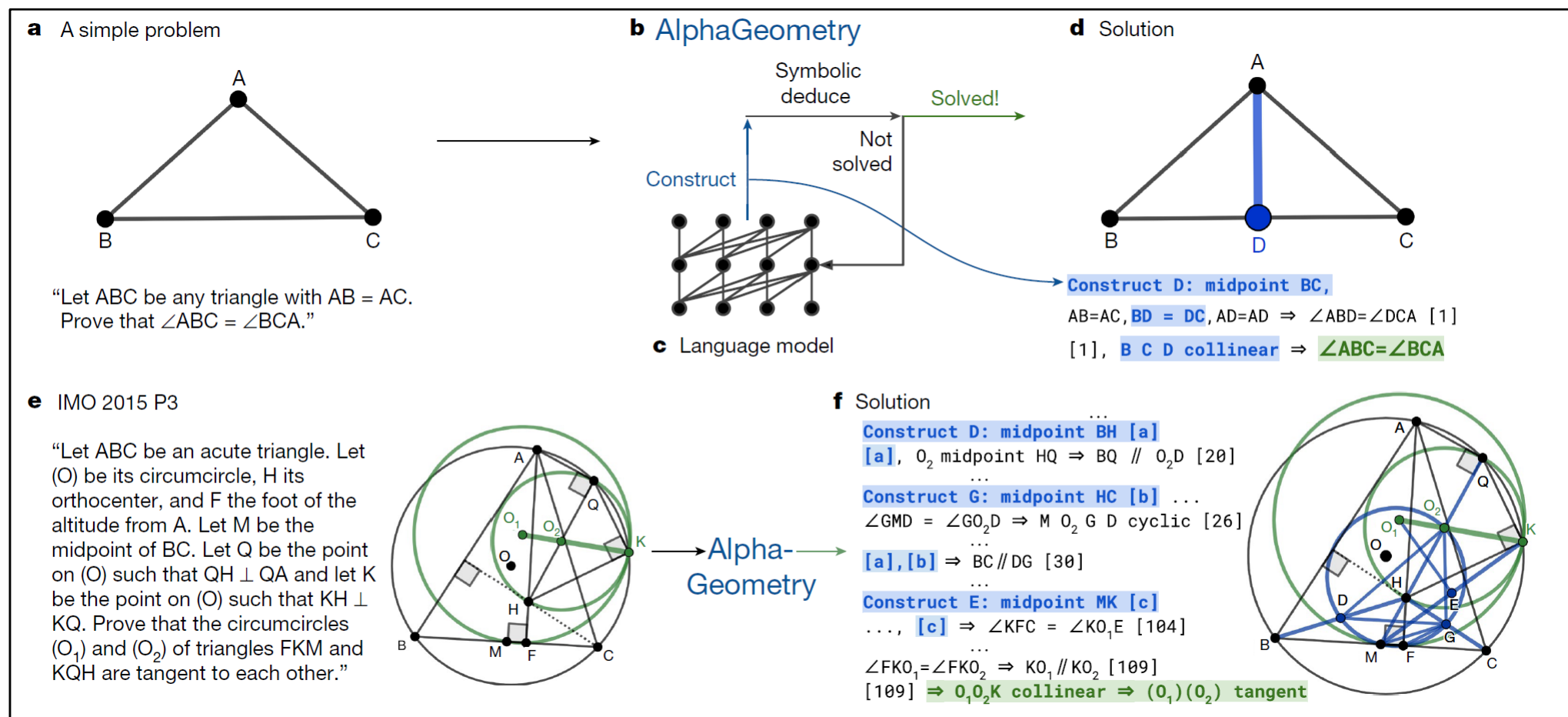
By [Davide Castelvecchi](#)



AlphaGeometry

<https://www.nature.com/articles/d41586-025-00406-7>

<https://www.nature.com/articles/s41586-023-06747-5>

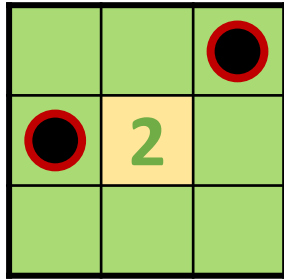
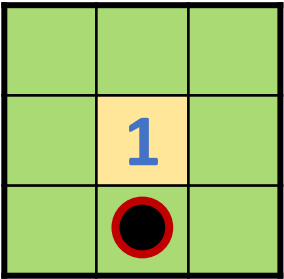


How do we solve Sudoku?

1			
	2	1	
		3	
			4

How do we play Minesweeper?

Numbers indicate how many mines are in the 8 adjacent cells



What are we trying to figure out?

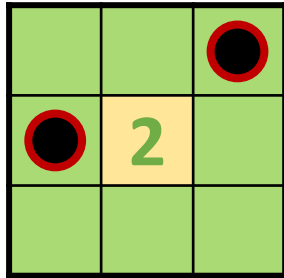
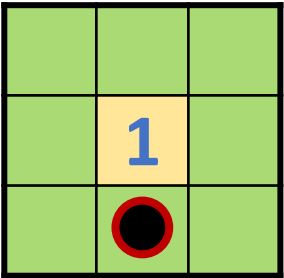
- A path (a sequence of actions)?
- A complete solution?



Image: Google Minesweeper game

Minesweeper

Numbers indicate how many mines are in the 8 adjacent cells

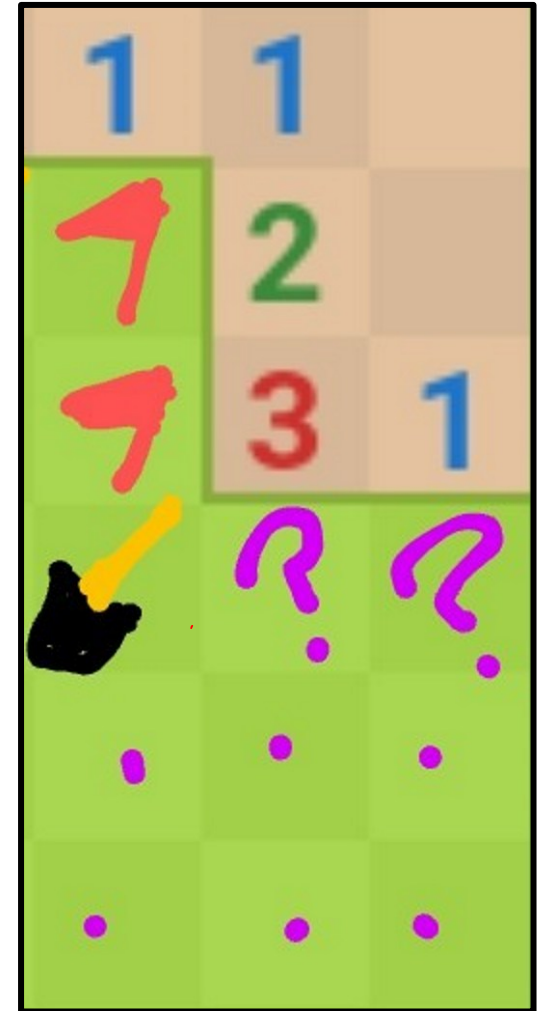
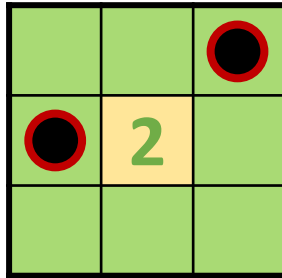
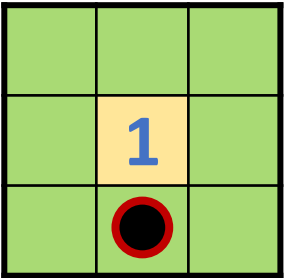


We're trying to figure out what to do next

- Which unvisited spaces that are **definitely safe**?
- Which unvisited spaces that are **definitely dangerous**?
- (What about the other spaces?)

