Logical Agents

Dr. Steven Bethard

Computer and Information Sciences
University of Alabama at Birmingham

11 Feb 2016

Outline

Outline

Knowledge Bases

Idea: Separate Knowledge from Reasoning

Inference Engine: domain-independent algorithms

Knowledge Base: domain-specific content

Knowledge Base (KB) Properties

- Contains a set of "sentences"
- Can TELL it new "sentences"
- Can ASK it "queries"

Knowledge Bases

Idea: Separate Knowledge from Reasoning

Inference Engine: domain-independent algorithms

Knowledge Base: domain-specific content

Knowledge Base (KB) Properties

- Contains a set of "sentences"
- Can TELL it new "sentences"
- Can ASK it "queries"

Knowledge-Based Agents

```python
class KnowledgeBaseAgent(object):
    def __init__(self, knowledge_base):
        self.knowledge_base = knowledge_base
        self.time = 0
    def take_action(self, percept):
        # convert the percept to knowledge
        percept_sentence = self.percept_to_sentence(percept, self.time)
        self.knowledge_base.tell(percept_sentence)
        # select an action based on the knowledge
        action_query = self.make_action_query(self.time)
        action = self.knowledge_base.ask(action_query)
        # update the knowledge base with the planned action
        action_sentence = self.action_to_sentence(action, self.time)
        self.knowledge_base.tell(action)
        self.time += 1
        # perform the action
        return action
```

Knowledge-Based Agents

```python
class KnowledgeBaseAgent(object):
    def __init__(self, knowledge_base):
        self.knowledge_base = knowledge_base
        self.time = 0
    def take_action(self, percept):
        # convert the percept to knowledge
        percept_sentence = self.percept_to_sentence(percept, self.time)
        self.knowledge_base.tell(percept_sentence)
        # select an action based on the knowledge
        action_query = self.make_action_query(self.time)
        action = self.knowledge_base.ask(action_query)
        # update the knowledge base with the planned action
        action_sentence = self.action_to_sentence(action, self.time)
        self.knowledge_base.tell(action)
        self.time += 1
        # perform the action
        return action
```

Knowledge-Based Agents

```
class KnowledgeBaseAgent(object):
    def __init__(self, knowledge_base):
        self.knowledge_base = knowledge_base
        self.time = 0
    def take_action(self, percept):
        # convert the percept to knowledge
        percept_sentence = self.percept_to_sentence(percept, self.time)
        self.knowledge_base.tell(percept_sentence)
        # select an action based on the knowledge
        action_query = self.make_action_query(self.time)
        action = self.knowledge_base.ask(action_query)
        # update the knowledge base with the planned action
        action_sentence = self.action_to_sentence(action, self.time)
        self.knowledge_base.tell(action)
        self.time += 1
        # perform the action
        return action
```

Knowledge-Based Agents

```python
class KnowledgeBaseAgent(object):
    def __init__(self, knowledge_base):
        self.knowledge_base = knowledge_base
        self.time = 0
    def take_action(self, percept):
        # convert the percept to knowledge
        percept_sentence = self.percept_to_sentence(percept, self.time)
        self.knowledge_base.tell(percept_sentence)
        # select an action based on the knowledge
        action_query = self.make_action_query(self.time)
        action = self.knowledge_base.ask(action_query)
        # update the knowledge base with the planned action
        action_sentence = self.action_to_sentence(action, self.time)
        self.knowledge_base.tell(action)
        self.time += 1
        # perform the action
        return action
```

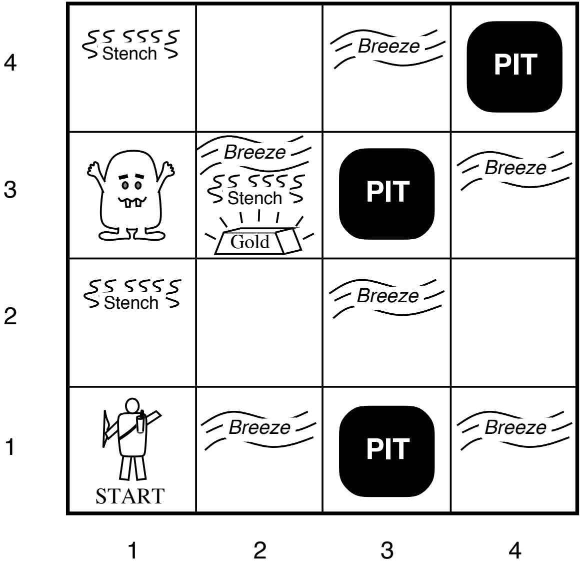
Knowledge-Based Agents

```
class KnowledgeBaseAgent(object):
    def __init__(self, knowledge_base):
        self.knowledge_base = knowledge_base
        self.time = 0
    def take_action(self, percept):
        # convert the percept to knowledge
        percept_sentence = self.percept_to_sentence(percept, self.time)
        self.knowledge_base.tell(percept_sentence)
        # select an action based on the knowledge
        action_query = self.make_action_query(self.time)
        action = self.knowledge_base.ask(action_query)
        # update the knowledge base with the planned action
        action_sentence = self.action_to_sentence(action, self.time)
        self.knowledge_base.tell(action)
        self.time += 1
        # perform the action
        return action
```

Knowledge-Based Agents

```python
class KnowledgeBaseAgent(object):
    def __init__(self, knowledge_base):
        self.knowledge_base = knowledge_base
        self.time = 0
    def take_action(self, percept):
        # convert the percept to knowledge
        percept_sentence = self.percept_to_sentence(percept, self.time)
        self.knowledge_base.tell(percept_sentence)
        # select an action based on the knowledge
        action_query = self.make_action_query(self.time)
        action = self.knowledge_base.ask(action_query)
        # update the knowledge base with the planned action
        action_sentence = self.action_to_sentence(action, self.time)
        self.knowledge_base.tell(action)
        self.time += 1
        # perform the action
        return action
```

The Wumpus World



Wumpus World Description

Environment:

- 4×4 rooms, 1 with gold, 1 with wumpus, k with pits
- Agent starts in (1,1), facing right, holding 1 arrow

Performance Measure:

- gold +1000, death -1000, -1 per step, -10 per arrow

Percepts:

- STENCH/BREEZE in squares adjacent to wumpus/pit
- GLITTER in squares with gold
- BUMP when running into wall
- SCREAM when wumpus is killed by arrow

Actions:

- FORWARD, TURNLEFT, TURNRIGHT, GRAB, SHOOT

Wumpus World Description

Environment:

- 4×4 rooms, 1 with gold, 1 with wumpus, k with pits
- Agent starts in (1,1), facing right, holding 1 arrow

Performance Measure:

- gold +1000, death -1000, -1 per step, -10 per arrow

Percepts:

- STENCH/BREEZE in squares adjacent to wumpus/pit
- GLITTER in squares with gold
- BUMP when running into wall
- SCREAM when wumpus is killed by arrow

Actions:

- FORWARD, TURNLEFT, TURNRIGHT, GRAB, SHOOT

Wumpus World Description

Environment:

- 4×4 rooms, 1 with gold, 1 with wumpus, k with pits
- Agent starts in (1,1), facing right, holding 1 arrow

Performance Measure:

- gold +1000, death -1000, -1 per step, -10 per arrow

Percepts:

- STENCH/BREEZE in squares adjacent to wumpus/pit
- GLITTER in squares with gold
- BUMP when running into wall
- SCREAM when wumpus is killed by arrow

Actions:

- FORWARD, TURNLEFT, TURNRIGHT, GRAB, SHOOT

Wumpus World Description

Environment:

- 4×4 rooms, 1 with gold, 1 with wumpus, k with pits
- Agent starts in (1,1), facing right, holding 1 arrow

Performance Measure:

- gold +1000, death -1000, -1 per step, -10 per arrow

Percepts:

- STENCH/BREEZE in squares adjacent to wumpus/pit
- GLITTER in squares with gold
- BUMP when running into wall
- SCREAM when wumpus is killed by arrow

Actions:

- FORWARD, TURNLEFT, TURNRIGHT, GRAB, SHOOT

Wumpus World Description

Environment:

- 4×4 rooms, 1 with gold, 1 with wumpus, k with pits
- Agent starts in (1,1), facing right, holding 1 arrow

Performance Measure:

- gold +1000, death -1000, -1 per step, -10 per arrow

Percepts:

- STENCH/BREEZE in squares adjacent to wumpus/pit
- GLITTER in squares with gold
- BUMP when running into wall
- SCREAM when wumpus is killed by arrow

Actions:

- FORWARD, TURNLEFT, TURNRIGHT, GRAB, SHOOT

Wumpus World Properties

Observable?	No, only local perception
Deterministic?	Yes, state+action determines outcome
Episodic?	No, involves a sequence of actions
Static?	Yes, pits and wumpus are stationary
Discrete?	Yes, no real-valued states or actions
Single-Agent?	Yes, wumpus takes no (real) actions

Wumpus World Properties

Observable?	No, only local perception
Deterministic?	Yes, state+action determines outcome
Episodic?	No, involves a sequence of actions
Static?	Yes, pits and wumpus are stationary
Discrete?	Yes, no real-valued states or actions
Single-Agent?	Yes, wumpus takes no (real) actions

Wumpus World Properties

Observable?	No, only local perception
Deterministic?	Yes, state+action determines outcome
Episodic?	No, involves a sequence of actions
Static?	Yes, pits and wumpus are stationary
Discrete?	Yes, no real-valued states or actions
Single-Agent?	Yes, wumpus takes no (real) actions

Wumpus World Properties

Observable?	No, only local perception
Deterministic?	Yes, state+action determines outcome
Episodic?	No, involves a sequence of actions
Static?	Yes, pits and wumpus are stationary
Discrete?	Yes, no real-valued states or actions
Single-Agent?	Yes, wumpus takes no (real) actions

Wumpus World Properties

Observable?	No, only local perception
Deterministic?	Yes, state+action determines outcome
Episodic?	No, involves a sequence of actions
Static?	Yes, pits and wumpus are stationary
Discrete?	Yes, no real-valued states or actions
Single-Agent?	Yes, wumpus takes no (real) actions

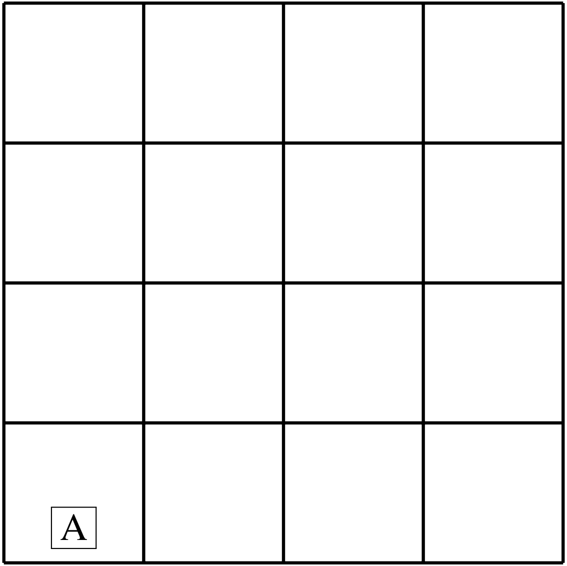
Wumpus World Properties

Observable?	No, only local perception
Deterministic?	Yes, state+action determines outcome
Episodic?	No, involves a sequence of actions
Static?	Yes, pits and wumpus are stationary
Discrete?	Yes, no real-valued states or actions
Single-Agent?	Yes, wumpus takes no (real) actions

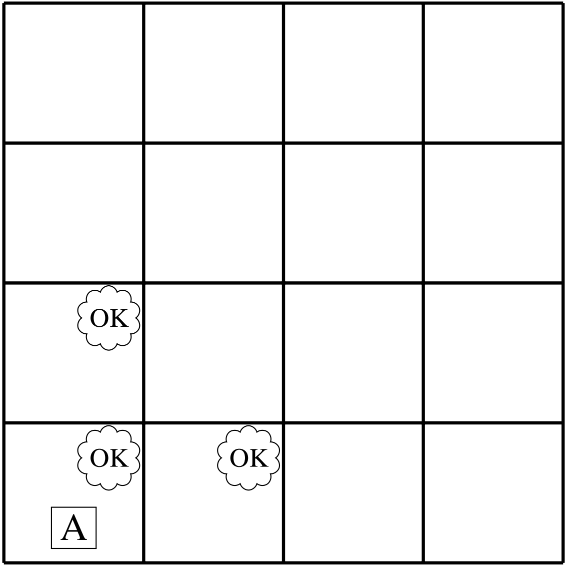
Wumpus World Properties

Observable?	No, only local perception
Deterministic?	Yes, state+action determines outcome
Episodic?	No, involves a sequence of actions
Static?	Yes, pits and wumpus are stationary
Discrete?	Yes, no real-valued states or actions
Single-Agent?	Yes, wumpus takes no (real) actions

