Introduction to Artificial Intelligence

Lecture: Logical Agents

Outline

- Knowledge-Based Agents
- Logic
- Propositional Logic
- Propositional Theorem Proving
- Effective Propositional Model Checking

Knowledge-based agents

- Supported by logic – a general class of representation
- Combine and recombine information to suit myriad purposes
- Knowledge base (KB): A set of sentences or facts
  - Each sentence represents some assertion about the world.
  - Axiom = the sentence that is not derived from other sentences
- Inference: Derive (infer) new sentences from old ones
  - Add new sentences to the knowledge base and query what is known

Knowledge-based agents

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

persistent: KB, a knowledge base

t, a counter, initially 0, indicating time

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

action ← ASK(KB, MAKE-ACTION-QUERY(t))

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

t ← t + 1

return action

Inference mechanisms are hidden inside TELL and ASK

Logic

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines the well-formed sentences in the language
- Semantics define the "meaning" of sentences
- **Models** are mathematical abstractions that fix the truth or falsehood of every relevant sentence.
- m satisfies (or is a model of) α if α is true in model m
- $M(\alpha)$ = the set of all models of α

Entailment

- A sentence follows logically from another sentence: $\boldsymbol{\alpha} \vDash \boldsymbol{\beta}$
- $\boldsymbol{\alpha} \vDash \boldsymbol{\beta}$ if and only if, in every model in which $\boldsymbol{\alpha}$ is true, $\boldsymbol{\beta}$ is also true.
  - $M(\alpha) \subseteq M(\beta)$
  - $x = 0$ entails $xy = 0$
- Entailment is a relationship between sentences that is based on semantics.

Logical inference

- $KB \models_i \alpha$ means α can be derived from KB by procedure i.
- Soundness: i is sound if whenever $KB \models_i \alpha$, it is also true that $KB \models \alpha$
- Completeness: i is complete if whenever $KB \models \alpha$, it is also true that $KB \models_i \alpha$

World and representation

- The reasoning agent gets its knowledge about the facts of the world as a sequence of logical sentences
- Conclusions must be drawn only from those → without agent's independent access to the world
- Thus it is very important that the agent's reasoning is sound.

Propositional Logic: Syntax

- Propositional logic: the simplest logic illustrating basic ideas
- Constants: TRUE or FALSE
- Symbols stand for propositions (sentences): P, Q, W, etc.
- Logical connectives

NOT	¬	Negation
AND	∧	Conjunction
OR	∨	Disjunction
IMPLIES	⇒	Implication (if..then)
IFF	⇔	Equivalence, biconditional

- Literal: atomic sentence (P) or negated atomic sentence (¬P)

Propositional Logic: Syntax

$$
\begin{aligned}
\mathit{Sentence} &\rightarrow \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} &\rightarrow \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} &\rightarrow (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
&\mid \neg\, \mathit{Sentence} \\
&\mid \mathit{Sentence} \wedge \mathit{Sentence} \\
&\mid \mathit{Sentence} \vee \mathit{Sentence} \\
&\mid \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
&\mid \mathit{Sentence} \Leftrightarrow \mathit{Sentence}
\end{aligned}
$$

$$
\text{OPERATOR PRECEDENCE} \quad : \quad \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
$$

Semantics

- Each model specifies true/false for each proposition symbol

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*

A simple inference procedure

- Given: a set of sentences, $\boldsymbol{KB}$, and sentence $\boldsymbol{\alpha}$
- Goal: answer $\boldsymbol{KB} \models \boldsymbol{\alpha}$? = “Does $\boldsymbol{KB}$ semantically entail $\boldsymbol{\alpha}$?”
- Approaches
  - Model-checking approach (Inference by enumeration)
  - Inference rules
  - Conversion to the inverse SAT problem (Resolution refutation)

Model-checking approach

- Check if α is true in every model in which KB is true
- Draw a truth table for checking

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

Inference by (depth-first) 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 // when KB is false, always return true
    else do
        P ← FIRST(symbols)
        rest ← REST(*symbols*)
        return (TT-CHECK-ALL(KB,α,rest,model ∪ {P = true})
                and TT-CHECK-ALL(KB,α,rest,model ∪ {P = false}))

Model-checking approach

- Given a KB containing the following rules and facts

  R1: IF hot AND smoky THEN fire

  R2: IF alarm_beeps THEN smoky

  R3: IF fire THEN sprinklers_on

  F1: alarm_beeps

  F2: hot

- Represent the KB in propositional logic with given symbols
  - H = hot, S = smoky, F = fire, A = alarms_beeps,
  - R = sprinklers_on
- Answer the question “Sprinklers_on?” by using the model- checking approach.

