Warm-up:

What is the relationship between number of constraints and number of possible solutions?

In other words, as the number of the constraints increases,

does the number of possible solutions:

A) Increase
B) Decrease
C) Stay the same

Announcements

Assignments:

- HW5 (written)
  - Due Tues 10/10, 10 pm
- P2: Optimization
  - Due Thurs 10/5, 10 pm
- P3: Logic and Classical Planning
  - Out Thursday
  - FIRST HALF!! Due Friday 10/13, 10 pm
  - ALL!! Due Friday 10/27, 10 pm
  - Grading

AI: Representation and Problem Solving

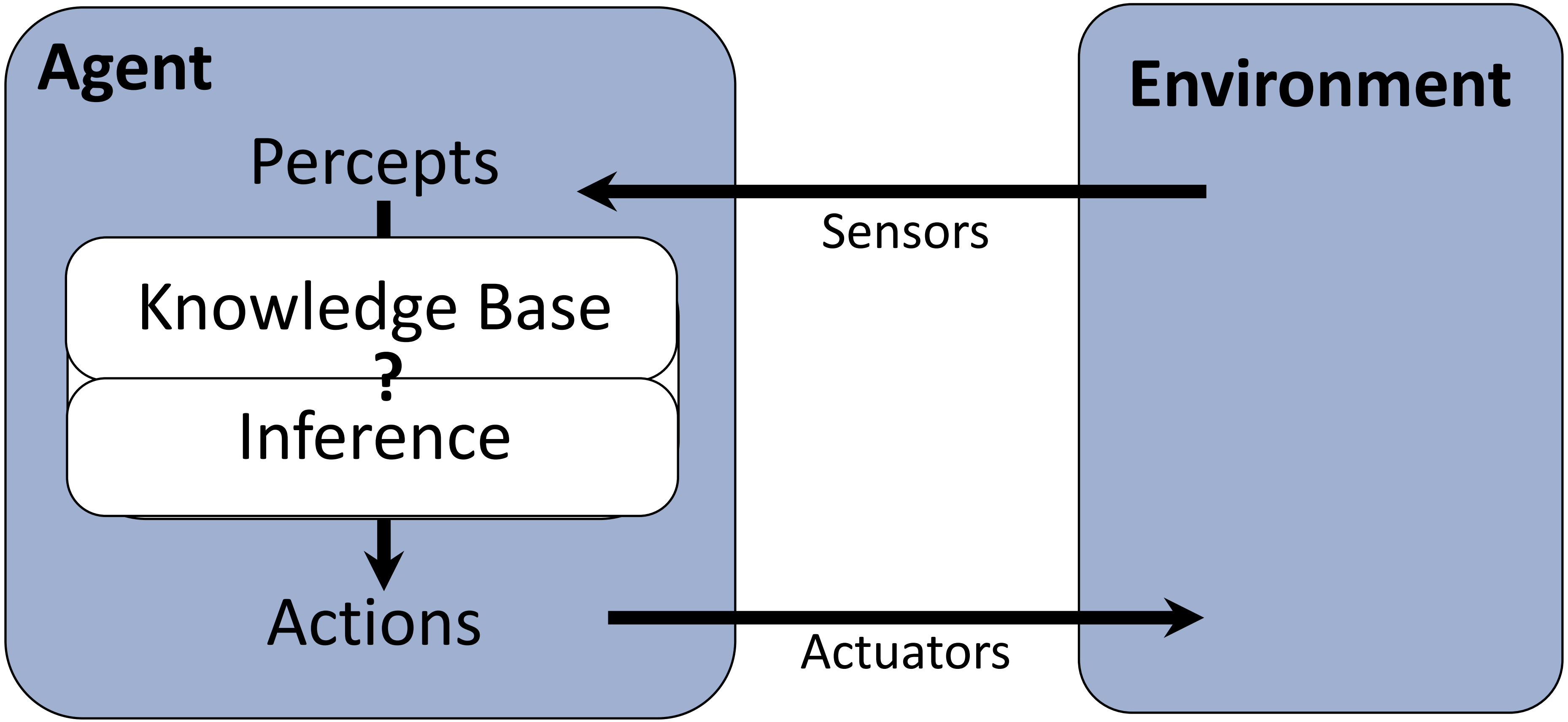
Propositional Logic and Logical Agents

Instructors: Vincent Conitzer and Aditi Raghunathan

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

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)
  - Vince ate too much candy today

What happens when we ASK the agent?

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

Models

How do we represent possible worlds with models and knowledge bases?

How do we then do inference with these representations?

Logic Language

Natural language?

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 is α 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_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.

Propositional Logic

Propositional Logic

Symbol:

- Variables that can be true or false
- We'll try to use capital letters, e.g. A, B, $P_{1,2}$
- Often include True and False

Operators:

- $\neg$ A: not A
- A $\wedge$ B: A and B (conjunction)
- A $\vee$ B: A or B (disjunction) Note: this is not an "exclusive or"
- A $\Rightarrow$ B: A implies B (implication). If A then B
- A $\Leftrightarrow$ B: A if and only if B (biconditional)

Sentences

Propositional Logic Syntax

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 Logical Vocab

Vocab Alert!

Literal

- Atomic sentence: True, False, Symbol, ¬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

Notes on Operators

$\boldsymbol{\alpha} \vee \boldsymbol{\beta}$ is inclusive or, not exclusive

Truth Tables

$\alpha \vee \beta$ is inclusive or, not exclusive

α	β	$\alpha \wedge \beta$
F	F	F
F	T	F
T	F	F
T	T	T

α	β	$\alpha \vee \beta$
F	F	F
F	T	T
T	F	T
T	T	T

Notes on Operators

$\boldsymbol{\alpha} \vee \boldsymbol{\beta}$ is inclusive or, not exclusive

$\boldsymbol{\alpha} \Rightarrow \boldsymbol{\beta}$ is equivalent to $\neg\boldsymbol{\alpha} \vee \boldsymbol{\beta}$

- Says who?

Truth Tables

$\alpha \Rightarrow \beta$ is equivalent to $\neg\alpha \vee \beta$

α	β	$\alpha \Rightarrow \beta$	$\neg\alpha$	$\neg\alpha \vee \beta$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	F
T	T	T	F	T

Notes on Operators

$\alpha \lor \beta$ is inclusive or, not exclusive

$\alpha \Rightarrow \beta$ is equivalent to $\neg\alpha \lor \beta$

- Says who?

$\alpha \Leftrightarrow \beta$ is equivalent to $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$

- Prove it!

Truth Tables

$\alpha \Leftrightarrow \beta$ is equivalent to $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$

α	β	$\alpha \Leftrightarrow \beta$	$\alpha \Rightarrow \beta$	$\beta \Rightarrow \alpha$	$(\alpha\Rightarrow\beta) \land (\beta\Rightarrow\alpha)$
F	F	T	T	T	T
F	T	F	T	F	F
T	F	F	F	T	F
T	T	T	T	T	T

Equivalence: it's true in all models. Expressed as a logical sentence:

$$(\alpha \Leftrightarrow \beta) \Leftrightarrow [(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)]$$

Poll 1

If we know that $A \vee B$ and $\neg B \vee C$ are true,

what do we know about $A \vee C$?

i. $A \vee C$ is guaranteed to be true
ii. $A \vee C$ is guaranteed to be false
iii. We don’t have enough information to say anything definitive about $A \vee C$

Poll 1

If we know that $A \vee B$ and $\neg B \vee C$ are true, what do we know about $A \vee C$?

A	B	C	$A \vee B$	$\neg B \vee C$	$A \vee C$
false	false	false	false	true	false
false	false	true	false	true	true
false	true	false	true	false	false
false	true	true	true	true	true
true	false	false	true	true	true
true	false	true	true	true	true
true	true	false	true	false	true
true	true	true	true	true	true

Poll 1

If we know that $A \lor B$ and $\neg B \lor C$ are true, what do we know about $A \lor C$?

A	B	C	$A \lor B$	$\neg B \lor C$	$A \lor C$
false	false	false	false	true	false
false	false	true	false	true	true
false	true	false	true	false	false
false	true	true	true	true	true
true	false	false	true	true	true
true	false	true	true	true	true
true	true	false	true	false	true
true	true	true	true	true	true

Poll 1

If we know that $A \vee B$ and $\neg B \vee C$ are true,

what do we know about $A \vee C$?

i. $A \vee C$ is guaranteed to be true

ii. $A \vee C$ is guaranteed to be false

iii. We don't have enough information to say anything definitive about $A \vee C$

Poll 2

If we know that $A \vee B$ and $\neg B \vee C$ are true,

what do we know about A?

i. A is guaranteed to be true

ii. A is guaranteed to be false

iii. We don't have enough information to say anything definitive about A

Poll 2

If we know that $A \vee B$ and $\neg B \vee C$ are true, what do we know about A?

A	B	C	$A \vee B$	$\neg B \vee C$	$A \vee C$
false	false	false	false	true	false
false	false	true	false	true	true
false	true	false	true	false	false
false	true	true	true	true	true
true	false	false	true	true	true
true	false	true	true	true	true
true	true	false	true	false	true
true	true	true	true	true	true

Poll 2

If we know that $A \vee B$ and $\neg B \vee C$ are true,

what do we know about A?

i. A is guaranteed to be true
ii. A is guaranteed to be false
iii. We don't have enough information to say anything definitive about A

Logic Representation of World Models

- Knowledge Base of things we know to be true (logical sentences):

  $P \vee (\neg Q \wedge R)$; $X_1 \Leftrightarrow (\text{Raining} \Rightarrow \text{Sunny})$

- Possible world model (assignment of variables to values):

  {P=true, Q=true, R=false, S=true} or 1101

- Semantics: $\alpha \wedge \beta$ is true in a world iff is α true and β is true (etc.)

