Artificial Intelligence (CS303)

Lecture 10: Logical Agents

Hints for this lecture

- Human not only act based on instinct (gene? Program?), but also act based on knowledge.
- Represent, store, and exploit knowledge should also be important (or at least useful) for AI.

Outline of this lecture

- **Knowledge-based Agents**
- **Represent Knowledge with Logic**
- **(Propositional) Logic**
- **Inference with Propositional Logic**

I. Knowledge-based Agents

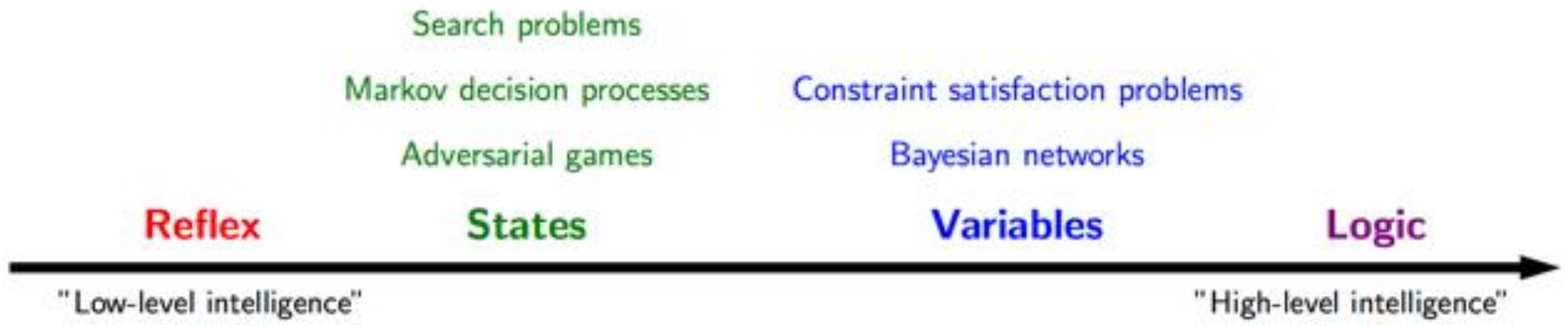
Knowledge-based agents

Logical AI:

The idea is that an agent can represent knowledge of its world, its goals and the current situation by sentences in logic and decide what to do by inferring that a certain action or course of action is appropriate to achieve its goals.

John McCarthy in Concepts of logical AI, 2000.

Knowledge-based agents



Knowledge-based agents

- Intelligent agents need **knowledge** about the world to choose good actions/decisions.
- Knowledge = **{sentences}** in a knowledge representation language (formal language).
- A sentence is an assertion about the world.
- A knowledge-based agent is composed of:
  1. Knowledge base: **domain-specific** content.
  2. Inference mechanism: **domain-independent** algorithms.

Knowledge-based agents

- The agent **must be able to**:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions
- **Declarative** approach to building an agent:
  - Add new sentences: **Tell** it what it needs to know
  - Query what is known: **Ask** itself what to do – answers should follow from the KB

Knowledge-based agents

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

  static: KB, a knowledge base

    t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(*percept*, t))

  $action \leftarrow$ ASK(KB, MAKE-ACTION-QUERY(t))

  TELL(KB, MAKE-ACTION-SENTENCE($action$, t))

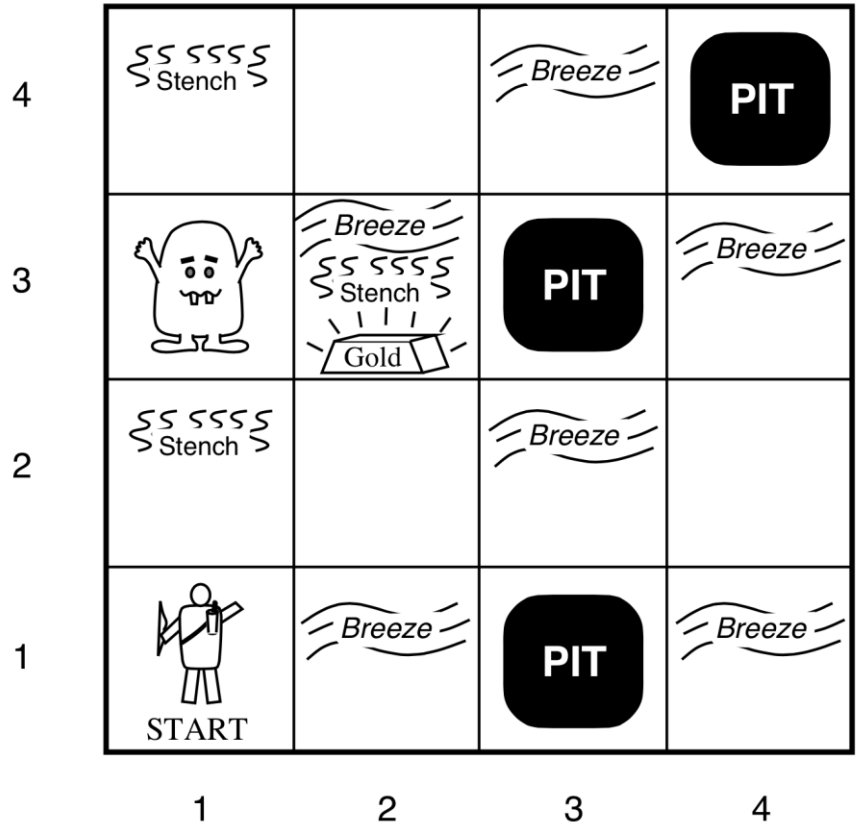
  $t \leftarrow t + 1$

  return *action*

The Wumpus World

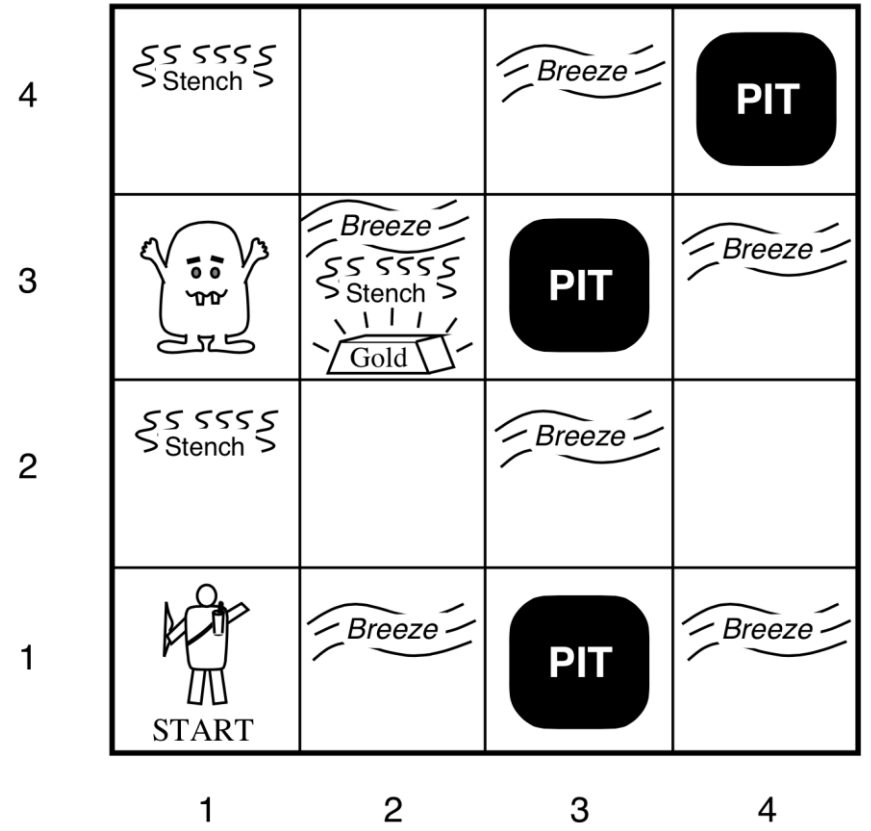
An example to see the power of knowledge-based agent

Gregory Yob (1975)



The Wumpus World

- 4 X 4 grid of rooms
- Squares adjacent to Wumpus are smelly and squares adjacent to pit are breezy
- Glitter iff gold is in the same square
- Shooting kills Wumpus if you are facing it
- Wumpus emits a horrible scream when it is killed that can be heard anywhere
- Shooting uses up the only arrow
- Grabbing picks up gold if in same square
- Releasing drops the gold in same square



Wumpus World PEAS

- **Performance measure:** gold +1000, death (eaten or falling in a pit) -1000, -1 per action taken, -10 for using the arrow. The games ends either when the agent dies or comes out of the cave.
- **Environment**
  - 4 X 4 grid of rooms
  - Agent starts in square [1, 1] facing to the right
  - Locations of the gold, and Wumpus are chosen randomly with a uniform distribution from all squares except [1,1]
  - Each square other than the start can be a pit with probability of 0.2

Wumpus World PEAS

- **Actuators:**
  - Left turn, Right turn, Forward, Grab, Release, Shoot
- **Sensors:**
  - Stench, Breeze, Glitter, Bump, Scream
  - Represented as a 5-element list
  - Example: [Stench, Breeze, None, None, None]

Wumpus World properties

- Partially observable
- Static
- Discrete
- Single-agent
- Deterministic
- Sequential

Exploring Wumpus World

Agent's first steps:

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

(a)

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(b)

Exploring Wumpus World

Agent's later steps:

1,4	2,4	3,4	4,4
1,3 **W!**	2,3	3,3	4,3
1,2 **A** **S** **OK**	2,2 **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

(a)

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4 **P?**	3,4	4,4
1,3 **W!**	2,3 **A** **S** **G** **B**	3,3 **P?**	4,3
1,2 **S** **V** **OK**	2,2 **V** **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

(b)

Knowledge-based Agent

- In the above situations, agent can obtain conclusion based on existing information/premise.

- If the information used/premise is right, then the conclusion is right.

- But how to contruct such an agent?
  - How to represent the known information
  - How to do the inference?
  - Whether the inference is sound or complete?

II. Represent Knowledge with Logic

Logic-Basic Concepts

- Logics are formal languages for representing knowledge to extract conclusions.
- **Knowledge base**: a set of sentences in a formal representation

- **Syntax**: defines well-formed sentences in the language
- **Semantic**: defines the truth or meaning of sentences in a world

- **Inference**: a procedure to derive a new sentence from other ones.

Sentences

Logics are formal languages for representing information
such that conclusions can be drawn

Syntax defines the sentences in the language

Semantics define the "meaning" of sentences;
i.e., define truth of a sentence in a world

E.g., the language of arithmetic

$x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

Logic is a **formal** language for representing knowledge

Relationship Between Sentences

Entailment means that one thing **follows from** another:

$$KB \models \alpha$$

Knowledge base KB entails sentence α
if and only if
α is true in all worlds where KB is true

E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"

“Worlds”

Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

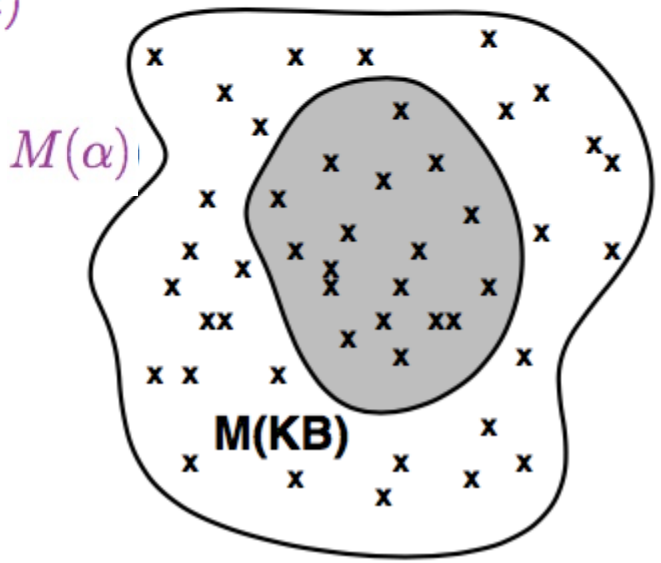
We say m is a model of a sentence α if α is true in m

$M(\alpha)$ is the set of all models of α

Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

E.g. $KB =$ Giants won and Reds won

$\alpha =$ Giants won



Inference

Inference: the **procedure** of deriving a sentence from another sentence

Model Checking: A basic (and general) idea to inference

$KB \vdash_i \alpha =$ sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.
Entailment $=$ needle in haystack; inference $=$ finding it

Soundness: i is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

III. (Propositional) Logic

Propositional logic

- Propositional logic (PL) is the simplest logic.
- **Syntax of PL:** defines the allowable sentences or propositions.
- **Definition (Proposition):** A proposition is a declarative statement that's either True or False.
- **Atomic proposition:** single proposition symbol. Each symbol is a proposition. Notation: upper case letters and may contain subscripts.
- **Compound proposition:** constructed from atomic propositions using parentheses and **logical connectives**.

Atomic proposition

Examples of atomic propositions:

- 2+2=4 is a true proposition
- $W_{1,3}$ is a proposition. It is true if there is a Wumpus in [1,3]
- "How are you?" or "Hello!" are not propositions. In general, statement that are questions, commands, or opinions are not propositions.