Inference rules approach

- Theorem proving: Apply rules of inference directly to the sentences in KB to construct a proof of the desired sentence without consulting models

Logical equivalence

- Two sentences α and β are logically equivalent if they are true in the same set of models.

$$\boldsymbol{\alpha \equiv \beta \; iff \; \alpha \models \beta \; and \; \beta \models \alpha}$$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validity

- A sentence is valid if it is true in all models.
- Valid sentences are also known as tautologies.
- Validity is connected to inference via the Deduction Theorem

$$\alpha \vDash \beta \; iff \; \alpha \Rightarrow \beta \; is \; valid$$

Satisfiability

- A sentence is satisfiable if it is true in some model.
- A sentence is unsatisfiable if it is true in no models.
- Satisfiability is connected to inference via the following

$$\alpha \models \beta \; iff \; \alpha \wedge \neg\beta \; is \; unsatisfiable$$

→ Refutation or proof by contradiction

- The SAT problem determines the satisfiability of sentences in propositional logic (NP-complete).

Validity and Satisfiability

1. $A \vee B \Rightarrow A \wedge C$
2. $A \wedge B \Rightarrow A \vee C$
3. $(A \vee B) \wedge (\neg B \vee C) \Rightarrow A \vee C$
4. $(A \vee \neg B) \Rightarrow A \wedge B$

Inference and Proofs

- Proof: A chain of conclusions leads to the desired goal

$$\frac{\alpha \Rightarrow \beta \quad \alpha}{\therefore \beta}$$ Modus Ponens

$$\frac{\alpha \Rightarrow \beta \quad \neg\beta}{\therefore \neg\alpha}$$ Modus Tollens

$$\frac{\alpha \quad \beta}{\therefore \alpha \wedge \beta}$$ AND-Introduction

$$\frac{\alpha \wedge \beta}{\therefore \alpha}$$ AND-Elimination

Inference and Proofs

KB
$P \wedge Q$
$P \Rightarrow R$
$Q \wedge R \Rightarrow S$

S?

No.	Sentences	Explanation
1	$P \wedge Q$	From KB
2	$P \Rightarrow R$	From KB
3	$Q \wedge R \Rightarrow S$	From KB
4	P	1 And-Elim
5	R	4,2 Modus Ponens
6	Q	1 And-Elim
7	$Q \wedge R$	5,6 And-Intro
8	S	3,7 Modus Ponens

Monotonicity

- The set of entailed sentences only increases as information is added to the knowledge base.

$$if\ KB \models \alpha\ then\ KB \wedge \beta \models \alpha$$

- Additional conclusions can be drawn without invalidating any conclusion α already inferred.

Proof by Resolution

- Proof by Inference Rules: sound but not complete
- Resolution: sound and complete, a single inference rule
- Given l_i and m are complementary literals
  - Unit resolution inference rule

$$\frac{l_1 \vee \cdots \vee l_k \qquad m}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k}$$

  - Full resolution rule

$$\frac{l_1 \vee \cdots \vee l_k \qquad m_1 \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

Proof by Resolution

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

...

R_{11}: $\neg B_{1,2}$

R_{12}: $B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$

R_{13}: $\neg P_{2,2}$

R_{14}: $\neg P_{1,3}$

R_{15}: $P_{1,1} \vee P_{2,2} \vee P_{3,1}$

R_{16}: $P_{1,1} \vee P_{3,1}$

R_{17}: $P_{3,1}$

$\neg P_{2,2}$ resolves with $P_{2,2}$

$\neg P_{1,1}$ resolves with $P_{1,1}$

Proof by Resolution

- Factoring: the resulting clause should contain only one copy of each literal.
- For any sentences α and β in propositional logic, a resolution-based theorem prover can decide whether $\alpha \vDash \beta$.