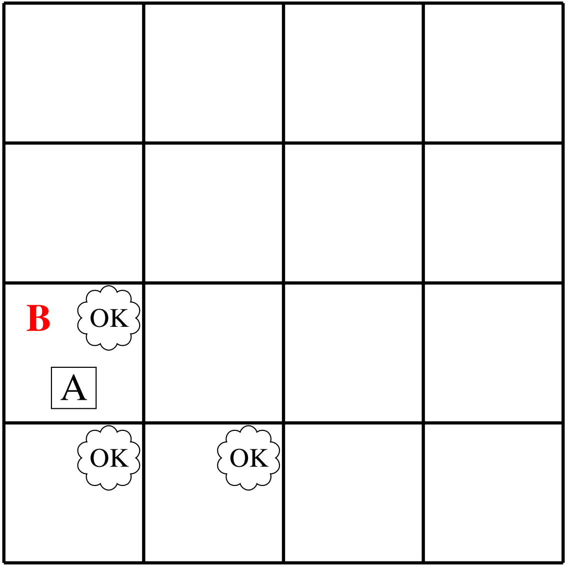
Exploring a Wumpus World



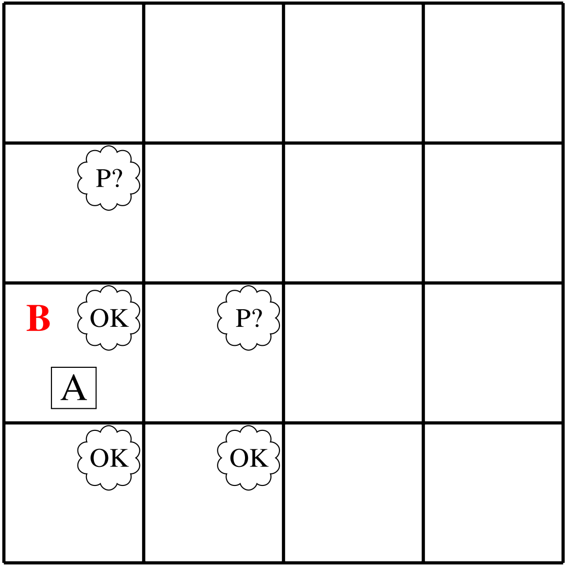
Exploring a Wumpus World



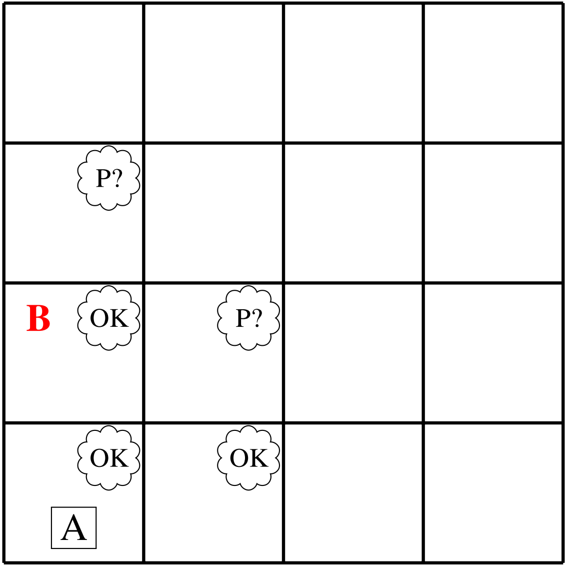
Exploring a Wumpus World



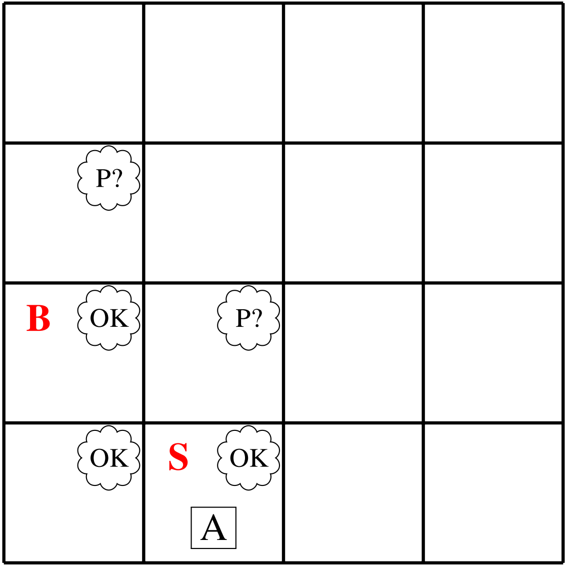
Exploring a Wumpus World



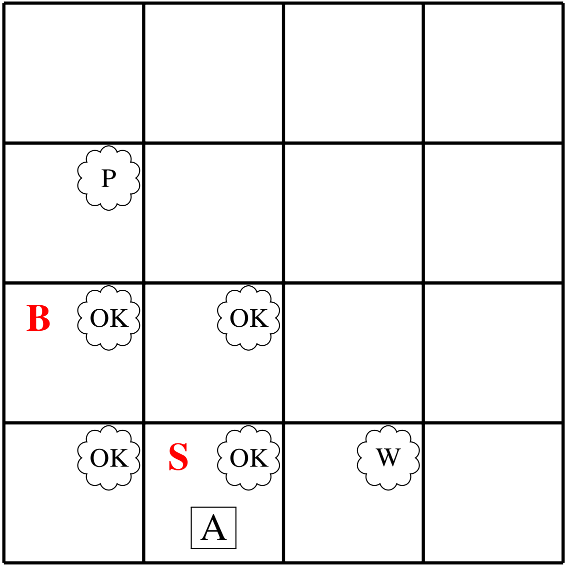
Exploring a Wumpus World



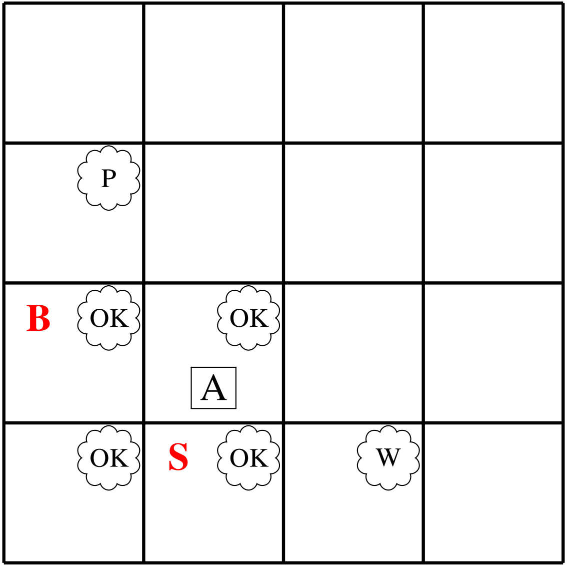
Exploring a Wumpus World



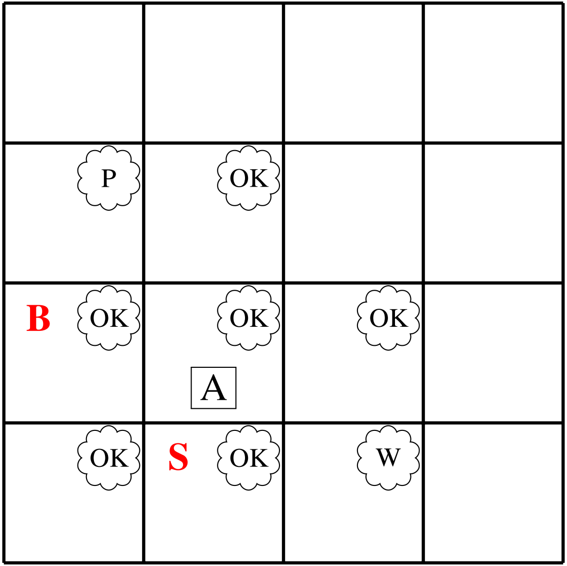
Exploring a Wumpus World



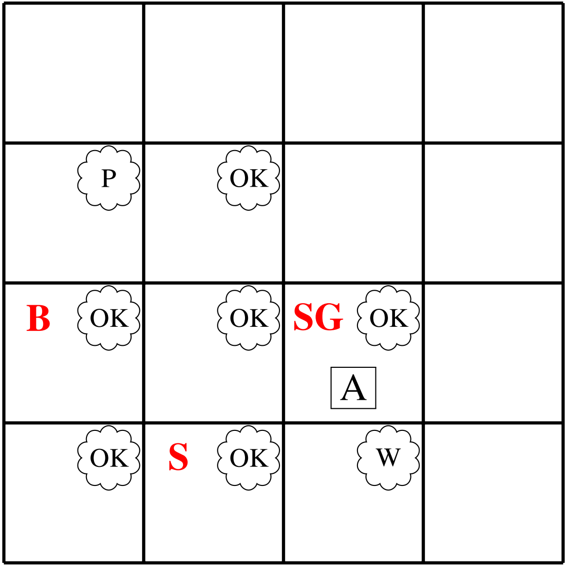
Exploring a Wumpus World



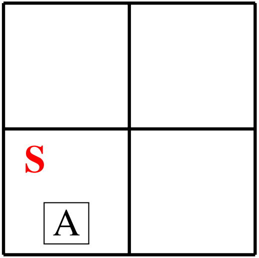
Exploring a Wumpus World



Exploring a Wumpus World



Difficult Wumpus World Situations

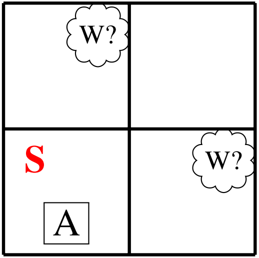


Solution: Coercion

SHOOT an arrow

- Scream
  ⇒ Wumpus above
- No scream
  ⇒ Wumpus on right

Difficult Wumpus World Situations



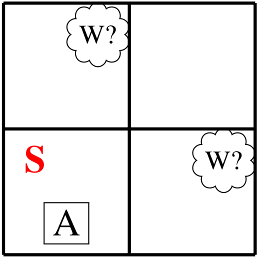
Solution: Coercion

Shoot an arrow

- Scream
  $\Rightarrow$ Wumpus above
- No scream
  $\Rightarrow$ Wumpus on right

- 86% pit in (2,2)
- 31% pit in (3,1) or (1,3)

Difficult Wumpus World Situations

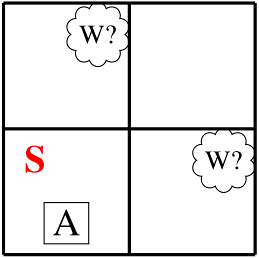


Solution: Coercion

Shoot an arrow

- Scream
  ⇒ Wumpus above
- No scream
  ⇒ Wumpus on right

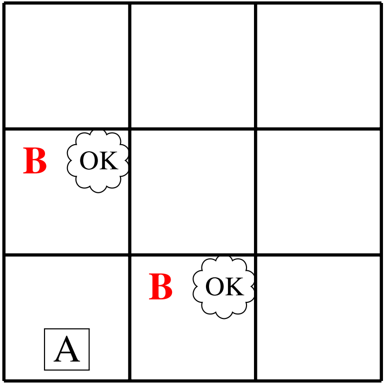
Difficult Wumpus World Situations



Solution: Coercion

Shoot an arrow

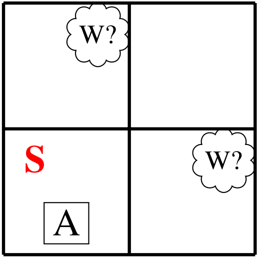
- Scream
  ⇒ Wumpus above
- No scream
  ⇒ Wumpus on right

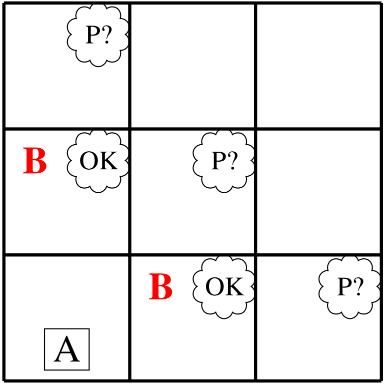


Solution: Probability

- 86% pit in (2,2)
- 31% pit in (3,1) or (1,3)

Difficult Wumpus World Situations





Solution: Coercion

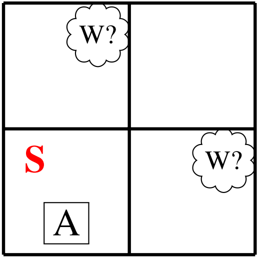
SHOOT an arrow

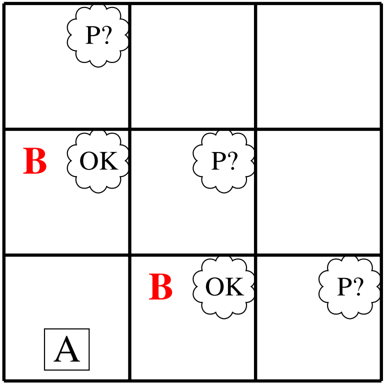
- Scream
  ⇒ Wumpus above
- No scream
  ⇒ Wumpus on right

Solution: Probability

- 86% pit in (2,2)
- 31% pit in (3,1) or (1,3)

Difficult Wumpus World Situations





Solution: Coercion

SHOOT an arrow

- Scream
  $\Rightarrow$ Wumpus above
- No scream
  $\Rightarrow$ Wumpus on right

Solution: Probability

- 86% pit in (2,2)
- 31% pit in (3,1) or (1,3)

Logic

Key Ideas

Formal language for representing information

- **Syntax** defines structure of sentences
- **Semantics** defines meaning of sentences

Example: Arithmetic

Syntax
- Valid: $x + 2 > y$
- Invalid: $x2 + y >$

Semantics
- The sentence $x + 2 > y$ is true if:
  The sum of x and 2 is greater than y

Logic

Key Ideas

Formal language for representing information

Syntax	defines structure of sentences
Semantics	defines meaning of sentences

Example: Arithmetic

Syntax

Valid: $x + 2 > y$

Invalid: $x2 + y >$

Semantics

The sentence $x + 2 > y$ is true if:

The sum of x *and* 2 *is greater than* y

Models

Definitions

- A model is a possible state of the world
- If a sentence α is true in model m,
  then m is a model of α
- $M(\alpha)$ means all models of α

Example: Arithmetic

Given the sentence α which looks like $x + 2 > y$:

- Is $\{x = 7, y = 1\}$ a model of α? Yes
- Is $\{x = 3, y = 6\}$ a model of α? No

Models

Definitions

- A model is a possible state of the world
- If a sentence α is true in model m,
  then m is a model of α
- $M(\alpha)$ means all models of α

Example: Arithmetic

Given the sentence α which looks like $x + 2 > y$:

- Is $\{x = 7, y = 1\}$ a model of α? Yes
- Is $\{x = 3, y = 6\}$ a model of α? No

Models

Definitions

- A model is a possible state of the world
- If a sentence α is true in model m,
  then m is a model of α
- $M(\alpha)$ means all models of α

Example: Arithmetic

Given the sentence α which looks like $x + 2 > y$:

- Is $\{x = 7, y = 1\}$ a model of α? Yes
- Is $\{x = 3, y = 6\}$ a model of α? No

Models

Definitions

- A model is a possible state of the world
- If a sentence α is true in model m,
  then m is a model of α
- $M(\alpha)$ means all models of α

Example: Arithmetic

Given the sentence α which looks like $x + 2 > y$:

- Is $\{x = 7, y = 1\}$ a model of α? Yes
- Is $\{x = 3, y = 6\}$ a model of α? No

Models

Definitions

- A model is a possible state of the world
- If a sentence α is true in model m,
  then m is a model of α
- $M(\alpha)$ means all models of α

Example: Arithmetic

Given the sentence α which looks like $x + 2 > y$:

- Is $\{x = 7, y = 1\}$ a model of α? Yes
- Is $\{x = 3, y = 6\}$ a model of α? No

Models

Definitions

- A model is a possible state of the world
- If a sentence α is true in model m,
  then m is a model of α
- $M(\alpha)$ means all models of α

Example: Arithmetic

Given the sentence α which looks like $x + 2 > y$:

- Is $\{x = 7, y = 1\}$ a model of α? Yes
- Is $\{x = 3, y = 6\}$ a model of α? No

