Artificial Intelligence

Knowledge

Nguyễn Văn Diêu

HO CHI MINH CITY UNIVERSITY OF TRANSPORT

2025

Kiến thức - Kỹ năng - Sáng tạo - Hội nhập

Sứ mệnh - Tầm nhìn

Triết lý Giáo dục - Giá trị cốt lõi

Outline I

Outline II

Outline III

Outline IV

Outline V

Knowledge-Based Agent

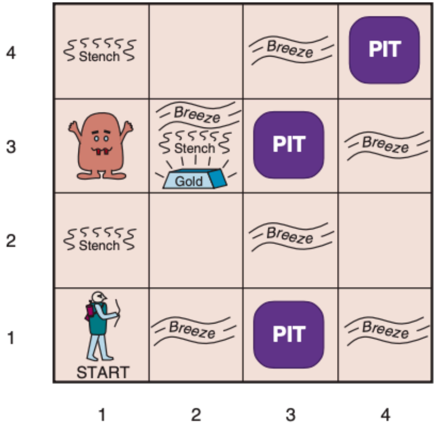
```
function KB-AGENT(percept) returns an action
   persistent: KB, a knowledge base
               t, a counter, initially 0, indicating time

   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
```

Each time the agent program is called, it does three things:

1. **TELL** the knowledge base what it perceives
2. **ASK** the knowledge base what action it should perform. Outcomes of possible action sequences
3. **TELL** the knowledge base which action was chosen. Returns the action so that it can be executed

Wumpus world game



Move around a square board looking for **Gold** while avoiding **Pits** and the **Wumpus**.

Performance Measure

- $+1000$ reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environmen

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators

- Left turn.
- Right turn.
- Move forward.
- Grab.
- Release.
- Shoot.

Sensors

- **Stench** if the room adjacent to the **Wumpus**.
- **Breeze** if the room adjacent to the **Pit**.
- **Glitter** in the room where the **Gold** is present.
- **Bump** if walks into a **Wall**.
- **Scream** when the Wumpus is **shot** anywhere in the cave.
- There are five element percepts list.

  e.g. if agent perceives Stench, Breeze, but no Glitter, no Bump, and no Scream then it can be represented as:

  [Stench, Breeze, None, None, None].

Wumpus world, first step

(a) (b)

(a) The initial situation, after percept:

[None, None, None, None, None].

(b) After moving to [2,1], perceiving:

[None, Breeze, None, None, None].

Wumpus world, two later stages

(a) (b)

(a) After moving to [1,1] and then [1,2], and perceiving:
[Stench,None,None,None,None].

(b) After moving to [2,2] and then [2,3], and perceiving:
[Stench,Breeze,Glitter,None,None].

Logic in general

- **Sentences**: A technical term. It describe the logic.
- **Knowledge bases**: A set of sentences.
- **Syntax**: Syntax of the representation language.
- **Semantics**: Meaning of sentence.
- **Truth**: The semantics of sentences.
- **Standard logics**: **true** or **false** - there is no “in between.”

Model - Entailment

- **Model**: Mathematical abstractions. It is Truth value (true or false) for every relevant sentence
- **Satisfaction**: If sentence α is true in model m, saying m satisfies α or sometimes m is a **model** of α
- $M(\alpha)$: Set of all **models** of α
- **Entailment** a sentence *follows logically* from another sentence:

$$\alpha \models \beta$$

  Sentence α **entails** the sentence β **if and only if**, in every model in which α is **true**, β is also **true**.

$$\alpha \models \beta \quad \textit{if and only if} \quad M(\alpha) \subseteq M(\beta)$$

Inference

Inference is an algorithm i can derive α from KB, we write:

$$KB \vdash_i \alpha$$

Pronounce: "α is derived from KB by i" or "i derives α from KB"

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 $KB \vdash_i \alpha$

Propositional Logic

- **Syntax** of propositional logic defines the allowable sentences.
- **Proposition symbol**: P, Q, R, $W_{1,3}$ and $FacingEast$ ...
- **Atomic sentences** consists of a single proposition symbol.
- **Complex sentences** are constructed from simpler sentences, using parentheses and operators called **logical connectives**.
- five **connectives** in common use:
  - $\neg$ (not). **negation** of.
  - $\wedge$ (and). **conjunction**.
  - $\vee$ (or). **disjunction**.
  - $\Rightarrow$ (implies). sometimes written $\supset$ or $\rightarrow$. **rules** or **if - then**
  - $\Leftrightarrow$ (if only if - iff). **biconditional**

BNF (Backus–Naur Form)

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} & \rightarrow & (\, \textit{Sentence} \,) \\
& \mid & \neg\, \textit{Sentence} \\
& \mid & \textit{Sentence} \land \textit{Sentence} \\
& \mid & \textit{Sentence} \lor \textit{Sentence} \\
& \mid & \textit{Sentence} \;\Rightarrow\; \textit{Sentence} \\
& \mid & \textit{Sentence} \;\Leftrightarrow\; \textit{Sentence} \\
\\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

BNF grammar of sentences in propositional logic.

Semantics

The semantics defines the rules for determining the truth of a sentence.

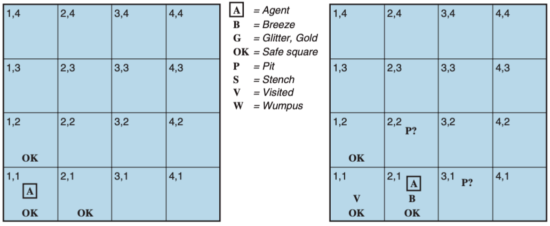
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Truth tables for the five logical connectives.

In any model m:

- $\neg P$ is *true* iff P is *false* in m.
- $P \wedge Q$ is *true* iff both P and Q are *true* in m.
- $P \vee Q$ is *true* iff either P or Q is *true* in m.
- $P \Rightarrow Q$ is *true* unless P is *true* and Q is *false* in m.
- $P \Leftrightarrow Q$ is *true* iff P and Q are both *true* or both *false* in m.

KB for Wumpus World



(a) (b)

Symbols for each $[x, y]$ location:

- $P_{x,y}$ is *true* if there is a **Pit** in $[x, y]$.
- $W_{x,y}$ is *true* if there is a **Wumpus** in $[x, y]$, dead or alive.
- $B_{x,y}$ is *true* if there is a **Breeze** in $[x, y]$.
- $S_{x,y}$ is *true* if there is a **Stench** in $[x, y]$.
- $L_{x,y}$ is *true* if the agent is in **Location** $[x, y]$.

KB for Wumpus World

We concern 4 squares, it is represent knowledge base for Wumpus World: **[1,2], [2,1], [2,2]** and **[3,1]**.

R_i: Label sentence so that we can refer to them:

- There is no **Pit** in [1,1]:
  $R_1 : \neg P_{1,1}$
- A square is **Breezy** iff there is a **Pit** in a neighboring square:
  $R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
  $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
- **Breeze** percepts for the first two squares:
  $R_4 : \neg B_{1,1}$
  $R_5 : B_{2,1}$

KB for Wumpus World

7 symbol: $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$.
$2^7 = 128$ possible models.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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 for the knowledge base. KB is true if R_1 - R_5 are true, which occurs in just 3 of the 128 rows.
In all 3 rows, $P_{1,2}$ is false, so there is no **Pit** in **[1,2]**.
On the other hand, there might (or might not) be a **Pit** in **[2,2]**.

Truth-table algorithm

```
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
      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 }))
```

TT: Truth-Table algorithm for deciding propositional entailment.
PL-TRUE?: Propositional Logic True?

Propositional Theorem Proving

Logical equivalence: α and β are logically equivalent if they are *true* in the same set of models: $\alpha \equiv \beta$

$\equiv$ is used to make claims about sentences, while

$\Leftrightarrow$ is used as part of a sentence.

$\alpha \equiv \beta$ *if and only if* $\alpha \models \beta$ and $\beta \models \alpha$

Validity, **Tautology**:

A sentence is valid if it is *true* **in all models**.

Deduction theorem:

α and β, $\alpha \models \beta$ *iff sentence* $(\alpha \Rightarrow \beta)$ is **valid**.

Satisfiability: if the sentence is true in, or satisfied by, some model.
Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence.
It is **SAT** problem (Boolean satisfiability problem), NP-complete.

Theorem Proving

Validity and satisfiability are connected:
α is valid iff $\neg\alpha$ is unsatisfiable.
α is satisfiable iff $\neg\alpha$ is not valid.

$$\alpha \models \beta \text{ iff the sentence } (\alpha \land \neg\beta) \textbf{ is unsatisfiable}.$$

Proof by **contradiction**:

Proving β from α by checking the unsatisfiability of $(\alpha \land \neg\beta)$.