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")

Sentences as Constraints

Adding a sentence to our knowledge base constrains the number of possible models:

KB: Nothing

Possible Models

P	Q	R
false	false	false
false	false	true
false	true	false
false	true	true
true	false	false
true	false	true
true	true	false
true	true	true

Sentences as Constraints

Adding a sentence to our knowledge base constrains the number of possible models:

KB: Nothing

KB: $[(P \land \neg Q) \lor (Q \land \neg P)] \Rightarrow R$

Possible Models

P	Q	R
false	false	false
false	false	true
~~false~~	~~true~~	~~false~~
false	true	true
~~true~~	~~false~~	~~false~~
true	false	true
true	true	false
true	true	true

Sentences as Constraints

Adding a sentence to our knowledge base constrains the number of possible models:

KB: Nothing

KB: $[(P \land \neg Q) \lor (Q \land \neg P)] \Rightarrow R$

KB: R, $[(P \land \neg Q) \lor (Q \land \neg P)] \Rightarrow R$

Possible Models

P	Q	R
~~false~~	~~false~~	~~false~~
false	false	true
~~false~~	~~true~~	~~false~~
false	true	true
~~true~~	~~false~~	~~false~~
true	false	true
~~true~~	~~true~~	~~false~~
true	true	true

Sherlock Entailment

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth" – *Sherlock Holmes via Sir Arthur Conan Doyle*

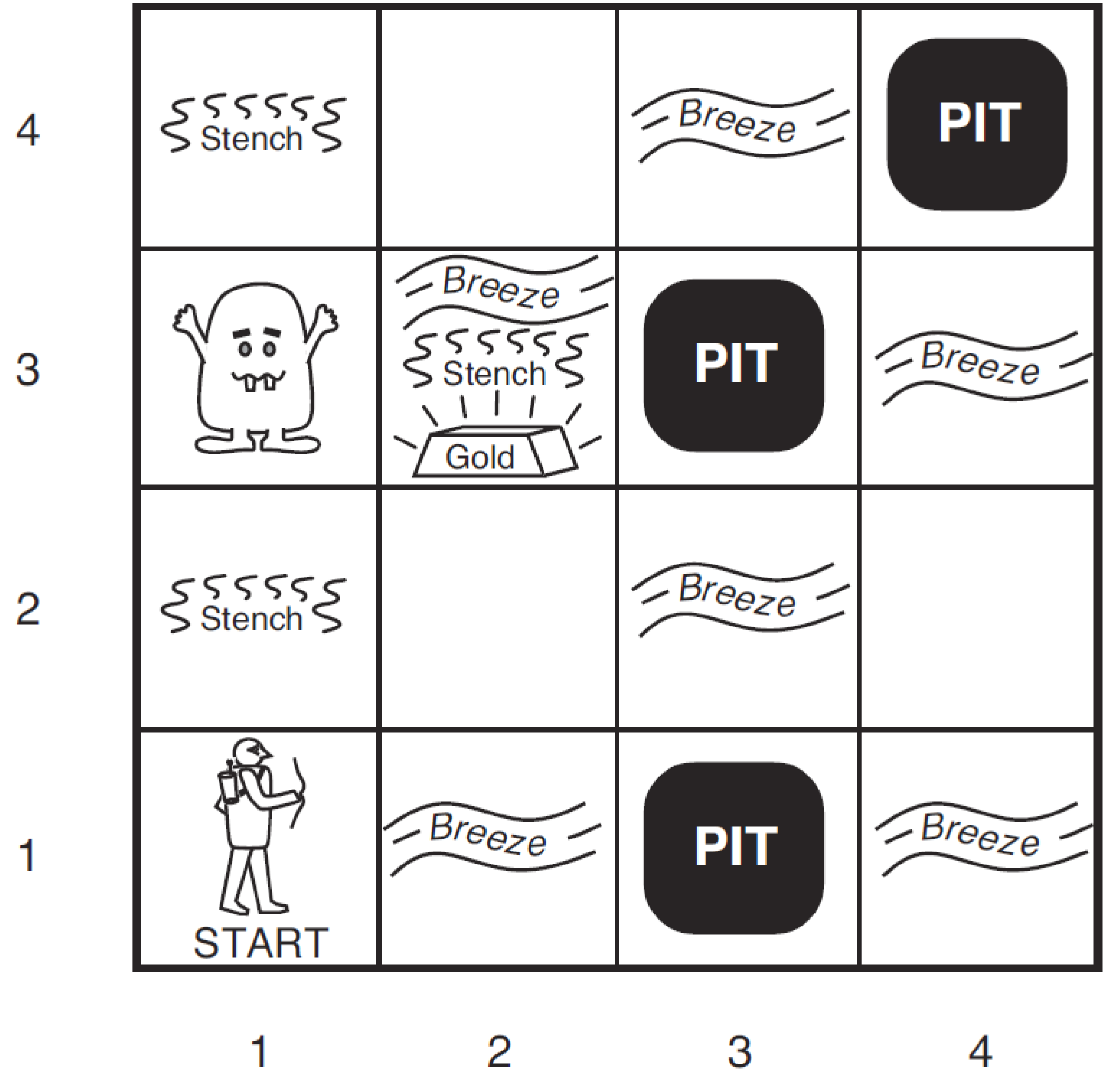
(Not quite)

- Knowledge base and inference allow us to remove impossible models, helping us to see what is true in all of the remaining models

Wumpus World

Logical Reasoning as a CSP

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

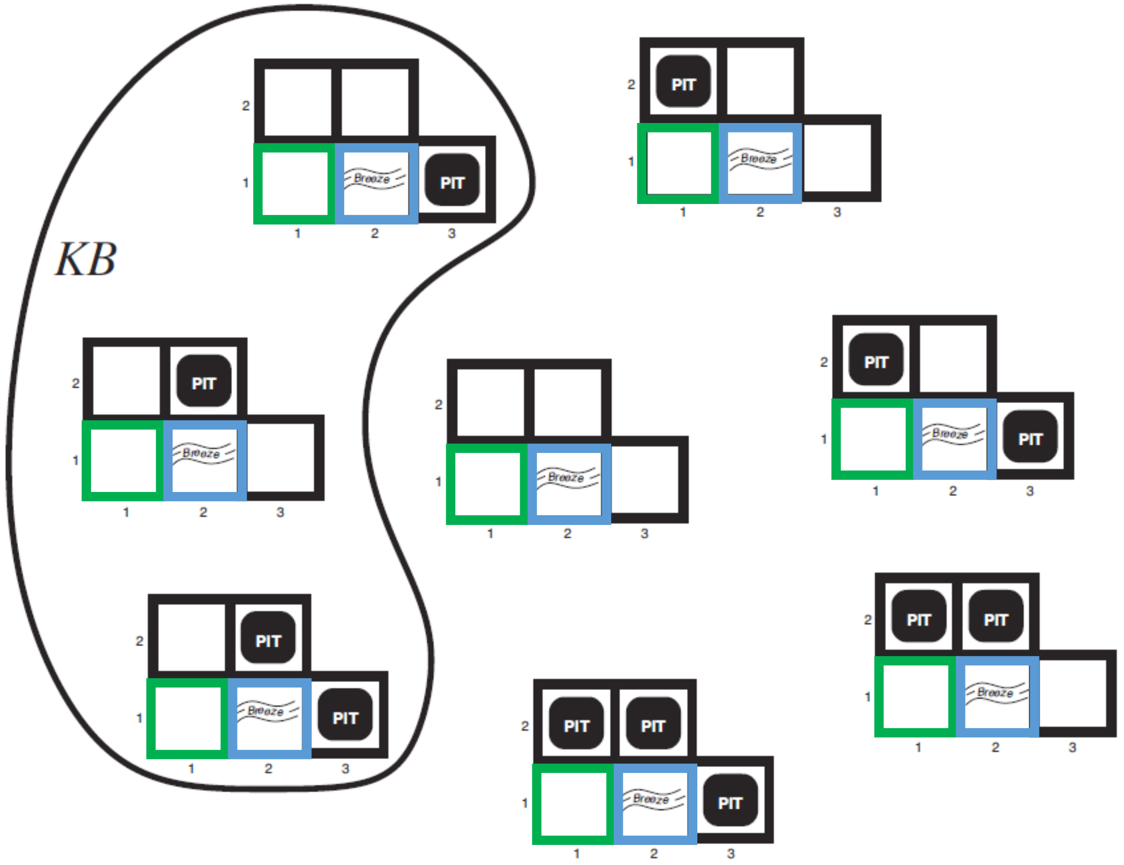


http://thiagodnf.github.io/wumpus-world-simulator/

Wumpus World

Possible Models

- $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
- Knowledge base
  - Breeze ⇒ Adjacent Pit
  - Nothing in [1,1]
  - Breeze in [2,1]



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)]

Usually, we want to know if *KB* $\models$ *query*

- *models*(*KB*) $\subseteq$ *models*(*query*)
- In other words
  - *KB* removes all impossible models (any model where *KB* is false)
  - If *query* is true in all of these remaining models, we conclude that *query* must be true