Compound proposition

Examples of compound/complex propositions:

Let p, p_1, and p_2 be propositions

- **Negation** $\neg p$ is also a proposition. We call a **literal** either an atomic proposition or its negation. E.g., $W_{1,3}$ is a positive literal, and $\neg W_{1,3}$ is a negative literal.
- **Conjunction** $p_1 \wedge p_2$. E.g., $W_{1,3} \wedge P_{3,1}$
- **Disjunction** $p_1 \vee p_2$. E.g., $W_{1,3} \vee P_{3,1}$
- **Implication** $p_1 \rightarrow p_2$. E.g., $W_{1,3} \wedge P_{3,1} \rightarrow \neg W_{2,2}$
- **If and only if** $p_1 \leftrightarrow p_2$. E.g., $W_{1,3} \leftrightarrow \neg W_{2,2}$

Semantic of Propositional Logic

- The semantics define the rules to determine the truth of a sentence.
- Semantics can be specified by truth tables.
- Boolean values domain: T, F
- n-tuple: $(x_1, x_2, \ldots, x_n)$
- Operator on n-tuples: $g(x_1 = v_1, x_2 = v_2, \ldots, x_n = v_n)$
- Definition: A truth table defines the truth value of an operator g on n-tuples by specifying a Boolean value for each tuple
- Number of rows in a truth table? $R = 2^n$

Truth Table

Negation:

p	$\neg p$
T	F
F	T

Truth Table

Conjunction:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Truth Table

Disjunction:

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Truth Table

Exclusive or:

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Truth Table

Implication:

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Truth Table

Biconditional or If and only if (IFF):

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Precedence of operators

- Just like arithmetic operators, there is an operator precedence when evaluating logical operators as follows:
  1. Expressions in parentheses are processed (inside to outside)
  2. Negation
  3. AND
  4. OR
  5. Implication
  6. Biconditional
  7. Left to right
- **Use parentheses whenever you have any doubt!**

Truth Tables

p	q	r	¬r	p ∨ q	p ∨ q → ¬r
T	T	T	F	T	F
T	T	F	T	T	T
T	F	T	F	T	F
T	F	F	T	T	T
F	T	T	F	T	F
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	T	F	T

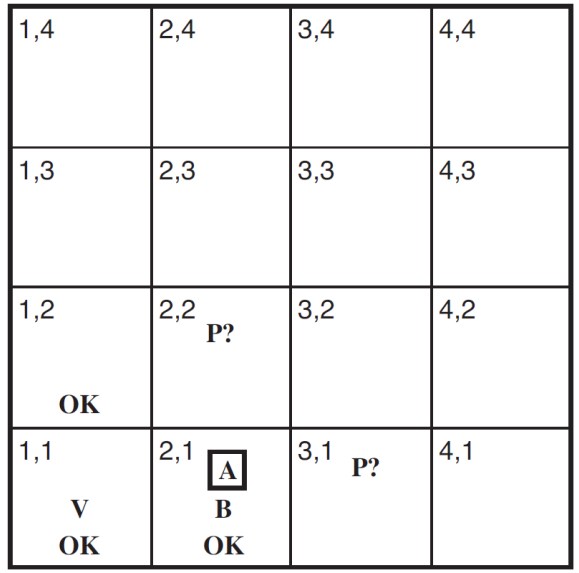
Truth Tables for connectives

Summary:

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

IV. Inference with Propositional Logic

Wumpus world KB



- Let's build the KB for the reduced Wumpus world.
- Let $P_{i,j}$ be true if there is a pit in $[i, j]$
- Let $B_{i,j}$ be true if there is a breeze in $[i, j]$

  R_1: $\neg P_{1,1}$
- "A square is breezy if and only if there is an adjacent pit"

  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: $B_{2,1}$
- Questions: Based on this KB, is $KB \models P_{1,2}$? Is $KB \models P_{2,2}$?

Model Checking by Enumeration

- Truth Table for inference
- Model: assignment T/F to every propositional symbol.
- Check that is true in every model in which KB is true.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*

Model Checking by Enumeration

```
function TT-ENTAILS?(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic

  symbols ← a list of the proposition symbols in KB and α
  return TT-CHECK-ALL(KB, α, symbols, [])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
  if EMPTY?(symbols) then
     if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
     else return true
  else do
     P ← FIRST(symbols); rest ← REST(symbols)
     return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model) and
                         TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model)
```

图 7.10 用于判定命题蕴涵的真值表枚举算法。TT 代表真值表。如果语句包含在模型中，PL-TRUE?返回真。变量 *model* 表示一个不完全模型——只是对某些变量的一个赋值。函数调用 EXTEND(*P*, *true*, *model*)，返回一个新的不完全模型，在该模型中 *P* 的值为真

Inference by Enumeration

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
fals												
fals												
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*

$O(2^n)$ for n symbols; problem is **co-NP-complete**

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

Entailment and Inference

- **Semantics**: Determine entailment by **Model Checking**, that is enumerate all models and show that the sentence must hold in all models.

$$KB \models \alpha$$

- **Syntax**: Determine entailment by **Theorem Proving**, that is apply rules of inference to KB to build a proof of α *without enumerating and checking all models*.

$$KB \vdash \alpha$$

- But how are entailment and inference related?

Soundness & Completeness

- We want an inference algorithm that is:
  1. **Sound**: does not infer false formulas, that is, derives only entailed sentences.
  $$\{\alpha | KB \vdash \alpha\} \subseteq \{\alpha | KB \models \alpha\}$$
  2. **Complete**: derives ALL entailed sentences.
  $$\{\alpha | KB \vdash \alpha\} \supseteq \{\alpha | KB \models \alpha\}$$

Logical equivalence

- Two propositions p and q are logically equivalent if and only if the columns in the truth table giving their truth values agree.
- We write this as $p \equiv q$.

p	q	$\neg p$	$\neg p \vee q$	$p \rightarrow q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of operators

- Commutativity:
  $p \wedge q \equiv q \wedge p$
  $p \vee q \equiv q \vee p$
- Associativity:
  $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
  $(p \vee q) \vee r \equiv q \vee (p \vee r)$
- Identity element:
  $p \wedge \mathit{True} \equiv p$
  $p \vee \mathit{True} \equiv \mathit{True}$
- $\neg(\neg p) \equiv p$
- $p \wedge p \equiv p$ and $p \vee p \equiv p$
- Distributivity:
  $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
  $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
- $p \wedge (\neg p) \equiv \mathit{False}$ and $p \vee (\neg p) \equiv \mathit{True}$
- DeMorgan's laws:
  $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$
  $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$

Tautology and contradiction

- **Tautology** is a proposition which is always true
- **Contradiction** is a proposition which is always false
- **Contingency** is a proposition which is neither a tautology or a contradiction

P	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

Contrapositive, inverse, etc.

- Given an **implication** $p \to q$
- The **converse** is: $q \to p$
- The **contrapositive** is: $\neg q \to \neg p$
- The **inverse** is: $\neg p \to \neg q$

Example: Hot is a sufficient condition for me to go to the beach.

- The **implication** is:
- The **converse** is:
- The **contrapositive** is:
- The **inverse** is:

Inference Rule: Modus Ponens

$$\frac{p \quad p \to q}{q}$$

$$\frac{warm \quad warm \to sunny}{sunny}$$

Modus ponens is sound but incomplete

a → b , b → c
can not get a→ c

A	B	$A \Rightarrow B$
False	*False*	*True*
False	*True*	*True*
True	*False*	*False*
True	*True*	*True*

Inference (Modus Ponens)

Horn clauses: a proposition of the form:

$$p_1 \wedge \ldots \wedge p_n \rightarrow q$$

Modus Ponens deals with **Horn clauses**:

$$\frac{p_1, \ldots, p_n \qquad (p_1 \wedge \cdots \wedge p_n) \rightarrow q}{q}$$

Inference (Modus Tollens)

$$\frac{\neg q \qquad p \rightarrow q}{\neg p}$$

$$\frac{\neg beach \qquad hot \rightarrow beach}{\neg hot}$$

Common Rules

- Addition: $\dfrac{p}{p \vee q}$
- Simplification: $\dfrac{p \wedge q}{q}$
- Disjunctive-syllogism: $\dfrac{p \vee q \quad \neg p}{q}$
- Hypothetical-syllogism: $\dfrac{p \rightarrow q \quad q \rightarrow r}{p \rightarrow r}$

Inference: Wumpus world

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

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

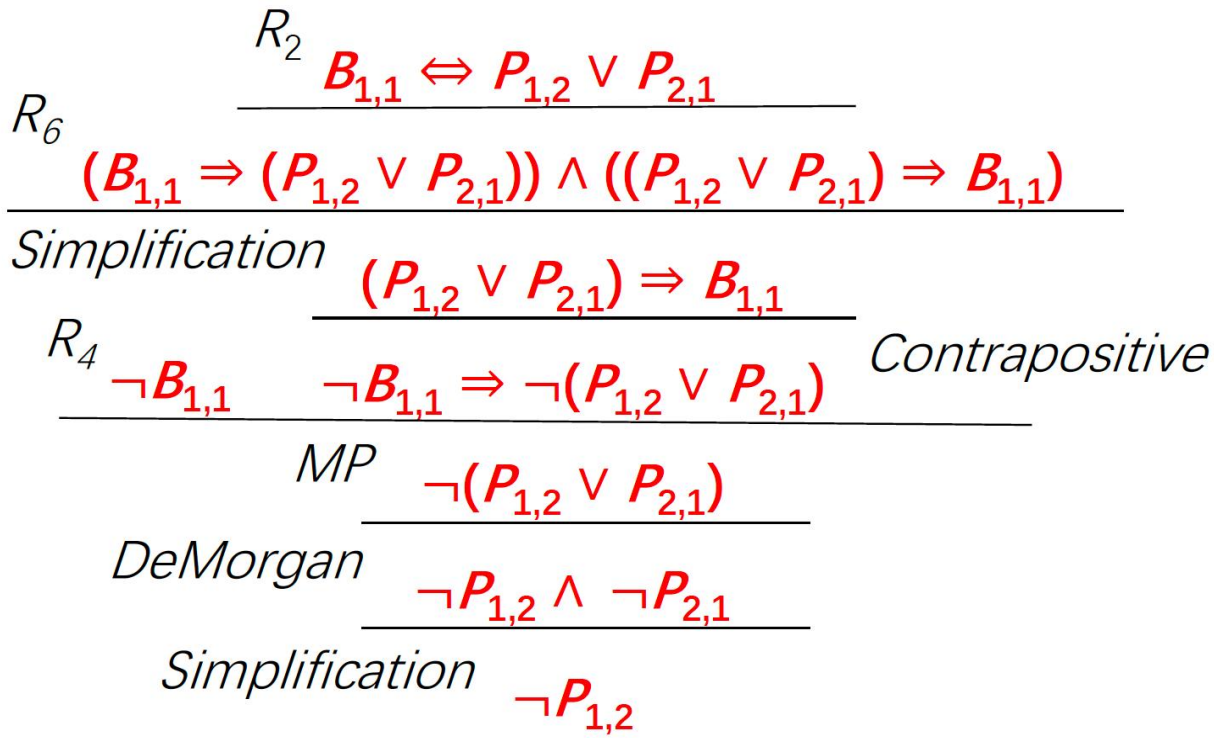
R_3: $B_{2,1} \Leftrightarrow P_{1,1} \vee P_{2,2} \vee P_{3,1}$

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

R_5: $B_{2,1}$

$KB \models P_{1,2}$?

Inference using Rules



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

Inference as a search problem

- **Initial state:** The initial KB
- **Actions:** all inference rules applied to all sentences that match the top of the inference rule
- **Results:** add the sentence in the bottom half of the inference rule
- **Goal:** a state containing the sentence we are trying to prove.

Completeness Issue: if the inference rules to use are not sufficient, the goal can not be obtained.

Theorem proving

- Search for proofs is a more efficient way than enumerating models (We can ignore irrelevant information)
- Truth tables have an exponential number of models.
- The idea of inference is to repeat applying *inference rules* to the KB.
- Inference can be applied whenever suitable premises are found in the KB.
- Inference is sound. How about completeness?

Theorem proving

- Two ways to ensure completeness:
  - **Proof by resolution**: use powerful inference rules (resolution rule)
  - **Forward or Backward chaining**: use of modus ponens on a restricted form of propositions (Horn clauses)
- Resolution: ONE single inference rule
- Invented by Robinson, 1965
- Resolution + Search = complete inference algorithm.

Proof by Resolution

- **Unit resolution:**

$$\frac{\ell_1 \vee \cdots \vee \ell_k \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

where ℓ_i and m are complementary literals.

- Example:

$$\frac{P_{1,3} \vee P_{2,2} \qquad \neg P_{2,2}}{P_{1,3}}$$

- We call a **clause** a disjunction of literals.
- Unit resolution: Clause + Literal = New clause.

Proof by Resolution

- **Resolution inference rule (for CNF):**

$$\frac{\ell_1 \vee \cdots \vee \ell_k \quad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

  where ℓ_i and m_j are complementary literals.