Assumes a sentence β to be false and shows that this leads to a **contradiction** with known axioms α. It is meant the sentence $(\alpha \land \neg\beta)$ is unsatisfiable

Logical equivalences

Two kind of logical operator:

- **Entail, equivalences:** $\models$, $\equiv$
- **Connectives:** $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

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

Inference and proofs

Modus Ponens Give $\alpha \Rightarrow \beta$ and α, then β.

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

e.g. Give $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$
and $(WumpusAhead \wedge WumpusAlive)$, then $Shoot$.

And-Elimination from a $\wedge$ conjunction, any of the conjuncts can be inferred:

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

e.g. Give $(WumpusAhead \wedge WumpusAlive)$, then $WumpusAlive$.
All of the logical equivalences can be used as inference rules.

Inference and proofs

e.g.

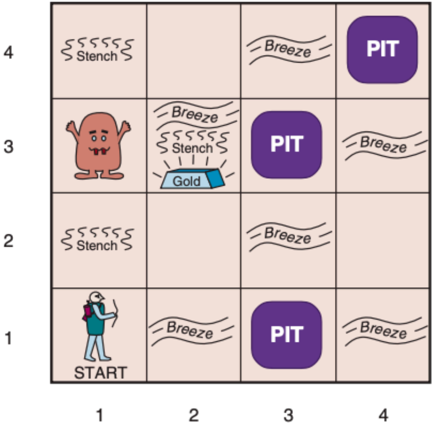
$$\frac{\alpha \Rightarrow \beta}{\neg\alpha \vee \beta}, \quad \frac{\neg\alpha \vee \beta}{\alpha \Rightarrow \beta}, \quad \frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

monotonicity: increase as information.

Any sentences α and β,

$$\textbf{if } KB \models \alpha \textbf{ then } KB \wedge \beta \models \alpha$$

Wumpus world



Wumpus world

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

1. "$\Leftrightarrow$" to R_2:
   $R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. And-Elimination to R_6:
   $R_7 : ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
3. "$\Rightarrow$" to R_7:
   $R_8 : (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$
4. Modus Ponens with R_8 and the percept R_4 $(\neg B_{1,1})$:
   $R_9 : \neg(P_{1,2} \vee P_{2,1})$
5. De Morgan's rule, giving the conclusion:
   $R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$: **Neither [1,2] nor [2,1] contains a Pit.**

Inference and proofs

Searching algorithms can be used to find a sequence of the steps.

Proof problem as follows:

- **INITIAL STATE**: the initial knowledge base.
- **ACTIONS**: the set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule.
- **RESULT**: the result of an action is to add the sentence in the bottom half of the inference rule.
- **GOAL**: the goal is a state that contains the sentence we are trying to prove.

Proof by truth-table algorithm: It would be overwhelmed by the exponential explosion of models

Proof by resolution

clause disjunction of literal

Unit resolution rule

ℓ: literal

ℓ_i and m: Complementary literals (one is the negation of the other)

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k}$$

Full resolution rule

ℓ_i and m_j: Complementary literals

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

another represent:

$$\frac{\{p, r\}, \{\neg p, k\}}{\{r, k\}}$$

e.g.

- agent returns from [2,1] to [1,1] and then goes to [1,2]. Add the following facts to the KB:

$$R_{11} : \neg B_{1,2}$$
$$R_{12} : B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$$

- absence of pits in [2,2] and [1,3]

$$R_{13} : \neg P_{2,2}$$
$$R_{14} : \neg P_{1,3}$$

- apply biconditional elimination to R_3, followed by Modus Ponens with R_5, to obtain the fact that there is a pit in [1,1], [2,2], or [3,1]

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

e.g.

- a pit in one of [1,1], [2,2], and [3,1]? it's not in [2,2], then it's in [1,1] or [3,1].
- apply resolution rule: $\neg P_{2,2}$ in R_{13} with $P_{2,2}$ in R_{15}:

  $R_{16} : P_{1,1} \vee P_{3,1}$
- a pit in [1,1] or [3,1]? it's not in [1,1], then it's in [3,1].
- apply resolution rule: $\neg P_{1,1}$ in R_1 with $P_{1,1}$ in R_{16}:

  $R_{17} : P_{3,1}$: a pit in [3,1]

Conjunctive normal form

Every sentence of propositional logic is logically equivalent to a conjunction of clauses.

Grammar for conjunctive normal form, Horn clauses, and definite clauses:

$$
\begin{array}{rcl}
\mathit{CNFSentence} & \rightarrow & \mathit{Clause}_1 \wedge \cdots \wedge \mathit{Clause}_n \\
\mathit{Clause} & \rightarrow & \mathit{Literal}_1 \vee \cdots \vee \mathit{Literal}_m \\
\mathit{Fact} & \rightarrow & \mathit{Symbol} \\
\mathit{Literal} & \rightarrow & \mathit{Symbol} \mid \neg \mathit{Symbol} \\
\mathit{Symbol} & \rightarrow & P \mid Q \mid R \mid \ldots \\
\mathit{HornClauseForm} & \rightarrow & \mathit{DefiniteClauseForm} \mid \mathit{GoalClauseForm} \\
\mathit{DefiniteClauseForm} & \rightarrow & \mathit{Fact} \mid (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{Symbol} \\
\mathit{GoalClauseForm} & \rightarrow & (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{False}
\end{array}
$$

Converting to CNF

e.g. $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ converting to **CNF**

The steps are as follows:

1. 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. 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. CNF requires $\neg$ to appear only in literals, so using:
   $\neg(\neg\alpha) \equiv \alpha$ (double-negation)
   $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ (De Morgan)
   $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ (De Morgan)
   the e.g. result: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply the distributivity law $\wedge$ and $\vee$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$
   Sentence is now in **CNF**, as a conjunction of three clauses

Resolution algorithm

- Using the principle of proof by contradiction
- To show $KB \models \alpha$, we show that $(KB \wedge \neg\alpha)$ is **unsatisfiable**

Algorithm $(\textit{show } KB \models \alpha)$

1. Convert $(KB \wedge \neg\alpha)$ into **CNF**
2. Apply resolution rule to the resulting clauses
   - Each pair that contains complementary literals is resolved to produce a new clause
   - added to the set if it is not already present
3. Loop step 2 until one of two things happens:
   - **No new clauses that can be added**: $KB \not\models \alpha$ or,
   - 2 clauses resolve to yield the **empty clause**: $KB \models \alpha$.

Resolution algorithm

PL-Resolve returns the set of all possible clauses obtained by resolving its two inputs.

```
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 ← { }
   while true 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
```

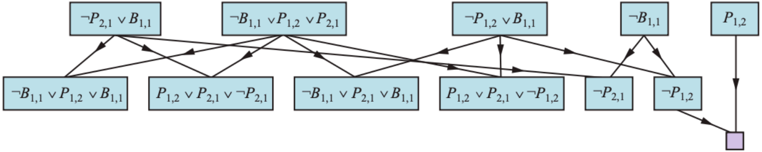
e.g. Agent is in [1,1], no breeze, so no pits in neighboring squares:

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

- $KB = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- Prove $\alpha = \neg P_{1,2}$
- Convert $(KB \wedge \neg\alpha)$ into **CNF**

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

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



Query $\neg P_{1,2}$, PL-RESOLUTION yield the **empty clause**.
So that the query is proven.

e.g. Resolution alg.

Consider the following six statements, all of which to be true:

1. If you go swimming you will get wet.
2. If it is raining and you are outside then you will get wet.
3. If it is warm and there is no rain then it is a pleasant day.
4. You are not wet.
5. You are outside.
6. It is a warm day.

Determine following statements must be true:

a. You are not swimming.
b. It is not raining.
c. It is a pleasant day.

e.g. Resolution alg.