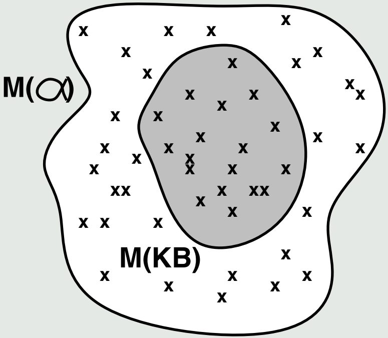
Entailment

Definition

$\beta \models \alpha$ if and only if
$M(\beta) \subseteq M(\alpha)$

A sentence β entails a sentence α if and only if α is true in all worlds where β is true

$M(\mathrm{KB}) \subseteq M(\alpha)$



Entailment Examples

Definition (Again)

$\beta \models \alpha$ if and only if $M(\beta) \subseteq M(\alpha)$

- $(x + y = 4) \models (4 = x + y)$
- $(x = 0) \models (xy = 0)$
- "The Crimson Tide won and the Blazers won"
  $\models$ "The Crimson Tide won or the Blazers won"

Entailment Examples

Definition (Again)

$\beta \models \alpha$ if and only if $M(\beta) \subseteq M(\alpha)$

- $(x + y = 4) \models (4 = x + y)$
- $(x = 0) \models (xy = 0)$
- "The Crimson Tide won and the Blazers won"
  $\models$ "The Crimson Tide won or the Blazers won"

Entailment Examples

Definition (Again)

$\beta \models \alpha$ if and only if $M(\beta) \subseteq M(\alpha)$

- $(x + y = 4) \models (4 = x + y)$
- $(x = 0) \models (xy = 0)$
- "The Crimson Tide won and the Blazers won"
  $\models$ "The Crimson Tide won or the Blazers won"

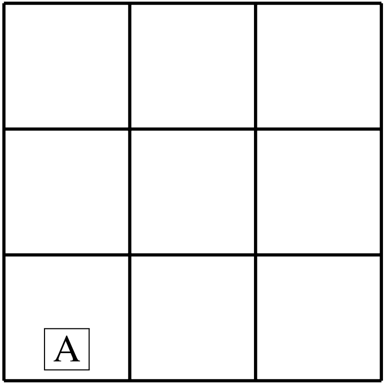
Entailment Examples

Definition (Again)

$\beta \models \alpha$ if and only if $M(\beta) \subseteq M(\alpha)$

- $(x + y = 4) \models (4 = x + y)$
- $(x = 0) \models (xy = 0)$
- "The Crimson Tide won and the Blazers won"
  $\models$ "The Crimson Tide won or the Blazers won"

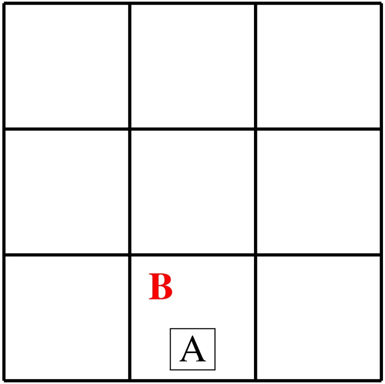
Wumpus Entailment Example



Models

Number of possible models when placing pits in three squares:

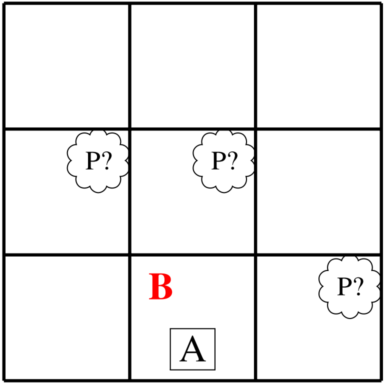
Wumpus Entailment Example



Models

Number of possible models when placing pits in three squares:

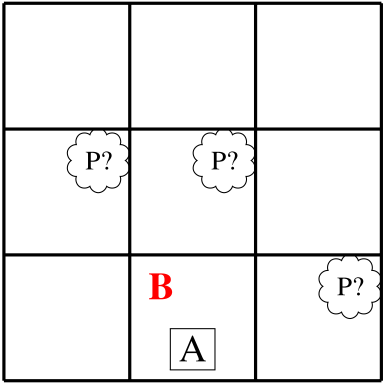
Wumpus Entailment Example



Models

Number of possible models when placing pits in three squares:

Wumpus Entailment Example

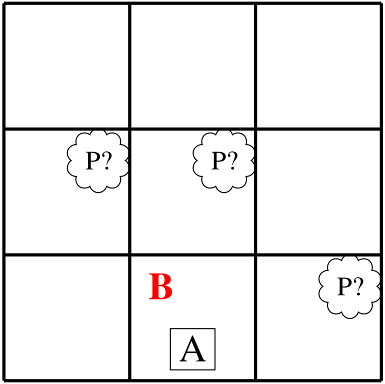


Models

Number of possible models when placing pits in three squares:

$2^3 = 8$

Wumpus Entailment Example

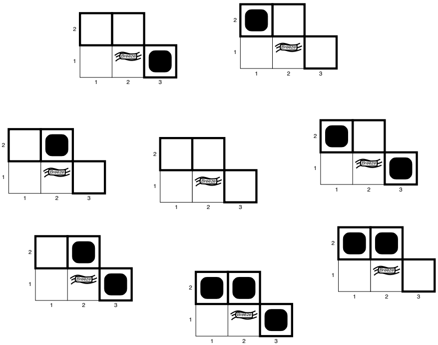


Models

Number of possible models when placing pits in three squares:

$2^3 = 8$

Wumpus Entailment Example

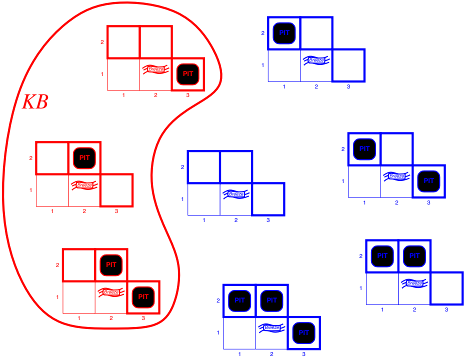


KB = rules + observations

α_1 = “[1,2] is safe”
KB $\models \alpha_1$

α_2 = “[2,2] is safe”
KB $\not\models \alpha_2$

Wumpus Entailment Example

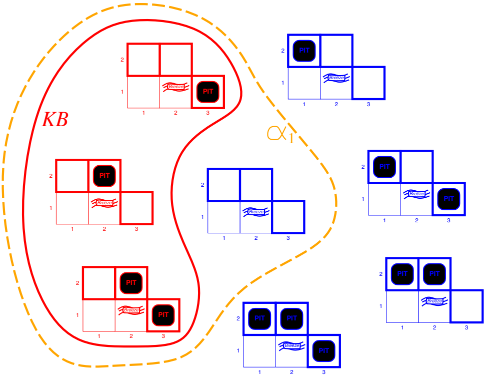


KB = rules + observations

α_1 = "[1,2] is safe"
KB $\models \alpha_1$

α_2 = "[2,2] is safe"
KB $\not\models \alpha_2$

Wumpus Entailment Example



KB = rules + observations

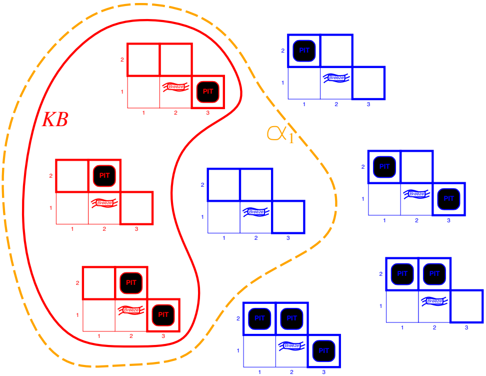
α_1 = "[1,2] is safe"

KB $\models \alpha_1$

α_2 = "[2,2] is safe"

KB $\not\models \alpha_2$

Wumpus Entailment Example

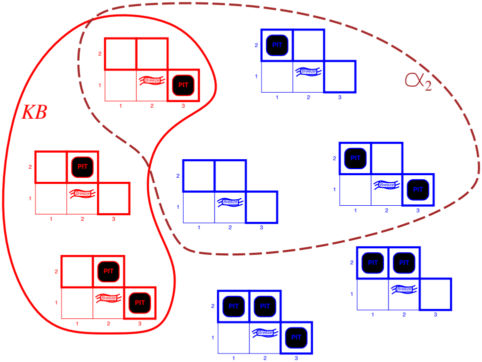


KB = rules + observations

α_1 = "[1,2] is safe"
KB $\models \alpha_1$

α_2 = "[2,2] is safe"
KB $\not\models \alpha_2$

Wumpus Entailment Example

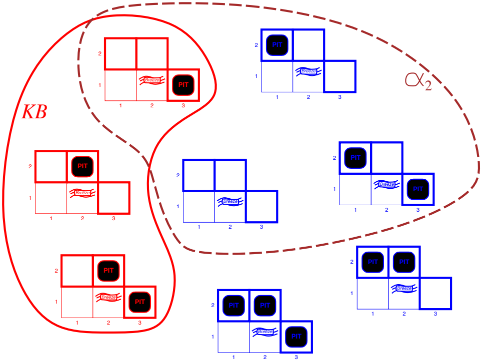


KB = rules + observations

α_1 = “[1,2] is safe”
KB $\models \alpha_1$

α_2 = “[2,2] is safe”
KB $\not\models \alpha_2$

Wumpus Entailment Example



KB = rules + observations

α_1 = "[1,2] is safe"
KB $\models \alpha_1$

α_2 = "[2,2] is safe"
KB $\not\models \alpha_2$

Outline

Propositional Logic Syntax

Basics

- Symbols look like P, Q, R, etc.
- Connectives look like:

$\neg$	negation, a.k.a. “not”
$\wedge$	conunction, a.k.a. “and”
$\vee$	disjunction, a.k.a. “or”
$\Rightarrow$	implication, a.k.a. “implies”
$\Leftrightarrow$	biconditional, a.k.a. “equivalent”

Examples

P $\quad$ $\neg Q$ $\quad$ $\neg Q \wedge P$ $\quad$ $R \Rightarrow \neg Q \wedge P$

Propositional Logic Syntax

Basics

- Symbols look like P, Q, R, etc.
- Connectives look like:

$\neg$	negation, a.k.a. “not”
$\wedge$	conunction, a.k.a. “and”
$\vee$	disjunction, a.k.a. “or”
$\Rightarrow$	implication, a.k.a. “implies”
$\Leftrightarrow$	biconditional, a.k.a. “equivalent”

Examples

P $\qquad$ $\neg Q$ $\qquad$ $\neg Q \wedge P$ $\qquad$ $R \Rightarrow \neg Q \wedge P$

Propositional Logic Semantics

Symbols

A model specifies *true* or *false* for each symbol
E.g. $\{P = true, Q = false, R = true\}$

Connectives

$\neg S$	is true iff	S is *false*		
$S_1 \wedge S_2$	is true iff	S_1 is *true*	and	S_2 is *true*
$S_1 \vee S_2$	is true iff	S_1 is *true*	or	S_2 is *true*
$S_1 \Rightarrow S_2$	is true iff	S_1 is *false*	or	S_2 is *true*
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is *true*	and	$S_2 \Rightarrow S_1$ is *true*

Propositional Logic Semantics

Symbols

A model specifies *true* or *false* for each symbol

E.g. $\{P = \mathit{true}, Q = \mathit{false}, R = \mathit{true}\}$

Connectives

$\neg S$	is true iff	S is *false*		
$S_1 \wedge S_2$	is true iff	S_1 is *true*	and	S_2 is *true*
$S_1 \vee S_2$	is true iff	S_1 is *true*	or	S_2 is *true*
$S_1 \Rightarrow S_2$	is true iff	S_1 is *false*	or	S_2 is *true*
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is *true*	and	$S_2 \Rightarrow S_1$ is *true*

Propositional Logic Complex Expressions

Given

$P = \mathit{true} \quad Q = \mathit{false} \quad R = \mathit{true}$

Evaluate

$P \vee R \Rightarrow \neg(Q \wedge \neg R)$

1. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \neg \mathit{true})$
2. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \mathit{false})$
3. $\mathit{true} \vee \mathit{true} \Rightarrow \neg \mathit{false}$
4. $\mathit{true} \vee \mathit{true} \Rightarrow \mathit{true}$
5. $\mathit{true} \Rightarrow \mathit{true}$
6. true

Propositional Logic Complex Expressions

Given

$P = \mathit{true} \quad Q = \mathit{false} \quad R = \mathit{true}$

Evaluate

$P \lor R \Rightarrow \neg(Q \land \neg R)$

1. $\mathit{true} \lor \mathit{true} \Rightarrow \neg(\mathit{false} \land \neg \mathit{true})$
2. $\mathit{true} \lor \mathit{true} \Rightarrow \neg(\mathit{false} \land \mathit{false})$
3. $\mathit{true} \lor \mathit{true} \Rightarrow \neg \mathit{false}$
4. $\mathit{true} \lor \mathit{true} \Rightarrow \mathit{true}$
5. $\mathit{true} \Rightarrow \mathit{true}$
6. true

Propositional Logic Complex Expressions

Given

$P = \mathit{true} \quad Q = \mathit{false} \quad R = \mathit{true}$

Evaluate

$P \vee R \Rightarrow \neg(Q \wedge \neg R)$

1. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \neg \mathit{true})$
2. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \mathit{false})$
3. $\mathit{true} \vee \mathit{true} \Rightarrow \neg \mathit{false}$
4. $\mathit{true} \vee \mathit{true} \Rightarrow \mathit{true}$
5. $\mathit{true} \Rightarrow \mathit{true}$
6. true

Propositional Logic Complex Expressions

Given

$P = \mathit{true} \quad Q = \mathit{false} \quad R = \mathit{true}$

Evaluate

$P \vee R \Rightarrow \neg(Q \wedge \neg R)$

1. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \neg \mathit{true})$
2. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \mathit{false})$
3. $\mathit{true} \vee \mathit{true} \Rightarrow \neg \mathit{false}$
4. $\mathit{true} \vee \mathit{true} \Rightarrow \mathit{true}$
5. $\mathit{true} \Rightarrow \mathit{true}$
6. true

Propositional Logic Complex Expressions

Given

$P = \mathit{true} \quad Q = \mathit{false} \quad R = \mathit{true}$

Evaluate

$P \vee R \Rightarrow \neg(Q \wedge \neg R)$

1. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \neg \mathit{true})$
2. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \mathit{false})$
3. $\mathit{true} \vee \mathit{true} \Rightarrow \neg \mathit{false}$
4. $\mathit{true} \vee \mathit{true} \Rightarrow \mathit{true}$
5. $\mathit{true} \Rightarrow \mathit{true}$
6. true