- Resolution **applies only to clauses**
- **Fact**: Every sentence in PL is logically equivalent to a conjunction of clauses.
- Conjunctive Normal Form (CNF): Conjunction of disjunction of literals: example: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- Resolution inference rule (for CNF): sound and complete for propositional logic

Conjunctive Normal Form (CNF)

- To make inference more efficient/effective, we need more
- Any sentence of propositional logic is equivalent to a conjunction of clauses.

$$
\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots
\end{aligned}
$$

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ 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. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ 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 $\neg$ inwards using de Morgan's rules 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 law ($\vee$ over $\wedge$) and flatten:

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

Satisfiability Problem

- Propositional model checking could be done by solving a satisfiability problem.
  - Put the KB and ASK into CNF, and solve the SAT problem.

A sentence is valid if it is true in **all** models,
e.g., $True$, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:
$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in **some** model
e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in **no** models
e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:
$KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

SAT: (Boolean) Satisfiability Problem

Resolution algorithm

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  loop do
    for each C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

- there are no new clauses that can be added, in which case KB does not entail α; or,
- two clauses resolve to yield the *empty* clause, in which case KB entails α.

Proof by Resolution

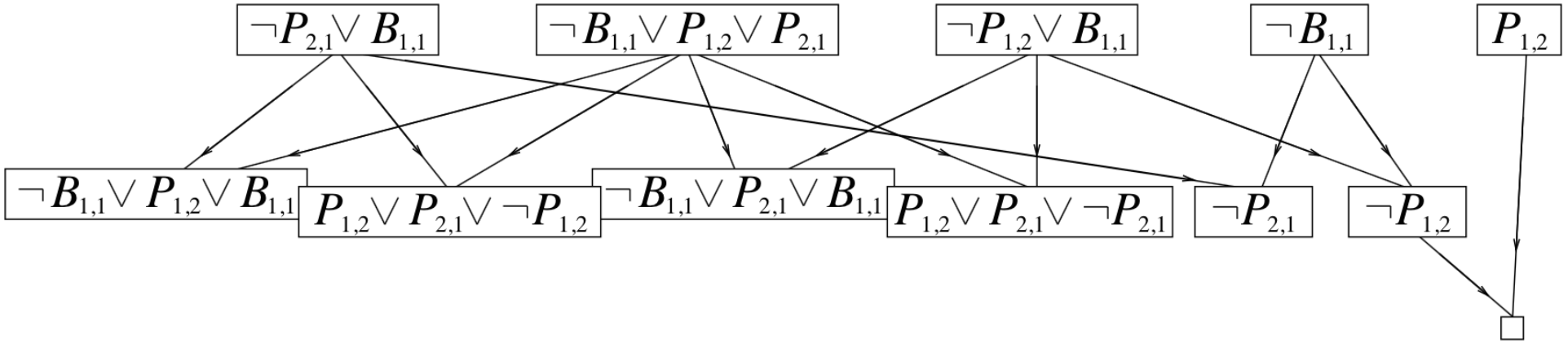
- Resolution & Wumpus world:

1,4	2,4	3,4	4,4
1,3 **W!**	2,3	3,3	4,3
1,2 **A** **S** **OK**	2,2 **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

Resolution example

$$KB = R_4 \wedge R_2 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

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



Forward/backward chaining

- KB = conjunction of Horn clauses
- Horn clauses: logic proposition of the form: $p_1 \wedge \ldots \wedge p_n \rightarrow q$
- Modus Ponens (for Horn Form): complete for Horn KBs with atomic proposition to entail

$$\frac{p_1, \ldots, p_n \qquad (p_1 \wedge \cdots \wedge p_n) \rightarrow q}{q}$$

- Can be used with **forward chaining** or **backward chaining**.
- These algorithms are very natural and run in linear time

Forward chaining

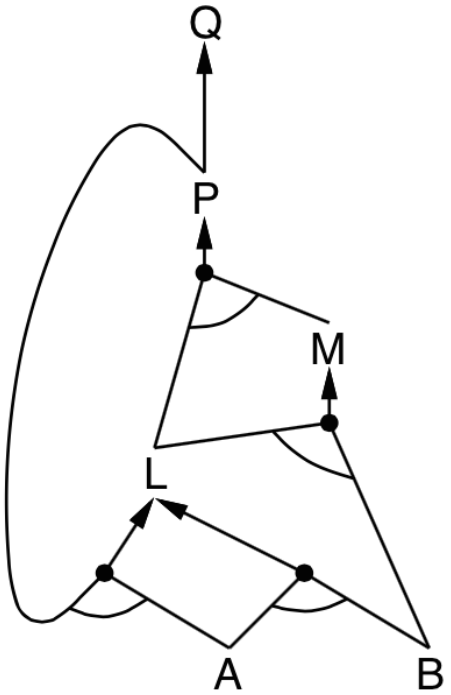
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional Horn clauses
          q, the query, a proposition symbol
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known in KB