In propositional logic:

1. $swimming \Rightarrow wet$
2. $(rain \land outside) \Rightarrow wet$
3. $(warm \land \neg rain) \Rightarrow pleasant$
4. $\neg wet$
5. $outside$
6. $warm$

Statements:

a. $\neg swimming$

b. $\neg rain$

c. $pleasant$

e.g. Resolution alg.

a) $\{1, 2, 3, 4, 5, 6\} \models \neg \mathit{swimming}$

Contradition method and convert to CNF:

1. $\neg \mathit{swimming} \vee \mathit{wet}$
2. $\neg \mathit{rain} \vee \neg \mathit{outside} \vee \mathit{wet}$
3. $\neg \mathit{warm} \vee \mathit{rain} \vee \mathit{pleasant}$
4. $\neg \mathit{wet}$
5. $\mathit{outside}$
6. warm
7. $\mathit{swimming}$

exercises:

b. $\neg \mathit{rain}$

PL-Resolution:

8. $(1) \wedge (4) : \neg \mathit{swimming}$
9. $(7) \wedge (8) : \blacksquare$

$\neg \mathit{swimming}$

e.g. The Power of False

$P \wedge \neg P \models Z$

We know: $P \wedge \neg P \equiv false$

$P \wedge \neg P \wedge \neg Z$

PL-Resolution:

$P \wedge \neg P$: $\blacksquare$

Z

Horn clauses and definite clauses

- **Definite clause**: **disjunction** of literals of which **exactly one is positive**.

  e.g. $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ is a definite clause,

  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ is not, because it has two positive clauses
- **Horn clause**: **disjunction** of literals of which **at most one is positive**.

  all definite clauses are Horn clauses.

  KB containing only definite clauses are interesting:

  1. Every definite clause can be written as an implication.

  e.g. $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ can be written as the implication
  $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$

Horn clauses and definite clauses

- **Horn clause**

  premise is call **body**

  conclusion is call **head**

  sentence consisting of a single positive literal is call **fact**

  e.g. $L_{1,1}$

  fact $L_{1,1}$ can be written in implication form: $True \Rightarrow L_{1,1}$

  2. Inference with Horn clauses can be done through the **forward-chaining** and **Backward-chaining** algorithms

  3. Entailment with Horn clauses in linear time with the size of knowledge base

Horn clauses and definite clauses

- **Goal clauses**: clauses with no positive literals.
- **Modus Ponens** (for Horn Form): complete for Horn KBs

$$\frac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta, \quad \alpha_1, \ldots, \alpha_n}{\beta}$$

FW and BW Chaining

Inference Engine

Apply **rules** of **KB** to **infer** new information.

Modus Ponens

e.g.

A It is raining

$A \Rightarrow B$ if it is rainning i will carry an umbrella

B I will carry an umbrellaq (new knowledge)

- **Forward Chaining**: Start with atomic sentences in the KB and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
- **Backward Chaining**: Starts with the goal and works backward, chaining through rules to find known facts that support the goal.

FW and BW Chaining

e.g.

Forward Chaining

A	He exercises regularly.
$A \Rightarrow B$	if he is exercising regularly, he is fit.
B	He is fit.

Backward Chaining

B	He is fit.
$A \Rightarrow B$	if he is exercising regularly, he is fit.
A	He exercises regularly.

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 initially the number of symbols in clause c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  queue ← a queue of symbols, initially symbols known to be true in KB

  while queue is not empty do
     p ← POP(queue)
     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 queue
  return false
```

The forward-chaining algorithm for propositional logic

- **Forward chaining is sound and complete for** *Horn KB*

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Queue
0		A B

Inferred	
A	false
B	false
L	false
M	false
P	false
Q	false

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

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L. \quad A, \quad B$

No.	Reached	Queue
0		A B
1	A	B

Inferred	
A	**true**
B	false
L	false
M	false
P	false
Q	false

Count	
$P \Rightarrow Q$	1
$L \wedge M \Rightarrow P$	2
$B \wedge L \Rightarrow M$	2
$A \wedge P \Rightarrow L$	**1**
$A \wedge B \Rightarrow L$	**1**

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Queue
0		A B
1	A	B
2	B	**L**

Inferred	
A	true
B	**true**
L	false
M	false
P	false
Q	false

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

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$. A, B

No.	Reached	Queue
0		A B
1	A	B
2	B	L
3	L	**M**

Inferred	
A	true
B	true
L	**true**
M	false
P	false
Q	false

Count	
$P \Rightarrow Q$	1
$L \wedge M \Rightarrow P$	**1**
$B \wedge L \Rightarrow M$	**0**
$A \wedge P \Rightarrow L$	1
$A \wedge B \Rightarrow L$	0

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$. A, B

No.	Reached	Queue
0		A B
1	A	B
2	B	L
3	L	M
4	M	**P**

Inferred	
A	true
B	true
L	true
M	**true**
P	false
Q	false

Count	
$P \Rightarrow Q$	1
$L \wedge M \Rightarrow P$	**0**
$B \wedge L \Rightarrow M$	0
$A \wedge P \Rightarrow L$	1
$A \wedge B \Rightarrow L$	0

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Queue
0		A B
1	A	B
2	B	L
3	L	M
4	M	P
5	P	**L Q**

Inferred	
A	true
B	true
L	true
M	true
P	**true**
Q	false

Count	
$P \Rightarrow Q$	**0**
$L \wedge M \Rightarrow P$	0
$B \wedge L \Rightarrow M$	0
$A \wedge P \Rightarrow L$	**0**
$A \wedge B \Rightarrow L$	0

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Queue
0		A B
1	A	B
2	B	L
3	L	M
4	M	P
5	P	L Q
6	L	Q

Inferred	
A	true
B	true
L	true
M	true
P	true
Q	false

Count	
$P \Rightarrow Q$	0
$L \wedge M \Rightarrow P$	0
$B \wedge L \Rightarrow M$	0
$A \wedge P \Rightarrow L$	0
$A \wedge B \Rightarrow L$	0

e.g. Forward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L, \quad A \wedge B \Rightarrow L, \quad A, \quad B$

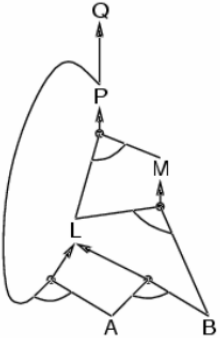
No.	Reached	Queue
0		A B
1	A	B
2	B	L
3	L	M
4	M	P
5	P	L Q
6	L	Q
7	**Q**	

Inferred	
A	true
B	true
L	true
M	true
P	true
Q	false

Count	
$P \Rightarrow Q$	0
$L \wedge M \Rightarrow P$	0
$B \wedge L \Rightarrow M$	0
$A \wedge P \Rightarrow L$	0
$A \wedge B \Rightarrow L$	0

e.g. Forward chaining

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



Backward chaining

Idea:

Check whether a particular fact Q is *true*

Backward chaining

Given a fact Q to be "proven",

1. See if Q is already in the **KB**. If so, return *true*.
2. Find all implications, I, whose conclusion "matches" Q.
3. Recursively establish the premises of all i in I via backward chaining.

e.g. Backward chaining

Goal Q

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Goal stack
0		Q

Inferred	
A	false
B	false
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$, A, B

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$

Inferred	
A	false
B	false
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$ ✔

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$
2	$P \Rightarrow Q$	$L \wedge M \Rightarrow P$

Inferred	
A	false
B	false
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$ ✔

$L \wedge M \Rightarrow P$ ✔

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$
2	$P \Rightarrow Q$	$L \wedge M \Rightarrow P$
3	$L \wedge M \Rightarrow P$	$A \wedge P \Rightarrow L, \quad B \wedge L \Rightarrow M$

Inferred	
A	false
B	false
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$ ✔

$L \wedge M \Rightarrow P$ ✔

$B \wedge L \Rightarrow M$ ✔

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L, \quad A, \quad B$

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$
2	$P \Rightarrow Q$	$L \wedge M \Rightarrow P$
3	$L \wedge M \Rightarrow P$	$A \wedge P \Rightarrow L, \quad B \wedge L \Rightarrow M$
4	$B \wedge L \Rightarrow M$	$A \wedge P \Rightarrow L, \quad B, \quad A \wedge B \Rightarrow L$

Inferred	
A	false
B	false
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$ ✔

$L \wedge M \Rightarrow P$ ✔

$B \wedge L \Rightarrow M$ ✔

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$ ✔, A, B

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$
2	$P \Rightarrow Q$	$L \wedge M \Rightarrow P$
3	$L \wedge M \Rightarrow P$	$A \wedge P \Rightarrow L,\ \ A \wedge B \Rightarrow L$
4	$B \wedge L \Rightarrow M$	$A \wedge P \Rightarrow L\ \ B,\ \ A \wedge B \Rightarrow L$
5	$A \wedge B \Rightarrow L$	$A \wedge P \Rightarrow L,\ \ B,\ \ A$

Inferred	
A	false
B	false
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$ ✔

$L \wedge M \Rightarrow P$ ✔

$B \wedge L \Rightarrow M$ ✔

$A \wedge P \Rightarrow L, \quad A \wedge B \Rightarrow L$ ✔, A ✔, B

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$
2	$P \Rightarrow Q$	$L \wedge M \Rightarrow P$
3	$L \wedge M \Rightarrow P$	$A \wedge P \Rightarrow L, \quad A \wedge B \Rightarrow L$
4	$B \wedge L \Rightarrow M$	$A \wedge P \Rightarrow L$
5	$A \wedge B \Rightarrow L$	$A \wedge P \Rightarrow L, \quad B, \quad A$
6	A	$A \wedge P \Rightarrow L, \quad B$

Inferred	
A	**true**
B	false
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$ ✔

$L \wedge M \Rightarrow P$ ✔

$B \wedge L \Rightarrow M$ ✔

$A \wedge P \Rightarrow L, \quad A \wedge B \Rightarrow L$ ✔, A ✔, B ✔

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$
2	$P \Rightarrow Q$	$L \wedge M \Rightarrow P$
3	$L \wedge M \Rightarrow P$	$A \wedge P \Rightarrow L, \quad A \wedge B \Rightarrow L$
4	$B \wedge L \Rightarrow M$	$A \wedge P \Rightarrow L$
5	$A \wedge B \Rightarrow L$	$A \wedge P \Rightarrow L, \quad B, \quad A$
6	A	$A \wedge P \Rightarrow L, \quad B$
7	B	$A \wedge P \Rightarrow L$

Inferred	
A	true
B	**true**
L	false
M	false
P	false
Q	false

e.g. Backward chaining

Goal Q ✔

$P \Rightarrow Q$ ✔

$L \wedge M \Rightarrow P$ ✔

$B \wedge L \Rightarrow M$ ✔, $A \wedge P \Rightarrow L$ ✔, $A \wedge B \Rightarrow L$ ✔, A ✔, B ✔

No.	Reached	Goal stack
0		Q
1	Q	$P \Rightarrow Q$
2	$P \Rightarrow Q$	$L \wedge M \Rightarrow P$
3	$L \wedge M \Rightarrow P$	$A \wedge P \Rightarrow L, \ A \wedge B \Rightarrow L$
4	$B \wedge L \Rightarrow M$	$A \wedge P \Rightarrow L$
5	$A \wedge B \Rightarrow L$	$A \wedge P \Rightarrow L, \ B, \ A$
6	A	$A \wedge P \Rightarrow L, \ B$
7	B	$A \wedge P \Rightarrow L$
8	$A \wedge P \Rightarrow L$	$\{\ \}$

Inferred	
A	true
B	true
L	**true**
M	**true**
P	**true**
Q	**true**

DPLL Algorithm

Davis-Putnam-Logemann-Loveland (DPLL)

The DPLL algorithm for **checking satisfiability** of a sentence in propositional logic.

- **Early termination**:
  - A **clause** is true if any literal is true.
  - A **sentence** is false if any clause is false.
- **Pure symbol heuristic**:
  - **Pure symbol**: always appears with the same "sign" in all clauses.
  - e.g. $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$. symbol A is pure; symbol B is pure; symbol C is impure.
  - **Make a pure symbol literal true**.
  - **Ignore clauses** that are already known to be **true** in the model constructed so far.
  - e.g. if the model contains $B = \mathit{false}$, then the clause $(\neg B \vee \neg C)$ is **true**, and in the remaining clauses C appears only as a positive literal; therefore C becomes **pure**.

DPLL Algorithm

- **Unit clause heuristic**:
  - **Unit clause**: only one literal in the clause.
  - The only literal in a unit clause must be **true**.
  - When a litteral assigned **false** by the model, the clause can simplifies by ignore that literal.
  - e.g. if the model contains $B = true$, then $(\neg B \vee \neg C)$ simplifies to $\neg C$, which is a unit clause.
  - Assigning one unit clause can create another unit clause.
  - e.g. when C is set to **false**, $(C \vee A)$ becomes a unit clause, causing **true** to be assigned to A.

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, { })

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})
```