Conjunctive Normal Form (CNF)

- Resolution applies only to clauses, i.e. disjunctions of literals
  - Convert all sentences in KB into clauses (CNF form)
- For example, convert $A \Leftrightarrow (B \vee C)$ into CNF

$$(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$$

Conversion to CNF

- Eliminate ⇔: $\alpha \Leftrightarrow \beta \equiv \alpha \Rightarrow \beta \wedge \beta \Rightarrow \alpha$
- Eliminate ⇒: $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
- The operator ¬ appears only in literals: “move ¬ inwards”

  $\neg\neg\alpha \equiv \alpha$ (double-negation elimination)

  $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$ (De Morgan)

  $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$ (De Morgan)

- Apply the distributivity law to distribute ∨ over ∧

  $$(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$

The resolution algorithm

- Proof by contradiction: To show that $KB \models \alpha$, prove $KB \wedge \neg\alpha$ is unsatisfiable

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 ← the set of clauses in the CNF representation of KB ∧ ¬α

new ← { }

loop do

for each pair of clauses C_i, C_j **in** clauses **do**

resolvents ← PL-RESOLVE(C_i , C_j)

if resolvents contains the empty clause **then return** true

new ← new ∪ resolvents

if new ⊆ clauses **then return** false

clauses ← clauses ∪ new

Horn clauses and Definite clauses

- Definite clause: a disjunction of literals of which exactly one is positive.
- Horn clause: a disjunction of literals of which at most one is positive.
- Goal clause: clauses with no positive literals

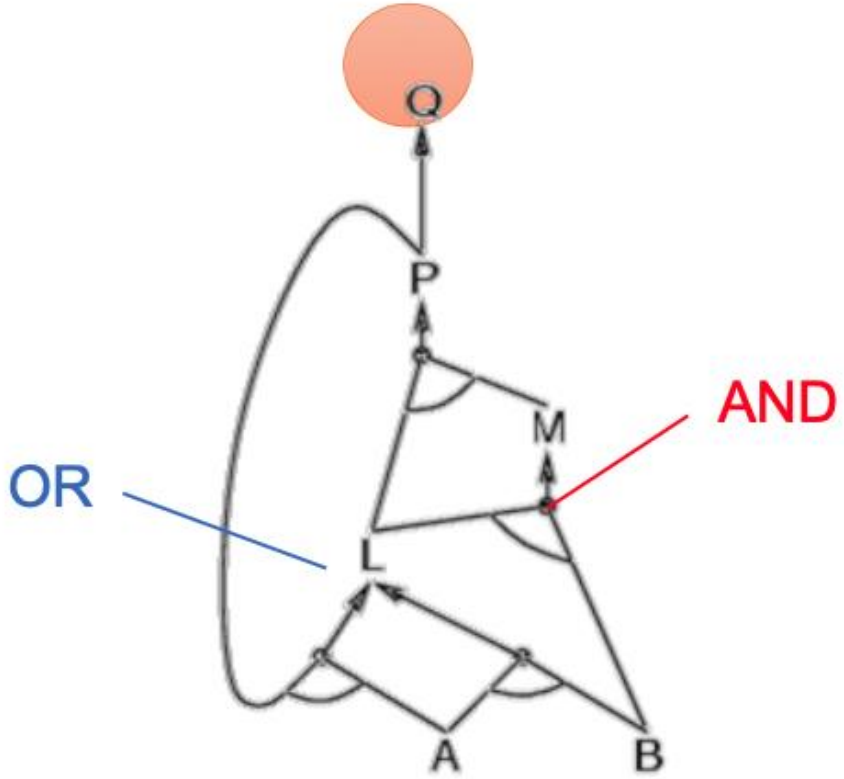
KB of definite clauses

- KB containing only definite clauses are interesting.
- Every definite clause can be written as an implication.
- Inference can be done with forward-chaining and backward-chaining algorithms.
- Deciding entailment can be done in linear time.