  while agenda is not empty do
      p ← POP(agenda)
      unless inferred[p] do
          inferred[p] ← true
          for each Horn clause c in whose premise p appears do
              decrement count[c]
              if count[c] = 0 then do
                  if HEAD[c] = q then return true
                  PUSH(HEAD[c], agenda)
  return false
```

Forward chaining

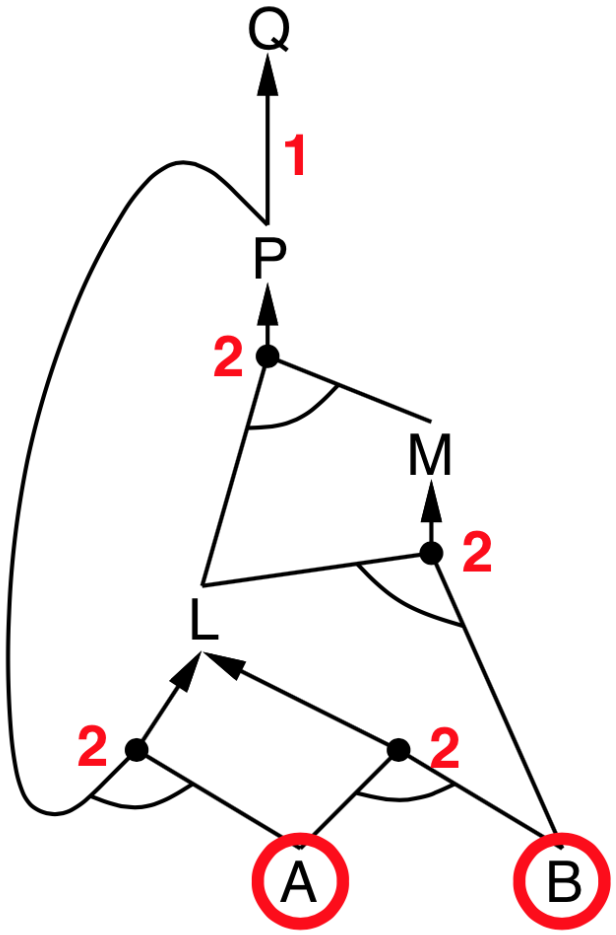
Idea:

Fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found

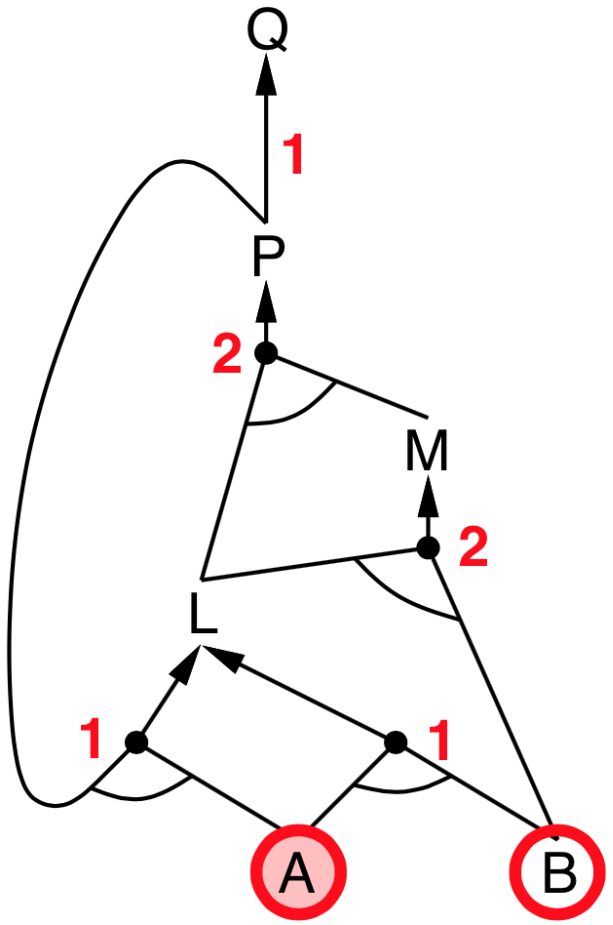
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



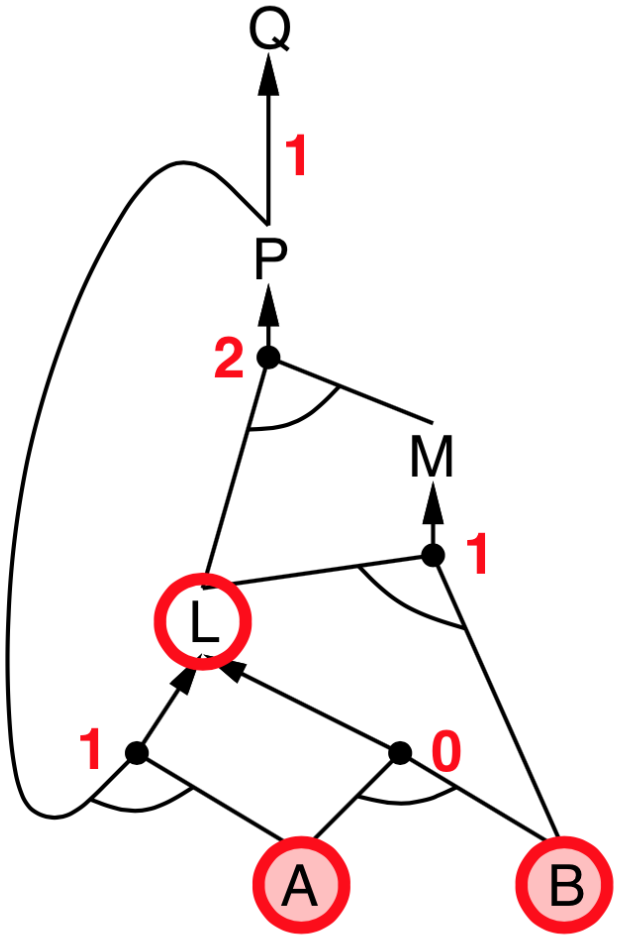
Forward chaining example



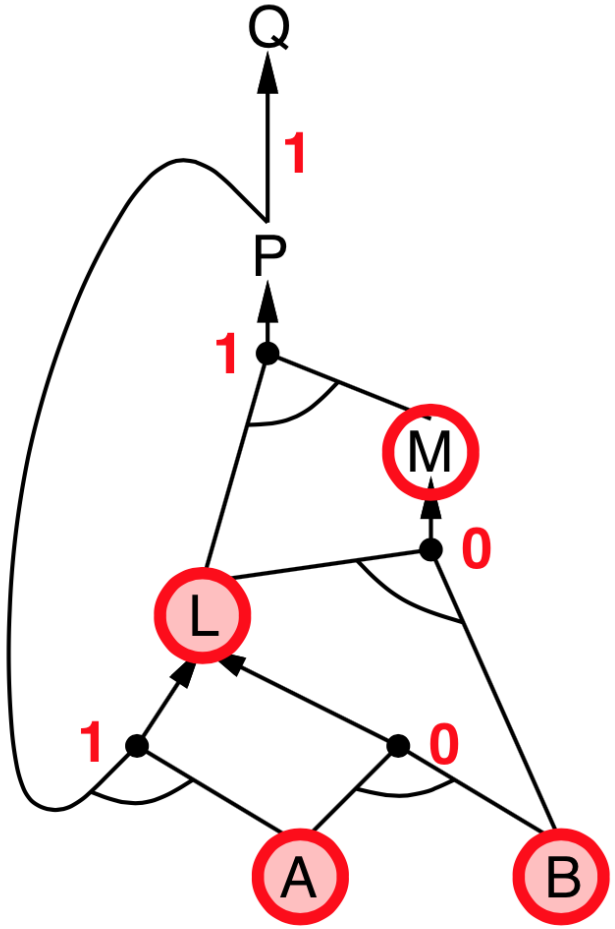
Forward chaining example



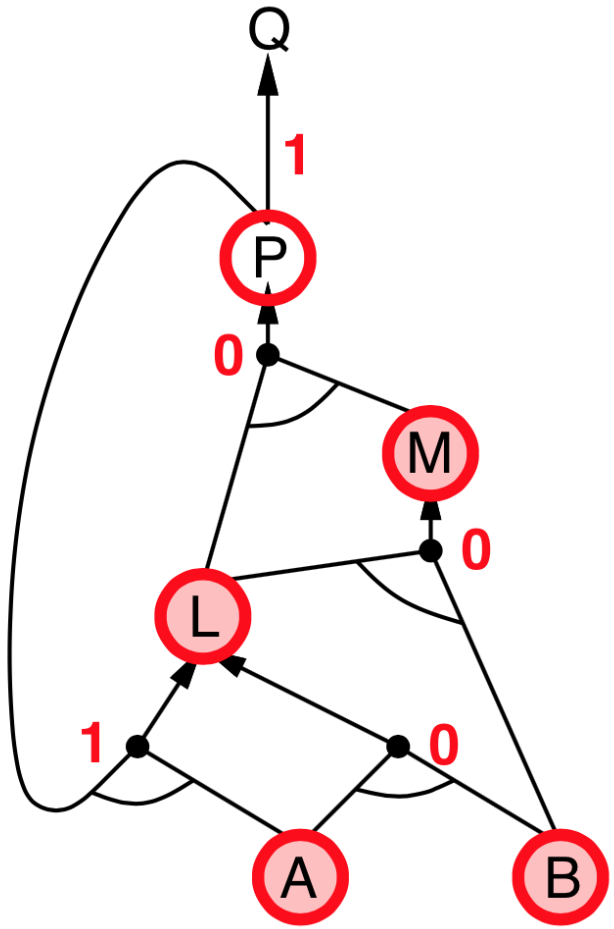
Forward chaining example



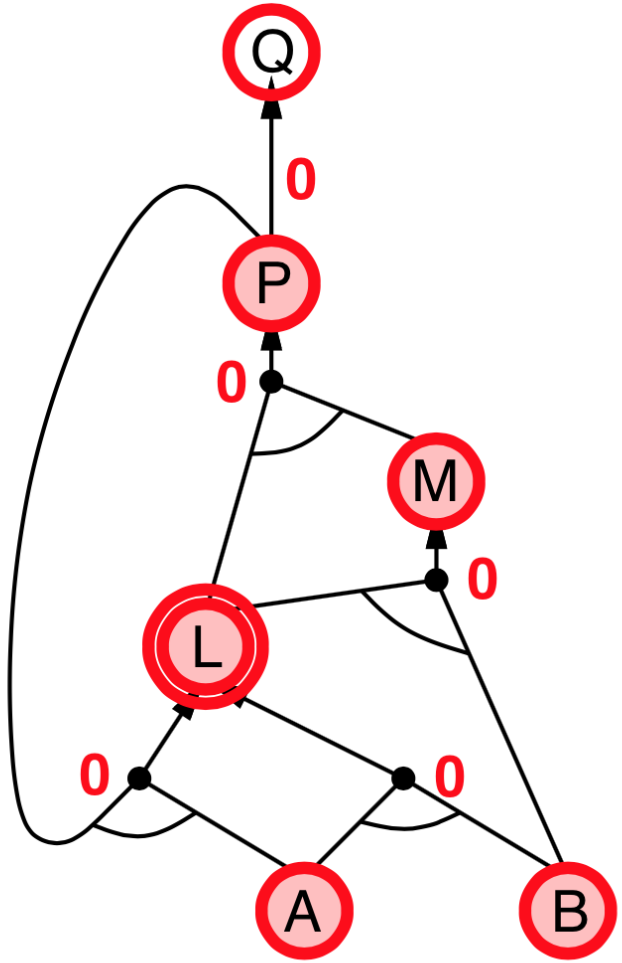
Forward chaining example



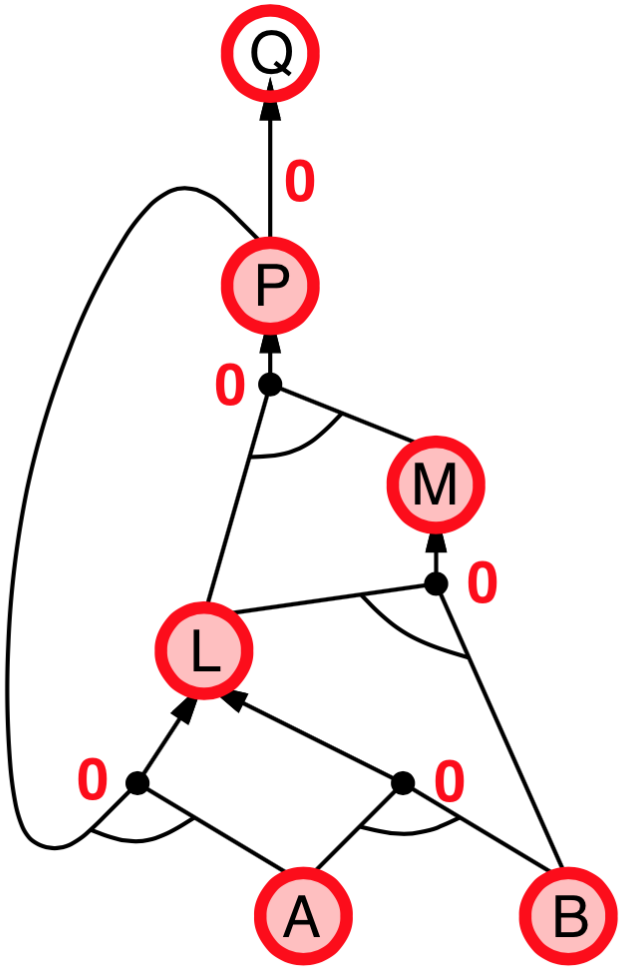
Forward chaining example



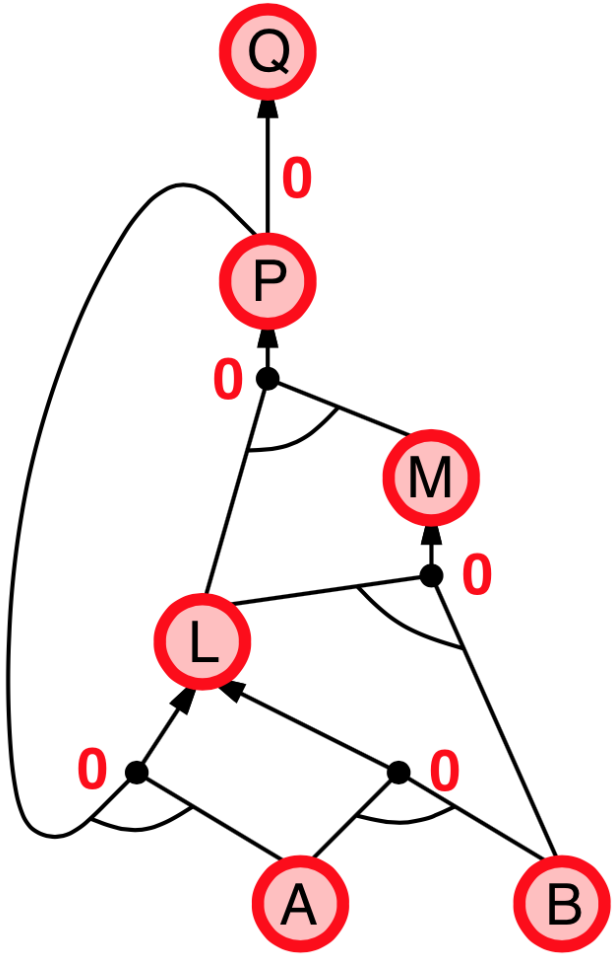
Forward chaining example



Forward chaining example



Forward chaining example

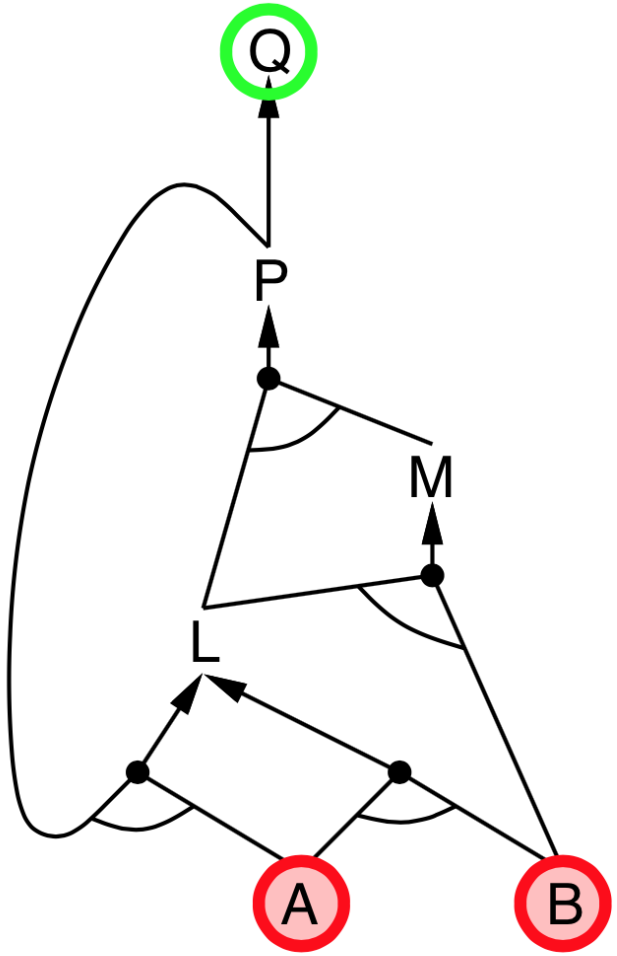


Backward chaining

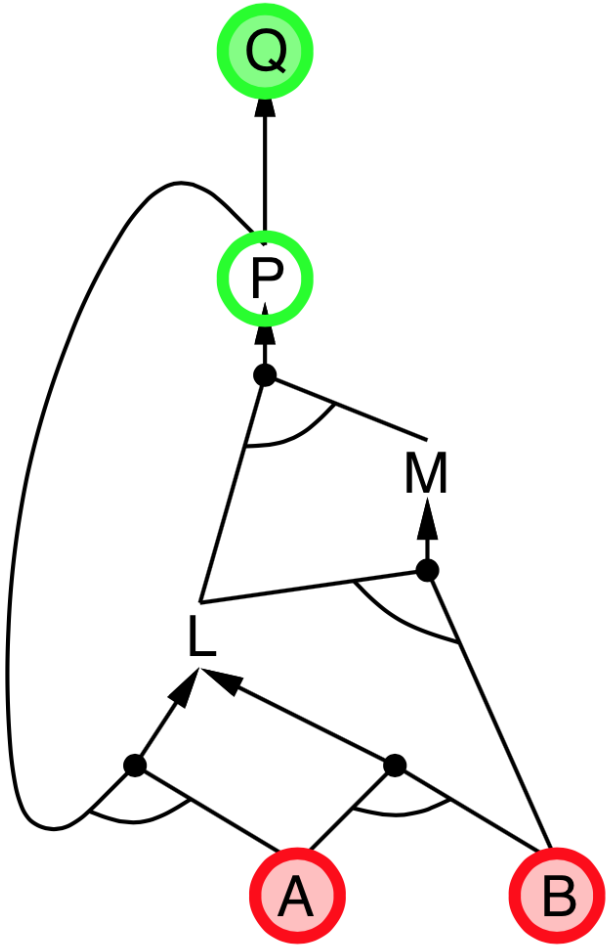
Idea: Works backwards from the query *q*

- to prove *q* by Backward Chaining:
  - Check if *q* is known already, or
  - Prove by Backward Chaining all premises of some rule concluding *q*
- Avoid loops: check if new subgoal is already on the goal stack
- Avoid repeated work: check if new subgoal
  - has already been proved true, or
  - has already failed

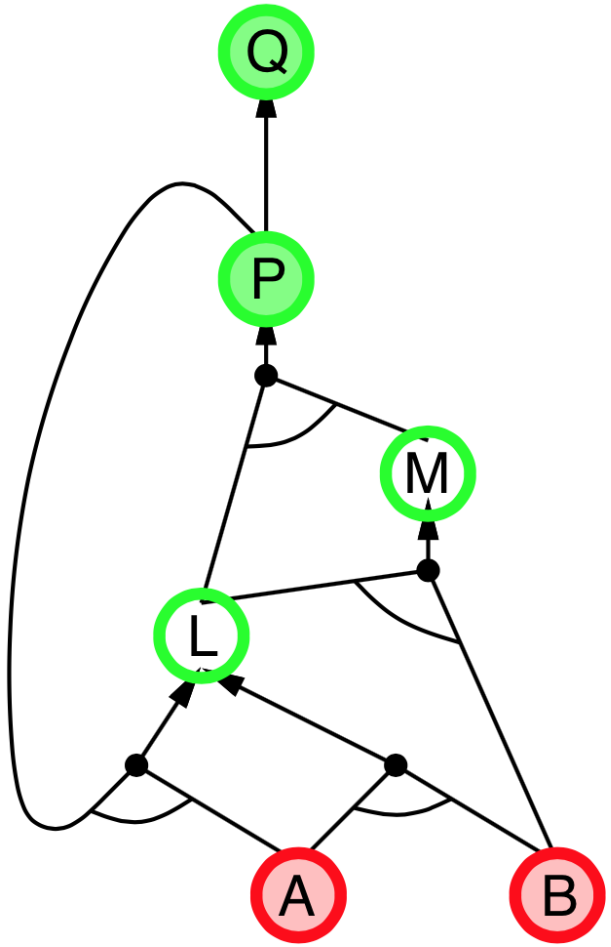
Backward chaining example



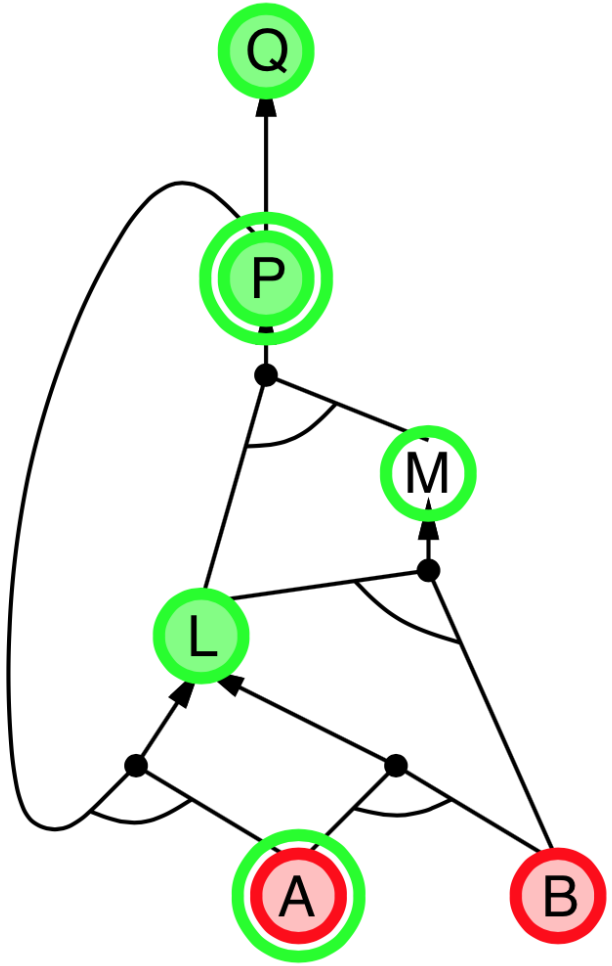
Backward chaining example



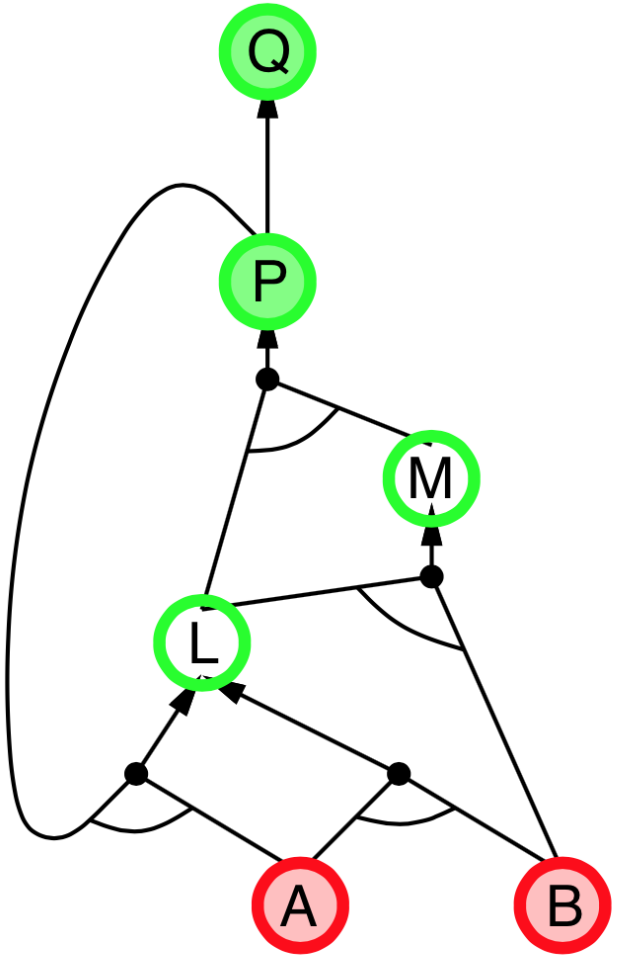
Backward chaining example



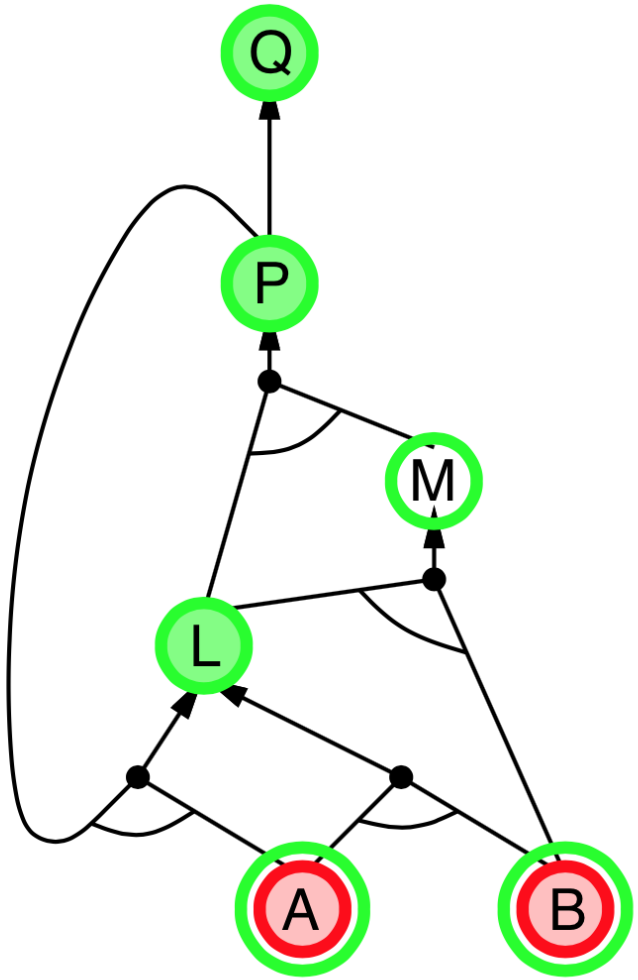
Backward chaining example



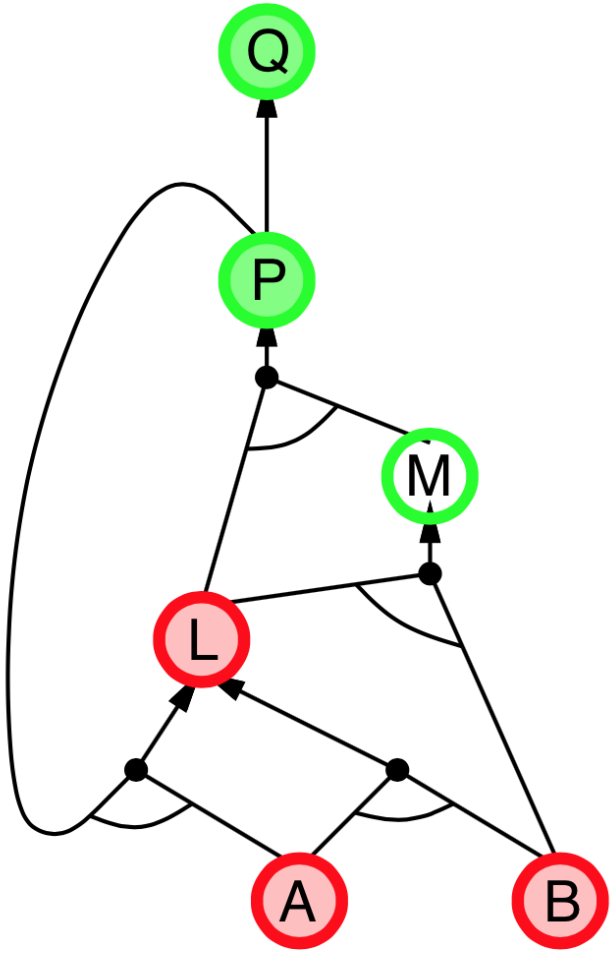
Backward chaining example



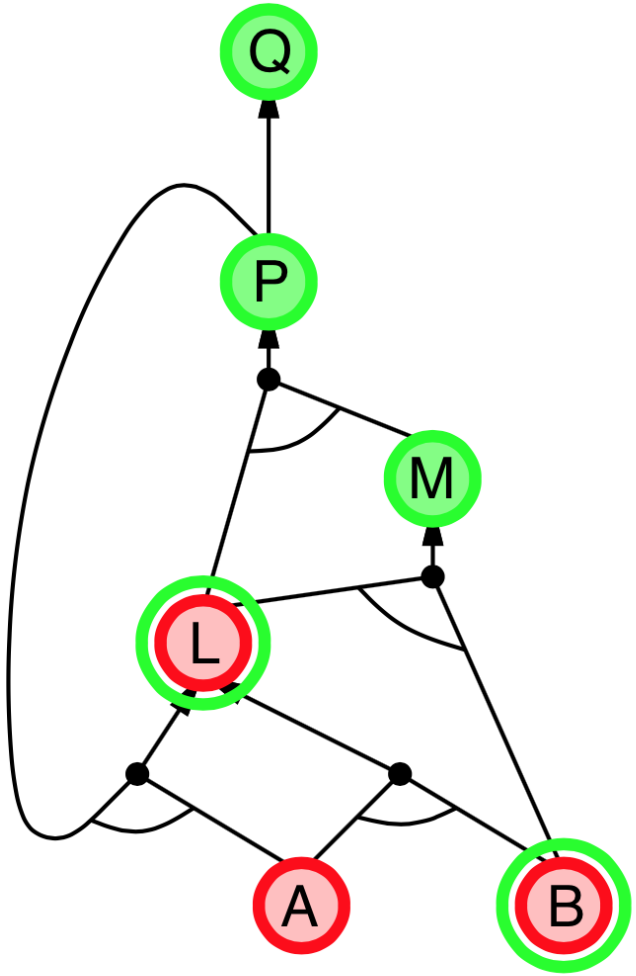
Backward chaining example



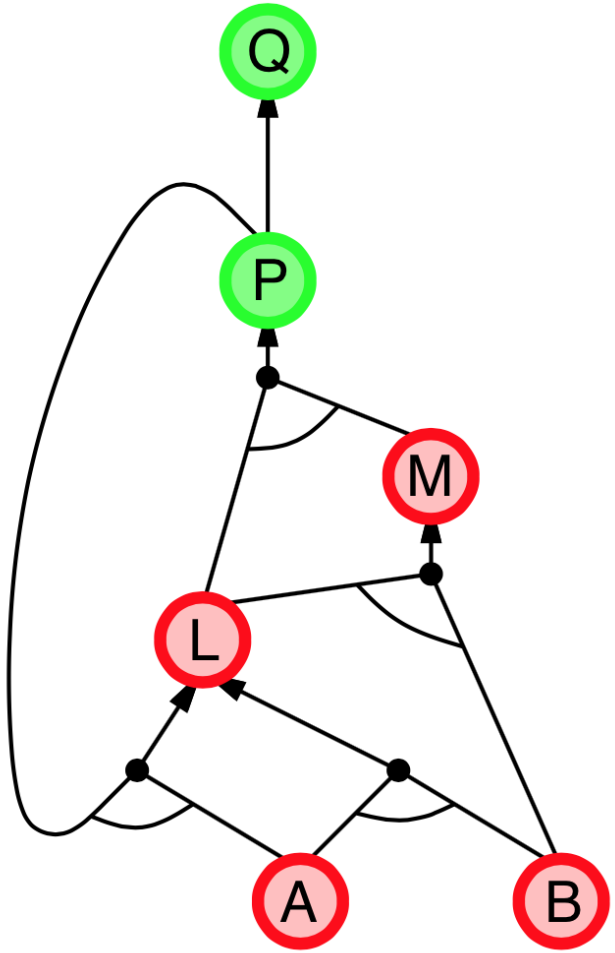
Backward chaining example



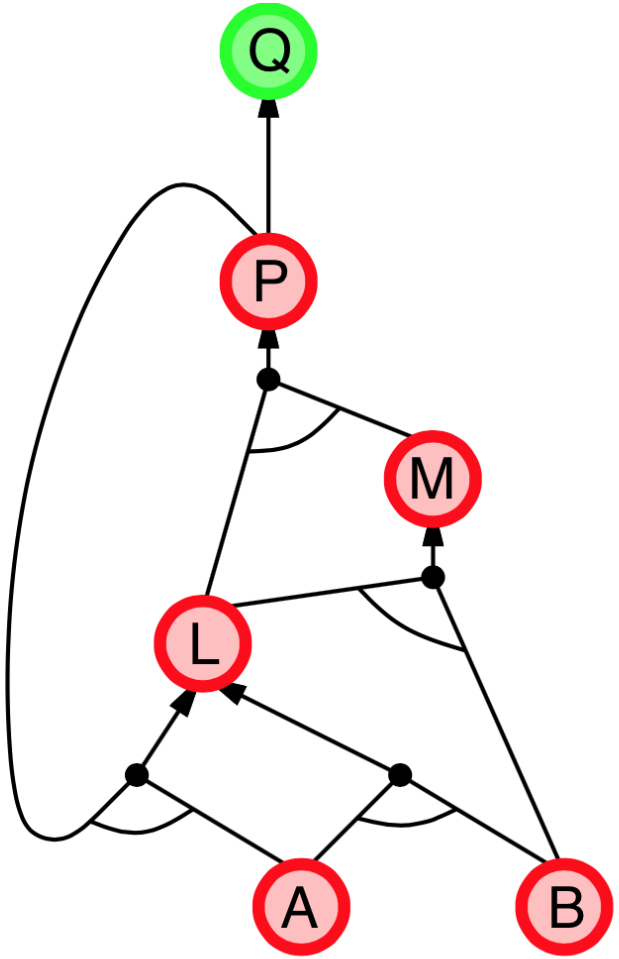
Backward chaining example



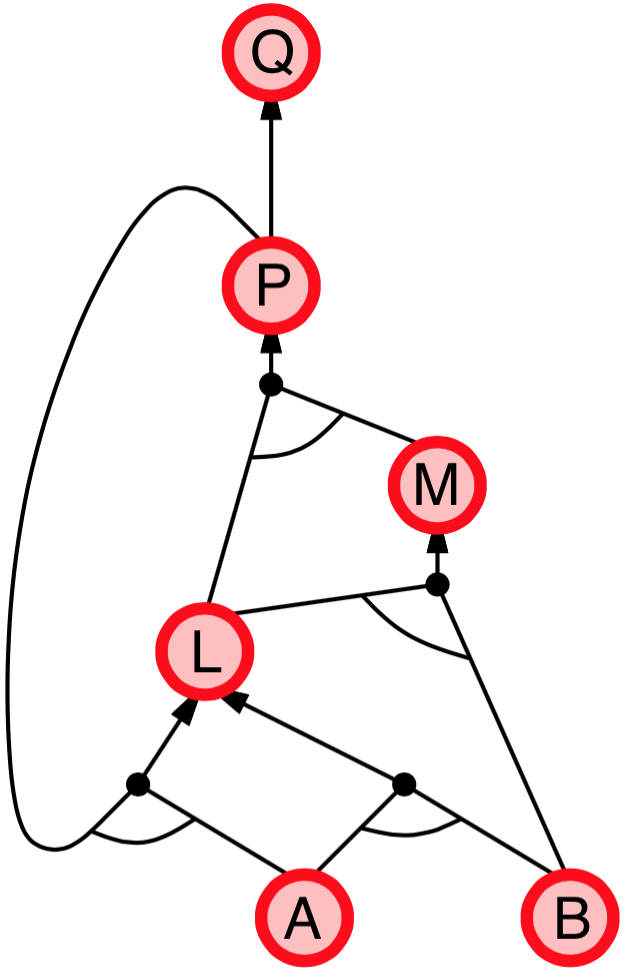
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. Backward