Propositional Logic Complex Expressions

Given

$P = \mathit{true} \quad Q = \mathit{false} \quad R = \mathit{true}$

Evaluate

$P \vee R \Rightarrow \neg(Q \wedge \neg R)$

1. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \neg \mathit{true})$
2. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \mathit{false})$
3. $\mathit{true} \vee \mathit{true} \Rightarrow \neg \mathit{false}$
4. $\mathit{true} \vee \mathit{true} \Rightarrow \mathit{true}$
5. $\mathit{true} \Rightarrow \mathit{true}$
6. true

Propositional Logic Complex Expressions

Given

$P = \mathit{true} \quad Q = \mathit{false} \quad R = \mathit{true}$

Evaluate

$P \vee R \Rightarrow \neg(Q \wedge \neg R)$

1. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \neg \mathit{true})$
2. $\mathit{true} \vee \mathit{true} \Rightarrow \neg(\mathit{false} \wedge \mathit{false})$
3. $\mathit{true} \vee \mathit{true} \Rightarrow \neg \mathit{false}$
4. $\mathit{true} \vee \mathit{true} \Rightarrow \mathit{true}$
5. $\mathit{true} \Rightarrow \mathit{true}$
6. true

Implication vs. Entailment

Implication ($\alpha \Rightarrow \beta$)

- α is *false* or β is *true*

Entailment ($\alpha \models \beta$)

- In all models where α is true, β is also true

Example

the earth is flat the moon is made of green cheese

Relation between Implication and Entailment

$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ in all models

Implication vs. Entailment

Implication ($\alpha \Rightarrow \beta$)

- α is *false* or β is *true*

Entailment ($\alpha \models \beta$)

- In all models where α is true, β is also true

Example

the earth is flat $\qquad$ the moon is made of green cheese

Relation between Implication and Entailment

$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ in all models

Implication vs. Entailment

Implication ($\alpha \Rightarrow \beta$)

- α is *false* or β is *true*

Entailment ($\alpha \models \beta$)

- In all models where α is true, β is also true

Example

the earth is flat $\stackrel{?}{\Rightarrow}$ the moon is made of green cheese

Relation between Implication and Entailment

$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ in all models

Implication vs. Entailment

Implication ($\alpha \Rightarrow \beta$)

- α is *false* or β is *true*

Entailment ($\alpha \models \beta$)

- In all models where α is true, β is also true

Example

the earth is flat $\Rightarrow$ the moon is made of green cheese

Relation between Implication and Entailment

$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ in all models

Implication vs. Entailment

Implication ($\alpha \Rightarrow \beta$)

- α is *false* or β is *true*

Entailment ($\alpha \models \beta$)

- In all models where α is true, β is also true

Example

the earth is flat $\overset{?}{\models}$ the moon is made of green cheese

Relation between Implication and Entailment

$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ in all models

Implication vs. Entailment

Implication ($\alpha \Rightarrow \beta$)

- α is *false* or β is *true*

Entailment ($\alpha \models \beta$)

- In all models where α is true, β is also true

Example

the earth is flat $\not\models$ the moon is made of green cheese

Relation between Implication and Entailment

$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ in all models

Implication vs. Entailment

Implication ($\alpha \Rightarrow \beta$)

- α is *false* or β is *true*

Entailment ($\alpha \models \beta$)

- In all models where α is true, β is also true

Example

the earth is flat $\not\models$ the moon is made of green cheese

Relation between Implication and Entailment

$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ in all models

Truth Tables

Key Idea

- Enumerate all possible values for symbols
- Calculate expression for each set of values

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Truth Table Exercise

Problem

Construct a truth table for $P \lor R \Rightarrow \neg(Q \land \neg R)$

Truth Table Exercise

Problem

Construct a truth table for $P \lor R \Rightarrow \neg(Q \land \neg R)$

P	Q	R	$P \lor R \Rightarrow \neg(Q \land \neg R)$
true	*true*	*true*	
true	*true*	*false*	
true	*false*	*true*	
true	*false*	*false*	
false	*true*	*true*	
false	*true*	*false*	
false	*false*	*true*	
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	
true	*false*	*true*	
true	*false*	*false*	
false	*true*	*true*	
false	*true*	*false*	
false	*false*	*true*	
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	*false*
true	*false*	*true*	
true	*false*	*false*	
false	*true*	*true*	
false	*true*	*false*	
false	*false*	*true*	
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	*false*
true	*false*	*true*	*true*
true	*false*	*false*	
false	*true*	*true*	
false	*true*	*false*	
false	*false*	*true*	
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	*false*
true	*false*	*true*	*true*
true	*false*	*false*	*true*
false	*true*	*true*	
false	*true*	*false*	
false	*false*	*true*	
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	*false*
true	*false*	*true*	*true*
true	*false*	*false*	*true*
false	*true*	*true*	*true*
false	*true*	*false*	
false	*false*	*true*	
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	*false*
true	*false*	*true*	*true*
true	*false*	*false*	*true*
false	*true*	*true*	*true*
false	*true*	*false*	*true*
false	*false*	*true*	
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	*false*
true	*false*	*true*	*true*
true	*false*	*false*	*true*
false	*true*	*true*	*true*
false	*true*	*false*	*true*
false	*false*	*true*	*true*
false	*false*	*false*	

Truth Table Exercise

Problem

Construct a truth table for $P \vee R \Rightarrow \neg(Q \wedge \neg R)$

P	Q	R	$P \vee R \Rightarrow \neg(Q \wedge \neg R)$
true	*true*	*true*	*true*
true	*true*	*false*	*false*
true	*false*	*true*	*true*
true	*false*	*false*	*true*
false	*true*	*true*	*true*
false	*true*	*false*	*true*
false	*false*	*true*	*true*
false	*false*	*false*	*true*

Logical Equivalences

$(\alpha \wedge \beta) \Leftrightarrow (\beta \wedge \alpha)$	Commutativity of $\wedge$
$(\alpha \vee \beta) \Leftrightarrow (\beta \vee \alpha)$	Commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \Leftrightarrow (\alpha \wedge (\beta \wedge \gamma))$	Associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \Leftrightarrow (\alpha \vee (\beta \vee \gamma))$	Associativity of $\vee$
$\neg(\neg\alpha) \Leftrightarrow \alpha$	Double-negation elimination
$(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\beta \Rightarrow \neg\alpha)$	Contraposition
$(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\alpha \vee \beta)$	Implication elimination
$(\alpha \Leftrightarrow \beta) \Leftrightarrow ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	Biconditional elimination
$\neg(\alpha \wedge \beta) \Leftrightarrow (\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta) \Leftrightarrow (\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \Leftrightarrow ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	Distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \Leftrightarrow ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	Distributivity of $\vee$ over $\wedge$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \wedge R \wedge Q)$

Given: $P \wedge R \Rightarrow \neg Q$

Given	$P \wedge R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \wedge R) \vee \neg Q$
De Morgan	$\neg P \vee \neg R \vee \neg Q$
De Morgan	$\neg(P \wedge R \wedge Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \wedge R \wedge Q)$
Given: $P \wedge R \Rightarrow \neg Q$

Given	$P \wedge R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \wedge R) \vee \neg Q$
De Morgan	$\neg P \vee \neg R \vee \neg Q$
De Morgan	$\neg(P \wedge R \wedge Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \land R \land Q)$

Given: $P \land R \Rightarrow \neg Q$

Given	$P \land R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \land R) \lor \neg Q$
De Morgan	$\neg P \lor \neg R \lor \neg Q$
De Morgan	$\neg(P \land R \land Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \wedge R \wedge Q)$

Given: $P \wedge R \Rightarrow \neg Q$

Given	$P \wedge R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \wedge R) \vee \neg Q$
De Morgan	$\neg P \vee \neg R \vee \neg Q$
De Morgan	$\neg(P \wedge R \wedge Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \wedge R \wedge Q)$

Given: $P \wedge R \Rightarrow \neg Q$

Given	$P \wedge R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \wedge R) \vee \neg Q$
De Morgan	$\neg P \vee \neg R \vee \neg Q$
De Morgan	$\neg(P \wedge R \wedge Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \wedge R \wedge Q)$

Given: $P \wedge R \Rightarrow \neg Q$

Given	$P \wedge R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \wedge R) \vee \neg Q$
De Morgan	$\neg P \vee \neg R \vee \neg Q$
De Morgan	$\neg(P \wedge R \wedge Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \wedge R \wedge Q)$
Given: $P \wedge R \Rightarrow \neg Q$

Given	$P \wedge R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \wedge R) \vee \neg Q$
De Morgan	$\neg P \vee \neg R \vee \neg Q$
De Morgan	$\neg(P \wedge R \wedge Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \land R \land Q)$
Given: $P \land R \Rightarrow \neg Q$

Given	$P \land R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \land R) \lor \neg Q$
De Morgan	$\neg P \lor \neg R \lor \neg Q$
De Morgan	$\neg(P \land R \land Q)$

Reasoning

Key Ideas

- Use logical equivalences etc. to prove things
- No need for a truth table!

Example

Prove: $\neg(P \land R \land Q)$

Given: $P \land R \Rightarrow \neg Q$

Given	$P \land R \Rightarrow \neg Q$
Implication Elimination	$\neg(P \land R) \lor \neg Q$
De Morgan	$\neg P \lor \neg R \lor \neg Q$
De Morgan	$\neg(P \land R \land Q)$

Additional Reasoning Patterns

Modus Ponens

$$\frac{\begin{array}{l}\alpha \Rightarrow \beta \\ \alpha\end{array}}{\beta}$$

$$\frac{\begin{array}{l}x > 2 \Rightarrow x \neq 1 \\ x > 2\end{array}}{x \neq 1}$$

And elimination

$$\frac{\alpha \wedge \beta}{\beta}$$

$$\frac{x = 0 \wedge y = 42}{y = 42}$$

Resolution

$$\frac{\begin{array}{l}\alpha \vee \beta \\ \neg\alpha\end{array}}{\beta}$$

$$\frac{\begin{array}{l}x = 1 \vee x = 2 \\ x \neq 1\end{array}}{x = 2}$$

Additional Reasoning Patterns

Modus Ponens

$$\frac{\alpha \Rightarrow \beta \qquad \alpha}{\beta}$$

$$\frac{x > 2 \Rightarrow x \neq 1 \qquad x > 2}{x \neq 1}$$

And elimination

$$\frac{\alpha \wedge \beta}{\beta}$$

$$\frac{x = 0 \wedge y = 42}{y = 42}$$

Resolution

$$\frac{\alpha \vee \beta \qquad \neg\alpha}{\beta}$$

$$\frac{x = 1 \vee x = 2 \qquad x \neq 1}{x = 2}$$

Additional Reasoning Patterns

Modus Ponens

$$\frac{\begin{array}{l}\alpha \Rightarrow \beta \\ \alpha\end{array}}{\beta}$$

$$\frac{\begin{array}{l}x > 2 \Rightarrow x \neq 1 \\ x > 2\end{array}}{x \neq 1}$$

And elimination

$$\frac{\alpha \wedge \beta}{\beta}$$

$$\frac{x = 0 \wedge y = 42}{y = 42}$$

Resolution

$$\frac{\begin{array}{l}\alpha \vee \beta \\ \neg\alpha\end{array}}{\beta}$$

$$\frac{\begin{array}{l}x = 1 \vee x = 2 \\ x \neq 1\end{array}}{x = 2}$$

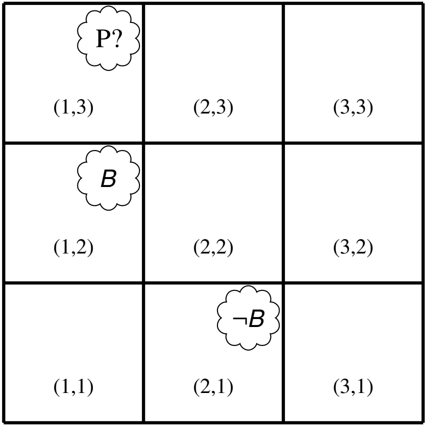
Reasoning Exercise

Prove: $P_{1,3}$

Given:

$B_{1,2}$
$\neg B_{2,1}$
$B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$
$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

The reasoning patterns are in your book on pages 249, 250 and 252



Prove: $P_{1,3}$

$B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$	Given
$B_{1,2} \Rightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$	Biconditional Elimination
$B_{1,2}$	Given
$P_{1,1} \vee P_{2,2} \vee P_{1,3}$	Modus Ponens
$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$	Given
$(P_{1,1} \vee P_{2,2} \vee P_{3,1}) \Rightarrow B_{2,1}$	Biconditional Elimination
$\neg B_{2,1} \Rightarrow \neg(P_{1,1} \vee P_{2,2} \vee P_{3,1})$	Contraposition
$\neg B_{2,1}$	Given
$\neg(P_{1,1} \vee P_{2,2} \vee P_{3,1})$	Modus Ponens
$\neg P_{1,1} \wedge \neg P_{2,2} \wedge \neg P_{3,1}$	De Morgan
$\neg P_{1,1}$	And-Elimination
$P_{2,2} \vee P_{1,3}$	Resolution
$\neg P_{2,2}$	And-Elimination
$P_{1,3}$	Resolution

Outline

Inference Algorithms

Definition: Derives

Procedure i derives β from α ($\alpha \vdash_i \beta$) if:

when given α, procedure i is able to conclude β

Definition: Soundness

Procedure i is sound if:

whenever $\alpha \vdash_i \beta$ it is also true that $\alpha \models \beta$

Definition: Completeness

Procedure i is complete if:

whenever $\alpha \models \beta$ it is also true that $\alpha \vdash_i \beta$

Inference Algorithms

Definition: Derives

Procedure i derives β from α ($\alpha \vdash_i \beta$) if:

when given α, procedure i is able to conclude β

Definition: Soundness

Procedure i is sound if:

whenever $\alpha \vdash_i \beta$ it is also true that $\alpha \models \beta$

Definition: Completeness

Procedure i is complete if:

whenever $\alpha \models \beta$ it is also true that $\alpha \vdash_i \beta$

Inference Algorithms

Definition: Derives

Procedure i derives β from α ($\alpha \vdash_i \beta$) if:

when given α, procedure i is able to conclude β

Definition: Soundness

Procedure i is sound if:

whenever $\alpha \vdash_i \beta$ it is also true that $\alpha \models \beta$

Definition: Completeness

Procedure i is complete if:

whenever $\alpha \models \beta$ it is also true that $\alpha \vdash_i \beta$

Inference by Truth Table

Prove:

$\neg P_{1,2}$

Given:

$$
\begin{aligned}
R_1 &: \neg P_{1,1} \\
R_2 &: B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1}) \\
R_3 &: B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \\
R_4 &: \neg B_{1,1} \\
R_5 &: B_{2,1}
\end{aligned}
$$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Inference by Truth Table