Minesweeper

Numbers indicate how many mines are in the 8 adjacent cells



We're trying to figure out what to do next

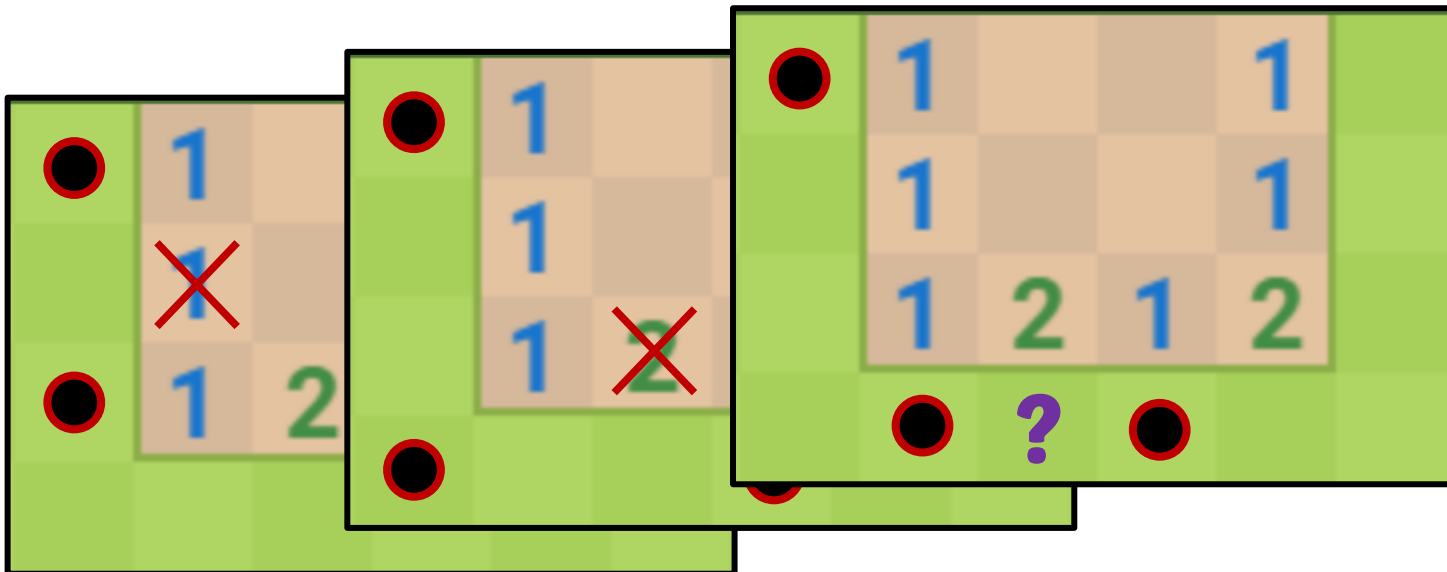
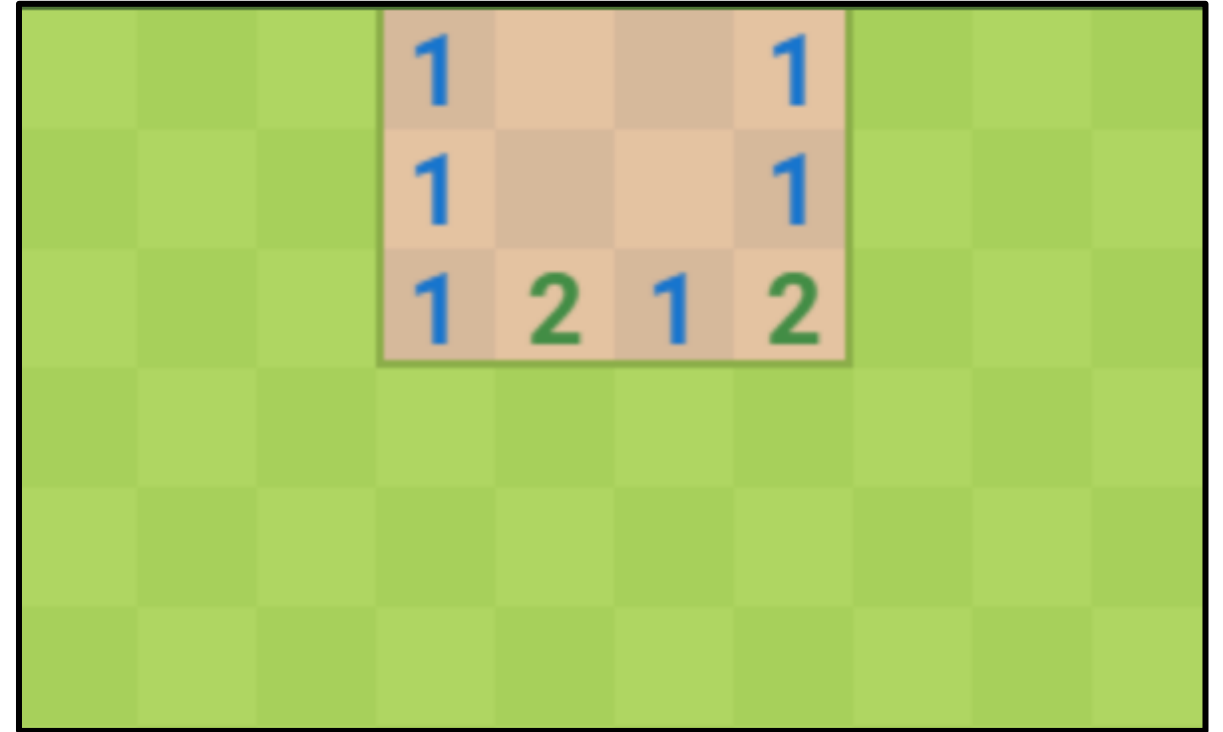
- Which unvisited spaces that are **definitely safe**?
- Which unvisited spaces that are **definitely dangerous**?
- (What about the other spaces?)

Minesweeper

It may take a few logical steps to reason about:

- 1) What is possible
- 2) What is impossible
- 3) What is still unknown

Example human inference steps:

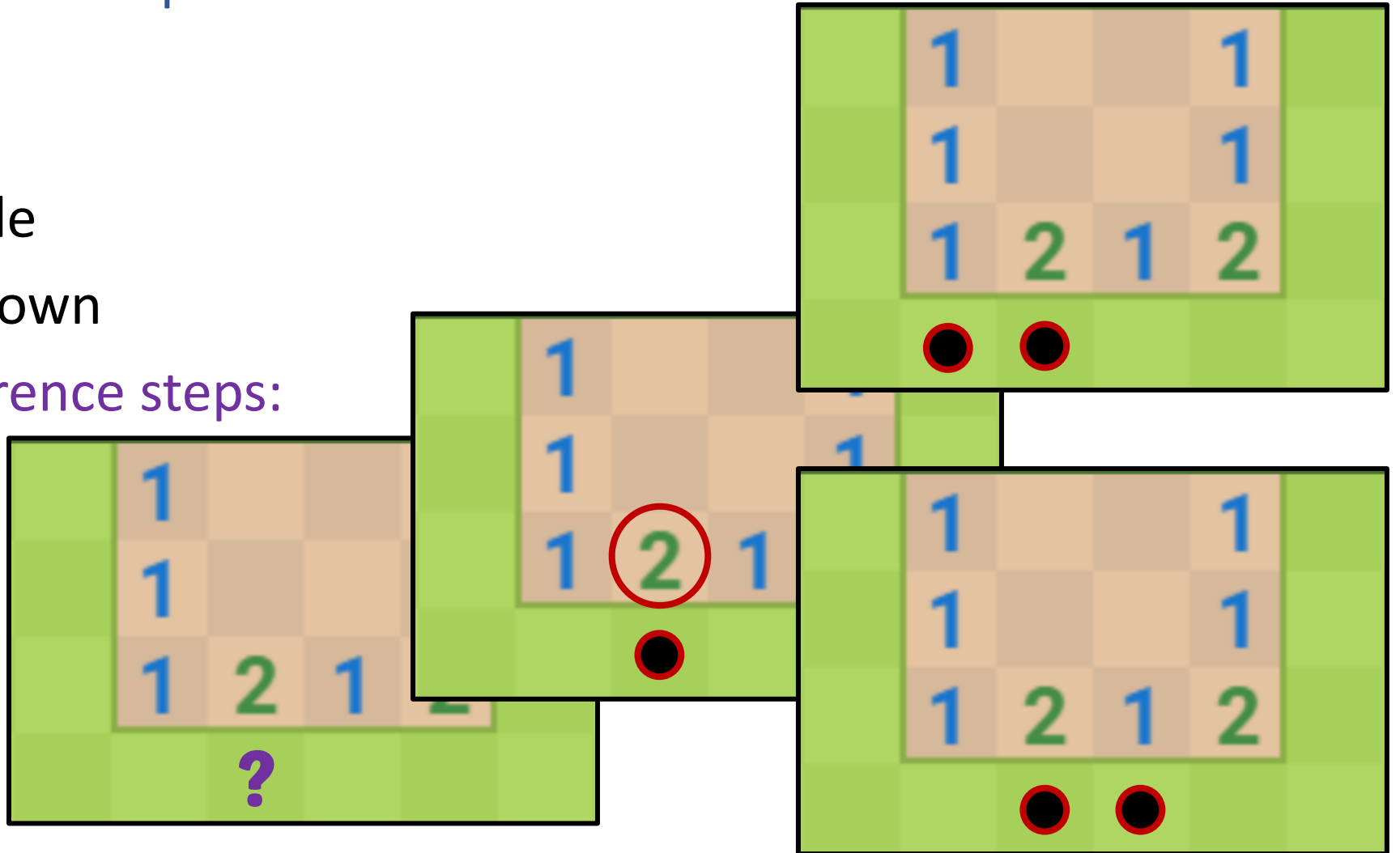


Minesweeper

It may take a few logical steps to reason about:

- 1) What is possible
- 2) What is impossible
- 3) What is still unknown

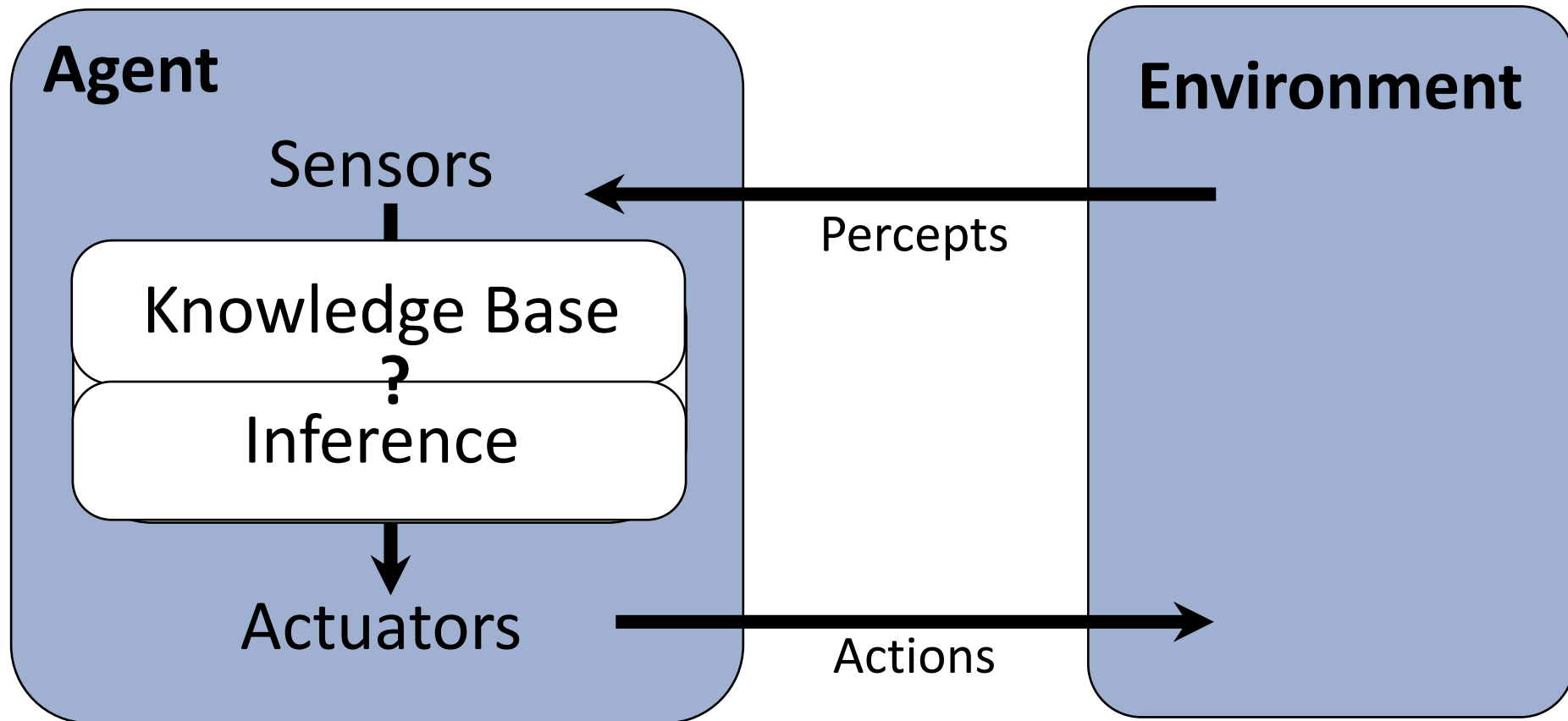
Example human inference steps:



High-level View: Logical Agents

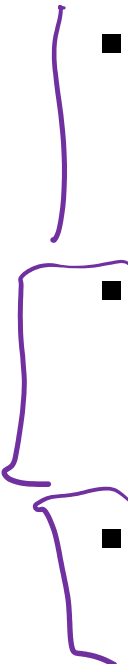
Logical Agents

Logical agents and environments



Logical Agents

So what do we TELL our knowledge base (KB)?

- 
- Facts (sentences)
 - The grass is green
 - The sky is blue
 - Rules (sentences)
 - Eating too much candy makes you sick
 - When you're sick you don't go to school
 - Percepts and Actions (sentences)
 - Pat ate too much candy today

What happens when we ASK the agent?

- Inference – new sentences created from old
 - Pat is not going to school today

A Knowledge-based Agent

function **KB-AGENT**(percept) returns an action

persistent: **KB**, a knowledge base

persistent: **t**, an integer, initially 0

TELL(**KB**, **PROCESS-PERCEPT**(percept, t))

action $\leftarrow$ **ASK**(**KB**, **PROCESS-QUERY**(t))

TELL(**KB**, **PROCESS-RESULT**(action, t))

t $\leftarrow$ t+1

return action

Pre-reading

Poll 1

KB: $[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$

→ one sent

- ✓ A) conj(KB)
- B) disj(KB)
- C) evaluate one at-a time
- D) IDK

Poll 2

KB:

$$\begin{array}{l} B \\ A \wedge B \Rightarrow C \end{array}$$

What will $A \wedge B \Rightarrow C$ evaluate to?

A T

B F

C Not enough info ←

Models and Knowledge Bases

Entailment and Satisfiability

Logical Agent Vocab

Model

- An specific assignment of all *symbols* to True/False

Sentence

- Logical statement
- Composition 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*.

Models and Knowledge Bases

Example: Sudoku

Model

Assignment of values to all variables

Knowledge Base

Collection of things we know to be true

- Rules of the world
- Observations
- Things we have figured out

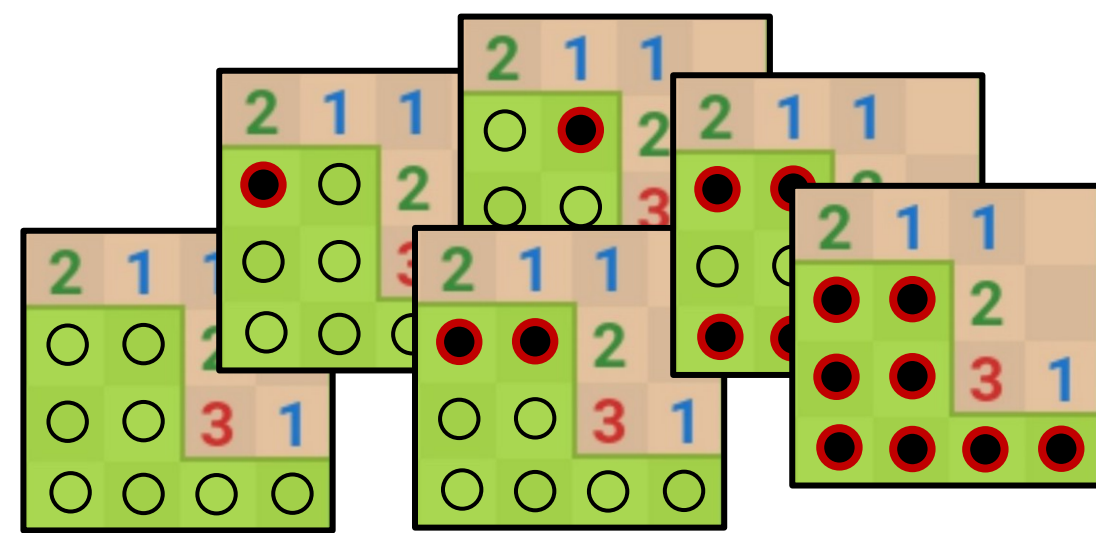
1			
	2	1	
		3	
			4

Models and Knowledge Bases

Example: Minesweeper

Model

Assignment of values to all variables



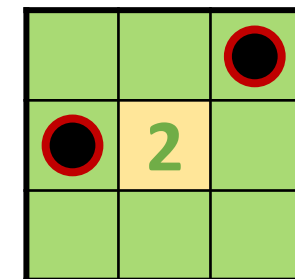
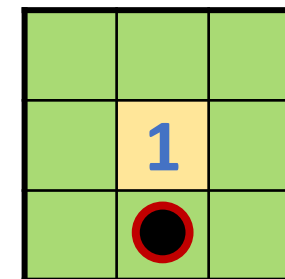
Knowledge Base

Collection of things we know to be true

- Rules of the world
- Observations
- Things we have figured out



Numbers indicate how many mines



Entailment and Satisfiability

More formally

- **Symbol** (variable)
- **Models** (all symbols assigned a value)
- **Satisfiable**: there exists (at least one) model that meets the constraints
- **Entailment**: statement is true for all models that meet the constraints

How do we get a computer to do this?

Entailment and Satisfiability

What reasoning are we doing?

- Can I click here? / Is this definitely safe?
 - Yes: For all possible configurations (models), none of them have a mine in that location
 - No: There exists (at least) one possible configuration with a mine in that location
- Is it possibly safe?
 - Yes: There exists (at least) one possible configuration with a mine in that location
 - No: For all possible configurations (models), all of them have a mine in that location → It's definitely dangerous