- Forward chaining:
  - Data-driven, automatic, unconscious processing,
  - May do lots of work that is irrelevant to the goal
- Backward chaining:
  - Goal-driven, appropriate for problem-solving,
  - Complexity of BC can be much less than linear in size of KB

Backtracking for SAT: Effective Model Checking

- Satisfiability is connected to inference via the following: $KB \models \alpha$ IFF $(KB \wedge \neg\alpha)$ is **unsatisfiable**, i.e., prove α by contradiction
- The DPLL algorithm for checking **satisfiability** (SAT) of a sentence in propositional logic.
- The DPLL algorithm is similar to Backtracking for CSP, but using various problem dependent information/heuristics, such as **Early Termination**, **Pure symbol heuristic** and **Unit clause heuristic**.

DPLL

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, { })

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {P=*true*}) **or**
        DPLL(*clauses*, *rest*, *model* ∪ {P=*false*}))

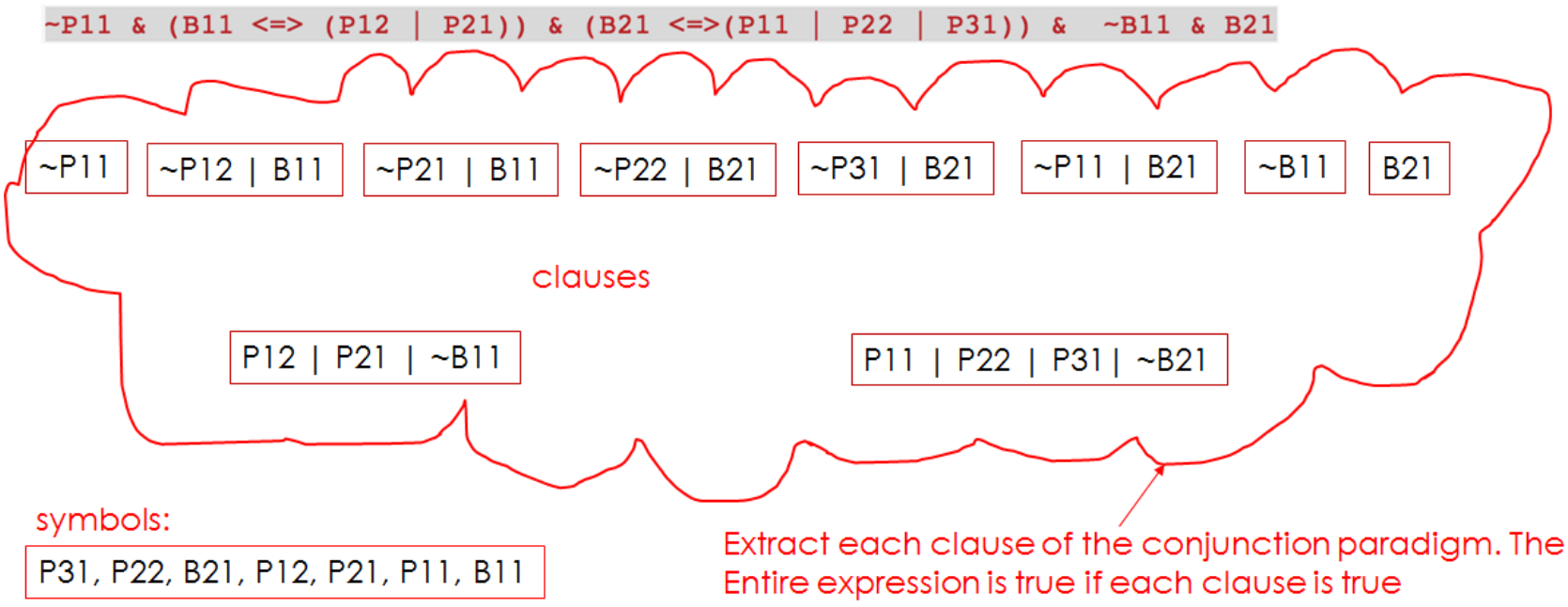
DPLL

- If there is a clause returning false, then return false as a whole.
- If all clauses are determined to be true in the model, return true as a whole.
- Look for pure symbols first: For example, if there is only A in all clauses, you can directly assign A to true; if there is only $\neg B$ in all clauses, you can directly assign B to false. We can ignore all clauses determined to be true with the built model.
- Find the unit clause in priority: clause with only one literal
  - a clause can also be considered as a unit clause if all other literals are assigned a value except one literal
  - E.g, A is a unit clause, and $\neg A$ is also a unit clause
  - A has been assigned a value of false, $A|\neg B$ is also a unit clause since B must be false here. When B is false, $A|B|C$ is also a unit clause, because C must be true
  - The process is a bit like constraint propagation, called unit propagation
- If none of the above symbols were found, look for the next symbol.
- Try all possible assignments of the symbol and recursively call the next layer.

DPLL Example

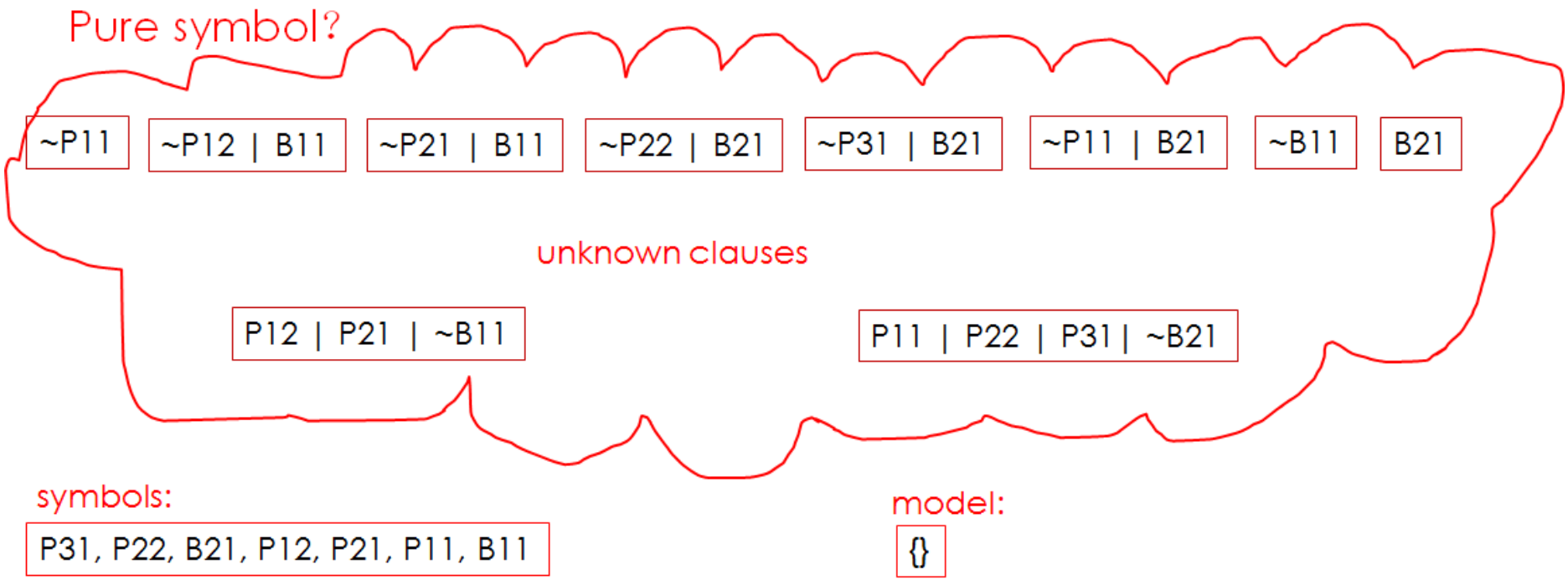
Transform into CNF and extract all conjunction sub-clauses