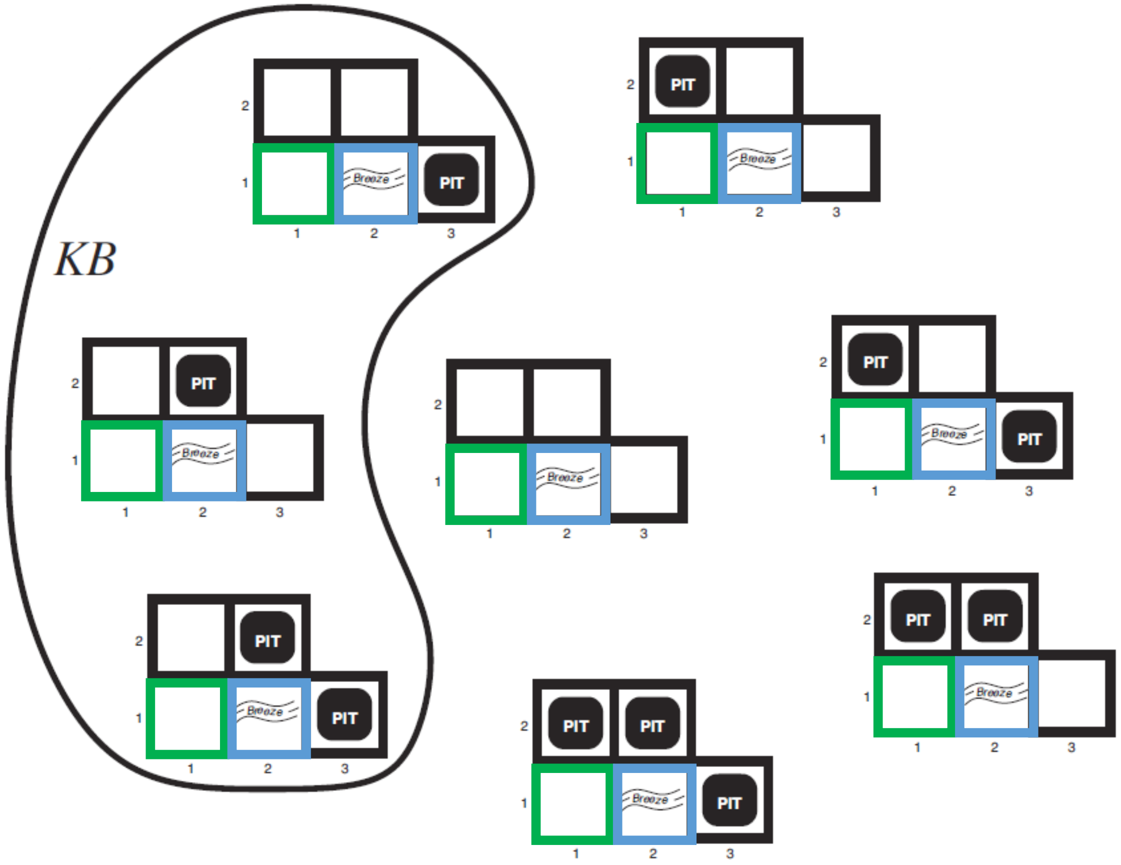
Entailment and implication are very much related

- However, entailment relates two sentences, while an implication is itself a sentence (usually derived via inference to show entailment)

Wumpus World

Possible Models

- $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
- Knowledge base
  - Breeze ⇒ Pit Adjacent
  - Pit ⇒ Breeze in all Adjacent
  - Nothing in [1,1]
  - Breeze in [2,1]



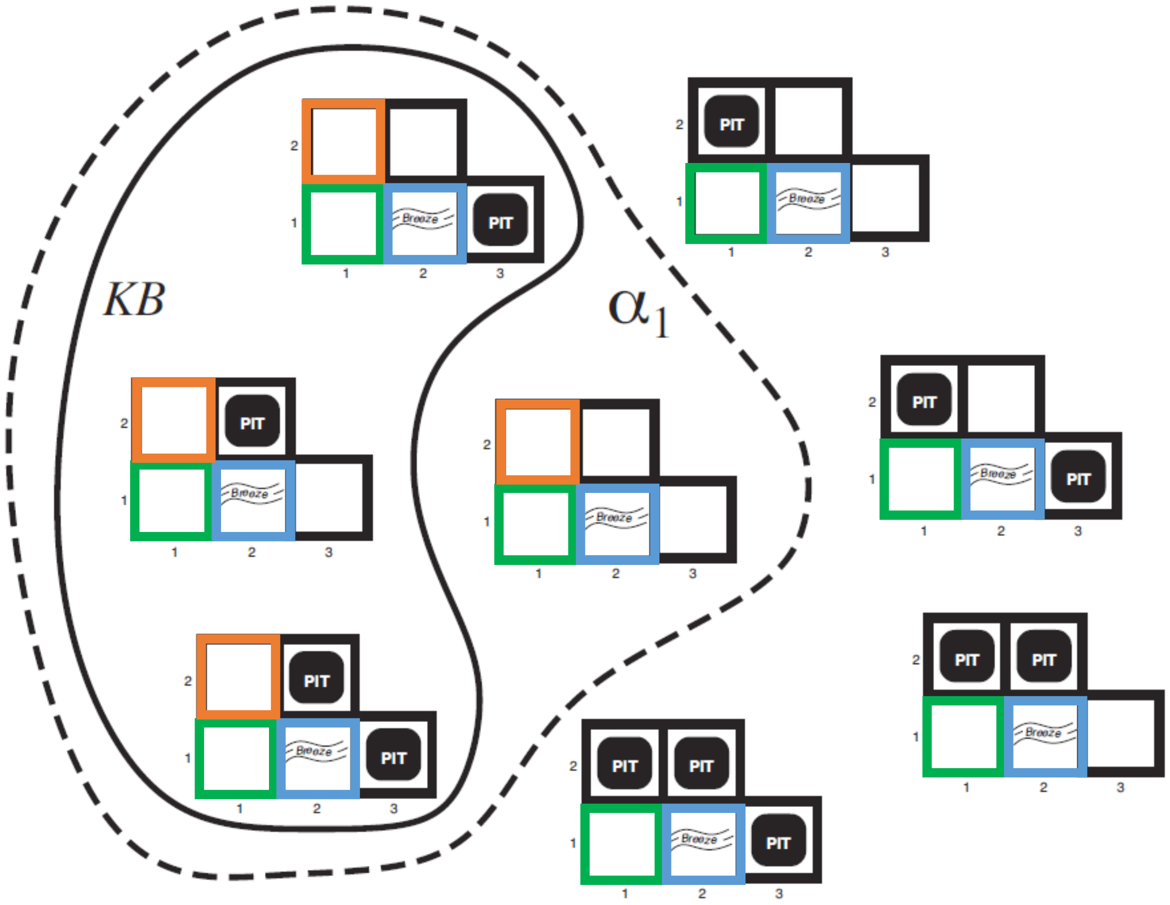
Entailment: KB $|= \alpha$

"KB entails α" iff in every world where KB is true, α is also true

Wumpus World

Possible Models

- $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
- Knowledge base
    - Breeze ⇒ Adjacent Pit
    - Pit ⇒ Breeze in all Adjacent
    - Nothing in [1,1]
    - Breeze in [2,1]
- Query α_1:
    - No pit in [1,2]



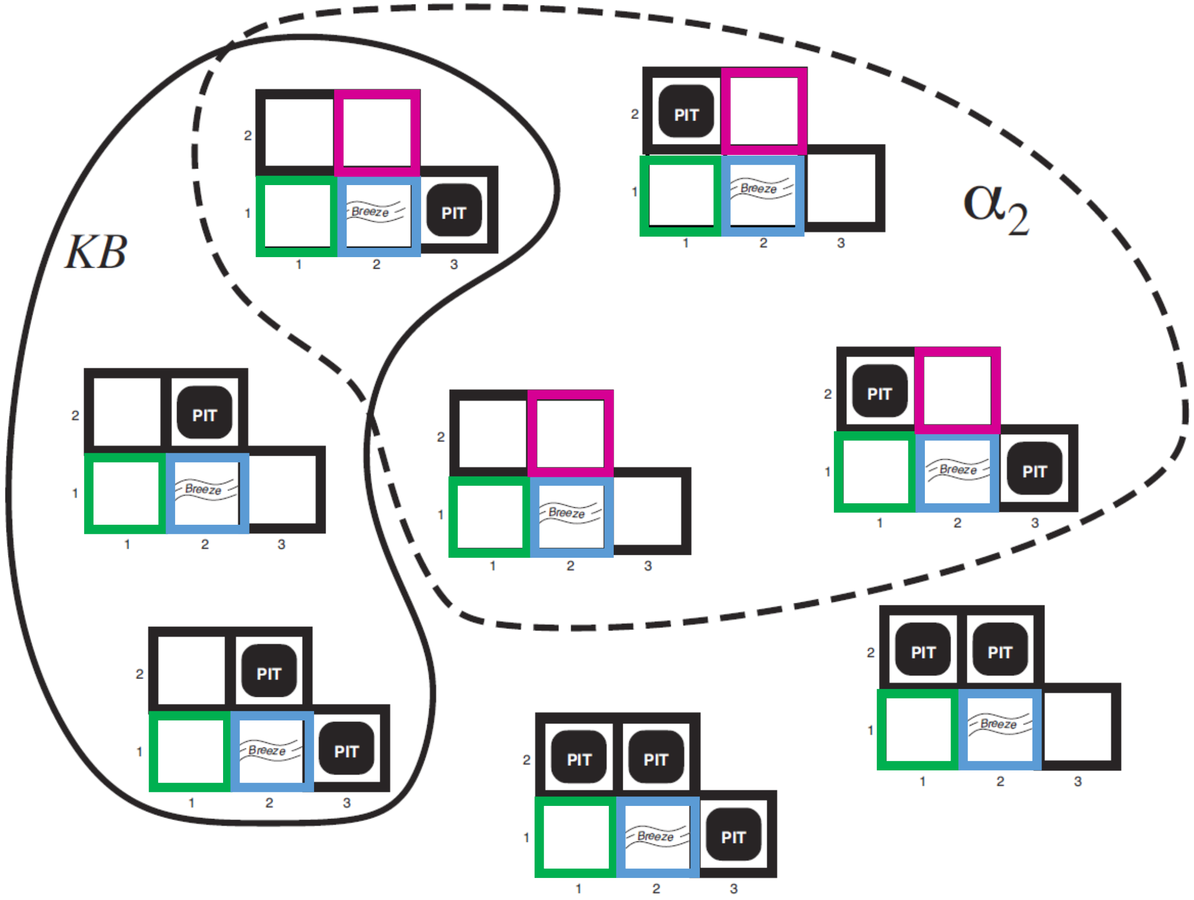
Entailment: KB |= α

"KB entails α" iff in every world where KB is true, α is also true

Wumpus World

Possible Models

- $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
- Knowledge base
  - Breeze ⇒ Adjacent Pit
  - Pit ⇒ Breeze in all Adjacent
  - Nothing in [1,1]
  - Breeze in [2,1]
- Query α_2:
  - No pit in [2,2]