Entailment: definitely safe

Satisfiability: possibly not safe

Satisfiability: possibly safe

Entailment: definitely not safe

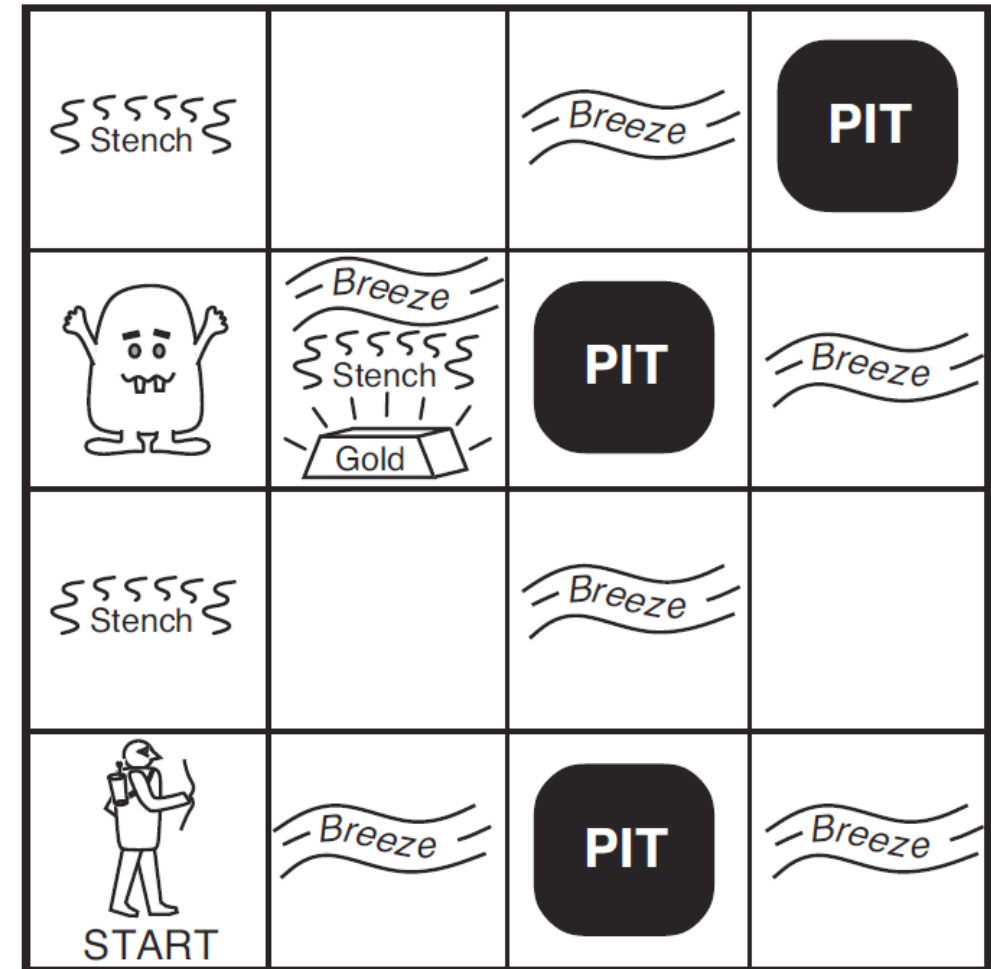
Wumpus World

We collect information as we move to a new grid in the world

- Breeze: if next to a Pit
- Stench: if next to a Wumpus
- Both
- Nothing
- Oh, and there's gold

We're trying to figure out what to do next

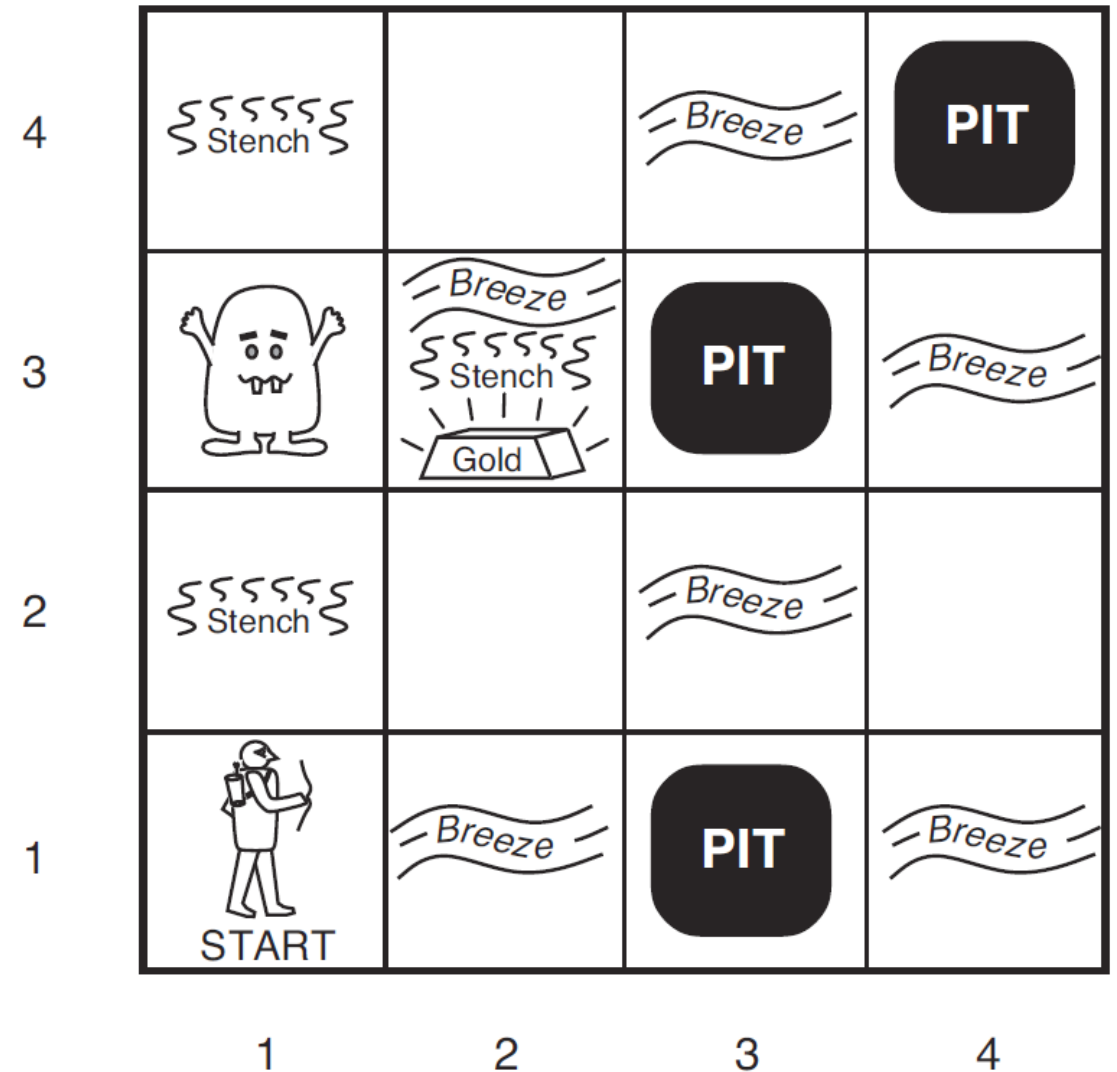
- Which unvisited spaces that are **definitely safe**?
- Which unvisited spaces that are **definitely dangerous**?
- (What about the other spaces?)



Wumpus World

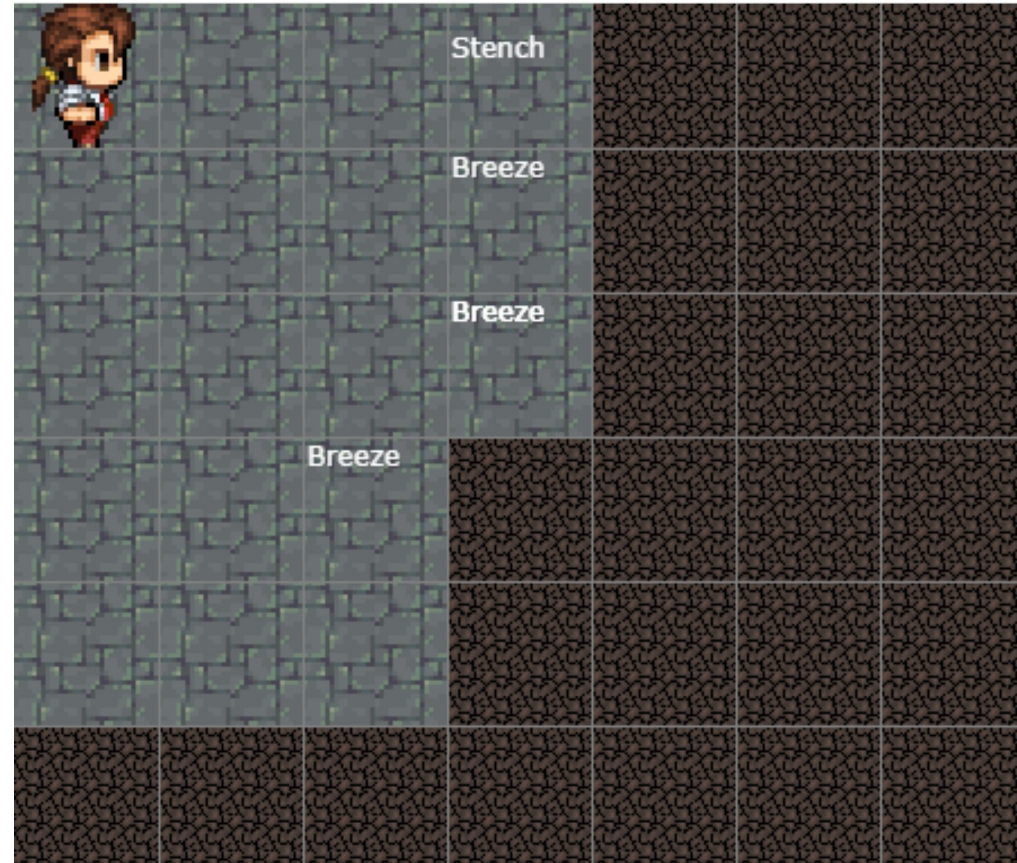
Symbols for Wumpus World

- B_{ij} = breeze felt
- S_{ij} = stench smelt
- P_{ij} = pit here
- W_{ij} = wumpus here
- G = gold



Wumpus World

Reasoning about how to safely get more information!