e.g. DPLL

let $(C \vee D)$, $(C \vee \neg D)$, $(\neg D \vee A)$, $(A \vee B)$, $(\neg A \vee \neg B)$, Satisfy?

Solution:

Pure: (C): $C = true$

Clauses: $(\neg D \vee A)$, $(A \vee B)$, $(\neg A \vee \neg B)$

Pure: $(\neg D)$: $D = false$

Clauses: $(A \vee B)$, $(\neg A \vee \neg B)$

not pure or unit clause

$A = true$

Clauses: $(\neg B)$

Pure: $(\neg B)$: $B = false$

Model: $(A = true, B = false, C = true, D = false)$

First Order Logic

- Propositional logic deals with atomic facts (i.e. atomic, non-structured propositional symbols; usually finitely many)
- FOL brings structure to facts, which can be built from:

1. **Objects**: people, houses, numbers, theories, Ronald McDonald, colors ...
2. **Relations**: unary relations or properties: red, round, bogus, prime ..., or brother of, bigger than, inside, part of, has color, occurred after, owns, comes between ...
3. **Functions**: father of, best friend, third inning of, one more than, beginning of ...

Logics in general

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Formal languages and their ontological and epistemological commitments.

Syntax

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{Predicate} \mid \textit{Predicate}(\textit{Term},\ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} & \rightarrow & (\ \textit{Sentence}\ ) \\
& \mid & \neg\ \textit{Sentence} \\
& \mid & \textit{Sentence} \wedge \textit{Sentence} \\
& \mid & \textit{Sentence} \vee \textit{Sentence} \\
& \mid & \textit{Sentence} \ \Rightarrow \ \textit{Sentence} \\
& \mid & \textit{Sentence} \ \Leftrightarrow \ \textit{Sentence} \\
& \mid & \textit{Quantifier}\ \textit{Variable},\ldots\ \textit{Sentence} \\
\\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term},\ldots) \\
& \mid & \textit{Constant} \\
& \mid & \textit{Variable} \\
\\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\textit{Predicate} & \rightarrow & \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} & \rightarrow & \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\text{OPERATOR PRECEDENCE} & : & \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{array}
$$

Example

"Socrates is a human."

$$Human(Socrates)$$

"All humans are mortal."

$$\forall x\ Human(x) \Rightarrow Mortal(x)$$

"There exists a person who loves Socrates."

$$\exists x\ Loves(x, Socrates)$$

"Every person who loves a dog also loves a cat."

$$\forall x\ \forall y\ Loves(x, y) \land Dog(y) \Rightarrow Loves(x, z) \land Cat(z)$$

"There exists a person who loves all dogs."

$$\exists x\ \forall y\ Dog(y) \Rightarrow Loves(x, y)$$

"No person loves everyone."

$$\forall x\ \exists y\ \neg Loves(x, y)$$

Example

"The father of every person is a male."

$$\forall x \; Person(x) \Rightarrow Male(Father(x))$$

"The mother of every child is a female."

$$\forall x \; Child(x) \Rightarrow Female(Mother(x))$$

"The sum of any two numbers is a number."

$$\forall x \; \forall y \; Number(x) \wedge Number(y) \Rightarrow Number(Sum(x, y))$$

"Every set is a subset of itself."

$$\forall x \; Set(x) \Rightarrow Subset(x, x)$$

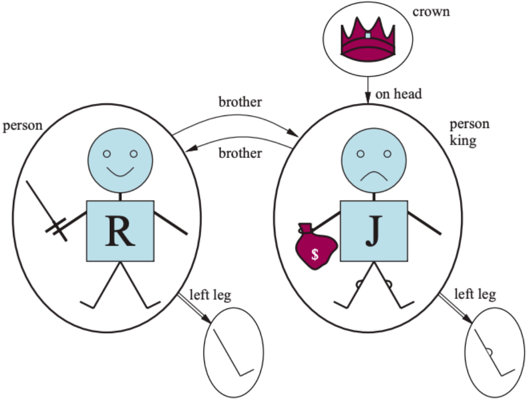
"The intersection of two sets is a set."

$$\forall x \; \forall y \; Set(x) \wedge Set(y) \Rightarrow Set(Intersect(x, y))$$

"The union of two sets is a set."

$$\forall x \; \forall y \; Set(x) \wedge Set(y) \Rightarrow Set(Union(x, y))$$

Model for FOL: Example



A model containing five objects, two binary relations (brother and on-head), three unary relations (person, king, and crown), and one unary function (left-leg).

Model for FOL

- **Object:** Things.
  Domain: Set of objects.
- **Relation:** Set of typles of objects.

$$\{\langle Richard\ the\ Lionheart, King\ John\rangle, \langle King\ John, Richard\ the\ Lionheart\rangle, ...\}$$

- **Function relations:** A mapping from a one-element tuple to an object.