Prove:

$\neg P_{1,2}$

Given:

$R_1 : \neg P_{1,1}$

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

$R_4 : \neg B_{1,1}$

$R_5 : B_{2,1}$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	*KB*
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Inference by Truth Table

Prove:

$\neg P_{1,2}$

Given:

$R_1: \quad \neg P_{1,1}$

$R_2: \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_3: \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

$R_4: \quad \neg B_{1,1}$

$R_5: \quad B_{2,1}$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Inference by Truth Table

Prove:

$\neg P_{1,2}$

Given:

$R_1 : \neg P_{1,1}$

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

$R_4 : \neg B_{1,1}$

$R_5 : B_{2,1}$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	*KB*
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Inference by Truth Table

Prove:

$\neg P_{1,2}$

Given:

$$R_1 : \neg P_{1,1} \qquad R_4 : \neg B_{1,1}$$
$$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}) \qquad R_5 : B_{2,1}$$
$$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	*KB*
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Truth Table Inference Code

```
def truth_table_entails(knowledge_base, query):
    # check all assignments of knowledge base and query symbols:
    # if the knowledge base is true the query should be true
    symbols = set.union(knowledge_base.symbols, query.symbols)
    return all(
        query.is_true_for(assignment)
        for assignment in all_models(symbols)
        if knowledge_base.is_true_for(assignment))

def all_models(symbols):
    # base case: no symbols, generate an empty assignment
    if not symbols:
        yield {}
    # recursive case: assign to the first symbol and recurse
    else:
        first, rest = symbols[0], symbols[1:]
        for assignment in all_models(rest):
            for value in [True, False]:
                assignment[first] = value
                yield assignment
```

Truth Table Inference Code

```
def truth_table_entails(knowledge_base, query):
    # check all assignments of knowledge base and query symbols:
    # if the knowledge base is true the query should be true
    symbols = set.union(knowledge_base.symbols, query.symbols)
    return all(
        query.is_true_for(assignment)
        for assignment in all_models(symbols)
        if knowledge_base.is_true_for(assignment))

def all_models(symbols):
    # base case: no symbols, generate an empty assignment
    if not symbols:
        yield {}
    # recursive case: assign to the first symbol and recurse
    else:
        first, rest = symbols[0], symbols[1:]
        for assignment in all_models(rest):
            for value in [True, False]:
                assignment[first] = value
                yield assignment
```

Truth Table Inference Code

```
def truth_table_entails(knowledge_base, query):
    # check all assignments of knowledge base and query symbols:
    # if the knowledge base is true the query should be true
    symbols = set.union(knowledge_base.symbols, query.symbols)
    return all(
        query.is_true_for(assignment)
        for assignment in all_models(symbols)
        if knowledge_base.is_true_for(assignment))

def all_models(symbols):
    # base case: no symbols, generate an empty assignment
    if not symbols:
        yield {}
    # recursive case: assign to the first symbol and recurse
    else:
        first, rest = symbols[0], symbols[1:]
        for assignment in all_models(rest):
            for value in [True, False]:
                assignment[first] = value
                yield assignment
```

Truth Table Inference Code

```
def truth_table_entails(knowledge_base, query):
    # check all assignments of knowledge base and query symbols:
    # if the knowledge base is true the query should be true
    symbols = set.union(knowledge_base.symbols, query.symbols)
    return all(
        query.is_true_for(assignment)
        for assignment in all_models(symbols)
        if knowledge_base.is_true_for(assignment))

def all_models(symbols):
    # base case: no symbols, generate an empty assignment
    if not symbols:
        yield {}
    # recursive case: assign to the first symbol and recurse
    else:
        first, rest = symbols[0], symbols[1:]
        for assignment in all_models(rest):
            for value in [True, False]:
                assignment[first] = value
                yield assignment
```

Truth Table Inference Code

```python
def truth_table_entails(knowledge_base, query):
    # check all assignments of knowledge base and query symbols:
    # if the knowledge base is true the query should be true
    symbols = set.union(knowledge_base.symbols, query.symbols)
    return all(
        query.is_true_for(assignment)
        for assignment in all_models(symbols)
        if knowledge_base.is_true_for(assignment))

def all_models(symbols):
    # base case: no symbols, generate an empty assignment
    if not symbols:
        yield {}
    # recursive case: assign to the first symbol and recurse
    else:
        first, rest = symbols[0], symbols[1:]
        for assignment in all_models(rest):
            for value in [True, False]:
                assignment[first] = value
                yield assignment
```

Truth Table Inference Properties

Definitions (Again)

Sound	if $\alpha \vdash_i \beta$ then $\alpha \models \beta$
Complete	if $\alpha \models \beta$ then $\alpha \vdash_i \beta$

Truth Table Inference Properties

Sound?	Yes, directly implements entailment definition
Complete?	Yes, explores all possibilities
Time?	Given n symbols, takes $O(2^n)$

Truth Table Inference Properties

Definitions (Again)

Sound	if $\alpha \vdash_i \beta$ then $\alpha \models \beta$
Complete	if $\alpha \models \beta$ then $\alpha \vdash_i \beta$

Truth Table Inference Properties

Sound?	Yes, directly implements entailment definition
Complete?	Yes, explores all possibilities
Time?	Given n symbols, takes $O(2^n)$

Truth Table Inference Properties

Definitions (Again)

Sound	if $\alpha \vdash_i \beta$ then $\alpha \models \beta$
Complete	if $\alpha \models \beta$ then $\alpha \vdash_i \beta$

Truth Table Inference Properties

Sound?	Yes, directly implements entailment definition
Complete?	Yes, explores all possibilities
Time?	Given n symbols, takes $O(2^n)$

Truth Table Inference Properties

Definitions (Again)

Sound	if $\alpha \vdash_i \beta$ then $\alpha \models \beta$
Complete	if $\alpha \models \beta$ then $\alpha \vdash_i \beta$

Truth Table Inference Properties

Sound?	Yes, directly implements entailment definition
Complete?	Yes, explores all possibilities
Time?	Given n symbols, takes $O(2^n)$

Forward and Backward Chaining

Key Idea

Inference is easier if all statements are Horn Clauses:

$P_1 \wedge \ldots \wedge P_n \Rightarrow Q$

Given:

$B \wedge L \Rightarrow M$

$A \wedge B \Rightarrow L$

A

B

Prove M:

Forward and Backward Chaining

Key Idea

Inference is easier if all statements are Horn Clauses:

$P_1 \wedge \ldots \wedge P_n \Rightarrow Q$

Given:

$B \wedge L \Rightarrow M$

$A \wedge B \Rightarrow L$

A

B

Prove M:

Forward and Backward Chaining

Key Idea

Inference is easier if all statements are Horn Clauses:

$$P_1 \wedge \ldots \wedge P_n \Rightarrow Q$$

Given:

$B \wedge L \Rightarrow M$

$A \wedge B \Rightarrow L$

A

B

Prove M:

A

B

$A \wedge B \Rightarrow L$

Forward and Backward Chaining

Key Idea

Inference is easier if all statements are Horn Clauses:

$$P_1 \wedge \ldots \wedge P_n \Rightarrow Q$$

Given:

$B \wedge L \Rightarrow M$

$A \wedge B \Rightarrow L$

A

B

Prove M:

A

B

$A \wedge B \Rightarrow L$

L by Modus Ponens

Forward and Backward Chaining

Key Idea

Inference is easier if all statements are Horn Clauses:

$$P_1 \wedge \ldots \wedge P_n \Rightarrow Q$$

Given:

$B \wedge L \Rightarrow M$

$A \wedge B \Rightarrow L$

A

B

Prove M:

A

B

$A \wedge B \Rightarrow L$

L by Modus Ponens

B

$B \wedge L \Rightarrow M$

Forward and Backward Chaining

Key Idea

Inference is easier if all statements are Horn Clauses:

$$P_1 \wedge \ldots \wedge P_n \Rightarrow Q$$

Given:

$B \wedge L \Rightarrow M$

$A \wedge B \Rightarrow L$

A

B

Prove M:

$$\frac{A \qquad B \qquad A \wedge B \Rightarrow L}{L} \quad \text{by Modus Ponens}$$

$$\frac{L \qquad B \qquad B \wedge L \Rightarrow M}{M} \quad \text{by Modus Ponens}$$

Forward Chaining Code

```python
def forward_chaining_entails(knowledge_base, query):
    # count the symbols in the premise of each clause
    counts = {}
    for clause in knowledge_base.get_clauses():
        counts[clause] = len(clause.premise)
    # start with known symbols and search for non-inferred
    inferred = set()
    agenda = knowledge_base.get_true_symbols()
    while agenda:
        symbol = agenda.pop()
        # if the query was on the agenda, it is known to be true
        if symbol == query:
            return True
        # do not repeat symbols that have already been checked
        if symbol not in inferred:
            inferred.add(symbol)
            # update counts and infer conclusions when possible
            for clause in knowledge_base.get_clauses():
                if symbol in clause.premise:
                    counts[clause] -= 1
                    if counts[clause] == 0:
                        agenda.append(clause.head)
    return False
```

Forward Chaining Code

```
def forward_chaining_entails(knowledge_base, query):
    # count the symbols in the premise of each clause
    counts = {}
    for clause in knowledge_base.get_clauses():
        counts[clause] = len(clause.premise)
    # start with known symbols and search for non-inferred
    inferred = set()
    agenda = knowledge_base.get_true_symbols()
    while agenda:
        symbol = agenda.pop()
        # if the query was on the agenda, it is known to be true
        if symbol == query:
            return True
        # do not repeat symbols that have already been checked
        if symbol not in inferred:
            inferred.add(symbol)
            # update counts and infer conclusions when possible
            for clause in knowledge_base.get_clauses():
                if symbol in clause.premise:
                    counts[clause] -= 1
                    if counts[clause] == 0:
                        agenda.append(clause.head)
    return False
```

Forward Chaining Code

```python
def forward_chaining_entails(knowledge_base, query):
    # count the symbols in the premise of each clause
    counts = {}
    for clause in knowledge_base.get_clauses():
        counts[clause] = len(clause.premise)
    # start with known symbols and search for non-inferred
    inferred = set()
    agenda = knowledge_base.get_true_symbols()
    while agenda:
        symbol = agenda.pop()
        # if the query was on the agenda, it is known to be true
        if symbol == query:
            return True
        # do not repeat symbols that have already been checked
        if symbol not in inferred:
            inferred.add(symbol)
            # update counts and infer conclusions when possible
            for clause in knowledge_base.get_clauses():
                if symbol in clause.premise:
                    counts[clause] -= 1
                    if counts[clause] == 0:
                        agenda.append(clause.head)
    return False
```

Forward Chaining Code

```
def forward_chaining_entails(knowledge_base, query):
    # count the symbols in the premise of each clause
    counts = {}
    for clause in knowledge_base.get_clauses():
        counts[clause] = len(clause.premise)
    # start with known symbols and search for non-inferred
    inferred = set()
    agenda = knowledge_base.get_true_symbols()
    while agenda:
        symbol = agenda.pop()
        # if the query was on the agenda, it is known to be true
        if symbol == query:
            return True
        # do not repeat symbols that have already been checked
        if symbol not in inferred:
            inferred.add(symbol)
            # update counts and infer conclusions when possible
            for clause in knowledge_base.get_clauses():
                if symbol in clause.premise:
                    counts[clause] -= 1
                    if counts[clause] == 0:
                        agenda.append(clause.head)
    return False
```

Forward Chaining Code

```
def forward_chaining_entails(knowledge_base, query):
    # count the symbols in the premise of each clause
    counts = {}
    for clause in knowledge_base.get_clauses():
        counts[clause] = len(clause.premise)
    # start with known symbols and search for non-inferred
    inferred = set()
    agenda = knowledge_base.get_true_symbols()
    while agenda:
        symbol = agenda.pop()
        # if the query was on the agenda, it is known to be true
        if symbol == query:
            return True
        # do not repeat symbols that have already been checked
        if symbol not in inferred:
            inferred.add(symbol)
            # update counts and infer conclusions when possible
            for clause in knowledge_base.get_clauses():
                if symbol in clause.premise:
                    counts[clause] -= 1
                    if counts[clause] == 0:
                        agenda.append(clause.head)
    return False
```

Forward Chaining Code

```
def forward_chaining_entails(knowledge_base, query):
    # count the symbols in the premise of each clause
    counts = {}
    for clause in knowledge_base.get_clauses():
        counts[clause] = len(clause.premise)
    # start with known symbols and search for non-inferred
    inferred = set()
    agenda = knowledge_base.get_true_symbols()
    while agenda:
        symbol = agenda.pop()
        # if the query was on the agenda, it is known to be true
        if symbol == query:
            return True
        # do not repeat symbols that have already been checked
        if symbol not in inferred:
            inferred.add(symbol)
            # update counts and infer conclusions when possible
            for clause in knowledge_base.get_clauses():
                if symbol in clause.premise:
                    counts[clause] -= 1
                    if counts[clause] == 0:
                        agenda.append(clause.head)
    return False
```

Forward Chaining Code

```python
def forward_chaining_entails(knowledge_base, query):
    # count the symbols in the premise of each clause
    counts = {}
    for clause in knowledge_base.get_clauses():
        counts[clause] = len(clause.premise)
    # start with known symbols and search for non-inferred
    inferred = set()
    agenda = knowledge_base.get_true_symbols()
    while agenda:
        symbol = agenda.pop()
        # if the query was on the agenda, it is known to be true
        if symbol == query:
            return True
        # do not repeat symbols that have already been checked
        if symbol not in inferred:
            inferred.add(symbol)
            # update counts and infer conclusions when possible
            for clause in knowledge_base.get_clauses():
                if symbol in clause.premise:
                    counts[clause] -= 1
                    if counts[clause] == 0:
                        agenda.append(clause.head)
    return False
```

Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

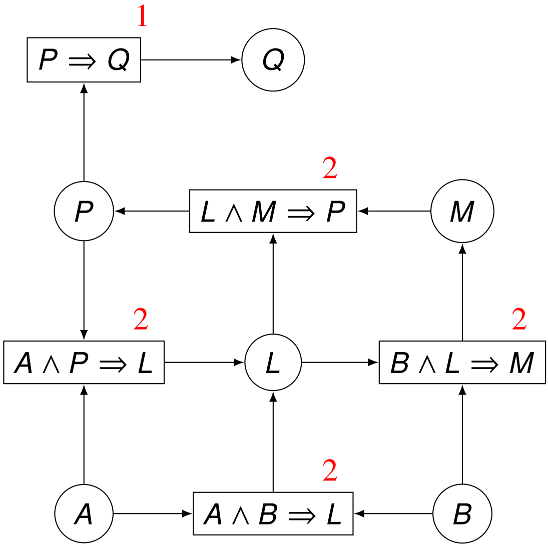
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