<http://thiagodnf.github.io/wumpus-world-simulator/>

Plan

Pre-reading:

- Propositional logic
- Models and Knowledge Bases
- Satisfiability and Entailment
- Model checking: Truth table entailment/SAT

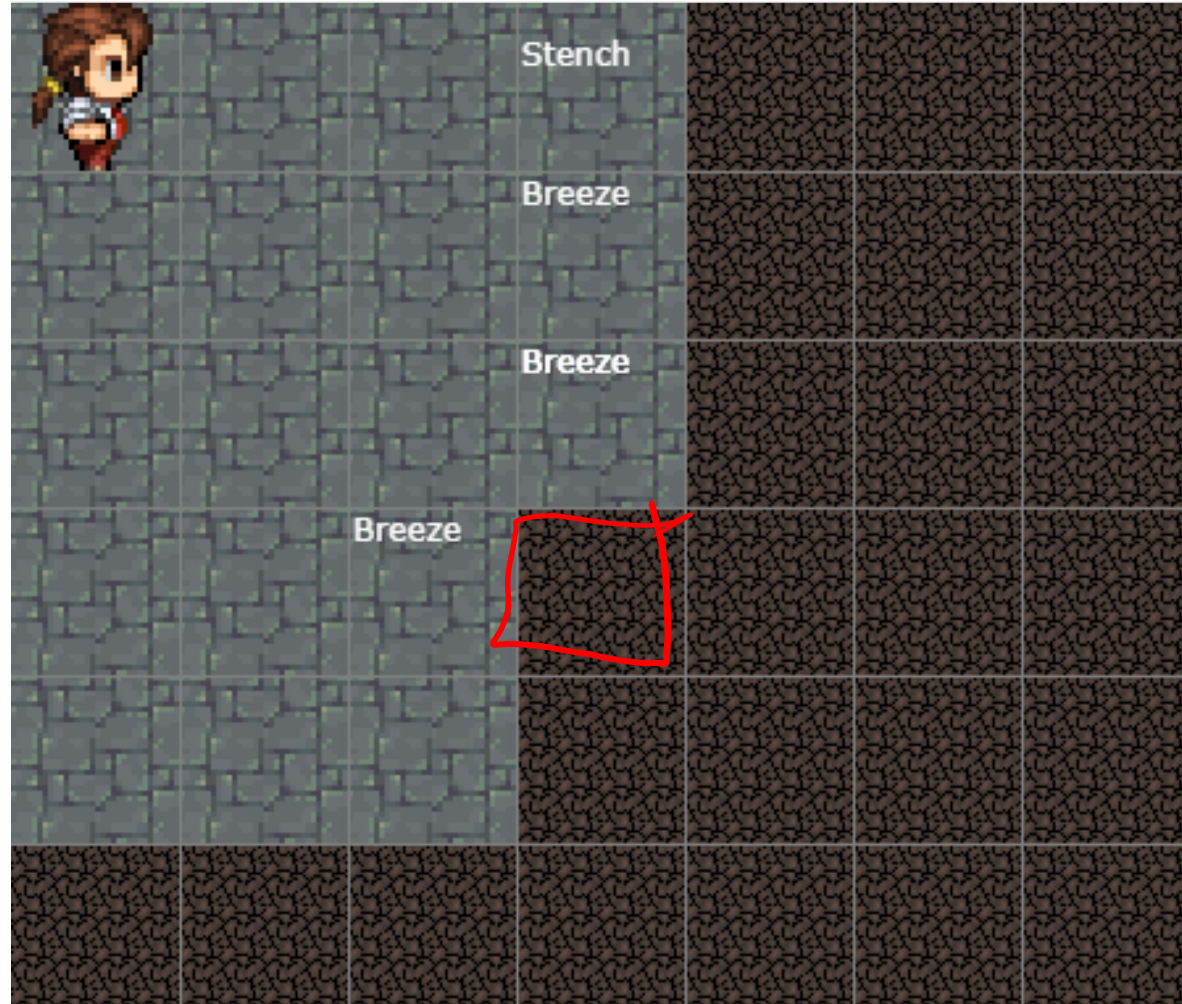
Today: Logical Agent Algorithms

- Model checking
 - Truth table entailment/SAT
 - Faster SAT: DPLL
- Theorem proving: Forward chaining
 - Planning with logic

Slow

FAST

Provably True, Provably False, or Unsure



<http://thiagodnf.github.io/wumpus-world-simulator/>

Logical Agent Vocab

Entailment (KB, φ)

- Input: sentence1, sentence2
- Each model that satisfies sentence1 must also satisfy sentence2
- "If I know 1 holds, then I know 2 holds"
- (ASK), TT-ENTAILS, FC-ENTAILS, RESOLUTION-ENTAILS

OO5cope

Satisfy

- Input: model, sentence
- Is this sentence true in this model?
- Does this model satisfy this sentence
- "Does this particular state of the world work?"
- PL-TRUE

Logical Agent Vocab

Satisfiable

SAT

- Input: sentence
- Can find at least one model that satisfies this sentence
 - (We often want to know what that model is)
- "Is it possible to make this sentence true?"
- DPLL

$KB \vee \neg q$

Valid

vocab

- Input: sentence
- sentence is true in all possible models

Propositional Logic

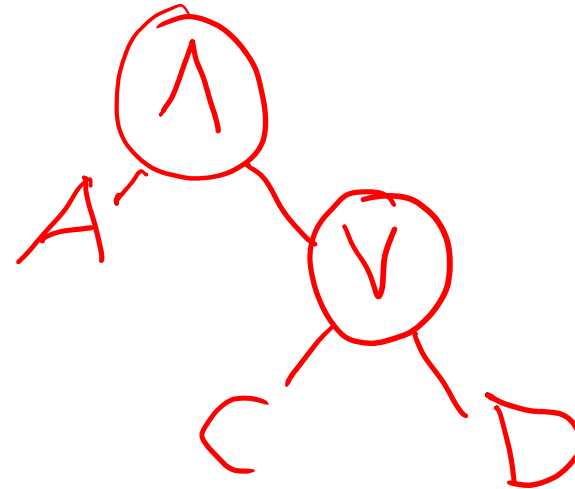
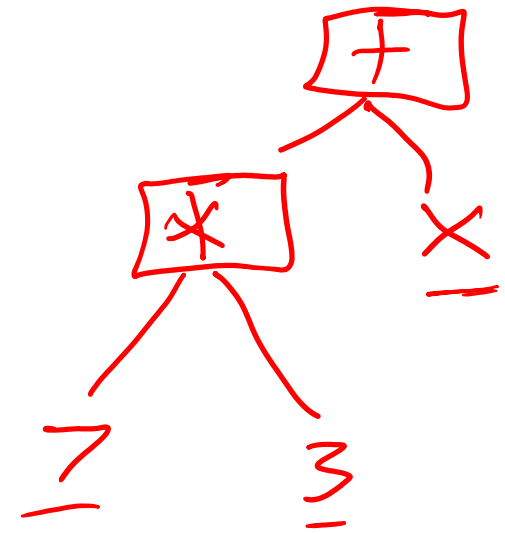
Check if sentence is true in given model

In other words, does the model *satisfy* the sentence?

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

But are models and propositional logic sentences α represented?

$$x + 7 * 3$$



Propositional Logic

Check if sentence is true in given model

In other words, does the model *satisfy* the sentence?

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))

etc.

(Sometimes called “recursion over syntax”)

Plan

Pre-reading:

- Propositional logic
- Models and Knowledge Bases
- Satisfiability and Entailment
- Model checking: Truth table entailment/SAT

Today: Logical Agent Algorithms

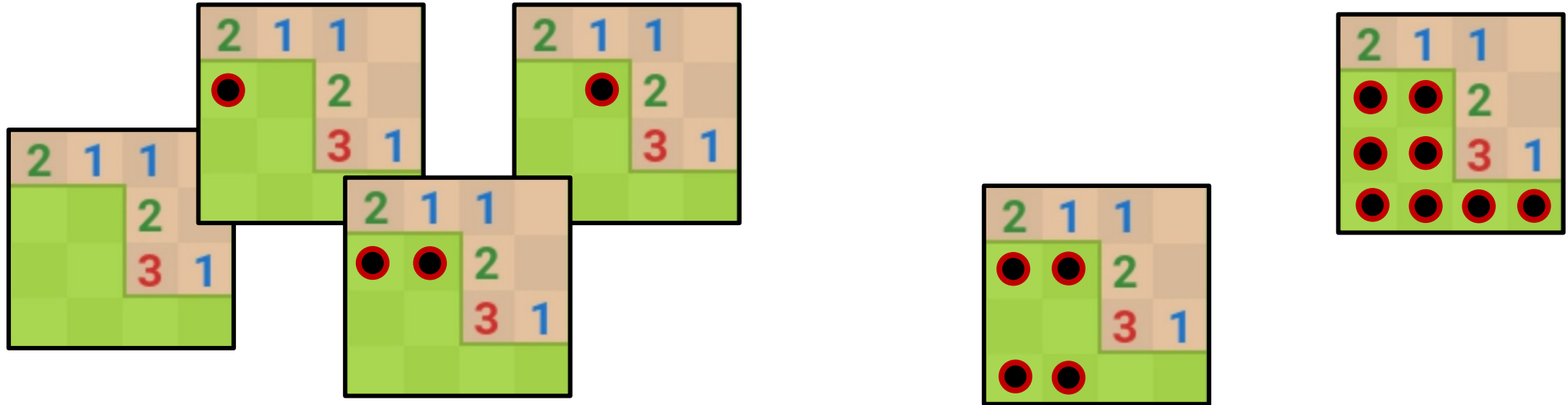
- Model checking
 - Truth table entailment/SAT
 - Faster SAT: DPLL
 - Theorem proving: Forward chaining
 - Planning with logic
- 

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 β is true too
- OK for propositional logic (finitely many worlds); not easy for first-order logic