  $\langle Richard\ the\ Lionheart\rangle \rightarrow Richard\text{'}s\ left\ leg$

  $\langle King\ John\rangle \rightarrow John\text{'}s\ left\ leg$

Symbols

Symbols stand for objects, relations, and functions.

3 kinds of symbol:

- **Constant symbols:** Stand for objects.
  e.g. $Richard, John$
- **Predicate symbols:** Stand for relations.
  e.g. $Brother, OnHead, Person, King, Crown$
- **Function symbols:** Stand for functions.
  e.g. $LeftLeg$

Interpretations:

- Every model must provide the information required to determine if any given sentence is true or false.
- Each model includes an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

e.g.

- *Richard* refers to **Richard the Lionheart** and *John* refers to the **evil King John**.
- *Brother* refers to the brotherhood relation.
- *OnHead* is a relation that holds between the **crown** and **King John**.
- *Person*, *King*, and *Crown* are unary relations that identify persons, kings, and crowns.
- *LeftLeg* refers to the “left leg” function.

Term

Term is a logical expression that refers to an object.

Complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol.

e.g.

$f(t_1, t_2, ..., t_n)$

$LeftLeg(John)$ refers to King John's left leg.

Atomic Sentences

Atomic Sentence $\rightarrow$ *Predicate* | *Predicate*(*Term*, ...) | *Term* $=$ *Term*

e.g.

$Brother(Richard, John)$

$Married(Father(Richard), Mother(John))$

Complex Sentences

$$
\begin{aligned}
\textit{Complex Sentence} \;\rightarrow\;& (\ \textit{Sentence}\ ) \\
\mid\;& \neg \textit{Sentence} \\
\mid\;& \textit{Sentence} \;\wedge\; \textit{Sentence} \\
\mid\;& \textit{Sentence} \;\vee\; \textit{Sentence} \\
\mid\;& \textit{Sentence} \;\Rightarrow\; \textit{Sentence} \\
\mid\;& \textit{Sentence} \;\Leftrightarrow\; \textit{Sentence} \\
\mid\;& \textit{Quantifier Variable}, \ldots\ \textit{Sentence}
\end{aligned}
$$

e.g.

$$\neg Brother(LeftLeg(Richard), John)$$

$$Brother(Richard, John) \wedge Brother(John, Richard)$$

$$King(Richard) \vee King(John)$$

$$\neg King(Richard) \Rightarrow King(John)$$

Quantifiers

Quantifier $\rightarrow \ \forall \mid \exists$

Universal quantification ($\forall$)

$\forall x\ P$, where P is any logical sentence, says that P is true for every object x.

"All Kings are persons":

$\forall x King(x) \Rightarrow Person(x)$: "For all x, if x is a King, then x is a person."

We can extend in five ways:

$x \rightarrow Richard\ the\ Lionheart$,
$x \rightarrow King\ John$,
$x \rightarrow Richard's\ left\ leg$,
$x \rightarrow John's\ left\ leg$,
$x \rightarrow the\ crown$.

Quantifiers

$\forall x King(x) \Rightarrow Person(x)$ is true in the original model if the sentence $King(x) \Rightarrow Person(x)$ is true under each of the five extended interpretations

Richard the Lionheart is a king $\rightarrow$ Richard the Lionheart is a person.

King John is a king $\rightarrow$ King John is a person.

Richard's left leg is a king $\rightarrow$ Richard's left leg is a person.

John's left leg is a king $\rightarrow$ John's left leg is a person.

The crown is a king $\rightarrow$ the crown is a person.

Quantifiers

Existential quantification ($\exists$)

Existential quantifier can make a statement about *some* object without naming it.

e.g.

$\exists x\ Crown(x) \wedge OnHead(x, John)$: "There exists x such that ..." or "For some x ..."

$\exists x\ P$: P is true for at least one object x.

at least one of the following is true:

- Richard the Lionheart is a crown $\wedge$ Richard the Lionheart is on John's head;
- King John is a crown $\wedge$ King John is on John's head;
- Richard's left leg is a crown $\wedge$ Richard's left leg is on John's head;
- John's left leg is a crown $\wedge$ John's left leg is on John's head;
- **The crown is a crown $\wedge$ the crown is on John's head** : $true$

Quantifiers

Nested quantifiers

e.g.

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

$$\forall x, y\ Sibling(x, y) \Leftrightarrow Sibling(y, x)$$

$$\forall x\ \exists y\ Loves(x, y)$$

$$\exists y\ \forall x\ Loves(x, y)$$

$$\forall x\ (Crown(x) \vee (\exists x\ Brother(Richard, x)))$$

Quantifiers

Connections between $\forall$ and $\exists$

De Morgan's rules:

$$
\begin{aligned}
\neg\exists x\ P &\equiv \forall x\ \neg P \\
\neg\forall x\ P &\equiv \exists x\ \neg P \\
\forall x\ P &\equiv \neg\exists x\ \neg P \\
\exists x\ P &\equiv \neg\forall x\ \neg P
\end{aligned}
\qquad\qquad
\begin{aligned}
\neg(P \lor Q) &\equiv \neg P \land \neg Q \\
\neg(P \land Q) &\equiv \neg P \lor \neg Q \\
P \land Q &\equiv \neg(\neg P \lor \neg Q) \\
P \lor Q &\equiv \neg(\neg P \land \neg Q)\,.
\end{aligned}
$$

e.g.

$$\forall x\ \neg Likes(x, Parsnips) \quad \equiv \quad \neg\exists x\ Likes(x, Parsnips)$$

$$\forall x\ Likes(x, IceCream) \quad \equiv \quad \neg\exists x\ \neg Likes(x, IceCream)$$

Equality

Equality symbol $=$

e.g.

$Father(John) = Henry$

To say that Richard has at least two brothers:

$\exists x, y \; Brother(x, Richard) \land Brother(y, Richard) \land \neg(x = y)$

Database semantics

Suppose that we believe that Richard has two brothers, John and Geoffrey:

$Brother(John, Richard) \land Brother(Geoffrey, Richard)$,

We need to add $John \neq Geoffrey$

and $John$ may have more brothers besides $John$ and $Geoffrey$

Thus, the correct translation of "Richard's brothers are John and Geoffrey" is as follows:

$$Brother(John, Richard) \land Brother(Geoffrey, Richard) \land John \neq Geoffrey$$
$$\land \forall x\ Brother(x, Richard) \Rightarrow (x = John \lor x = Geoffrey)$$

We call this **database semantics** to distinguish it from the **standard semantics** of first-order logic.

Assertions and Queries in FOL

Assertions: Sentences are added to a knowledge base using $TELL$

e.g.

We can assert that John is a king, Richard is a person, and all kings are persons:

$TELL(KB, King(John))$

$TELL(KB, Person(Richard))$

$TELL(KB, \forall x \ King(x) \Rightarrow Person(x))$

Queries or **Goals**: Questions asked with ASK to a knowledge base.

e.g.

$ASK(KB, Person(John))$

$ASKVARS(KB, Person(x))$

Family Domain

Ojects in **Family Domain** are people

Unary predicates: $Male$ and $Female$

Binary predicates: $Parent, Sibling, Brother, Sister, Child, Daughter, Son, Spouse$
$Wife, Husband, Grandparent, Grandchild, Cousin, Aunt$, and $Uncle$.

Function: $Mother, Father$

e.g. **Axiom of the family domain:**

one's mother is one's parent who is female:

$$\forall m, c \; Mother(c) = m \Leftrightarrow Female(m) \wedge Parent(m, c)$$

One's husband is one's male spouse:

$$\forall w, h \; Husband(h, w) \Leftrightarrow Male(h) \wedge Spouse(h, w)$$

Family Domain

Parent and child are inverse relations:

$$\forall p, c \; Parent(p, c) \Leftrightarrow Child(c, p)$$

A grandparent is a parent of one's parent:

$$\forall g, c \; Grandparent(g, c) \Leftrightarrow \exists p \; Parent(g, p) \land Parent(p, c)$$

A sibling is another child of one's parent:

$$\forall x, y \; Sibling(x, y) \Leftrightarrow x \neq y \land \exists p \; Parent(p, x) \land Parent(p, y)$$

Numbers

Natural numbers: predicate $NatNum$, function $S(successor)$

Peano axioms :

Natural numbers are defined recursively:

$$NatNum(0)$$

$$\forall n\ NatNum(n) \Rightarrow NatNum(S(n))$$

Constrain the successor function:

$$\forall n\ 0 \neq S(n)$$

$$\forall m, n\ m \neg n \Rightarrow S(m) \neq S(n)$$

Define addition in terms of the successor function:

$$\forall m\ NatNum(m) \Rightarrow +(0, m) = m$$

$$\forall m, n\ NatNum(m) \wedge NatNum(n) \Rightarrow +(S(m), n) = S(+(m, n))$$

Sets

Empty Set: $\{\ \}$

There is one **unary predicate** Set, which is true of sets.