Forward chaining

- Key idea: Fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until the 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

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

inputs: KB, the knowledge base, a set of propositional definite clauses
q, the query, a proposition symbol

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

inferred ← a table, where inferred[s] is initially false for all symbols

agenda ← a queue of symbols, initially symbols known to be true in KB

while agenda is not empty **do**

p ← POP(agenda)

if p = q **then return** true

if inferred[p] = false **then**

inferred[p] ← true

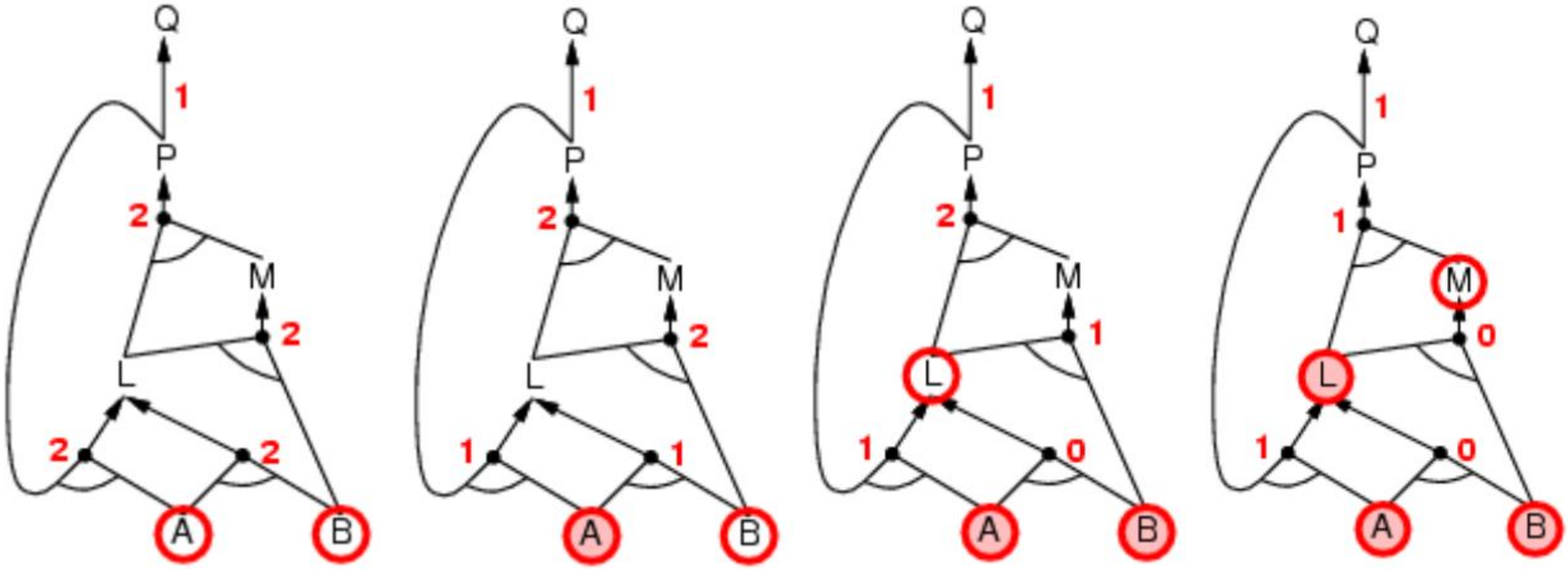
for each clause c **in** KB **where** p is in c.PREMISE **do**

decrement count[c]