$$\neg P11 \;\&\; \big(B11 \Leftrightarrow (P12|P21)\big) \;\&\; \big(B21 \Leftrightarrow (P11|P22|P31)\big) \;\&\; \neg B11 \;\&\; B21$$



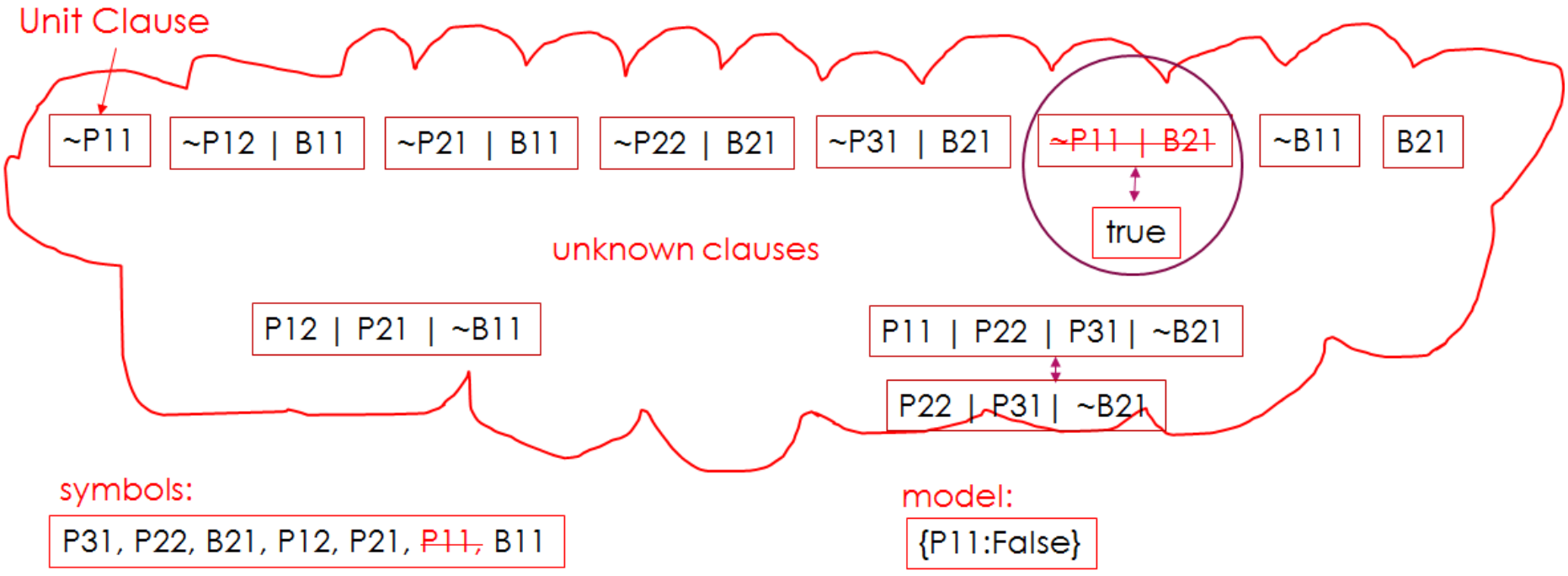
DPLL Example

Put priority in selecting pure literal and unit clauses



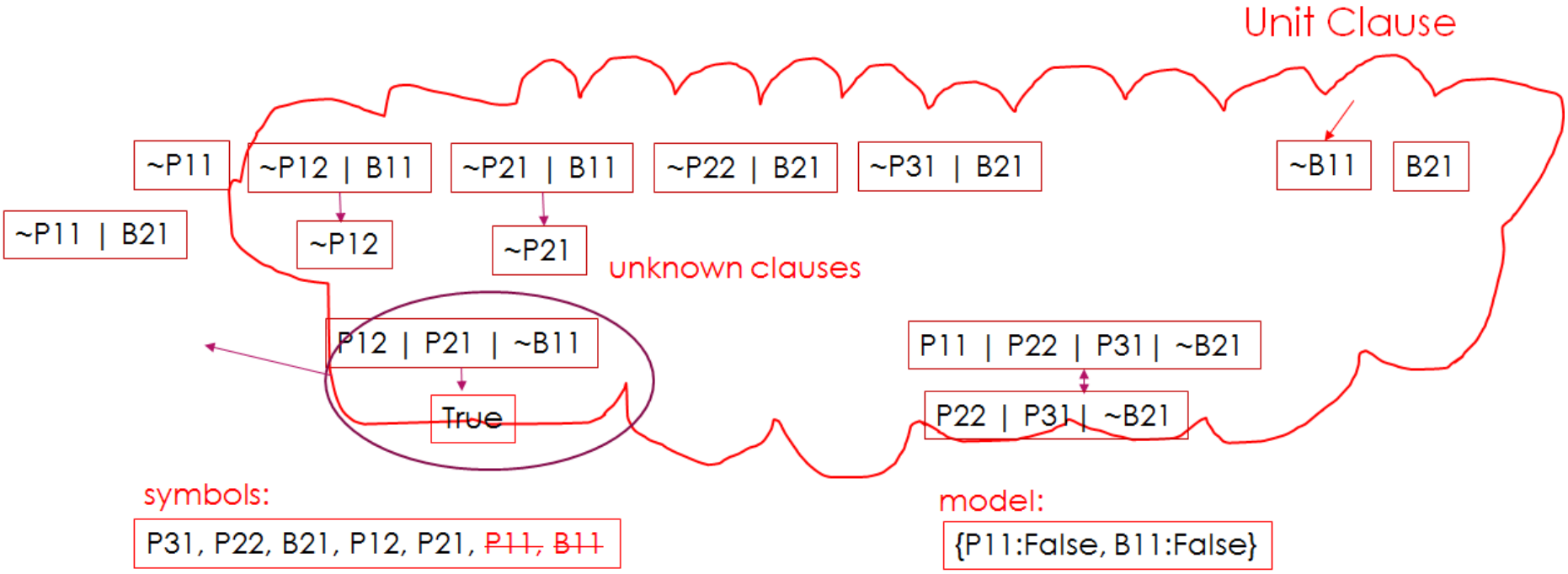
DPLL Example

Put priority in selecting pure literal and unit clauses



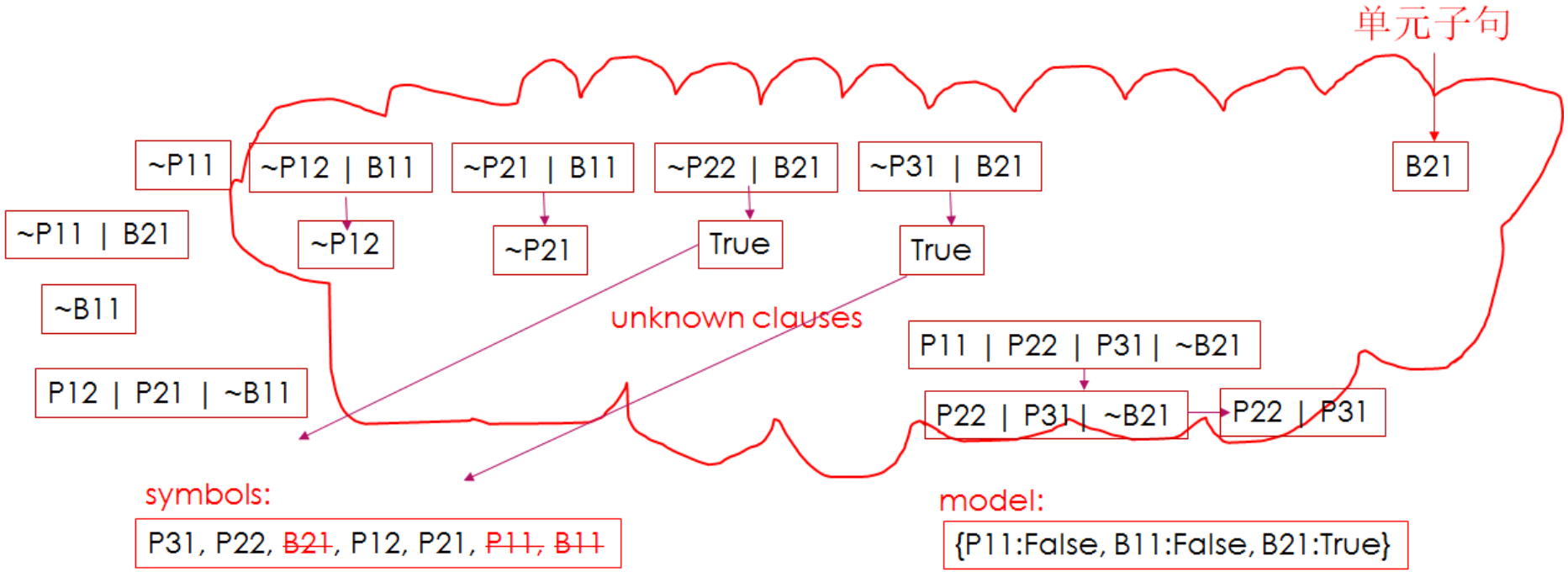
DPLL Example

Put priority in selecting pure literal and unit clauses



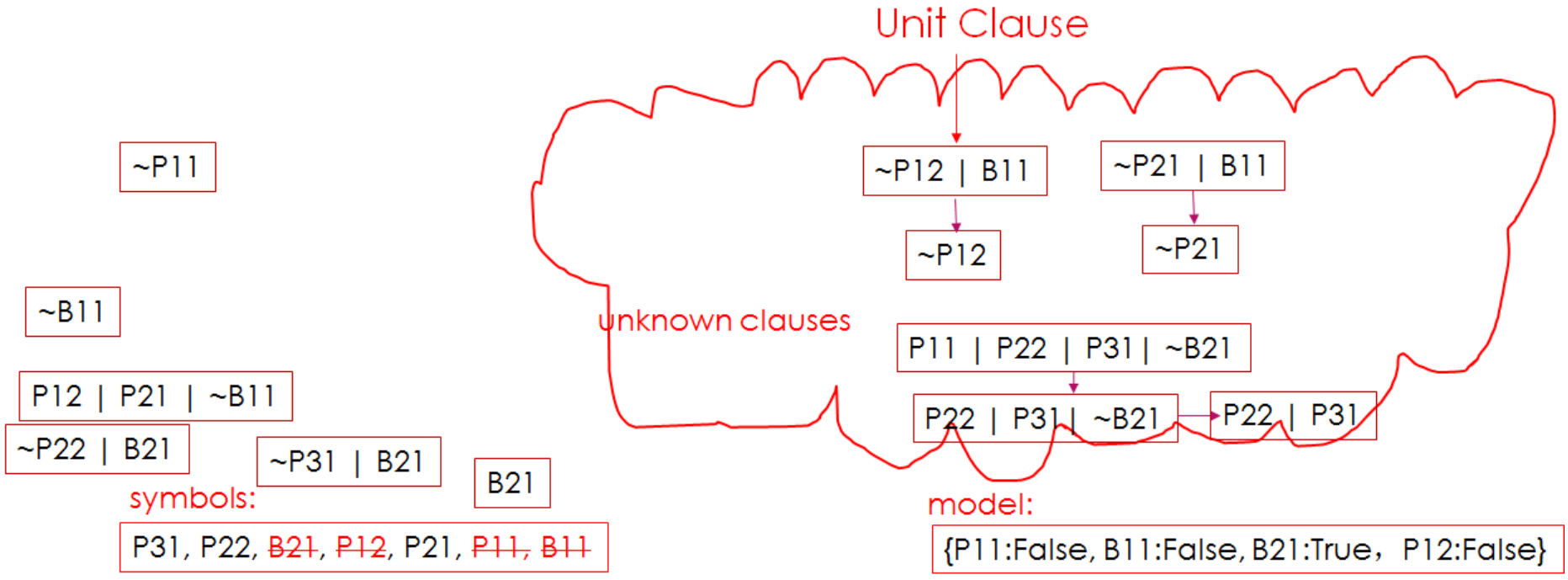
DPLL Example

Put priority in selecting pure literal and unit clauses



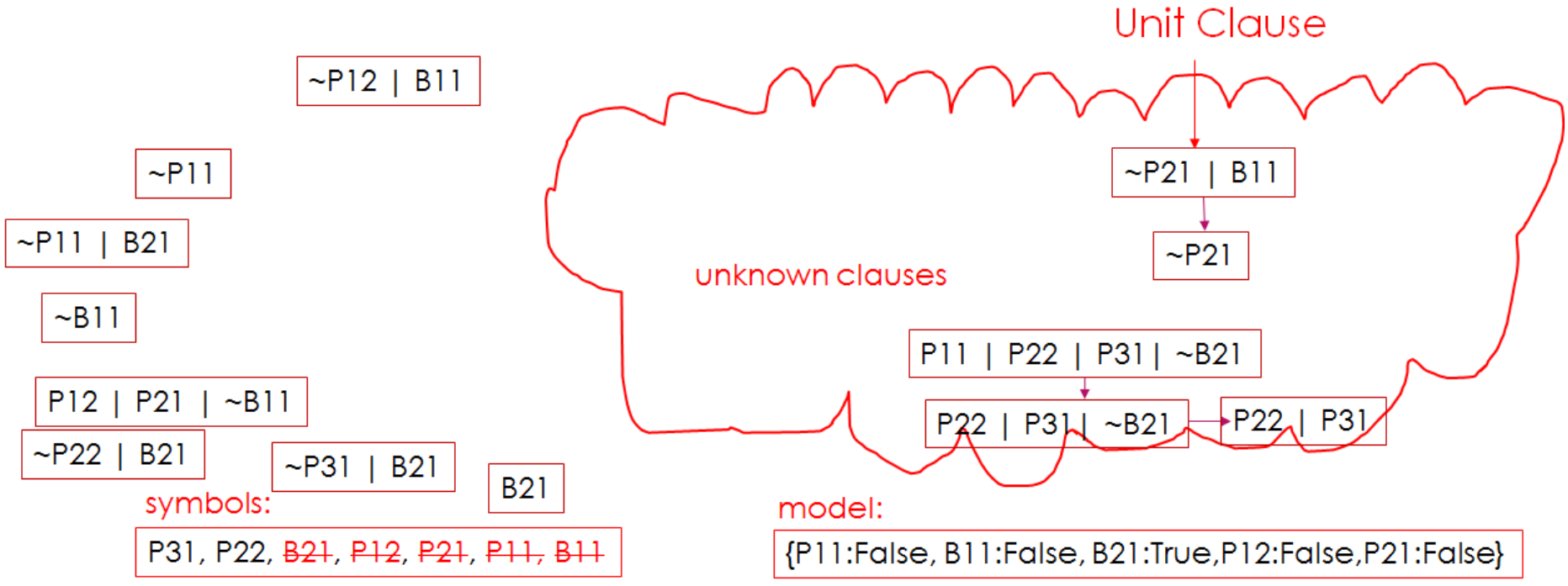
DPLL Example

Put priority in selecting pure literal and unit clauses



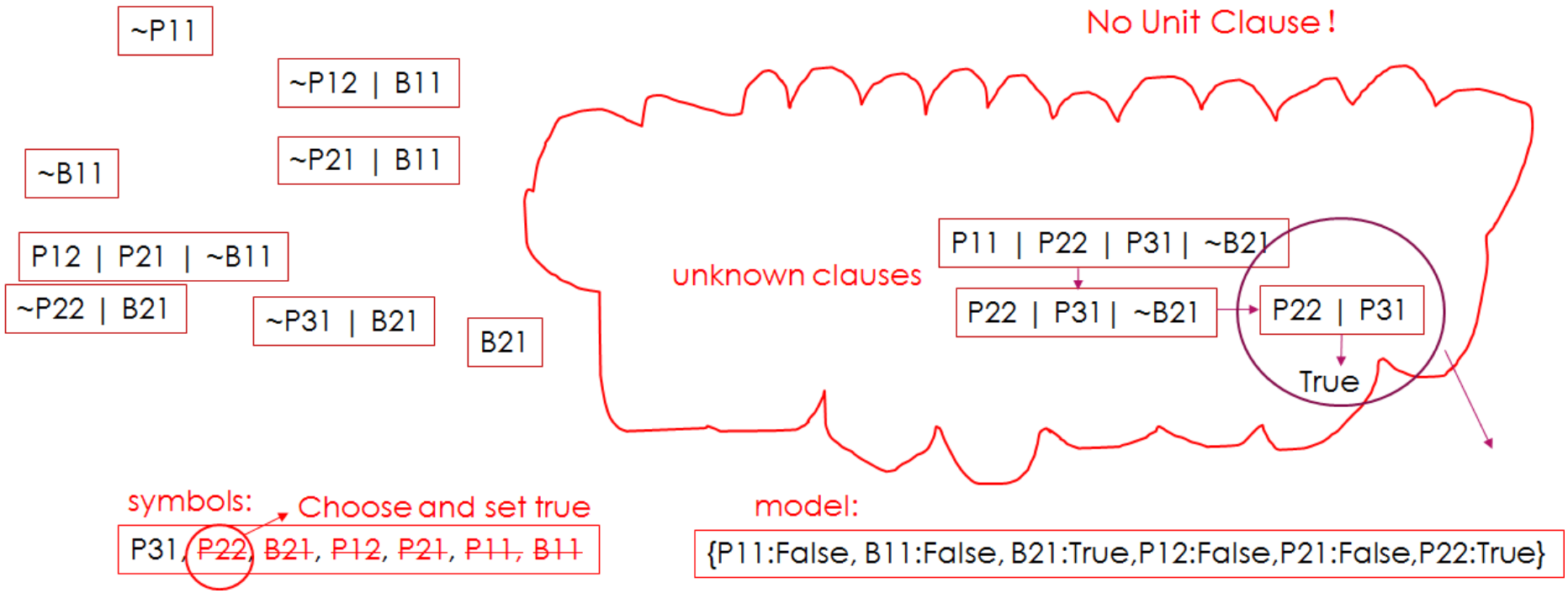
DPLL Example

Put priority in selecting pure literal and unit clauses



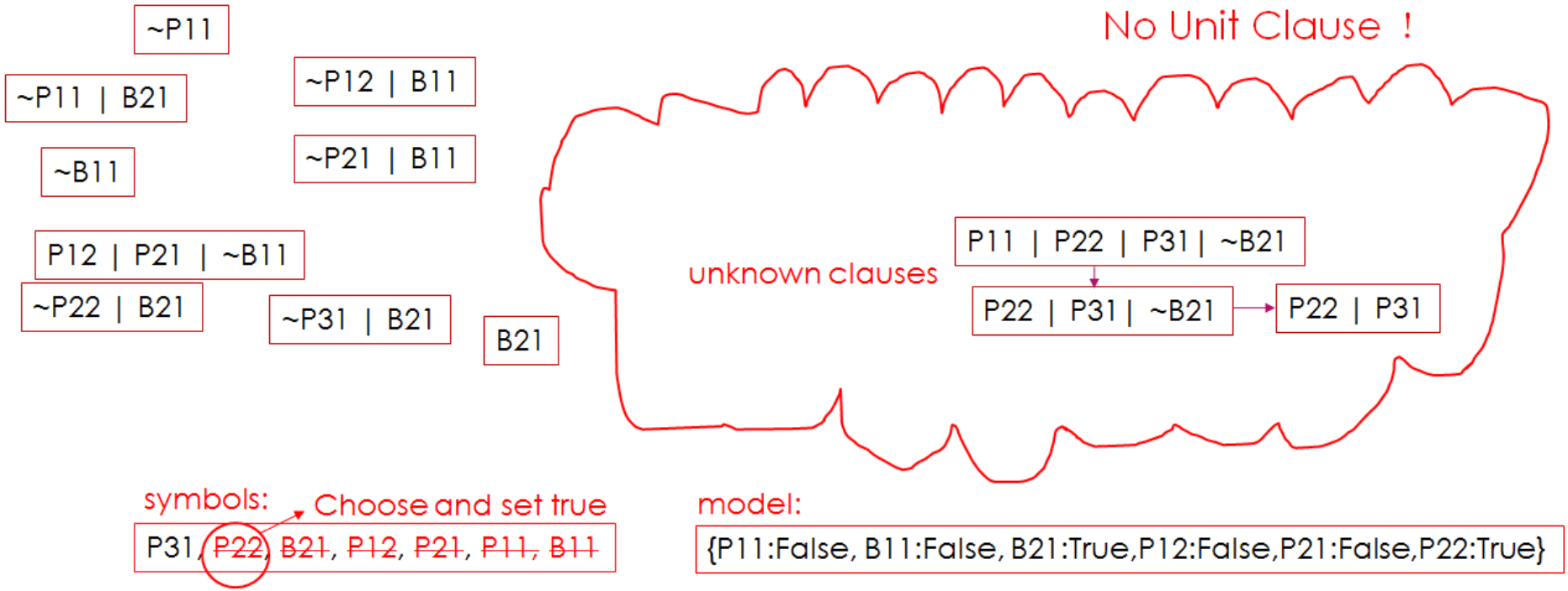
DPLL Example

No pure literals and no unit clauses. Select a literal and assign it to be true



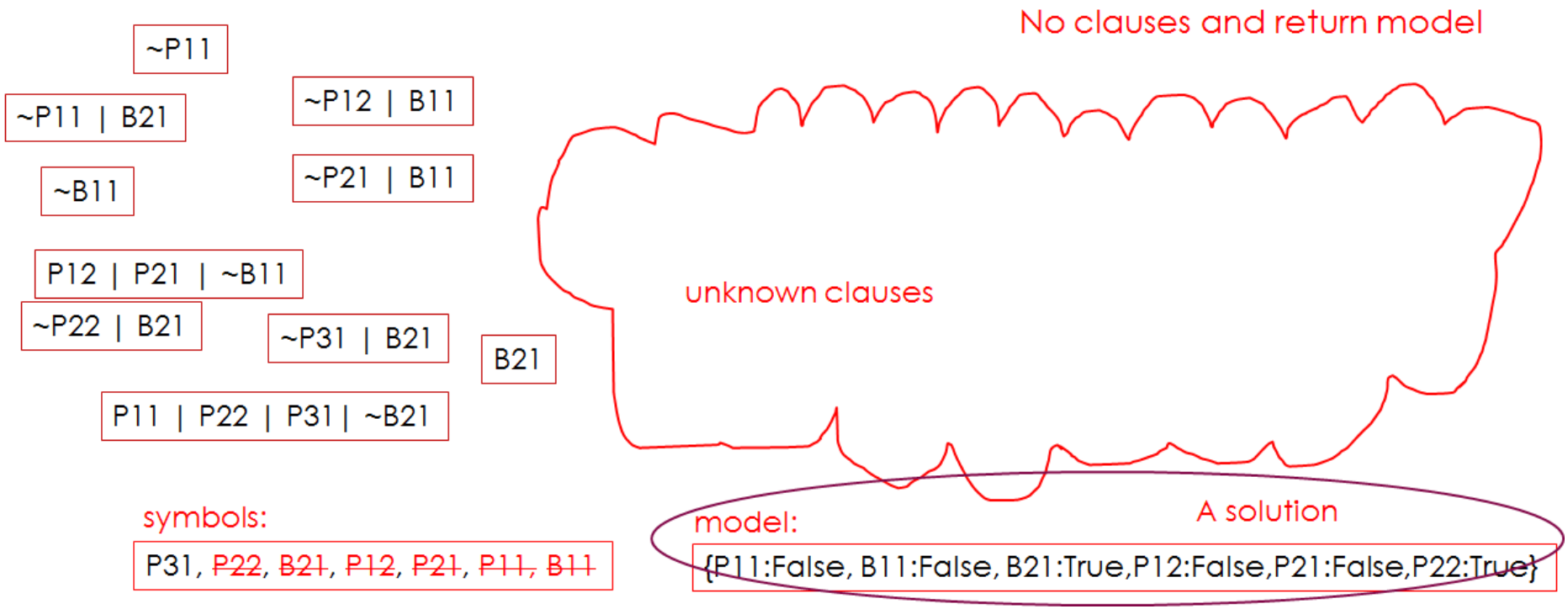
DPLL Example

No pure literals and no unit clauses. Select a literal and assign it to be true



DPLL Example

No pure literals and no unit clauses. Select a literal and assign it to be true



Propositional Logic

- Propositional Logic (PL) is a formal language to describe the world around us.
- Logic can be used by an agent to model the world.
- Sentences in PL have a fixed **syntax**.
- With symbols and connectives we can form logical sentences:
  - Symbols or terms that can be either True or False or unknown.
  - Logical connectives

  Example: $hot \wedge sunny \Rightarrow beach \vee pool$
- Syntax and **Semantics** represent two important and distinct aspects in PL.
- Semantic: configurations/instantiations of the world.

Propositional Logic

- Modus Ponens inference rule:

$$\frac{p_1, \ldots, p_n \qquad (p_1 \wedge \cdots \wedge p_n) \rightarrow q}{q}$$

- Example:

$$\frac{warm \qquad warm \rightarrow sunny}{sunny}$$

- Modus Ponens deals with Horn clauses
- Horn clauses: logic proposition of the form: $p_1 \wedge \ldots \wedge p_n \rightarrow q$
- **Inference**: we want an inference algorithm that is:
  1. sound (does not infer false formulas), and