Binary predicates:

$x \in s$, $s_1 \subseteq s_2$

Binary functions:

$s_1 \cap s_2$, $s_1 \cup s_2$, Add

Sets

Axiom of Set:

1. The only sets are the empty set and those made by adding something to a set:
   $\forall s\ Set(s) \Leftrightarrow (s = \{\ \}) \vee (\exists x, s_2\ Set(s_2) \wedge s = Add(x, s_2))$
2. The empty set has no elements added into it. In other words, there is no way to decompose $\{\ \}$ into a smaller set and an element:
   $\neg\exists x, s\ Add(x, s) = \{\ \}$
3. Adding an element already in the set has no effect:
   $\forall x, s\ x \in s \Leftrightarrow s = Add(x, s)$
4. The only members of a set are the elements that were added into it. We express this recursively, saying that x is a member of s if and only if s is equal to some element y added to some set s_2, where either y is the same as x or x is a member of s_2:
   $\forall x, s\ x \in s \Leftrightarrow \exists y, s_2\ (s = Add(y, s_2) \wedge (x = y \vee x \in s_2))$

Axiom of Set:

5. A set is a subset of another set if and only if all of the first set's members are members of the second set:

   $\forall s_1, s_2 \; s_1 \subseteq s_2 \Leftrightarrow (\forall x \; x \in s_1 \Rightarrow x \in s_2)$

6. Two sets are equal if and only if each is a subset of the other:

   $\forall s_1, s_2 \; (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \land s_2 \subseteq s_1)$

7. An object is in the intersection of two sets if and only if it is a member of both sets:

   $\forall x, s_1, s_2 \; x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \land x \in s_2)$

8. An object is in the union of two sets if and only if it is a member of either set:

   $\forall x, s_1, s_2 \; x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \lor x \in s_2)$

Lists

Lists are similar to sets.

Constant list with no elements:

Nil

Functions:

$Cons, Append, First$, and $Rest$

Predicate:

$Find$, $List$

The Wumpus World

Percept vector with five elements in Wumpus world.
e.g. One percept vector:
$Percept([Stench, Breeze, Glitter, None, None], 5)$
$Percept$ is a binary predicate, and $Stench$ and so on are constants placed in a list.

Actions in the wumpus world can be represented by logical terms:
$Turn(Right), Turn(Left), Forward, Shoot, Grab, Climb$

To determine which is best, the agent program executes the query
$ASKVARS(KB, BestAction(a, 5))$, : returns a list such as $\{a/Grab\ \}$
e.g. current state:
$\forall t, s, g, w, c\ Percept([s, Breeze, g, w, c], t) \Rightarrow Breeze(t)$
$forallt, s, g, w, c\ Percept([s, None, g, w, c], t) \Rightarrow \neg Breeze(t)$
$\forall t, s, b, w, c\ Percept([s, b, Glitter, w, c], t) \Rightarrow Glitter(t)$
$\forall t, s, b, w, c\ Percept([s, b, None, w, c], t) \Rightarrow \neg Glitter(t)$
...

The Wumpus World

Quantification over time t

$\forall t\ Glitter(t) \Rightarrow BestAction(Grab, t)$

Adjacency of any two squares can be defined as

$\forall x, y, a, b\ Adjacent([x, y], [a, b]) \Leftrightarrow$

$(x = a \land (y = b - 1 \lor y = b + 1)) \lor (y = b \land (x = a - 1 \lor x = a + 1))$

The agent's location changes over time

$At(Agent, s, t)$: the agent is at square s at time t

We can fix the wumpus to a specific location forever

$\forall t\ At(Wumpus, [1, 3], t)$

The Wumpus World

We can say that objects can be at only one location at a time

$$\forall x, s_1, s_2, t\ At(x, s_1, t) \wedge At(x, s_2, t) \Rightarrow s_1 = s_2$$

Given its current location, the agent can infer properties of the square from properties of its current percept

$$\forall s, t\ At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$$

$$\forall s\ Breezy(s) \Leftrightarrow \exists r\ Adjacent(r, s) \wedge Pit(r)$$

$$\forall t\ HaveArrow(t + 1) \Leftrightarrow (HaveArrow(t) \wedge \neg Action(Shoot, t))$$

Substitution

- **Substitution** is a fundamental operation performed on terms and formulas:

  $\theta = \{\nu/g\}$

Substitution θ substitute a variable ν to the **ground term** (a term without variables) g

- **Applying the substitution**

  $Subst(\{\nu/g\}, \alpha)$

  denote the result of applying the substitution θ to the sentence α

e.g.

$\theta = \{x/John\}$

$\alpha = King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

Universal Instantiation (UI)

Universal ($\forall$) Instantiation: Convert the first-order knowledge base to propositional logic and use propositional inference.

$$\forall x \; King(x) \land Greedy(x) \Rightarrow Evil(x)$$

From that we can infer any of the following sentences:

$$King(John) \land Greedy(John) \Rightarrow Evil(John)$$

$$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$$

$$King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John))$$

...

Universal Instantiation (UI)

$$\frac{\forall \nu \ \alpha}{Subst(\{\nu/g\}, \alpha)}$$

for any variable ν and ground term g

e.g.

$\forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x)$ ($\nu = x$ *and* $\alpha = King(x) \land Greedy(x) \Rightarrow Evil(x)$)

we obtain:

$Subst(\{\nu/g\}, \alpha) = \{x/John\}, \{x/Richard\}$ and $\{x/Father(John)\}$

Existential Instantiation (EI)

Existential (∃) Instantiation replaces an existentially quantified variable with a single new constant symbol. The formal statement is as follows: for any sentence α, variable ν, and constant symbol k that does not appear elsewhere in the knowledge base,

$$\frac{\exists \nu \ \alpha}{Subst(\{\nu/k\}, \alpha)}$$ for any variable ν constant symbol k

e.g.

$\exists x \ Crown(x) \land OnHead(x, John)$

we can infer the sentence:

$Crown(C_1) \land OnHead(C_1, John)$

as long as C_1 does not appear elsewhere in the knowledge base.

Reduction to propositional inference

Convert any first-order knowledge base into a propositional knowledge base.

- An existentially quantified sentence can be replaced by one instantiation.
- A universally quantified sentence can be replaced by the set of all possible instantiations.

e.g.

Suppose the KB contains just the following:

$\forall x\ King(x) \land Greedy(x) \Rightarrow Evil(x)$

$King(John)$

$Greedy(John)$

$Brother(Richard, John)$

Reduction to propositional inference

Apply Universal Instantiation to the first sentence using all possible substitutions:

$\{x/John\}$ and $\{x/Richard\}$

We obtain

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$.

$King(John)$

$Greedy(John)$

$Brother(Richard, John)$

Next replace ground atomic sentences: $JohnIsKing,\ JohnIsGreedy$ to use propositional logic inference we obtain conclusions: $JohnIsEvil$

The new **KB** is **Propositionalized**: proposition symbols are: $JohnIsEvil$

Unification

Unification algorithm takes two sentences and returns a unifier for them (a substitution) if one exists:

$Unify(p, q) = \theta$ where $Subst(\theta, p) = Subst(\theta, q)$.

e.g.

- We have the inference rule:

  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

- We have facts that (partially) match the precondition

  $King(John)$

  $\forall y \; Greedy(y)$

- We need to match them up with substitutions: $\theta = \{x/John, y/John\}$

  Unification

  Generalized Modus Ponens

Unification

$Unify(p,q) = \theta$ where $Subst(\theta,p) = Subst(\theta,q)$

p	q	$Unify(p,q) = \theta$
$Knows(John,x)$	$Knows(John,Jane)$	$\{x/Jane\}$
$Knows(John,x)$	$Knows(y,Mary)$	$\{x/Mary,y/John\}$
$Knows(John,x)$	$Knows(y,Mother(y))$	$\{y/John,x/Mother(John)\}$
$Knows(John,x)$	$Knows(x,Mary)$	$fail$

Standardizing apart: to avoid $fail$

$$Unify(Knows(John,x),Knows(x_{17},Elizabeth)) = \{x/Elizabeth,x_{17}/John\}$$

Unification Algorithm

- It is used to determine whether two expressions in FOL can be made identical by finding a substitution for their variables.
- If such a substitution exists, the algorithm outputs it; otherwise, it reports that unification failed.