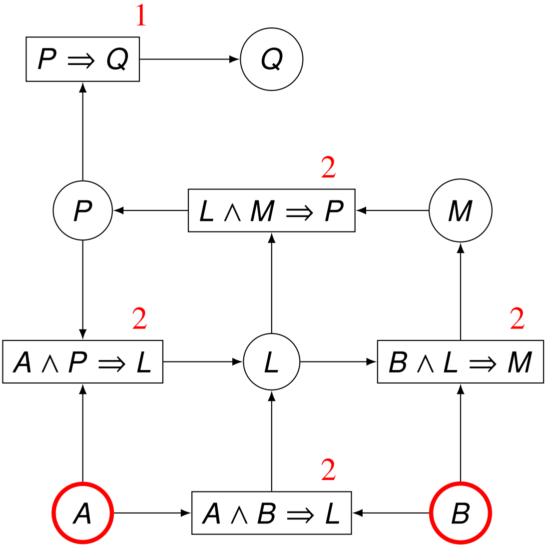
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

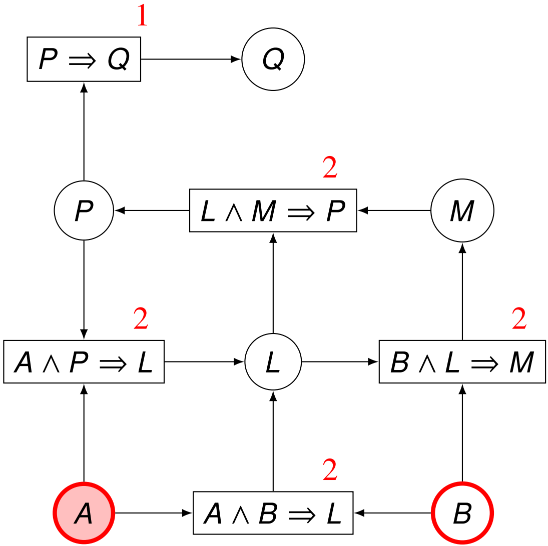
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

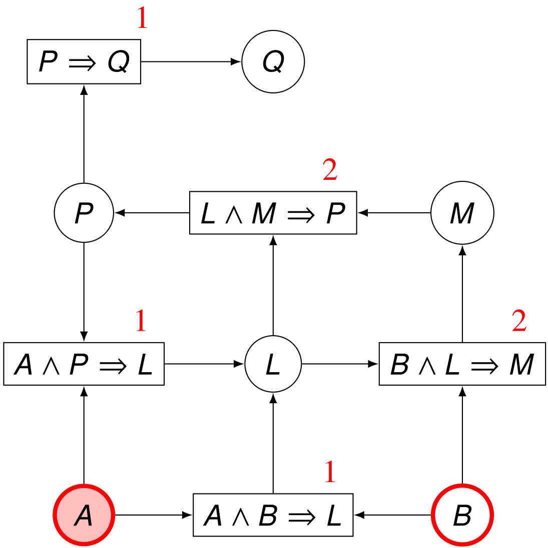
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

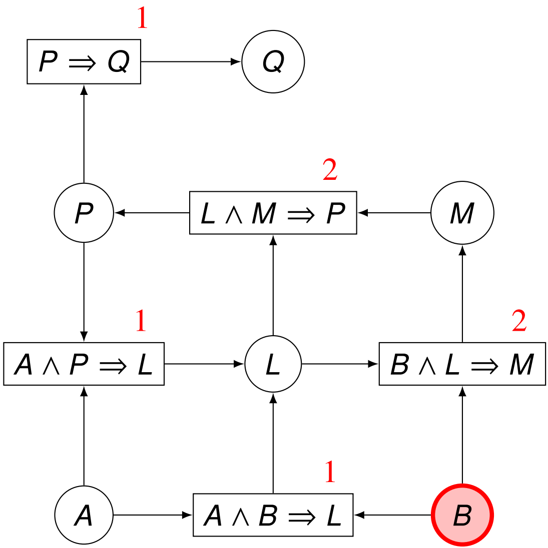
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

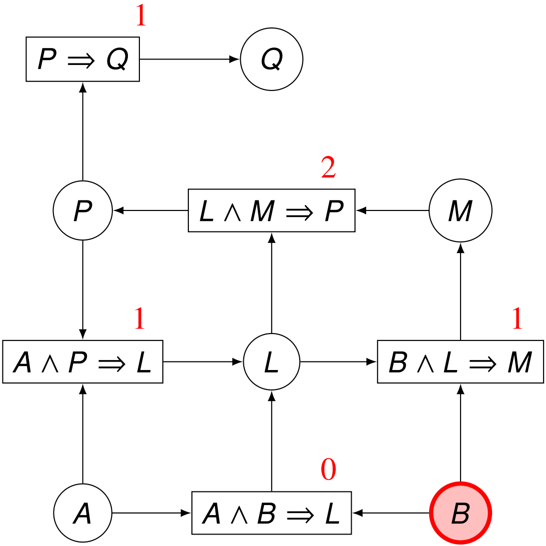
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

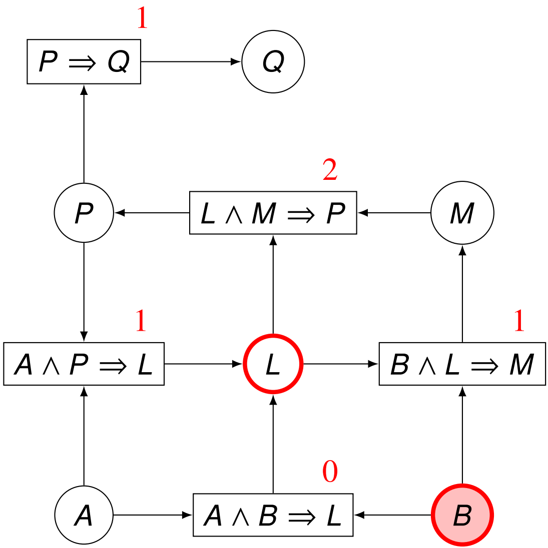
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \land M \Rightarrow P$

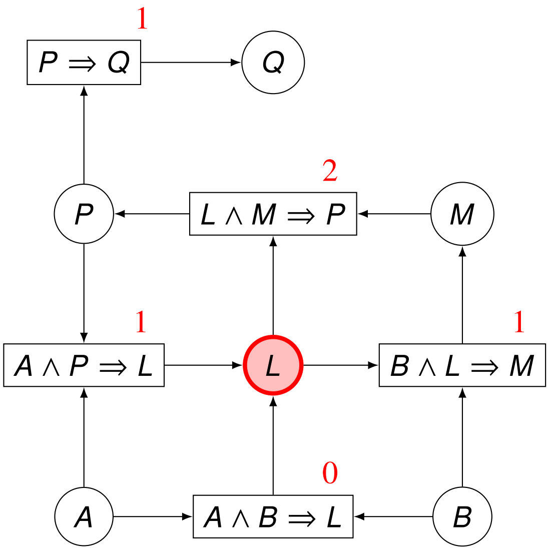
$B \land L \Rightarrow M$

$A \land P \Rightarrow L$

$A \land B \Rightarrow L$

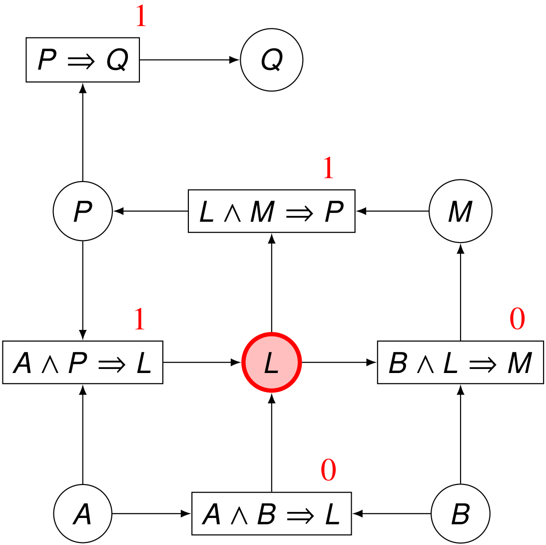
A

B



Forward Chaining Example

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

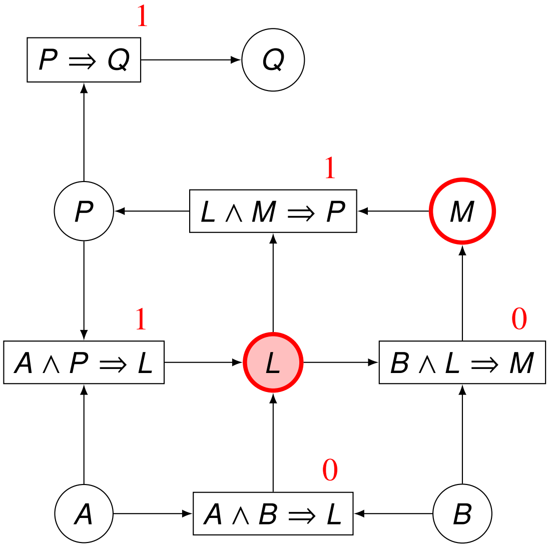
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

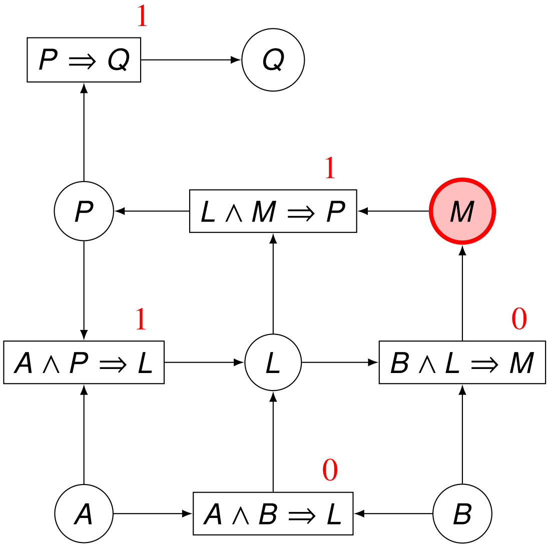
A

B



Forward Chaining Example

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

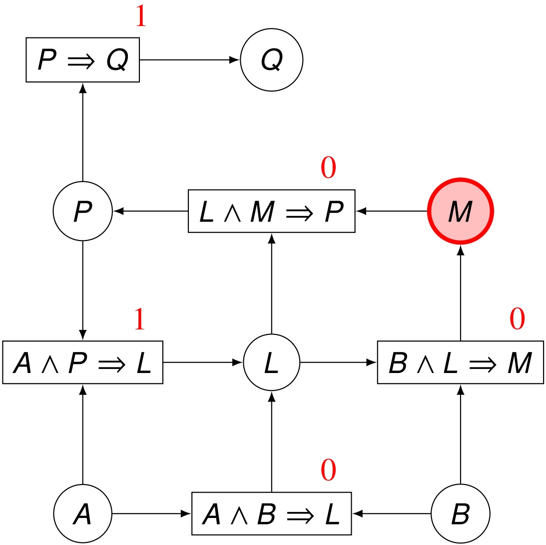
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

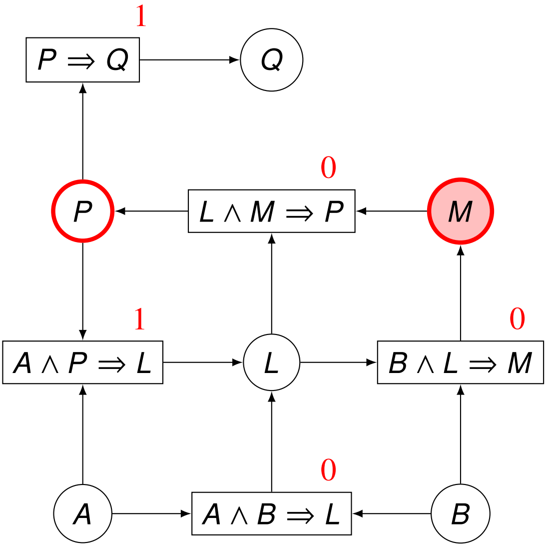
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

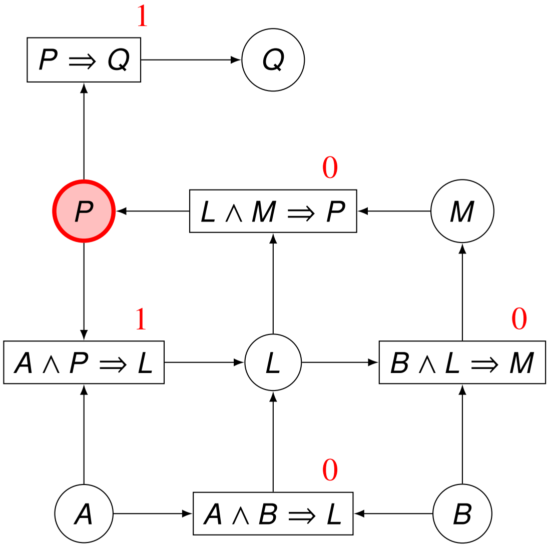
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

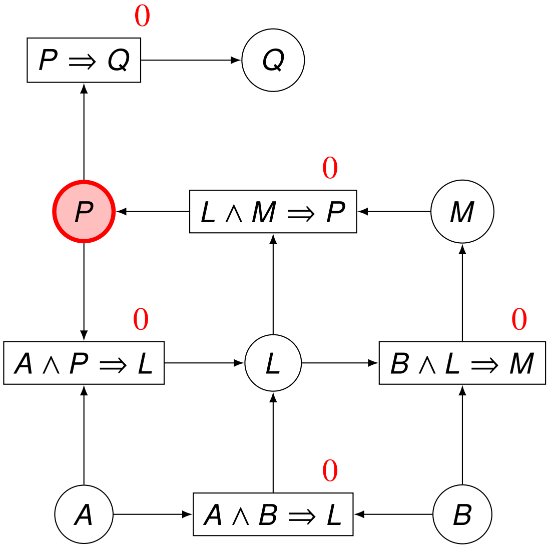
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