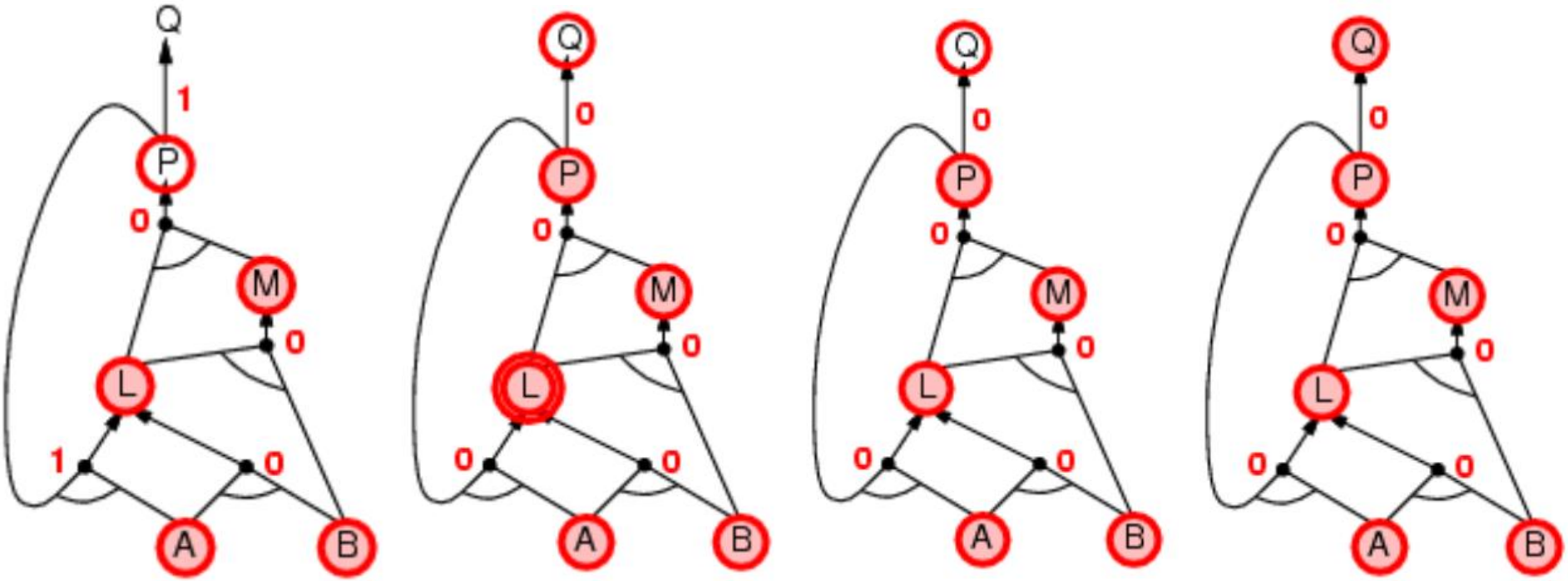
if count[c] = 0 **then** add c.CONCLUSION to agenda

return false

Forward chaining



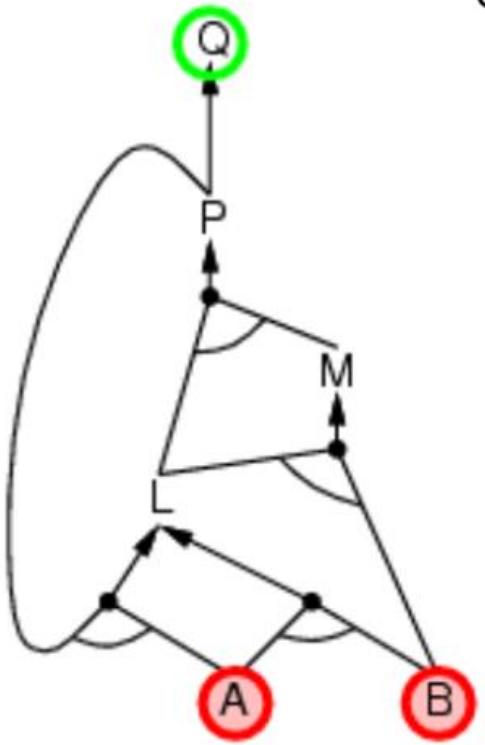
Forward chaining



Backward chaining

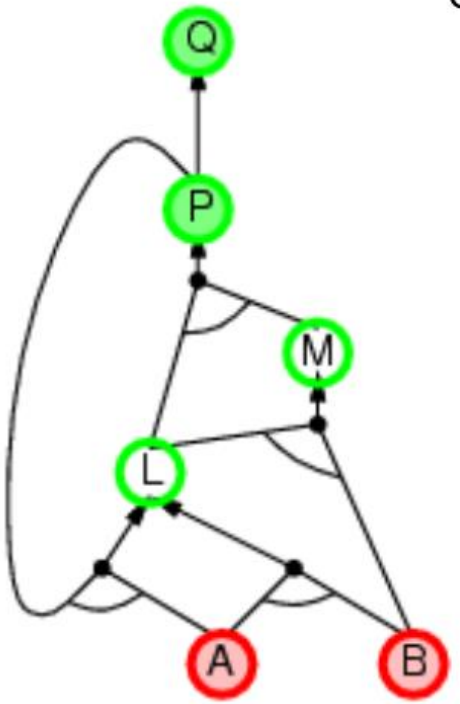
- Key idea: Work backwards from the query q
- Avoid loops: A new subgoal is already on the goal stack?
- Avoid repeated work: A new subgoal has already been proved true, or has already failed?