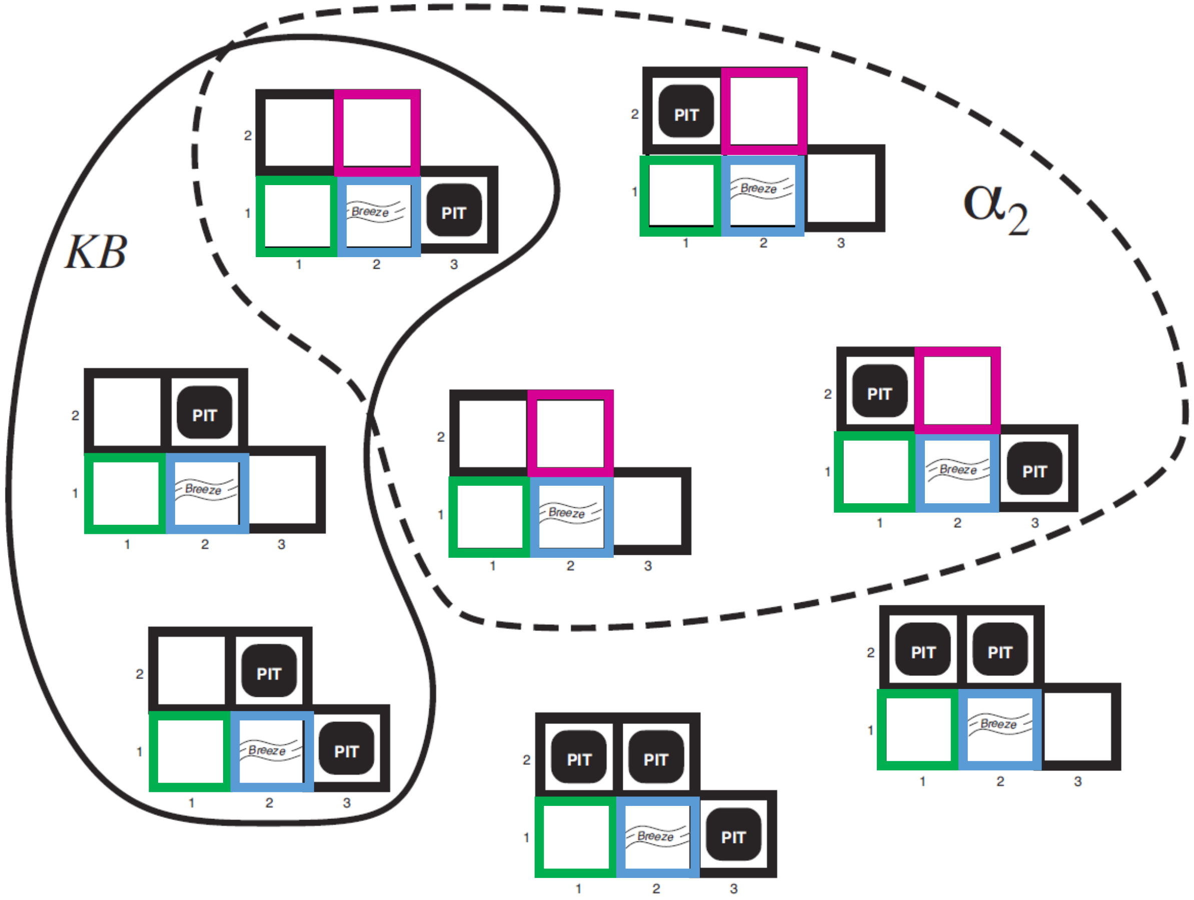
Entailment: KB |= α

"KB entails α" iff in every world where KB is true, α is also true

Wumpus World

Possible Models

- $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
- Knowledge base
  - Breeze ⇒ Adjacent Pit
  - Pit ⇒ Breeze in all Adjacent
  - Nothing in [1,1]
  - Breeze in [2,1]
- Query α_2:
  - No pit in [2,2] – UNSURE!!



Entailment: KB |= α

"KB entails α" iff in every world where KB is true, α is also true

Propositional Logic Models

All Possible Models

Model Symbols

A	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1
C	0	1	0	1	0	1	0	1

Poll 3

Does the KB entail query C?

Entailment: $\alpha \models \beta$

"α entails β" iff in every world where α is true, β is also true

All Possible Models

Model Symbols	A	0	0	0	0	1	1	1	1
	B	0	0	1	1	0	0	1	1
	C	0	1	0	1	0	1	0	1
Knowledge Base	A	0	0	0	0	1	1	1	1
	$B \Rightarrow C$	1	1	0	1	1	1	0	1
	$A \Rightarrow B \vee C$	1	1	1	1	0	1	1	1
Query	C	0	1	0	1	0	1	0	1

Poll 3

Does the KB entail query C?

Yes!

Entailment: $\alpha \models \beta$

"α entails β" iff in every world where α is true, β is also true

All Possible Models

Model Symbols	A	0	0	0	0	1	1	1	1
	B	0	0	1	1	0	0	1	1
	C	0	1	0	1	0	1	0	1
Knowledge Base	A	0	0	0	0	1	1	1	1
	$B \Rightarrow C$	1	1	0	1	1	1	0	1
	$A \Rightarrow B \vee C$	1	1	1	1	0	1	1	1
	KB	0	0	0	0	0	1	0	1
Query	C	0	1	0	1	0	1	0	1

Entailment

How do we implement a logical agent that proves entailment?

- Logic language
  - Propositional logic
  - First order logic

- Knowledge Base
  - Add known logical rules and facts

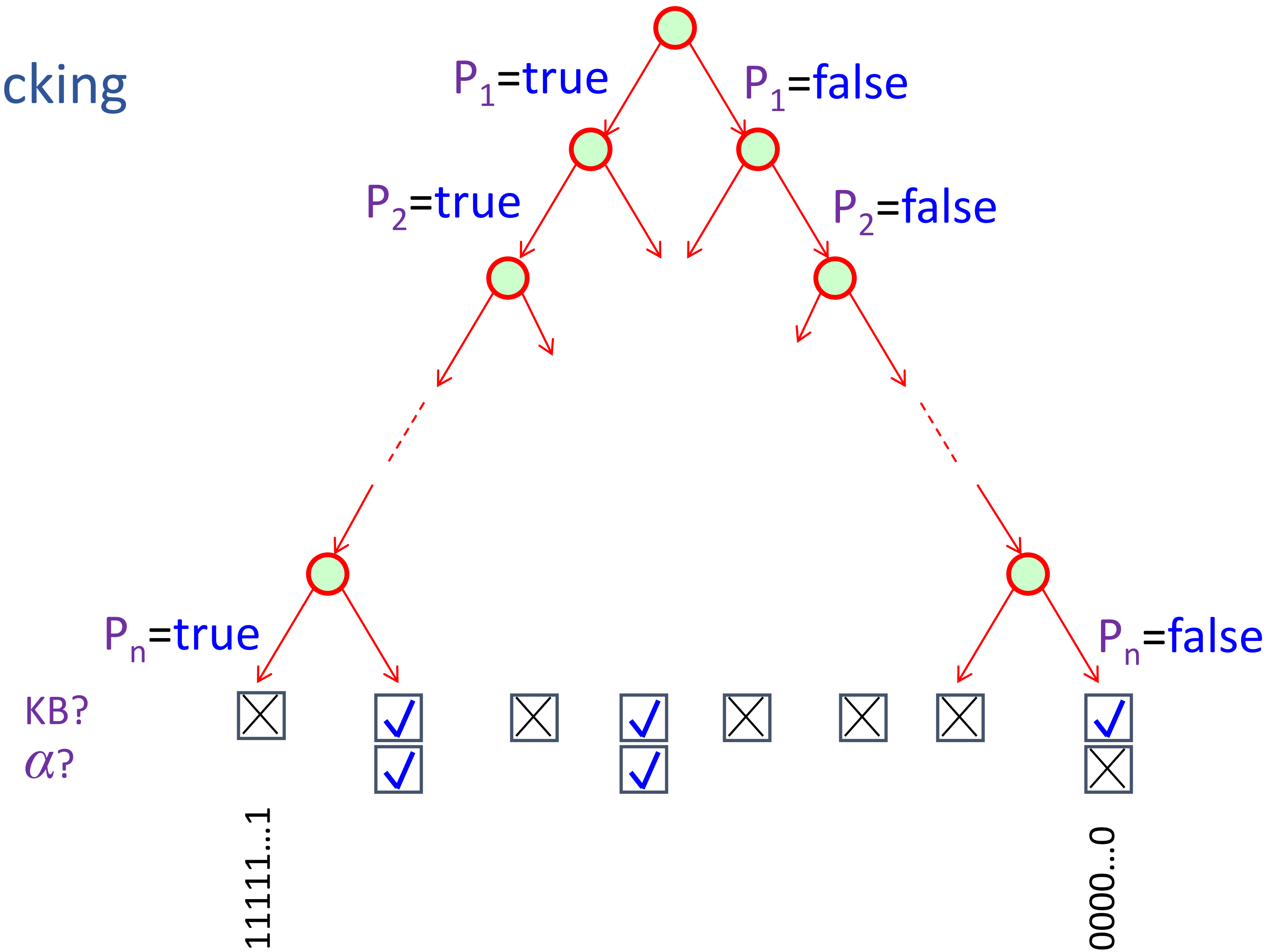
- Inference algorithms
  - Theorem proving
  - Model checking

Simple Model Checking

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

Simple Model Checking, contd.