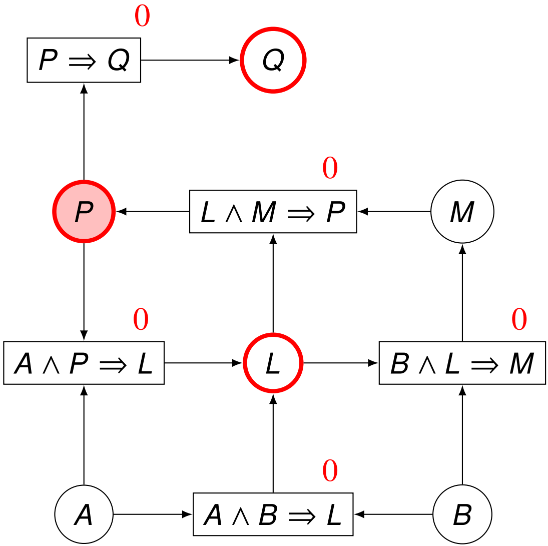
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

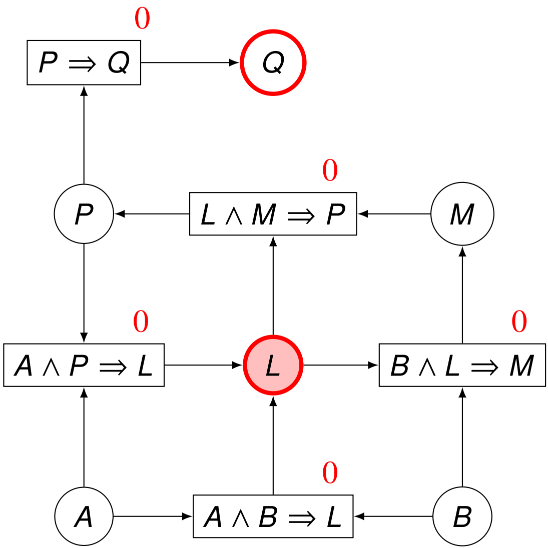
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

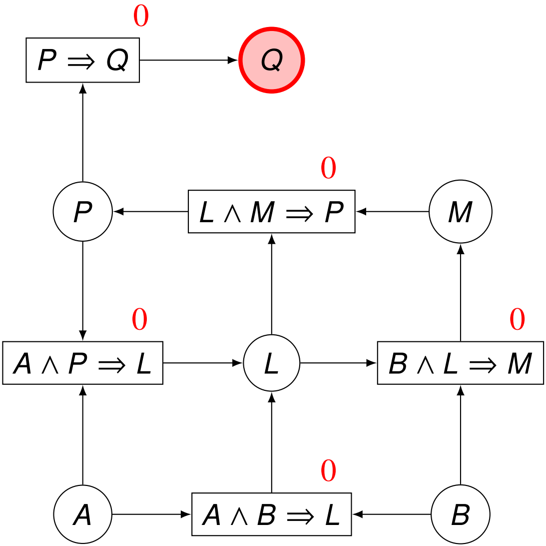
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward Chaining Properties

Properties

Sound?	Yes, uses Modus Ponens
Complete?	Yes (more on this in a moment)
Time?	$O(n)$ given n statements in KB

Reminder

- Requires Horn Clauses
- Won't work with Propositional logic in general

Forward Chaining Properties

Properties

Sound?	Yes, uses Modus Ponens
Complete?	Yes (more on this in a moment)
Time?	$O(n)$ given n statements in KB

Reminder

- Requires Horn Clauses
- Won't work with Propositional logic in general

Forward Chaining Properties

Properties

Sound?	Yes, uses Modus Ponens
Complete?	Yes (more on this in a moment)
Time?	$O(n)$ given n statements in KB

Reminder

- Requires Horn Clauses
- Won't work with Propositional logic in general

Forward Chaining Properties

Properties

Sound?	Yes, uses Modus Ponens
Complete?	Yes (more on this in a moment)
Time?	$O(n)$ given n statements in KB

Reminder

- Requires Horn Clauses
- Won't work with Propositional logic in general

Forward Chaining Properties

Properties

Sound?	Yes, uses Modus Ponens
Complete?	Yes (more on this in a moment)
Time?	$O(n)$ given n statements in KB

Reminder

- Requires Horn Clauses
- Won't work with Propositional logic in general

Forward Chaining Completeness

Prove: If $KB \models Q$ then $KB \vdash_{\textit{forward-chaining}} Q$

1. $KB \models Q$, so Q is *true* in every model of KB
2. Final `inferred` set is a model of KB
3. So Q is true in the `inferred` model
4. So $KB \vdash_{\textit{forward-chaining}} Q$

Prove: The final `inferred` set is a model of KB

1. Assume not, i.e. $P_1 \land \ldots \land P_n \Rightarrow Q$ is *false*
2. So $P_1 \land \ldots \land P_n$ is *true* and Q is *false* (= not inferred)
3. But then Q should have been inferred!
4. So `inferred` must be a model of KB

Forward Chaining Completeness

Prove: If $KB \models Q$ then $KB \vdash_{\textit{forward-chaining}} Q$

1. $KB \models Q$, so Q is *true* in every model of KB
2. Final `inferred` set is a model of KB
3. So Q is true in the `inferred` model
4. So $KB \vdash_{\textit{forward-chaining}} Q$

Prove: The final `inferred` set is a model of KB

1. Assume not, i.e. $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ is *false*
2. So $P_1 \wedge \ldots \wedge P_n$ is *true* and Q is *false* (= not inferred)
3. But then Q should have been inferred!
4. So `inferred` must be a model of KB

Forward Chaining Completeness

Prove: If $KB \models Q$ then $KB \vdash_{\textit{forward-chaining}} Q$

1. $KB \models Q$, so Q is *true* in every model of KB
2. Final `inferred` set is a model of KB
3. So Q is true in the `inferred` model
4. So $KB \vdash_{\textit{forward-chaining}} Q$

Prove: The final `inferred` set is a model of KB

1. Assume not, i.e. $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ is *false*
2. So $P_1 \wedge \ldots \wedge P_n$ is *true* and Q is *false* (= not inferred)
3. But then Q should have been inferred!
4. So `inferred` must be a model of KB

Forward Chaining Completeness

Prove: If $KB \models Q$ then $KB \vdash_{\textit{forward-chaining}} Q$

1. $KB \models Q$, so Q is *true* in every model of KB
2. Final `inferred` set is a model of KB
3. So Q is true in the `inferred` model
4. So $KB \vdash_{\textit{forward-chaining}} Q$

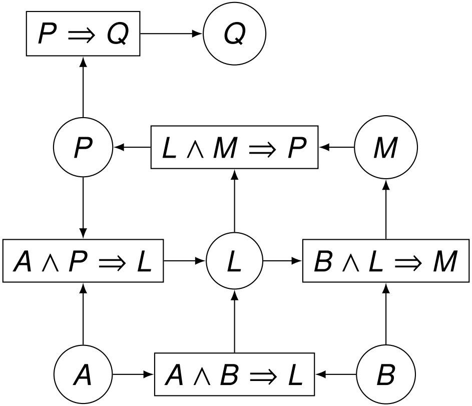
Prove: The final `inferred` set is a model of *KB*

1. Assume not, i.e. $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ is *false*
2. So $P_1 \wedge \ldots \wedge P_n$ is *true* and Q is *false* (= not inferred)
3. But then Q should have been inferred!
4. So `inferred` must be a model of KB

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

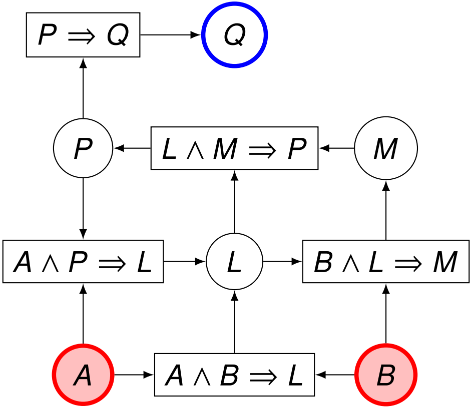
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

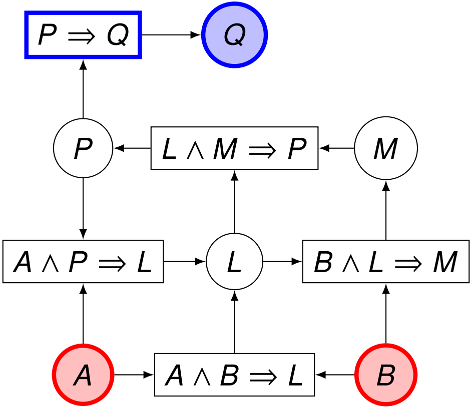
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

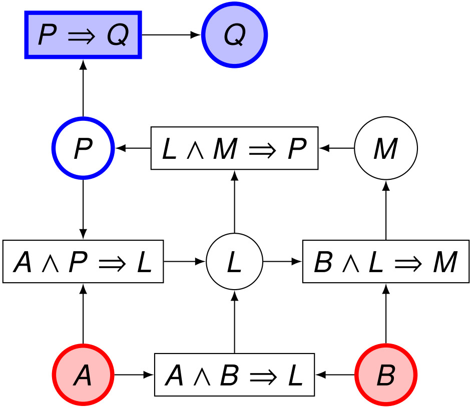
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

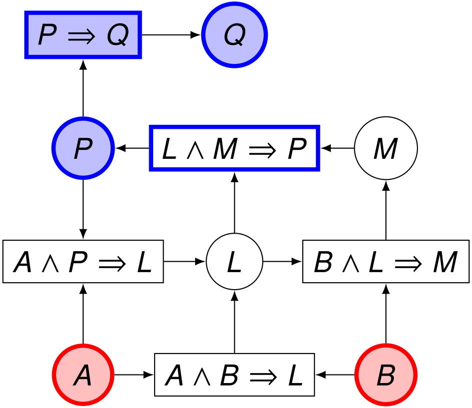
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

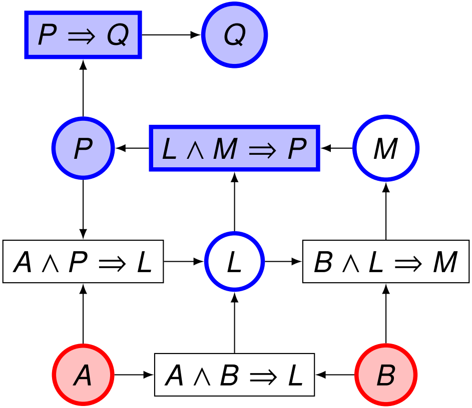
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

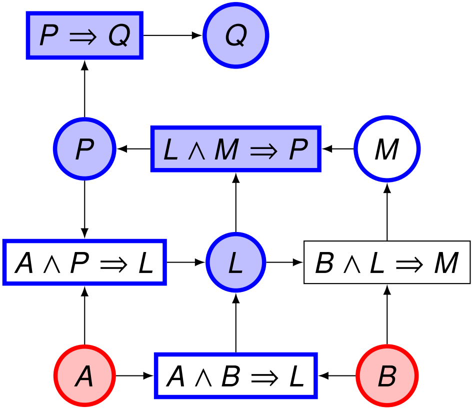
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \land \ldots \land P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

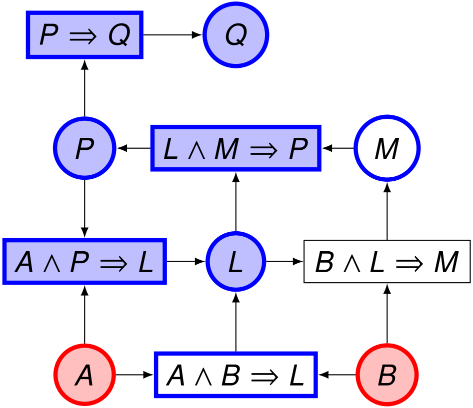
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \land \ldots \land P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

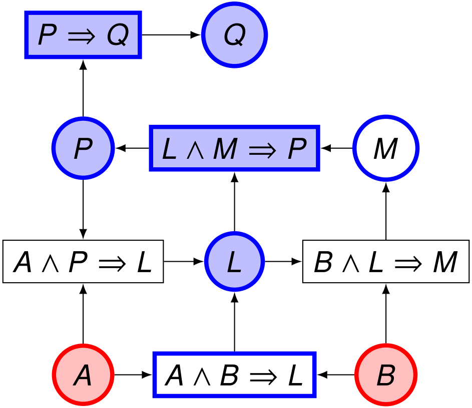
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

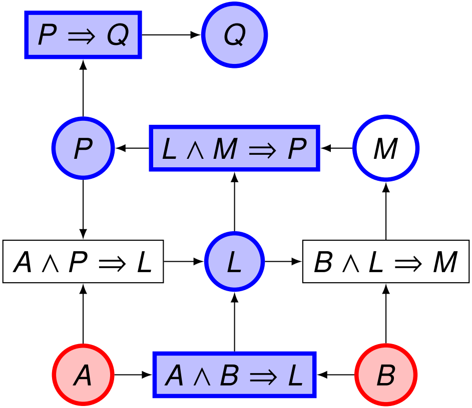
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

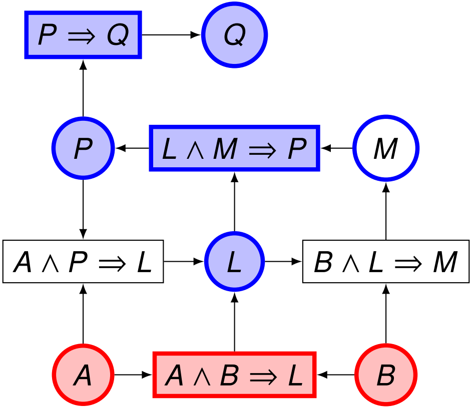
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

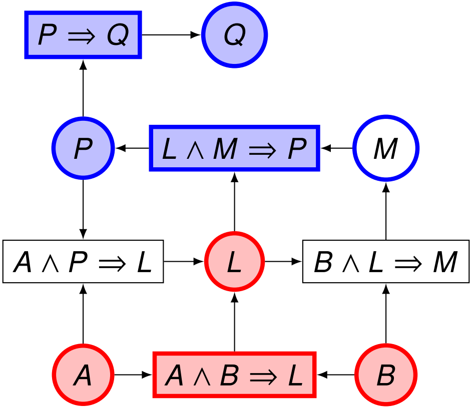
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

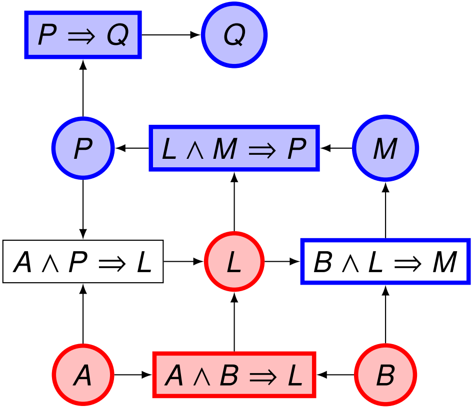
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