Backward chaining



Q?
P?

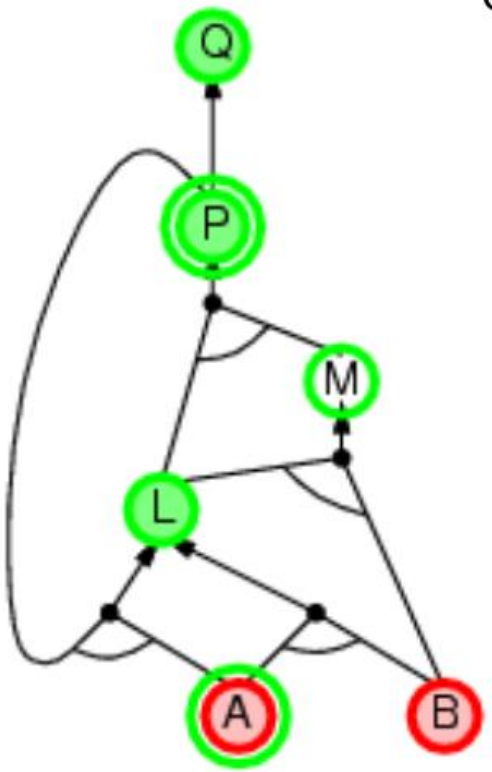
$P \Rightarrow Q$



Q?
P?
L?

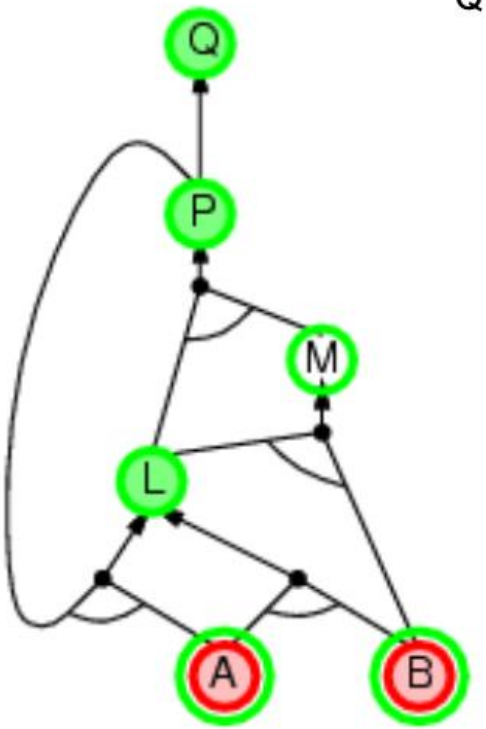
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$

Backward chaining



Q?
P?
L?
A?

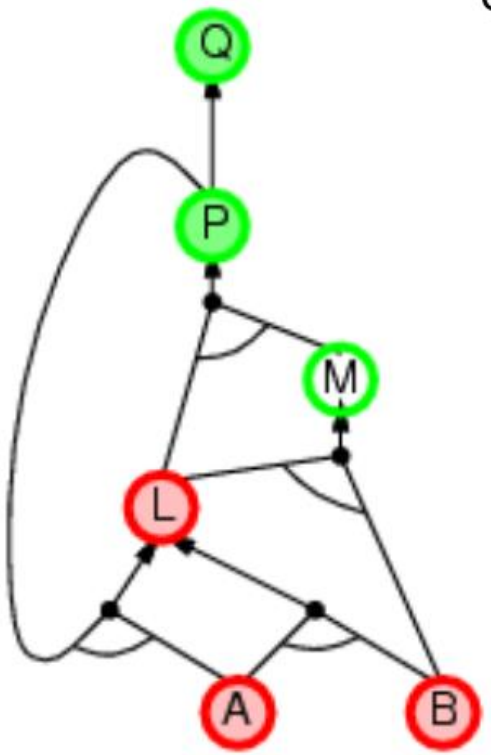
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$A \wedge B \Rightarrow L$
✓



Q?
P?
L?
A?
B?