Unification Algorithm

function UNIFY(x, y, θ=*empty*) **returns** a substitution to make x and y identical, or *failure*
    if θ = *failure* **then return** *failure*
    else if $x = y$ **then return** θ
    else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
    else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
    else if COMPOUND?(x) **and** COMPOUND?(y) **then**
        return UNIFY(ARGS(x), ARGS(y), UNIFY(OP(x), OP(y), θ))
    else if LIST?(x) **and** LIST?(y) **then**
        return UNIFY(REST(x), REST(y), UNIFY(FIRST(x), FIRST(y), θ))
    else return *failure*

function UNIFY-VAR(var, x, θ) **returns** a substitution
    if $\{var/val\} \in \theta$ for some *val* **then return** UNIFY(val, x, θ)
    else if $\{x/val\} \in \theta$ for some *val* **then return** UNIFY(var, val, θ)
    else if OCCUR-CHECK?(var, x) **then return** *failure*
    else return add $\{var/x\}$ to θ

Explanation of Unify Function

The **Unify** function attempts to unify two terms x and y with an initial substitution θ (default is an empty substitution if not provided).

1. **Base Case: Failure Check**

   if $\theta = failure$ then return $failure$

   ◦ if the substitution θ already contains a $failure$, return $failure$ immediately. This short-circuits the recursion if any prior steps have failed.

2. **Case 1: Terms are Identical**

   else if $x = y$ then return θ

   ◦ if x and y are identical, no additional substitution is needed, so return the current substitution θ.

Explanation of UNIFY Function

3. **Case 2: x is a Variable**

   else if $VARIABLE?(x)$ then return $UNIFY\text{-}VAR(x, y, \theta)$

   ◦ if x is a variable, call $UNIFY\text{-}VAR(x, y, \theta)$ to try and unify x with y while respecting θ.

4. **Case 3: y is a Variable**

   else if $VARIABLE?(y)$ then return $UNIFY\text{-}VAR(y, x, \theta)$

   ◦ if y is a variable, call $UNIFY\text{-}VAR(y, x, \theta)$ to try and unify y with x while respecting θ.

Explanation of UNIFY Function

5. **Case 4: Both x and y are Compound Expressions**

   else if $COMPOUND?(x)$ and $COMPOUND?(y)$ then

   return $Unify(ARGS(x), ARGS(y), Unify(OP(x), OP(y), \theta))$

   ◦ if x and y are compound expressions, recursively unify their operators ($OP(x)$ and $OP(y)$) and arguments ($ARGS(x)$ and $ARGS(y)$) under the current substitution θ.

6. **Case 5: Both x and y are Lists**

   else if $LIST?(x)$ and $LIST?(y)$ then

   return $Unify(REST(x), REST(y), Unify(FIRST(x), FIRST(y), \theta))$

   ◦ if x and y are lists, recursively unify the first elements ($FIRST(x)$ and $FIRST(y)$) and the remaining parts ($REST(x)$ and $REST(y)$) under the current substitution θ.

Explanation of UNIFY Function

7. Case 6: Failure

   else return $failure$

   - if none of the above cases apply, return $failure$.

Explanation of UNIFY-VAR Function

UNIFY-VAR unify a variable var with a term x under a given substitution θ.

1. **Check for Existing Substitutions for var**

   if $\{var/val\} \in \theta$ for some val then return $Unify(val, x, \theta)$

   ∘ if var already has a binding (e.g., $\{var/val\}$ exists in θ), unify val with x to ensure consistency.

2. **Check for Existing Substitutions for x**

   else if $\{x/val\} \in \theta$ for some val then return $Unify(var, val, \theta)$

   ∘ if x is already bound to some value (e.g. $\{x/val\}$ exists in θ), unify var with that value to maintain consistency.

Explanation of UNIFY-VAR Function

3. **Occurrence Check**

   else if $OCCUR\text{-}CHECK\ ?(var, x)$ then return $failure$

   ◦ Perform an occurrence check to ensure that var does not appear within x, which would create an infinite loop. If it does, return $failure$.

4. **Add New Binding to Substitution**

   else return add $\{var/x\}$ to θ

   ◦ if none of the above conditions apply, add the substitution $\{var/x\}$ to θ and return the updated substitution.

Example 1

e.g. Unify two expressions:

- $x = Loves(John, y)$
- $y = Loves(z, Mary)$

Step-by-Step Execution of the Unification Algorithm

1. **Initial Call:** $Unify(Loves(John, y), Loves(z, Mary), \varnothing)$

   - Start with an empty substitution set $\theta = \varnothing$.

   - Since $Loves(John, y)$ and $Loves(z, Mary)$ are compound terms, we proceed to unify their operators and arguments.

2. **Unify Operators:** $Unify(Loves, Loves, \varnothing)$

   - The operators ($Loves$ in both terms) are identical, so we continue without adding any substitutions.

Example 1

3. **Unify Arguments:** $Unify([John, y], [z, Mary], \varnothing)$

   • Move to the arguments of the $Loves$ expressions, which are $[John, y]$ and $[z, Mary]$.

   • This is a list unification case, so we proceed by unifying each corresponding pair of terms in these lists.

4. **Unify First Pair:** $Unify(John, z, \varnothing)$

   - Since John is a constant and z is a variable, we use $UNIFY\text{-}VAR$ to unify them.
   - **Call to** $Unify\text{-}VAR(z, John, \varnothing)$:
     - z is not yet in any substitution, so we add $\{z/John\}$ to θ.
   - Now, $\theta = \{z \rightarrow John\}$.

Example 1

5. **Unify Second Pair:** $Unify(y, Mary, \{z \to John\})$

• Here, y is a variable and $Mary$ is a constant. Again, we call $UNIFY\text{-}VAR$ to unify them.

• **Call to** $UNIFY\text{-}VAR(y, Mary, \{z \to John\})$:

  • y is not yet in any substitution, so we add $\{y/Mary\}$ to θ.

• Now, $\theta = \{\mathit{z} \to \mathit{John}, \mathit{y} \to \mathit{Mary}\}$.

Final Substitution

$$\theta = \{z \to John, y \to Mary\}$$

This substitution set makes the expressions $Loves(John, y)$ and $Loves(z, Mary)$ identical by replacing z with $John$ and y with $Mary$.

Example 1

Verification

Applying θ to both terms:

- **Substitute in** $x = Loves(John, y)$:

  $Loves(John, y) \to Loves(John, Mary)$ (using $y \to Mary$).

- **Substitute in** $y = Loves(z, Mary)$:

  $Loves(z, Mary) \to Loves(John, Mary)$ (using $z \to John$).

After applying the substitution θ, both expressions become $Loves(John, Mary)$, confirming that the unification was successful.

Summary

$\{z \to John, y \to Mary\}$ successfully unifies $Loves(John, y)$ with $Loves(z, Mary)$, making them identical.

Example 2

e.g. Unify the following two terms:

- $x = Knows(f(Alice), y)$
- $y = Knows(z, h(Bob))$

Step-by-Step Execution of the Unification Algorithm

1. **Initial Call:** $Unify(Knows(f(Alice), y), Knows(z, h(Bob)), \varnothing)$

   - Start with an empty substitution set $\theta = \varnothing$.

   - Both terms are compound terms with the same top-level operator *Knows*, so we proceed to unify their arguments.

Example 2

2. **Unify Operators:** $Unify(Knows, Knows, \varnothing)$

   - The operators are identical ($Knows$ in both terms), so no substitution is needed. We proceed with unifying their arguments:

$$Unify([f(Alice), y], [z, h(Bob)], \varnothing)$$

3. **Unify First Pair of Arguments:** $Unify(f(Alice), z, \varnothing)$

   - Here, $f(Alice)$ is a compound term, and z is a variable.
   - According to the algorithm, if one term is a variable, we can try to unify by substituting that variable.
   - **Call to** $UNIFY\text{-}VAR(z, f(Alice), \varnothing)$:
     - z is not yet in any substitution, so we add $\{z/f(Alice)\}$ to θ.
   - Now, $\theta = \{z \rightarrow f(Alice)\}$.

Example 2

4. **Unify Second Pair of Arguments:** $Unify(y, h(Bob), \{z \rightarrow f(Alice)\})$
   - Here, y is a variable and $h(Bob)$ is a compound term.
   - **Call to** $UNIFY\text{-}VAR(y, h(Bob), \{z \rightarrow f(Alice)\})$:
     - y is not yet in any substitution, so we add $\{y/h(Bob)\}$ to θ.
   - Now, $\theta = \{z \rightarrow f(Alice), y \rightarrow h(Bob)\}$.