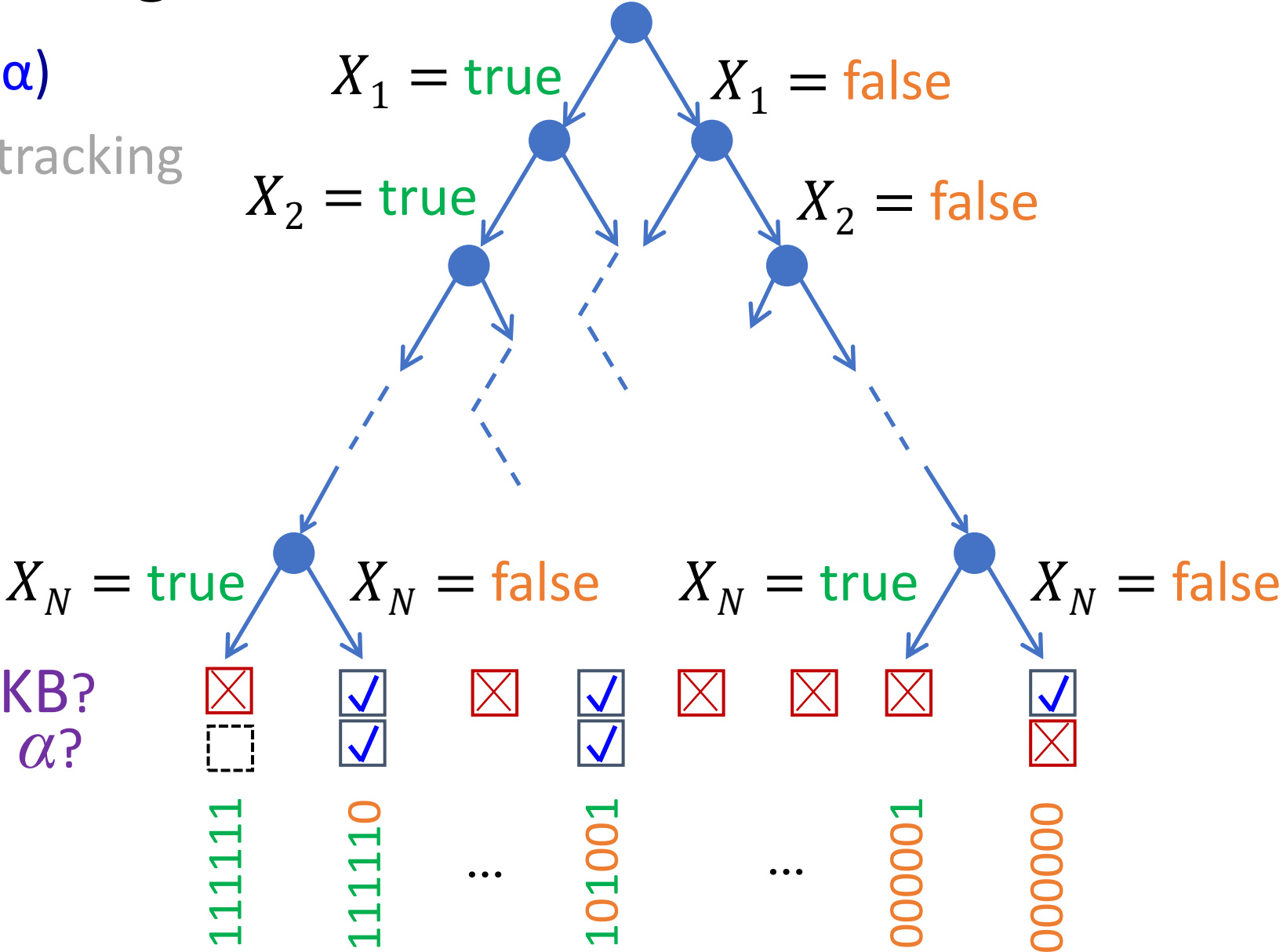


Simple Model Checking

function **TT-ENTAILS?**(KB, α) Returns true or false

Simple Model Checking

```
function TT-ENTAILS?(KB,  $\alpha$ )  
  Same recursion as backtracking
```



Simple Model Checking

function **TT-ENTAILS?**(KB, α) Returns true or false

 return **TT-CHECK-ALL**(KB, α , symbols(KB) $\cup$ symbols(α), {})

function **TT-CHECK-ALL**(KB, α , symbols, model) Returns true or false

 Recursively check to make sure all models
 that satisfy the KB also satisfy α

Simple Model Checking

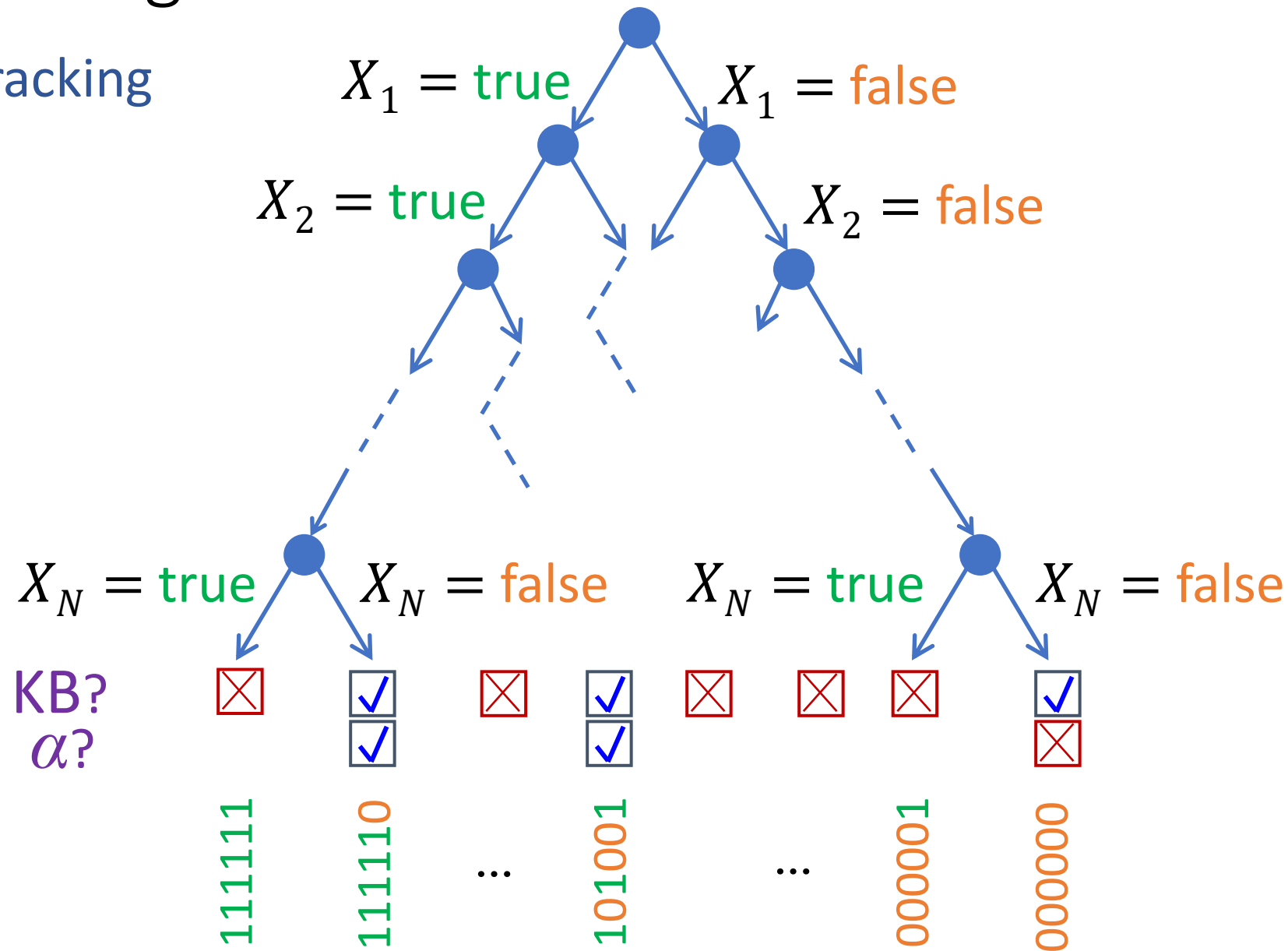
```
function TT-CHECK-ALL(KB,  $\alpha$ , symbols, model)    Returns true or false
    if empty?(symbols) then
        if PL-TRUE?(KB, model) then
            return PL-TRUE?( $\alpha$ , model)
        else
            return true
    else
         $X_i \leftarrow \text{first}(\text{symbols})$ 
        rest  $\leftarrow \text{rest}(\text{symbols})$ 
        return and ( TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup \{X_i = \text{true}\}$ )
                     TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup \{X_i = \text{false}\}$ ) )
```