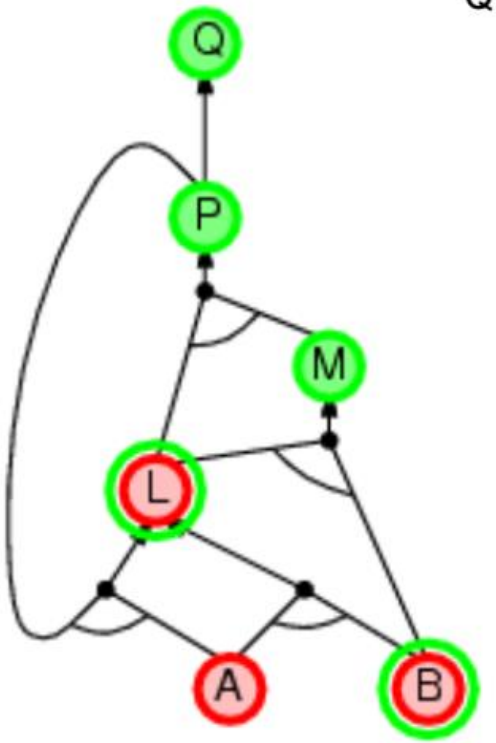
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$A \wedge B \Rightarrow L$
✓
✓

Backward chaining



Q?
P?
L? ✓
A? ✓
B? ✓

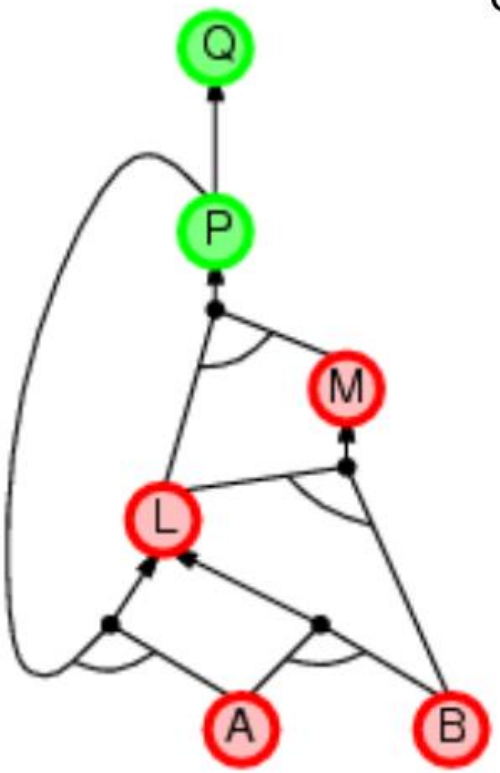
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$



Q?
P?
L? ✓
A? ✓
B? ✓
M?
L?
B?

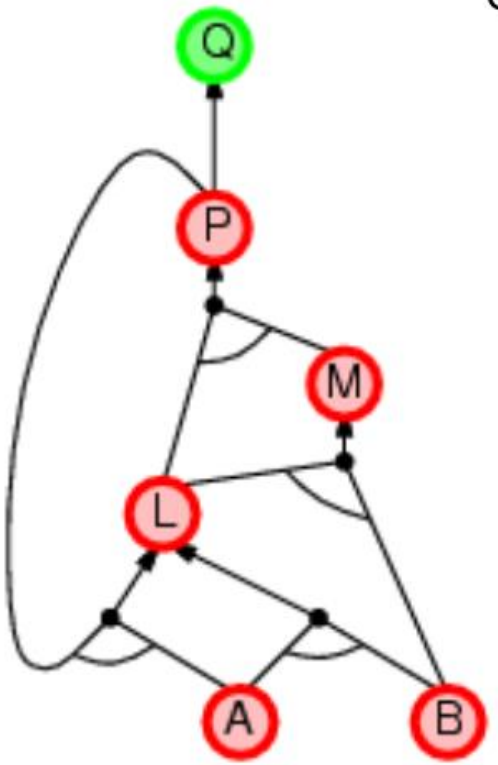
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$L \wedge B \Rightarrow M$