  2. ideally, complete too (derives all true formulas).
- **Inference in PL with horn clauses is sound and complete for atomic sentence to entail**

Propositional Logic

- Limits of PL?
  1. PL is not expressive enough to describe all the world around us. It can't express information about different object and the relation between objects.
  2. PL is not compact. It can't express a fact for a set of objects without enumerating all of them which is sometimes impossible.
- Example: We have a vacuum cleaner (Roomba) to clean a 10×10 squares in the classroom. Use PL to express information about the squares.

Propositional Logic

- The proposition $square_1\_is\_clean$ expresses information about square1 being clean. How can one write this in a less heavy way?
- How can we express that all squares in the room are clean?

  $square_1\_is\_clean \wedge square_2\_is\_clean \wedge \ldots \wedge square_{100}\_is\_clean$
- How can we express that some squares in the room are clean?

  $square_1\_is\_clean \vee square_2\_is\_clean \vee \ldots \vee square_{100}\_is\_clean$
- How can we express that some squares have chairs on them?

  $square_1\_has\_chair \vee square_2\_has\_chair \vee \ldots \vee square_{100}\_has\_chair$

To be continued

First Order Logic

- Alternative to PL: Another more powerful language, **First Order Logic (FOL).**
- Syntax of FOL:
  - Terms are either:
    - Constant symbols (e.g., A, 10, Shenzhen),
    - Variables (e.g., *x*, *y*)
    - Functions of terms, e.g., sqrt(x), sum(1,2).
  - Atomic formulas: predicates applied to terms, e.g., brother(x,y), above(A,B)
  - Connectives: $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$
  - Equality: $\equiv$
  - Quantifiers: $\forall$, $\exists$
  - Connectives, equality, quantifiers can be applied to atomic formulas to create sentences in FOL.

First Order Logic

All squares are clean:

$$\forall x \;\; Square(x) \Rightarrow Clean(x)$$

There exists some dirty squares:

$$\exists x \;\; Square(x) \wedge \neg Clean(x)$$

Note:

- $\forall x \;\; P(x)$ is like $P(A) \wedge P(B) \wedge \ldots$
- $\exists x \;\; P(x)$ is like $P(A) \vee P(B) \vee \ldots$