Same recursion as backtracking



Simple Model Checking

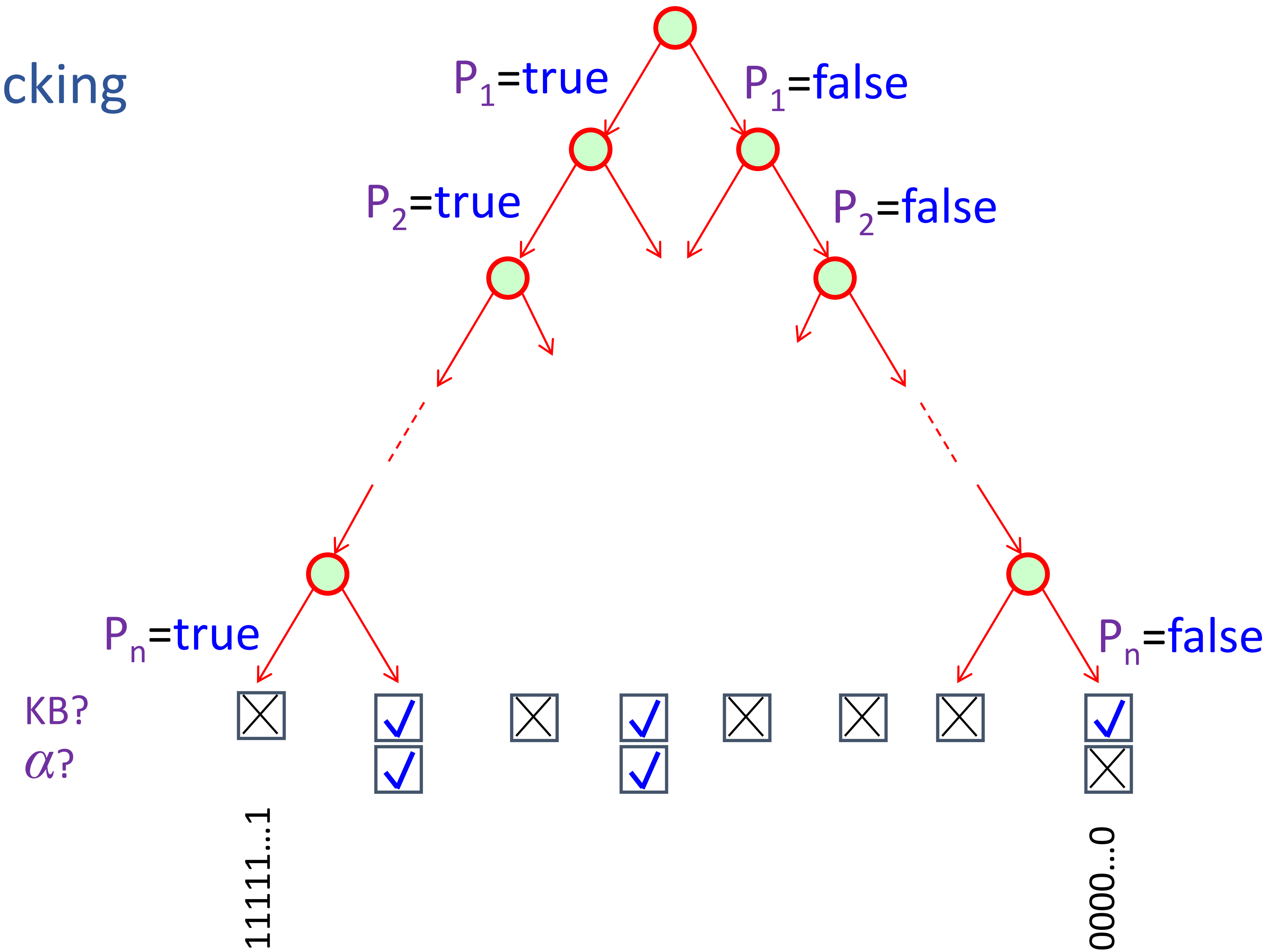
```
function TT-ENTAILS?(KB, α) returns true or false
    return TT-CHECK-ALL(KB, α, symbols(KB) U symbols(α),{})

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

Can we do better?



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*

Properties

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

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

Clauses

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

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

CLAUSES	*COUNT*	*INFERRED*	*AGENDA*
$P \Rightarrow Q$	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

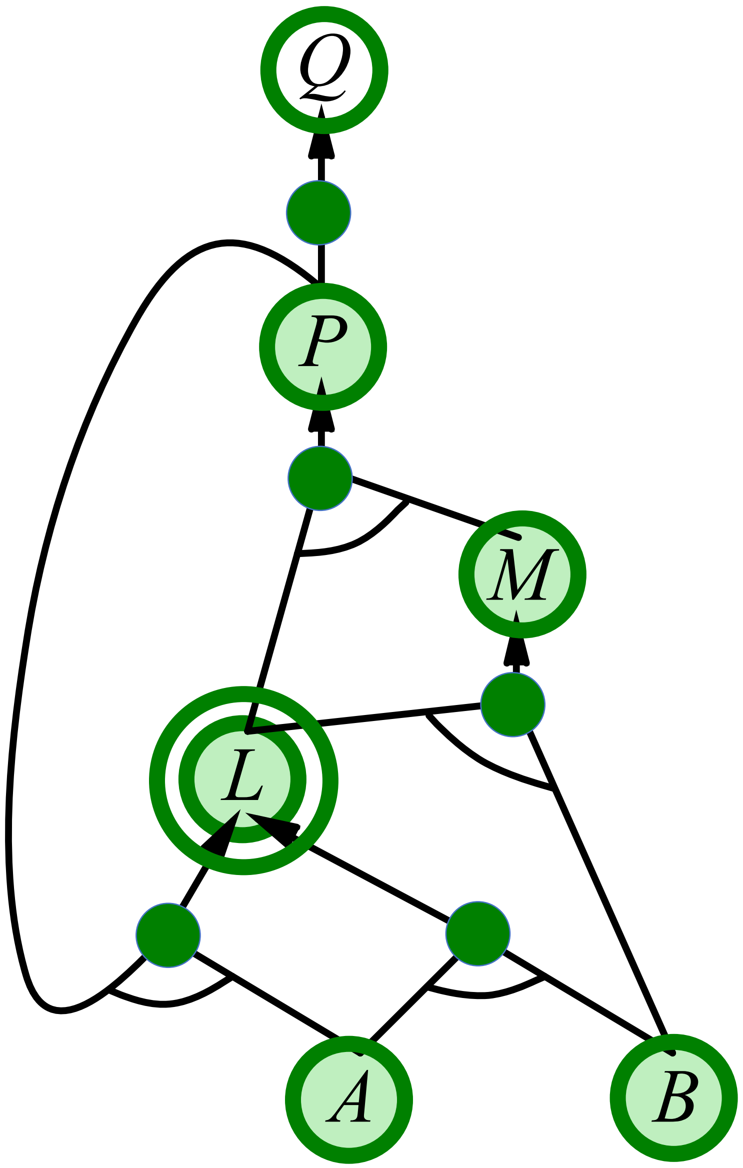
Clauses	Count
$P \Rightarrow Q$	~~1~~ ~~1~~ 0
$L \wedge M \Rightarrow P$	~~2~~ ~~1~~ 0
$B \wedge L \Rightarrow M$	~~2~~ ~~1~~ 0
$A \wedge P \Rightarrow L$	~~2~~ ~~1~~ 0
$A \wedge B \Rightarrow L$	~~2~~ ~~1~~ 0
A	0
B	0

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~~



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
```

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 for ***m***

   Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false for ***m***
   Then $a_1 \wedge \ldots \wedge a_k$ is true in ***m*** and b is false for ***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 xxxx true
B false xxxx true
L false xxxx true
M false xxxx true
P false xxxx true
Q false xxxx true

Does forward chaining work on this example?

$A \Rightarrow B$

$\neg A \Rightarrow B$

Inference Rules

Modus Ponens

Notation Alert!