Simple Model Checking

Same recursion as backtracking

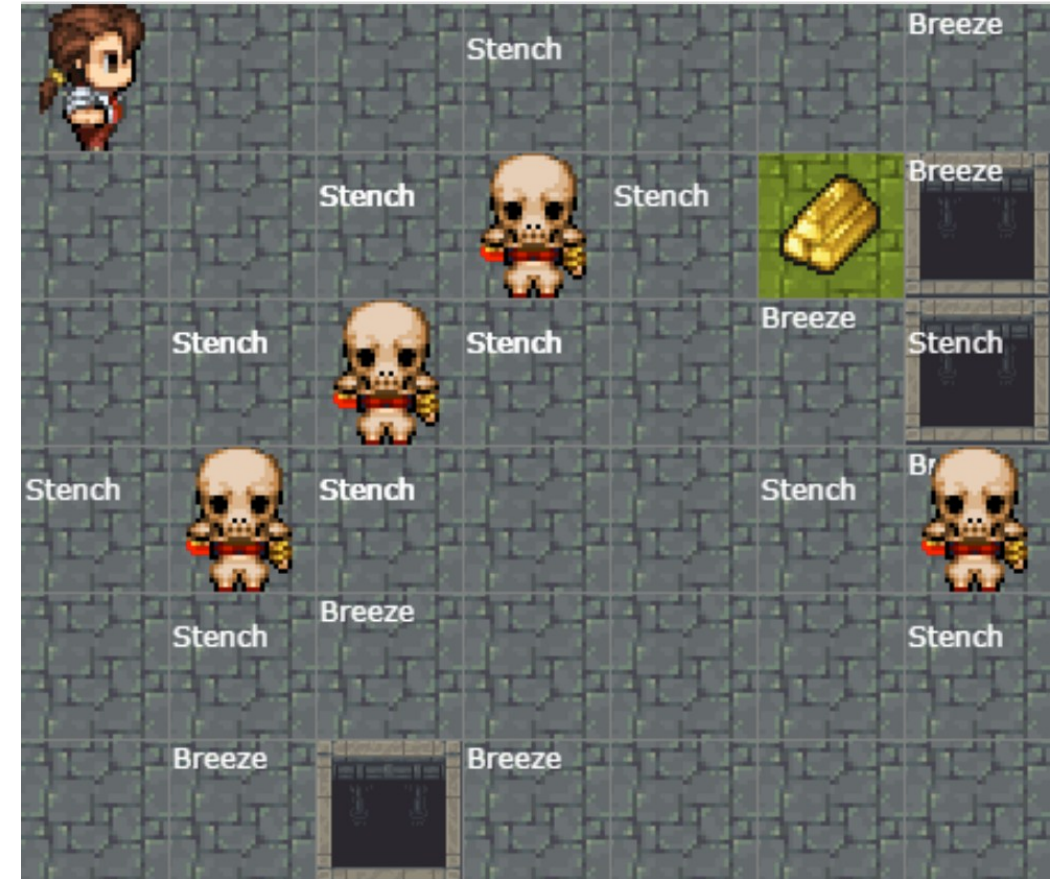
$O(2^N)$ time, linear space

Can we do better?



Satisfiability and Entailment

A sentence is *satisfiable* if it is true in at least one world (e.g. CSPs!)



<http://thiagodnf.github.io/wumpus-world-simulator/>

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*

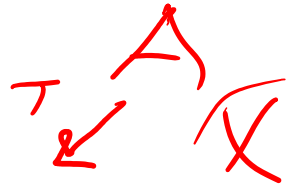
Efficient SAT solvers operate on *conjunctive normal form*

Efficient SAT solvers

DPLL

DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern solvers

Essentially a backtracking search over models with some extras:



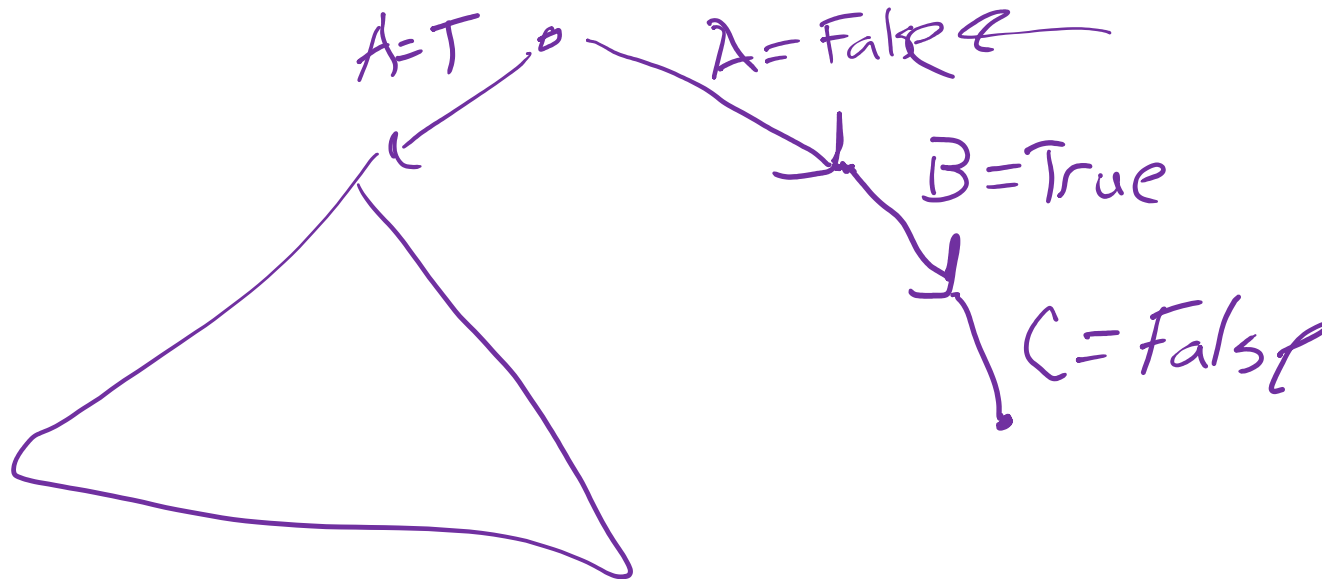
- *Early termination*: stop if
 - all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by $\{A=\text{true}\}$
 - any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by $\{A=\text{false}, B=\text{false}\}$
- *Pure literals*: if all occurrences of 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 $A=\text{false}$, $(A \vee B) \wedge (A \vee \neg C)$ 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.

Efficient SAT solvers

DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern solvers

Essentially a backtracking search over models with some extras:

- **Unit clauses**: if a clause is left with a single literal, set symbol to satisfy clause
 - E.g., if $A = \text{false}$, $(A \vee B) \wedge (A \vee \neg C)$ 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 $\leftarrow$ **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})))

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 β is 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*

Properties

- *Sound* algorithm: everything it claims to prove is in fact entailed
- *Complete*  algorithm: every sentence that is entailed can be proved

Inference Rules

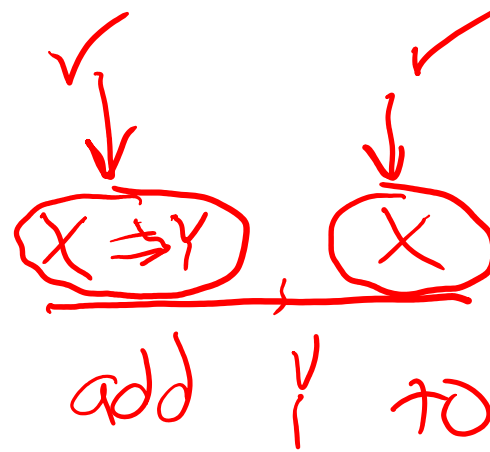
Notation Alert!

Modus Ponens

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

→ Forward Checking

KB



Out of Scope

Unit Resolution

$$\frac{a \vee b, \quad \neg b \vee c}{a \vee c}$$

General Resolution

$$\frac{a_1 \vee \cdots \vee a_m \vee b, \quad \neg b \vee c_1 \vee \cdots \vee c_n}{a_1 \vee \cdots \vee a_m \vee c_1 \vee \cdots \vee c_n}$$

Simple Theorem Proving: Forward Chaining

Forward chaining applies **Modus Ponens** to generate new facts:

- Given $X_1 \wedge X_2 \wedge \dots \wedge X_n \Rightarrow Y$ and $X_1, X_2, \dots, X_n$
- 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 $\text{True} \Rightarrow X$)

Forward Chaining Algorithm

function **PL-FC-ENTAILS?**(KB, q) Returns true or false

KB CLAUSES

Queue

$P \Rightarrow Q$

~~$L \wedge M \Rightarrow P$~~

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

~~$A \wedge B \Rightarrow L$~~

~~A~~

~~B~~

Model so far

A:T

B:T

L:T

M:T

P:T

Q:T

Forward Chaining Algorithm

function **PL-FC-ENTAILS?**(KB, q) returns true or false

count $\leftarrow$ a table, where count[c] is the number of symbols in c's premise