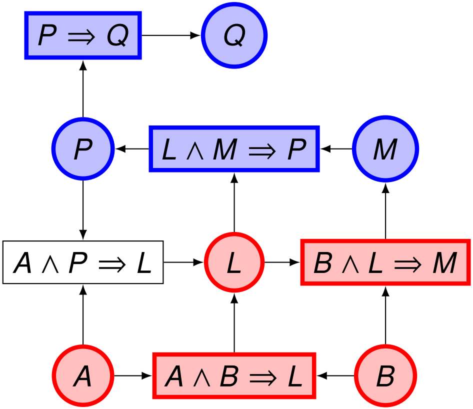
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

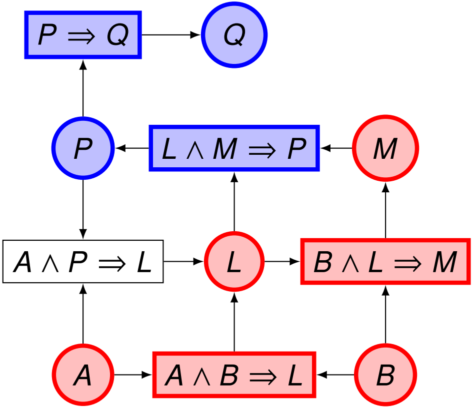
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \land \ldots \land P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

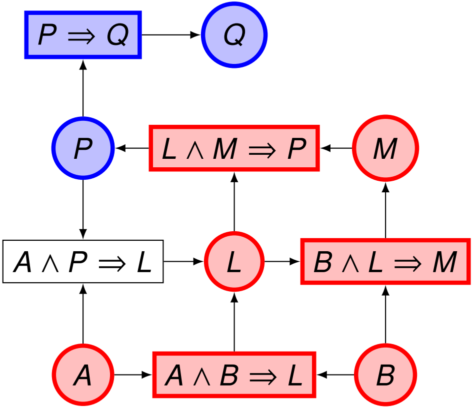
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

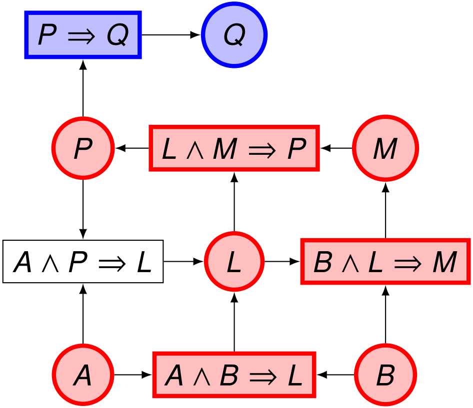
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

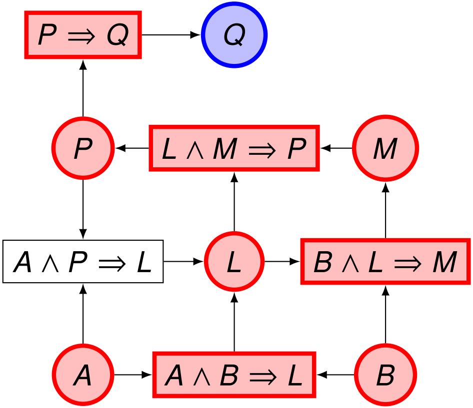
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

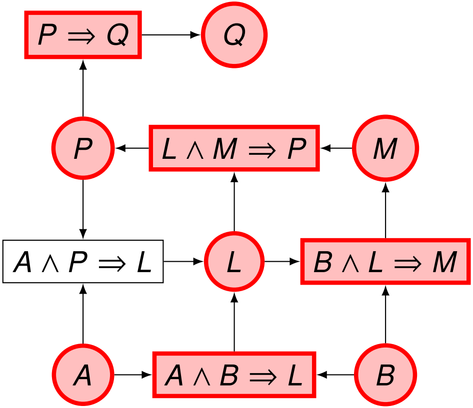
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound? Yes

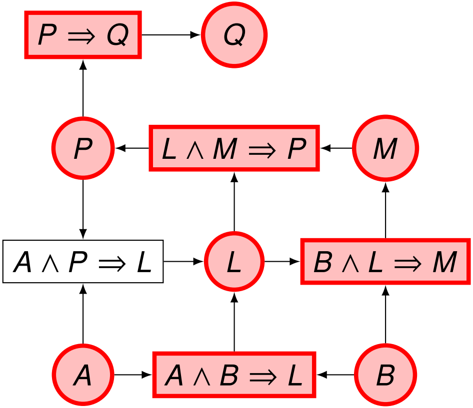
Complete? Yes

Time? $O(n)$ and often less

Backward Chaining

Key Idea: Recursion

- Start by trying to prove query Q
- If not *true*, try to prove $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ (recurse)



Properties

Sound?	Yes
Complete?	Yes
Time?	$O(n)$ and often less

Resolution (Again)

Example

$\neg IsSnowing \lor IsCold$

$\neg IsCold \lor WearCoat$

$\neg IsSnowing \lor WearCoat$

Intuition

- If *IsCold* is *true*, then *WearCoat* must be *true*
- If *IsCold* is *false*, then *IsSnowing* must be *false*
- So either *IsSnowing* is *false* or *WearCoat* is *true*

Resolution

Resolution Definition

$$\frac{P_1 \vee \ldots \vee P_n \qquad Q_1 \vee \ldots \vee Q_m \qquad P_i = \neg Q_j}{P_1 \vee \ldots \vee P_{i-1} \vee P_{i+1} \vee \ldots \vee P_n \vee Q_1 \vee \ldots \vee Q_{j-1} \vee Q_{j+1} \vee \ldots \vee Q_m}$$

Inference by Resolution

- Simplify all statements to use only $\wedge$, $\vee$ and $\neg$
- Assume $KB \wedge \neg\alpha$ (a.k.a $\neg(KB \Rightarrow \alpha)$)
- Apply resolution until *false* is concluded

Resolution Example

Prove: A

Proof:

Given:

R_1: $A \lor B$

R_2: $C \lor \neg B$

R_3: $A \lor \neg C$

Resolution Example

Prove: A

Given:
R_1: $A \vee B$
R_2: $C \vee \neg B$
R_3: $A \vee \neg C$

Proof:

R_4: $\neg A$ Assumed

Resolution Example

Prove: A

Given:

R_1:	$A \vee B$
R_2:	$C \vee \neg B$
R_3:	$A \vee \neg C$

Proof:

R_4:	$\neg A$	Assumed
R_5:	B	R_1, R_4

Resolution Example

Prove: A

Given:

R_1: $A \vee B$

R_2: $C \vee \neg B$

R_3: $A \vee \neg C$

Proof:

R_4:	$\neg A$	Assumed
R_5:	B	R_1, R_4
R_6:	$\neg C$	R_3, R_4

Resolution Example

Prove: A

Given:

R_1:	$A \vee B$
R_2:	$C \vee \neg B$
R_3:	$A \vee \neg C$

Proof:

R_4:	$\neg A$	Assumed
R_5:	B	R_1, R_4
R_6:	$\neg C$	R_3, R_4
R_7:	$\neg B$	R_2, R_6

Resolution Example

Prove: A

Given:

R_1:	$A \lor B$
R_2:	$C \lor \neg B$
R_3:	$A \lor \neg C$

Proof:

R_4:	$\neg A$	Assumed
R_5:	B	R_1, R_4
R_6:	$\neg C$	R_3, R_4
R_7:	$\neg B$	R_2, R_6
R_8:	*false*	R_5, R_7

Conjunctive Normal Form

Goal: Conjunction of Disjunctions of Literals

$(P_1 \vee \ldots \vee P_n) \wedge (Q_1 \vee \ldots \vee Q_m) \wedge \ldots$

Procedure

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
3. Move $\neg$ inward with De Morgan and Double Negation
4. Apply Distributivity of $\vee$ over $\wedge$

Conjunctive Normal Form Example

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply Distributivity of $\vee$ over $\wedge$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Conjunctive Normal Form Example

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply Distributivity of $\vee$ over $\wedge$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Conjunctive Normal Form Example

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$
   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$
4. Apply Distributivity of $\lor$ over $\land$
   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$

Conjunctive Normal Form Example

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply Distributivity of $\vee$ over $\wedge$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Conjunctive Normal Form Example

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply Distributivity of $\vee$ over $\wedge$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Conjunctive Normal Form Example

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply Distributivity of $\vee$ over $\wedge$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Conjunctive Normal Form Example

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply Distributivity of $\vee$ over $\wedge$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Conjunctive Normal Form Example

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$
2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$
   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$
3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$
4. Apply Distributivity of $\lor$ over $\land$
   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$

Conjunctive Normal Form Example

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

2. Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

3. Move $\neg$ inward with De Morgan and Double Negation
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Apply Distributivity of $\vee$ over $\wedge$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

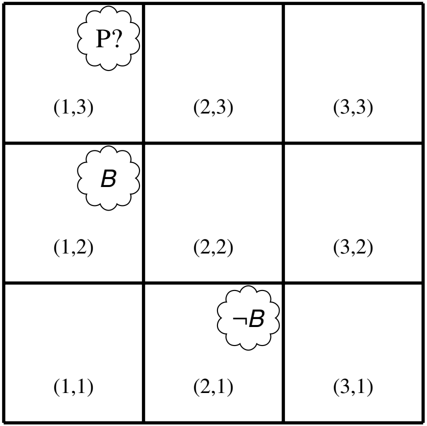
Resolution Exercise

Prove using Resolution:

$P_{1,3}$

Given:

$B_{1,2}$
$\neg B_{2,1}$
$B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$
$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$



Prove: $P_{1,3}$

After conversion to CNF:

R_1: $B_{1,2}$
R_2: $\neg B_{2,1}$
R_3: $\neg B_{1,2} \vee P_{1,1} \vee P_{2,2} \vee P_{1,3}$
R_4: $\neg P_{1,1} \vee B_{1,2}$
R_5: $\neg P_{2,2} \vee B_{1,2}$
R_6: $\neg P_{1,3} \vee B_{1,2}$
R_7: $\neg B_{2,1} \vee P_{1,1} \vee P_{2,2} \vee P_{3,1}$
R_8: $\neg P_{1,1} \vee B_{2,1}$
R_9: $\neg P_{2,2} \vee B_{2,1}$
R_{10}: $\neg P_{3,1} \vee B_{2,1}$

Using resolution:

R_{11}: $\neg P_{1,3}$	Assumed
R_{12}: $\neg P_{1,1}$	$R_2 + R_8$
R_{13}: $\neg P_{2,2}$	$R_2 + R_9$
R_{14}: $\neg B_{1,2} \vee P_{1,1} \vee P_{2,2}$	$R_{11} + R_3$
R_{15}: $P_{1,1} \vee P_{2,2}$	$R_{14} + R_1$
R_{16}: $P_{2,2}$	$R_{15} + R_{12}$
R_{17}: *false*	$R_{16} + R_{13}$

Resolution Properties

Properties

Sound?	Yes, uses Resolution
Complete?	Yes
Time?	Worst case exponential in # of symbols

Notes

- Works for Propositional logic in general
- Not just Horn Clauses!

Resolution Properties

Properties

Sound?	Yes, uses Resolution
Complete?	Yes
Time?	Worst case exponential in # of symbols

Notes

- Works for Propositional logic in general
- Not just Horn Clauses!

Resolution Properties

Properties

Sound?	Yes, uses Resolution
Complete?	Yes
Time?	Worst case exponential in # of symbols

Notes

- Works for Propositional logic in general
- Not just Horn Clauses!

Resolution Properties

Properties

Sound? Yes, uses Resolution

Complete? Yes

Time? Worst case exponential in # of symbols

Notes

- Works for Propositional logic in general
- Not just Horn Clauses!

Resolution Properties

Properties

Sound?	Yes, uses Resolution
Complete?	Yes
Time?	Worst case exponential in # of symbols

Notes

- Works for Propositional logic in general
- Not just Horn Clauses!

WalkSAT

Key Ideas

- Treat satisfiability checking as local search
- On each iteration flip a symbol's value
- Choice of symbol to flip is sometimes random, sometimes by MIN-CONFLICTS
- Quit after a certain number of steps

Properties

Sound?	Yes, only returns when KB is *true*
Complete?	No, could stop early
Time?	Can be quite fast

WalkSAT

Key Ideas

- Treat satisfiability checking as local search
- On each iteration flip a symbol's value
- Choice of symbol to flip is sometimes random, sometimes by MIN-CONFLICTS
- Quit after a certain number of steps

Properties

Sound? Yes, only returns when KB is *true*

Complete? No, could stop early

Time? Can be quite fast

WalkSAT

Key Ideas

- Treat satisfiability checking as local search
- On each iteration flip a symbol's value
- Choice of symbol to flip is sometimes random, sometimes by MIN-CONFLICTS
- Quit after a certain number of steps

Properties

Sound?	Yes, only returns when KB is *true*
Complete?	No, could stop early
Time?	Can be quite fast

WalkSAT

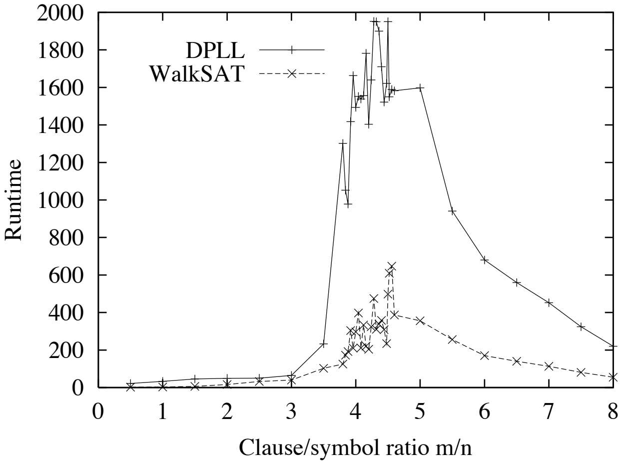
Key Ideas

- Treat satisfiability checking as local search
- On each iteration flip a symbol's value
- Choice of symbol to flip is sometimes random, sometimes by MIN-CONFLICTS
- Quit after a certain number of steps

Properties

Sound?	Yes, only returns when KB is *true*
Complete?	No, could stop early
Time?	Can be quite fast

WalkSAT Performance



DPLL
Truth table entailment + heuristics and pruning

Key Points

Propositional Logic

- Symbols: P, Q, R, ...
- Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$,
- Entailment: $\alpha \models \beta$ iff $\alpha \Rightarrow \beta$ in all worlds

Inference

- Enumerate all models through truth tables
- Forward/Backward chaining with Horn clauses
- Resolution using conjunctive normal form
- Local search, e.g. WalkSAT