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

Unit Resolution

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

General Resolution

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

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 $\land \neg\alpha$ cannot be satisfied

      Return true, KB entails α

   B. No new clauses added

      Return false, KB does not entail α

Resolution

General Resolution

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

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

Knowledge Base

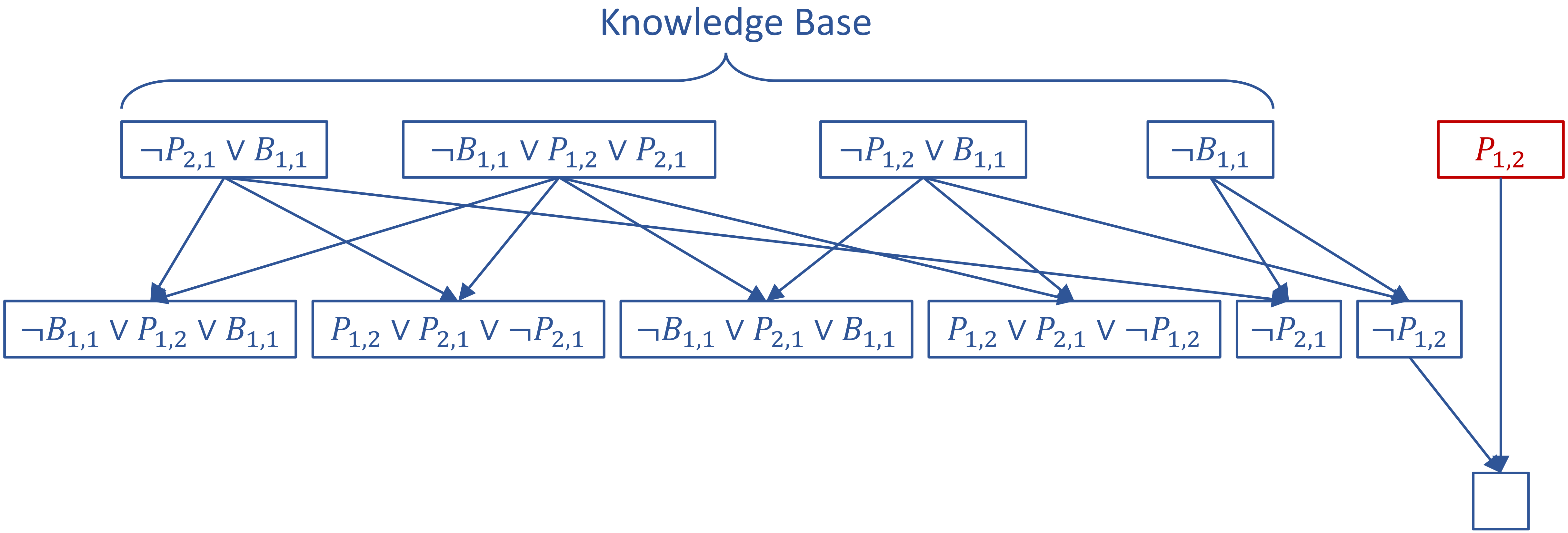
$\neg P_{2,1} \vee B_{1,1}$ | $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$ | $\neg P_{1,2} \vee B_{1,1}$ | $\neg B_{1,1}$ | $\neg\neg P_{1,2}$

Resolution

General Resolution

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

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



Resolution

```
function PL-RESOLUTION?(KB, α) returns true or false
  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  loop do
    for each pair of clauses C_i, C_j in clauses do
      resolvents ← PL-RESOLVE(C_i, C_j)
      if resolvents contains the empty clause then
        return true
      new ← new ∪ resolvants
    if new ⊆ clauses then
      return false
    clauses ← clauses ∪ new
```

Properties

Forward Chaining is:

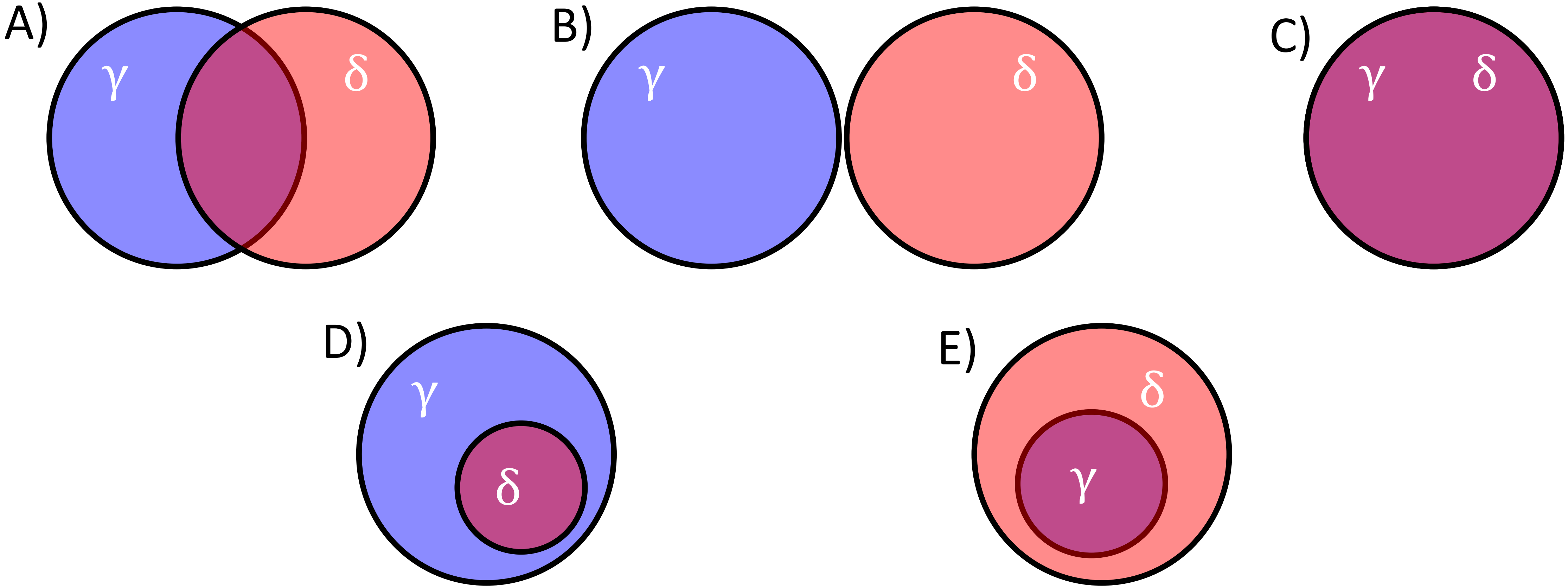
- Sound and complete for definite-clause KBs
- Complexity: linear time

Resolution is:

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

Poll 4

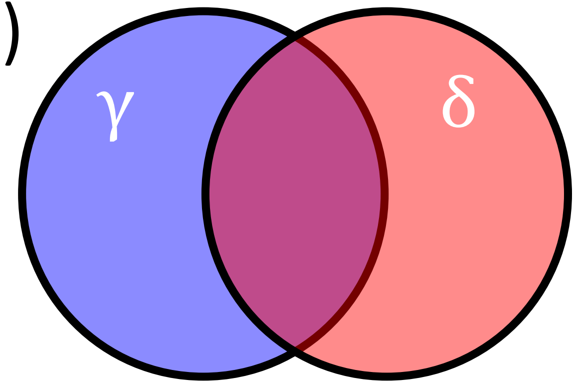
The regions below visually enclose the set of models that satisfy the respective sentence γ or δ. For which of the following diagrams is the sentence $\gamma \wedge \delta$ satisfiable? Select all that apply.



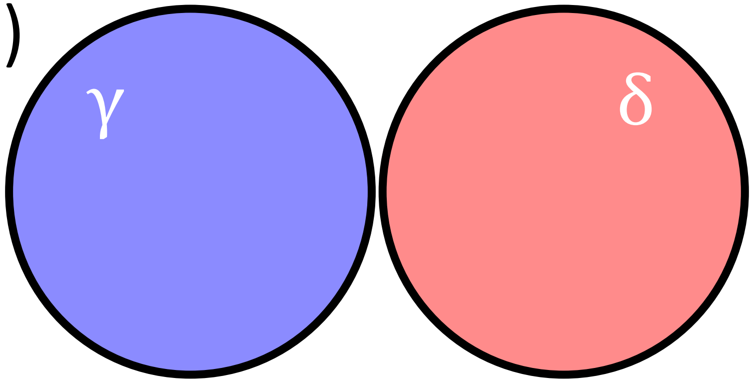
Poll 5

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.

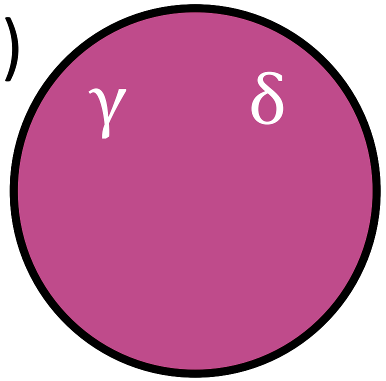
A)



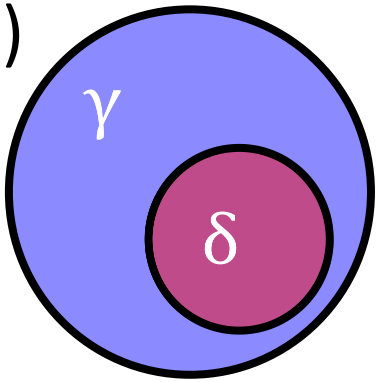
B)



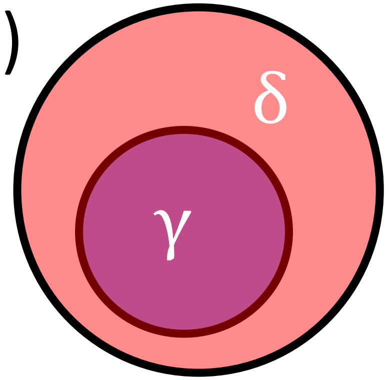
C)



D)



E)