inferred $\leftarrow$ a table, where inferred[s] is initially false for all s

agenda $\leftarrow$ a queue of symbols, initially symbols known to be true in KB

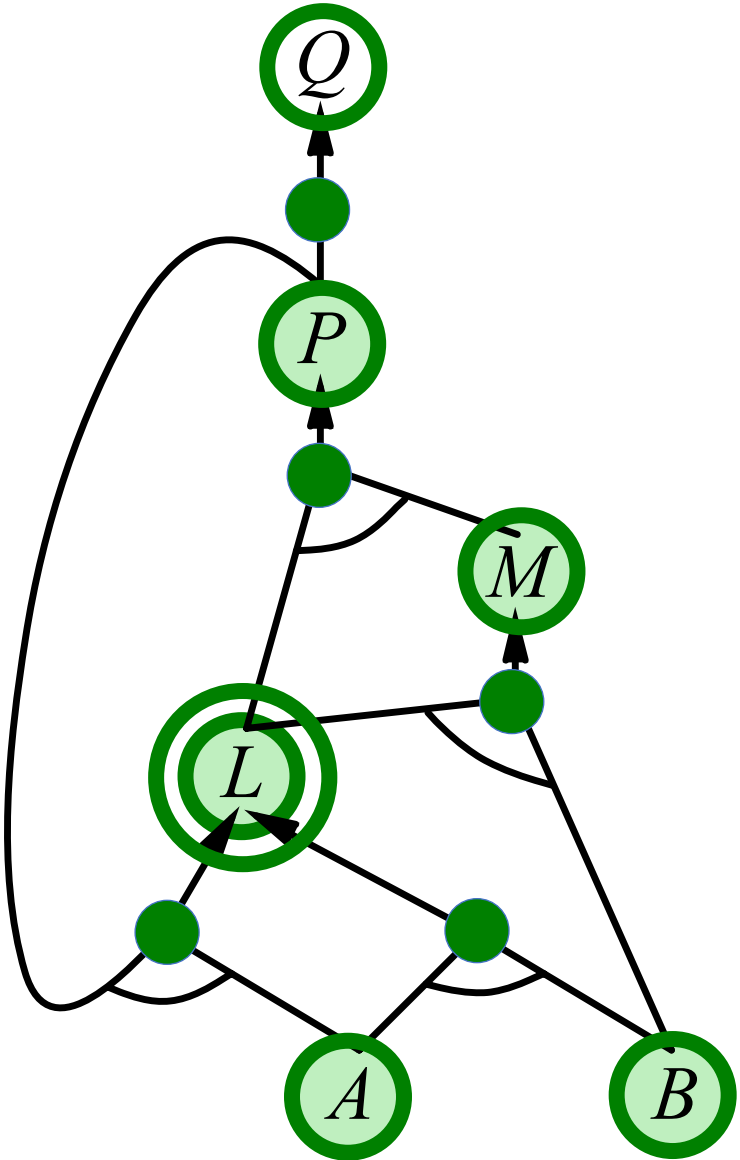
<i>CLAUSES</i>	<i>COUNT</i>	<i>INFERRED</i>	<i>AGENDA</i>
$P \Rightarrow Q$	1	A false	
$L \wedge M \Rightarrow P$	2	B false	
$B \wedge L \Rightarrow M$	2	L false	
$A \wedge P \Rightarrow L$	2	M false	
$A \wedge B \Rightarrow L$	2	P false	
A	0	Q false	
B	0		

Forward Chaining Example: Proving Q

<i>CLAUSES</i>	<i>COUNT</i>	<i>INFERRED</i>
$P \Rightarrow Q$	~~1~~ /0	A ~~false~~ true
$L \wedge M \Rightarrow P$	~~2~~ / ~~1~~ /0	B ~~false~~ true
$B \wedge L \Rightarrow M$	~~2~~ / ~~1~~ /0	L ~~false~~ true
$A \wedge P \Rightarrow L$	~~2~~ / ~~1~~ /0	M ~~false~~ true
$A \wedge B \Rightarrow L$	~~2~~ / ~~1~~ /0	P ~~false~~ true
A	0	Q ~~false~~ true
B	0	

AGENDA

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



Forward Chaining Algorithm

function **PL-FC-ENTAILS?**(KB, q) returns true or false

count $\leftarrow$ a table, where count[c] is the number of symbols in c's premise

inferred $\leftarrow$ a table, where inferred[s] is initially false for all s

agenda $\leftarrow$ a queue of symbols, initially symbols known to be true in KB

while agenda is not empty do

 p $\leftarrow$ Pop(agenda)

 if p = q then return true

 if inferred[p] = false then

 inferred[p] $\leftarrow$ 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

Properties

Forward Chaining is:

- Sound and complete for definite-clause KBs
- Complexity: linear time 😊

Resolution is another theorem-proving algorithm that is:

- Sound and complete for any PL KBs!
- Complexity: exponential time 😞

Vocab Reminder

Literal

- Atomic sentence:
 $T, F, \text{Symbol}, \neg \text{Symbol}$

Clause

- Disjunction of literals:
 $A \vee B \vee \neg C$

Definite clause

- Disjunction of literals, *exactly one* is positive
 $\neg A \vee B \vee \neg C$

Next Time

Planning with PL and SAT

Additional Slides

Theorem Proving: Resolution (out of scope)

Resolution

Algorithm Overview

function PL-RESOLUTION?(KB, α) returns true or false

We want to prove that KB entails α

In other words, we want to prove that we cannot satisfy (KB and **not** α)

1. Start with a set of CNF clauses, including the KB as well as $\neg \alpha$
2. Keep resolving pairs of clauses until

A. You resolve the empty clause

Contradiction found!

KB $\wedge \neg \alpha$ cannot be satisfied

Return true, KB entails α

B. No new clauses added

Return false, KB does not entail α

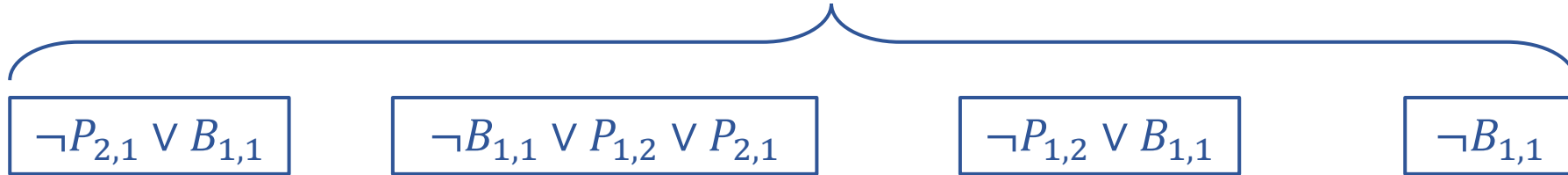
Resolution

Example trying to prove $\neg P_{1,2}$

General Resolution

$$\frac{a_1 \vee \dots \vee a_m \vee b, \quad \neg b \vee c_1 \vee \dots \vee c_n}{a_1 \vee \dots \vee a_m \vee c_1 \vee \dots \vee c_n}$$

Knowledge Base



$$\neg \neg P_{1,2}$$

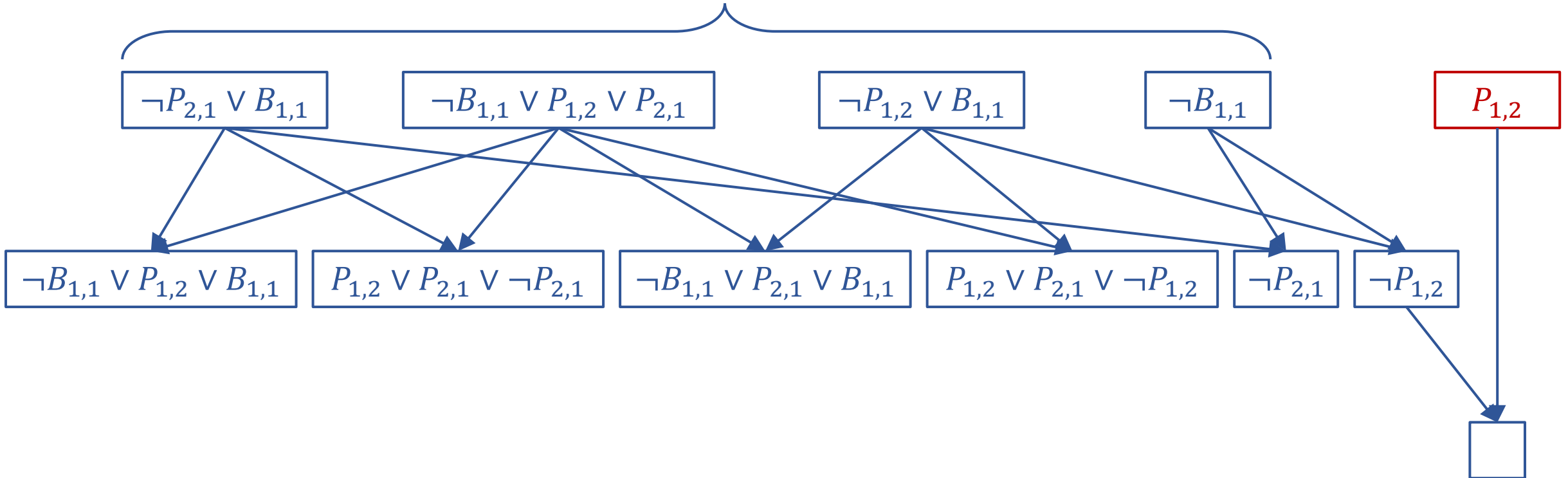
Resolution

Example trying to prove $\neg P_{1,2}$

General Resolution

$$\frac{a_1 \vee \dots \vee a_m \vee b, \quad \neg b \vee c_1 \vee \dots \vee c_n}{a_1 \vee \dots \vee a_m \vee c_1 \vee \dots \vee c_n}$$

Knowledge Base



Resolution

function PL-RESOLUTION?(KB, α) returns true or false

clauses $\leftarrow$ the set of clauses in the CNF representation of $\text{KB} \wedge \neg\alpha$

new $\leftarrow \{ \}$

loop do

for each pair of clauses C_i, C_j in clauses do

resolvents $\leftarrow$ PL-RESOLVE(C_i, C_j)

if resolvents contains the empty clause then

return true

new $\leftarrow$ new $\cup$ resolvents

if new $\subseteq$ clauses then

return false

clauses $\leftarrow$ clauses $\cup$ new