- $\neg \forall x \;\; P(x)$ is like $\exists x \;\; \neg P(x)$
- $\forall x \;\; \exists y \;\; likes(x, y)$ is NOT like $\exists y \;\; \forall x \;\; likes(x, y)$

First Order Logic

- All birds fly:

$$\forall x \ \ bird(x) \Rightarrow Fly(x)$$

- All birds except penguins fly:

$$\forall x \ \ bird(x) \wedge \neg penguin(x) \Rightarrow Fly(x)$$

- Every kid likes candy:

$$\forall x \ \ Kid(\mathrm{x}) \Rightarrow Likes(x, candy)$$

- Some kids like candy:

$$\exists x \ \ Kid(x) \wedge Likes(x, candy)$$

- Brothers are sibling:

$$\forall x, y \ \ Brothers(x, y) \Rightarrow Sibling(x, y)$$

- One's mother is one's female parent:

$$\forall x, y \ \ Mother(x, y) \Leftrightarrow Female(x) \wedge Parent(x, y)$$

First Order Logic

- **Inference:** While a bit more complicated than PL, there are procedures to do inference with a knowledge base of FOL formulas (Further optional reading: book chapter 8, 9).
- **Natural language:** The expressiveness of FOL suggests that it is possible to automate the conversion between natural language and logical expressions. This is very valuable for different applications, such as personal assistants (Siri), question/answering systems, and communicating with computers in general.

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - **Syntax**: formal structure of sentences
  - **Semantics**: truth of sentences wrt models
  - **Entailment**: necessary truth of one sentence given another
  - **Inference**: deriving sentences from other sentences
  - **Soundness**: derivations produce only entailed sentences
  - **Completeness**: derivations can produce all entailed sentences
- Forward, backward chaining are linear in time, complete for Horn clauses. Resolution is complete for propositional logic.

Summary

- Building logical agents was a main research trend in AI before the mid-nineties
- Logic is used in AI to represent the environment of the agent and reason about that environment
- Pros and cons of logical agents:
  - Do not handle uncertainty
  - Rule-based and do not use data (Machine Learning)
  - It is hard to model every aspect of the world
  + Intelligibility of models: models are encoded explicitly