Satisfiability and Entailment

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

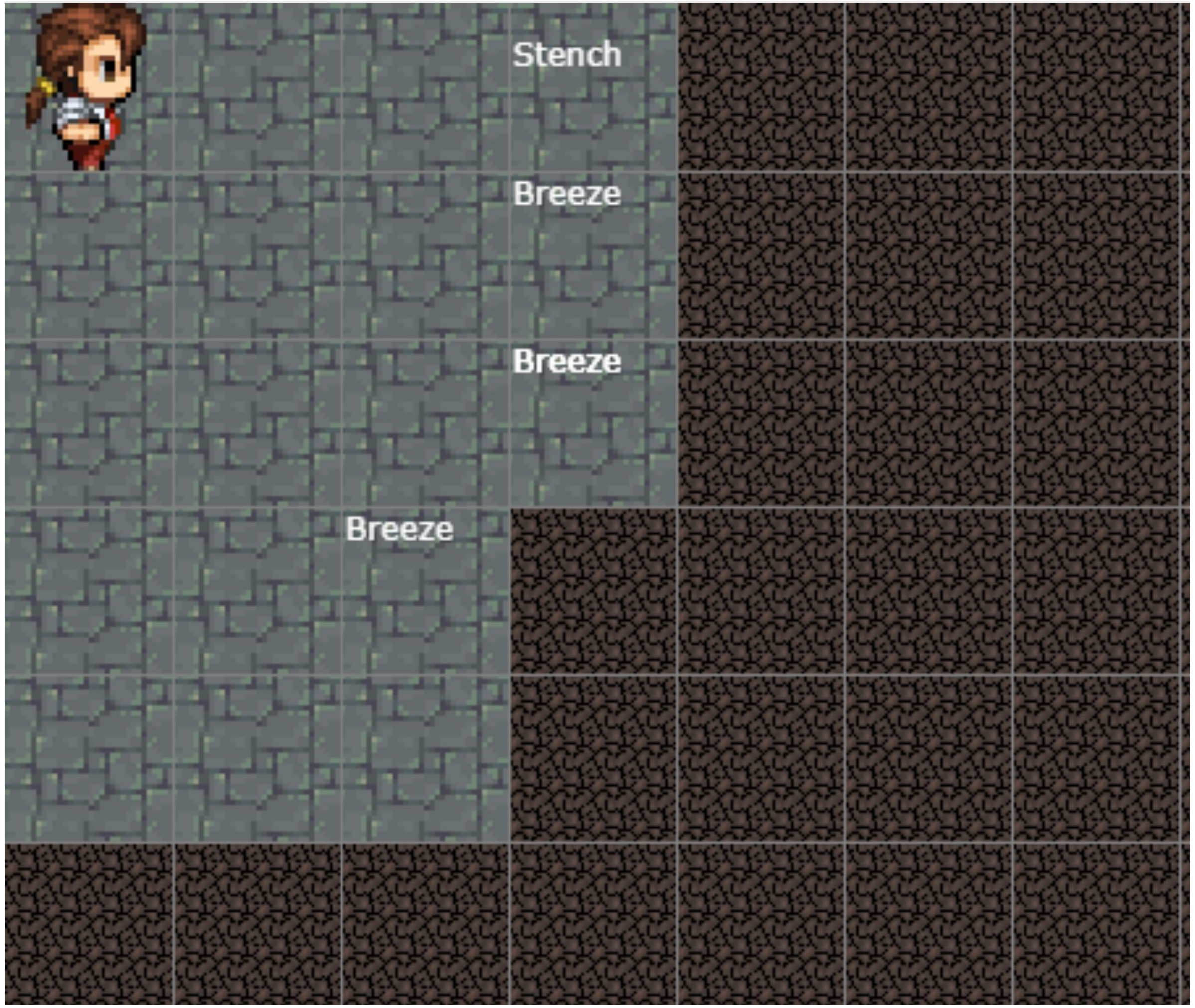
Suppose we have a hyper-efficient SAT solver; how can we use it to test entailment?

- 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***

Satisfiability and Entailment



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

Conjunctive Normal Form (CNF)

Every sentence can be expressed

Replace biconditional by two implications

Each clause is a disjunction of literal

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

Each literal is a symbol or a neg

Distribute v over $\wedge$

Conversion to CNF by a se

- At_1,1_0 $\Rightarrow$ (Wall_0,1 $\Leftrightarrow$ Block
- At_1,1_0 $\Rightarrow$ ((Wall_0,1 $\Rightarrow$ Blocked_W_0) $\wedge$ (Bloc
- $\neg$At_1,1_0 v (($\neg$Wall_0,1 v Blocked_W_0) $\wedge$ ($\neg$Blocked_W_0 v Wall_0,1))
- ($\neg$At_1,1_0 v $\neg$Wall_0,1 v Blocked_W_0) $\wedge$ ($\neg$At_1,1_0 v $\neg$Blocked_W_0 v Wall_0,1)

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., $(A \lor B) \land (A \lor \neg C)$ is satisfied by {A=true}
  - any clause is falsified; e.g., $(A \lor B) \land (A \lor \neg C)$ is falsified 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 $(A \lor B) \land (A \lor \neg C) \land (C \lor \neg B)$ 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, $(A \lor B) \land (A \lor \neg C)$ becomes $(\text{false} \lor B) \land (\text{false} \lor \neg C)$, i.e. $(B) \land (\neg C)$
  - Satisfying the unit clauses often leads to further propagation, new unit clauses, etc.

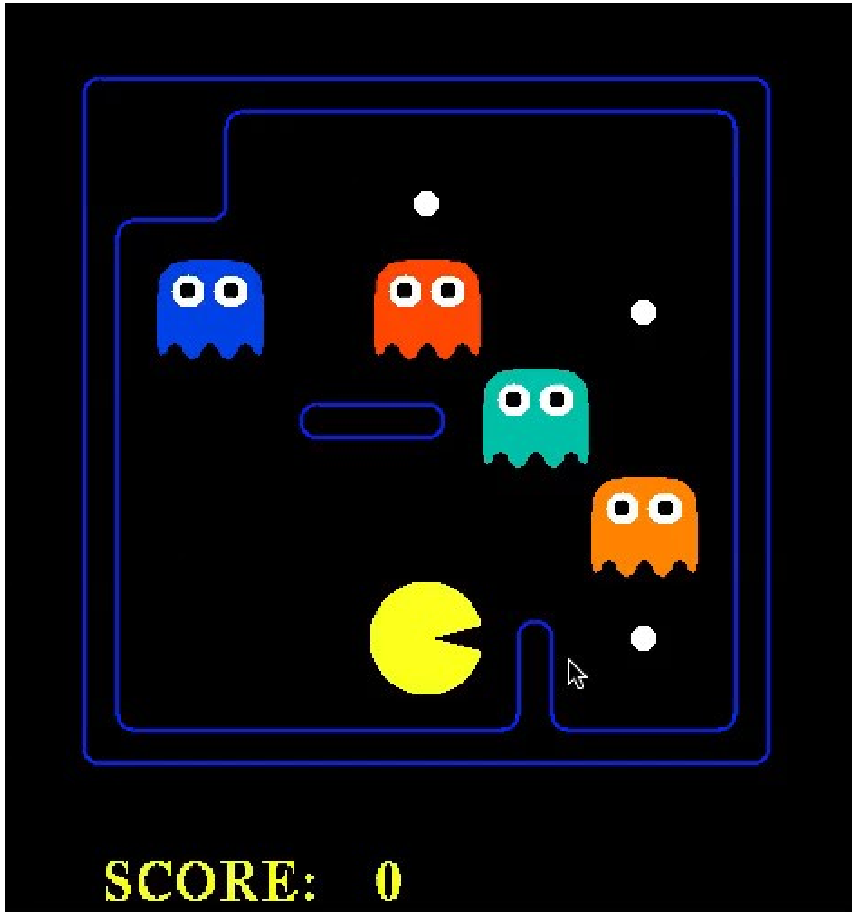
DPLL algorithm

```
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, model∪{P=value})

   P, value ←FIND-UNIT-CLAUSE(clauses, model)
   if P is non-null then return DPLL(clauses, symbols–P, model∪{P=value})

   P ← First(symbols)
   rest ← Rest(symbols)

   return or(DPLL(clauses, rest, model∪{P=true}),
             DPLL(clauses, rest, model∪{P=false}))
```

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 for actions etc.


SCORE: 0

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 for actions etc.

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
- *Precondition axioms*: At_1,1_0 $\wedge$ N_0 $\Rightarrow$ $\neg$Wall_1,2 etc.
- *Action exclusion axioms*: $\neg$(N_0 $\wedge$ W_0) $\wedge$ $\neg$(N_0 $\wedge$ S_0) $\wedge$ .. etc.

Initial State

The agent may know its initial location:

- At_1,1_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* – cannot be in two places at once!

- ¬(At_1,1_0 ∧ At_1,2_0) ∧ ¬(At_1,1_0 ∧ At_1,3_0) ∧ …
- ¬(At_1,1_1 ∧ At_1,2_1) ∧ ¬(At_1,1_1 ∧ At_1,3_1) ∧ …
- …

Fluents and Effect Axioms

A *fluent* is a state variable that changes over time

How does each *state variable* or *fluent* at each time gets its value?

Fluents for PL Pacman are Pacman_x,y_t , e.g., Pacman _3,3_17

Fluents and Successor-state Axioms

A *fluent* is a state variable that changes over time

How does each *state variable* or *fluent* at each time gets its value?