Backward chaining



Q?
P?
L? ✓
A? ✓
B? ✓
M? ✓
L? ✓
B? ✓

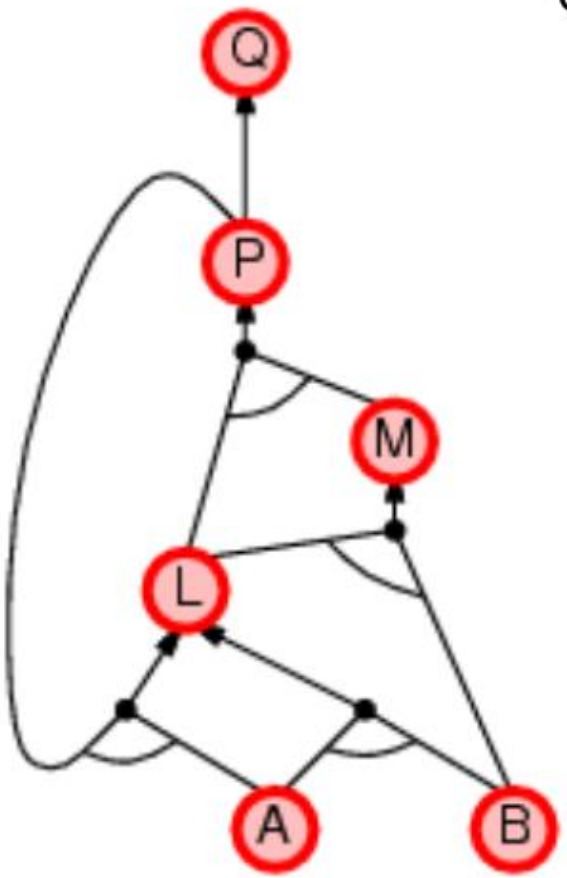
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$



Q? ✓
P? ✓
L? ✓
A? ✓
B? ✓
M? ✓
L? ✓
B? ✓

Backward chaining

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



Q? ✓
P? ✓
L? ✓
A? ✓
B? ✓
M? ✓
L? ✓
B? ✓

Forward vs. Backward chaining

- Forward chaining: data-driven, automatic, unconscious processing
- Backward chaining: goal-driven, good for problem-solving

Efficient propositional inference

- The SAT problem (checking satisfiability)
  - Testing entailment, $\alpha \models \beta$? = testing unsatisfiability of $\alpha \wedge \neg\beta$
- Two families of efficient algorithms for general propositional inference based on model checking
  - Complete backtracking search algorithms

    DPLL algorithm (Davis, Putnam, Logemann, Loveland)
  - Incomplete local search algorithms(hill-climbing)

    WalkSAT algorithm

The DPLL algorithm

- Determine whether an input propositional logic sentence (in CNF) is satisfiable
- Improvements over truth table enumeration

  1. Early termination
  2. Pure symbol heuristic
  3. Unit clause heuristic

The DPLL algorithm

- Early termination: A clause is true if any literal is true and a sentence is false if any clause is false.
- Pure symbol heuristic: A pure symbol always appears with the same "sign" in all clauses.
- Unit clause heuristic: there is only one literal in the clause and thus this literal must be true

The DPLL algorithm

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,{ })

The DPLL algorithm

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

if every clause in clauses is true in model **then** return true

if some clauses in clauses are 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}))

The WalkSAT algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max_flips, number of flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability *p* flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

The WalkSAT algorithm

- When the algorithm returns a model → The input sentence is indeed satisfiable
- When it returns false → The sentence is unsatisfiable OR we need to give it more time
- WalkSAT cannot always detect unsatisfiability
- It is most useful when a solution is expected to exist

PySAT - Glucose3

```
! pip install python-sat

from pysat.solvers import Glucose3

g = Glucose3()
g.add_clause([1])
g.add_clause([2])
g.add_clause([-1, 3])
g.add_clause([-2, -3, -4])
g.add_clause([-4])

print(g.solve())
# True

g.get_model()
# [1, 2, 3, -4]
```

References

- Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
- Lê Hoài Bắc, Tô Hoài Việt. 2014. Giáo trình Cơ sở Trí tuệ nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.
- Nguyễn Ngọc Thảo, Nguyễn Hải Minh. 2020. Bài giảng Cơ sở Trí tuệ Nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.