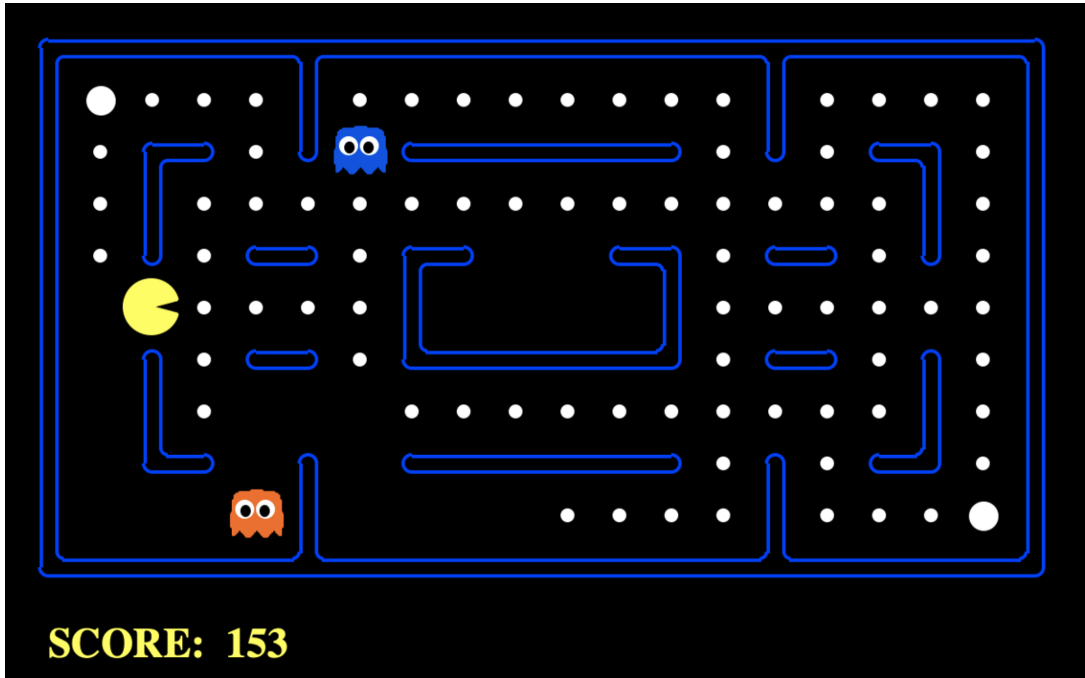
Fluents for PL Pacman are Pacman_x,y_t , e.g., Pacman _3,3_17

A state variable gets its value according to a *successor-state axiom*

- $X_t \Leftrightarrow [X_{t-1} \wedge \neg$(some action$_{t-1}$ made it false)] v
  [$\neg X_{t-1} \wedge$ (some action$_{t-1}$ made it true)]

Fluents and Successor-state Axioms

Write the *successor-state axiom* for pacman's location



Planning as Satisfiability

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

Why?

If I can find a satisfying set of variables that meet the constraints, then I have also found a plan as the set of action variables.

EXTRA SLIDES

Logical Agent Vocab

Model

- Complete assignment of 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

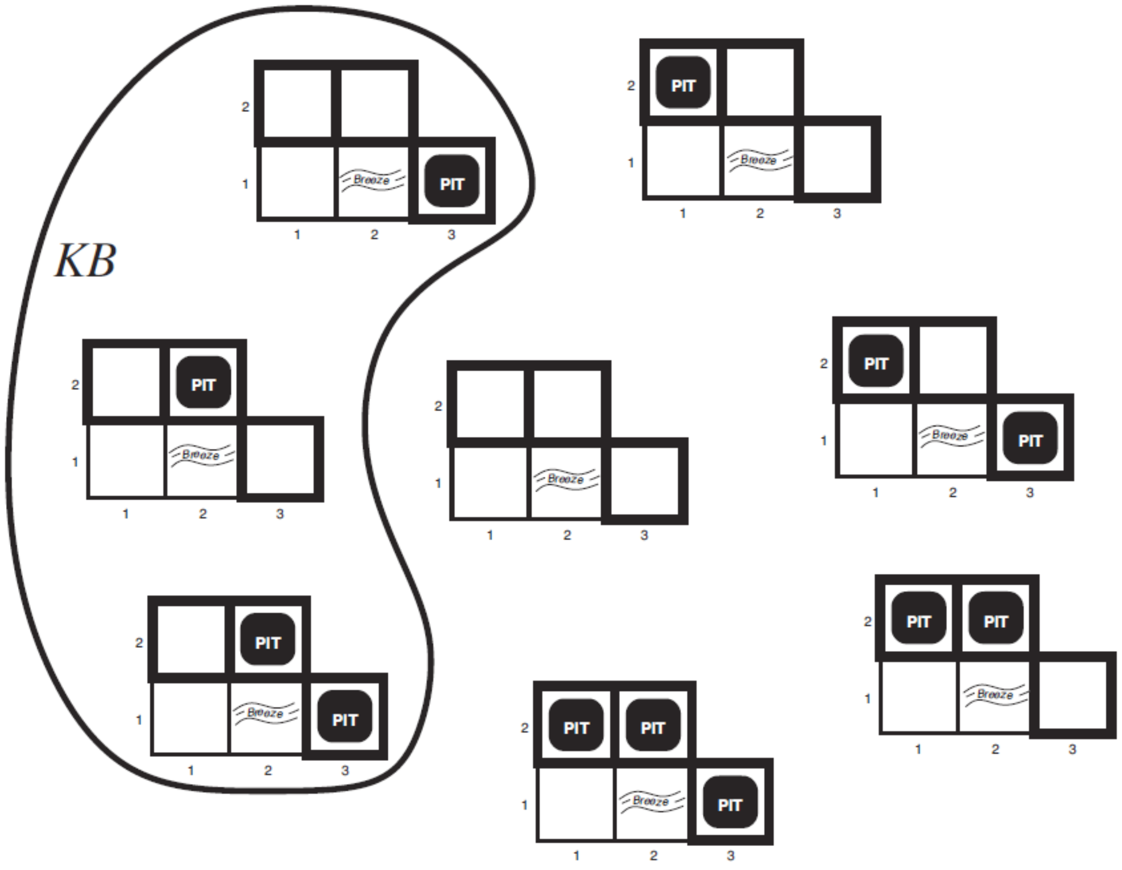
Query

- Sentence we want to know if it is *provably* True, *provably* False, or *unsure*.

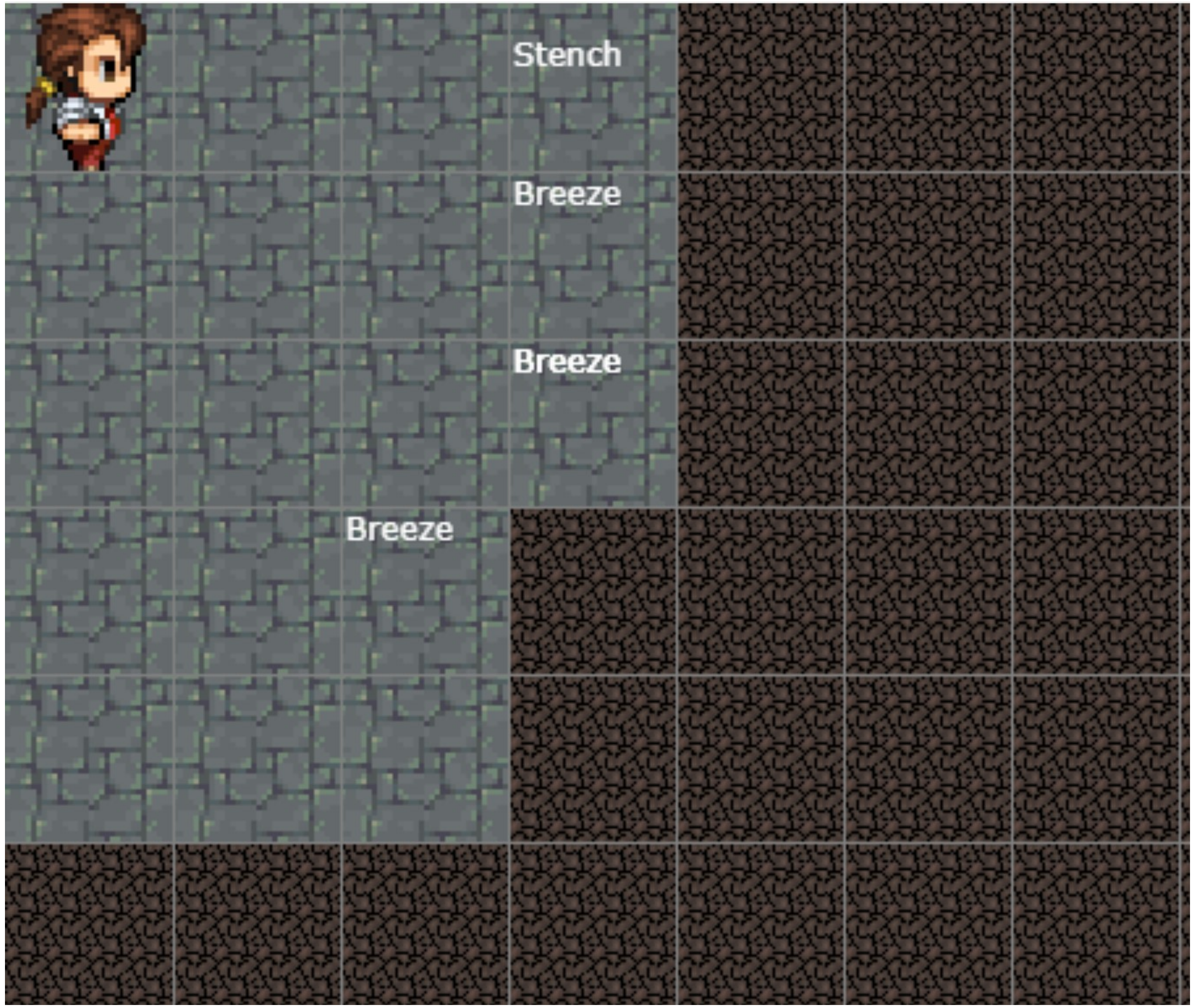
Entailment

Does the knowledge base entail my query?

- Query 1: $\neg P[1,2]$
- Query 2: $\neg P[2,2]$



Provably True, Provably False, or Unsure



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

Logical Agent Vocab

Entailment

- 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

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

- 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

Valid

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