Final Substitution Set

After processing all pairs of arguments, the substitution set we get is:

$$\theta = \{z \rightarrow f(Alice), y \rightarrow h(Bob)\}$$

Example 2

Verification

Let's verify by applying the substitution θ to both expressions:

- **Substitute in** $x = Knows(f(Alice), y)$:

  $Knows(f(Alice), y) \rightarrow Knows(f(Alice), h(Bob))$ (using $y \rightarrow h(Bob)$)

- **Substitute in** $y = Knows(z, h(Bob))$:

  $Knows(z, h(Bob)) \rightarrow Knows(f(Alice), h(Bob))$ (using $z \rightarrow f(Alice)$)

Summary

The final substitution $\{z \rightarrow f(Alice), y \rightarrow h(Bob)\}$

Propositional logic - Modus Ponens

Modus Ponens:

Give α and $\alpha \Rightarrow \beta$, then β.

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

e.g.

Give $(WumpusAhead \wedge WumpusAlive)$

and $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$

then $Shoot$.

Generalized Modus Ponens

We have rule:

$greedy\ kings$ are $evil$,
find some x such that x is a $king$ and x is $greedy$,
we can infer that this x is $evil$.

$King(John)$ and $Greedy(John)$

So we have substitution $\theta = x/John$ solves the query $Evil(x)$.

Now suppose that instead of knowing $Greedy(John)$, we know that everyone is greedy:

$\forall y\ Greedy(y)$

$King(x)$ and $Greedy(x)$
$King(John)$ and $Greedy(y)$

Applying the substitution $\theta = \{x/John, y/John\}$

Generalized Modus Ponens

Atomic sentences $p_i, p_i{}'$, and q,

where:

there is a substitution θ such that $Subst(\theta, p_i{}') = Subst(\theta, p_i)$, for all i,

$$\frac{p_1{}', p_2{}', ..., p_n{}', (p_1 \land p_2 \land ... \land p_n \Rightarrow q)}{Subst(\theta, q)}$$

e.g.

$p_1{}'$ *is* $King(John)$ $\quad$ p_1 *is* $King(x)$

$p_2{}'$ *is* $Greedy(y)$ $\quad$ p_2 *is* $Greedy(x)$

θ *is* $\{x/John, y/John\}$ $\quad$ q *is* $Evil(x)$

$$Subst(\theta, q) = Evil(John)$$

Generalized Modus Ponens is a lifted version of Modus Ponens - it raises Modus Ponens from ground (variable-free) propositional logic to first-order logic.

Forward Chaining

Propositional Logic: We have forward-chaining algorithm for knowledge bases of propositional definite clauses.

Conjunction: $\wedge$ Disjunction: $\vee$

$$p_1 \wedge p_2 \wedge \ldots \wedge p_k \Rightarrow q$$

$$\equiv \neg(p_1 \wedge p_2 \wedge \ldots \wedge p_k) \vee q$$

$\equiv \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_k \vee q$: disjunction with only q is positive literal.

$$Definite\ Clause\ Form\ :\ Fact \mid (Symbol_1 \wedge \cdots \wedge Symbol_k) \Rightarrow Symbol$$

Forward Chaining

In FOL inference we also using definite clauses.

First-order definite clauses:

- Disjunctions of literals of which *exactly one is positive*
- Existential quantifiers are not allowed
- If x in a definite clause, that means there is an implicit $\forall x$ quantifier
- A typical first-order definite clause looks like this:

  $King(x) \land Greedy(x) \Rightarrow Evil(x)$
  $King(John)$
  $Greedy(y)$

Forward Chaining Algorithm

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  inputs: KB, the knowledge base, a set of first-order definite clauses
          α, the query, an atomic sentence

  while true do
     new ← { }      // The set of new sentences inferred on each iteration
     for each rule in KB do
        (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
        for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ... ∧ p'n)
                    for some p'1, ..., p'n in KB
            q' ← SUBST(θ, q)
            if q' does not unify with some sentence already in KB or new then
               add q' to new
               φ ← UNIFY(q', α)
               if φ is not failure then return φ
     if new = { } then return false
     add new to KB
```

Forward Chaining Algorithm

- *FOL-FC-ASK*(KB, α): Answer a query α by repeatedly applying rules from a knowledge base (*KB*) to infer new facts until the query is derived or no further inferences can be made.
- $STANDARDIZE\text{-}VARIABLES(rule)$: Replaces all variables in its arguments with new ones that have not been used before.
- $Subst(\theta, expression)$: Applies substitution θ to an expression.
- $Unify(expression_1, expression_2)$: Checks if two expressions can unify, returning the substitution if they do.

Explanation of the Algorithm

1. **Initialize the New Inferences Set:**

   $new \leftarrow \{\ \}$

   - new will hold all new facts inferred in each iteration.

2. **Loop Through Each Rule in KB:**

   for each *rule* in *KB* do

   $(p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow STANDARDIZE\text{-}VARIABLES(rule)$

   - For each rule in the knowledge base, standardize the variables to prevent conflicts with variables in other rules. This is necessary because variables might have different bindings across different rules.

   - Here, each rule has the form $p_1 \land p_2 \land \ldots \land p_n \Rightarrow q$, where $p_1, p_2, \ldots, p_n$ are the premises and q is the conclusion.

Explanation of the Algorithm

3. **Check if Premises Match Facts in KB:**

   for each θ such that $Subst(\theta, p_1 \wedge ... \wedge p_n) = Subst(\theta, p_1{}' \wedge ... \wedge p_n{}')$

   • For each possible substitution θ, try to match the premises $p_1, ..., p_n$ with some facts already in the knowledge base. If such a substitution exists, it means the premises hold under θ.

4. **Infer New Fact:**

   $q' \leftarrow Subst(\theta, q)$

   • If the premises match, apply the substitution θ to the conclusion q to infer a new fact q'.

Explanation of the Algorithm

5. **Check for Redundancy:**

   if $q^{'}$ does not unify with some sentence already in KB or new then

   add $q^{'}$ to new

   • Before adding $q^{'}$ to the set of new facts, check that it isn't already in the KB or in new.

   This avoids redundant inferences and prevents infinite loops.

6. **Check if Inferred Fact Answers the Query:**

   $\phi \leftarrow Unify(q^{'}, \alpha)$

   if ϕ is not $failure$ then return ϕ

   • If $q^{'}$ unifies with the query α, return the substitution ϕ that satisfies α with $q^{'}$ (i.e., the answer to the query).

Explanation of the Algorithm

7. **Add New Inferences to KB:**

if $new = \{\ \}$ then return $false$

add new to *KB*

- If no new inferences were made (i.e., new is empty), and α hasn't been derived, return $false$, indicating that the query cannot be answered with the current knowledge base.

- Otherwise, add all new inferences in new to the *KB* and repeat the process.

Summary of *FOL-FC-ASK*

Repeatedly applies rules in the knowledge base to infer new facts until either:

- The query α is derived (success).

- No new facts can be inferred, meaning the query is not entailed by the knowledge base (failure).

Example 1

e.g. The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

1. "... it is a crime for an American to sell weapons to hostile nations":

Crime rule:

$$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$$

Example 1

2. "Nono ... has some missiles":

   $\exists x\ (Owns(Nono, x) \land Missile(x))$

   - Using existential instantiation, introduce a constant M_1 for a specific missile:

   Fact: $Owns(Nono, M_1)$
   Fact: $Missile(M_1)$

3. "All of its missiles were sold to it by Colonel West":

Missile Ownership Rule:
$Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

4. We will also need to know that missiles are weapons:

Missile-to-Weapon Rule:
$Missile(x) \Rightarrow Weapon(x)$

Example 1

5. and we must know that an enemy of America counts as "hostile":

Hostility Rule:

$Enemy(x, America) \Rightarrow Hostile(x)$

6. "West, who is American ...":

Fact:

$American(West)$

7. "The country Nono, an enemy of America ...":

Fact:

$Enemy(Nono, America)$

Determine if Colonel West is a criminal: $Criminal(West)$

Example 1

Datalog knowledge base (KB):

Rule:

- $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
- $Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- $Missile(x) \Rightarrow Weapon(x)$
- $Enemy(x, America) \Rightarrow Hostile(x)$

Fact:

- $Missile(M_1)$
- $Owns(Nono, M_1)$
- $American(West)$
- $Enemy(Nono, America)$

Query: Determine if Colonel West is a criminal: $Criminal(West)$

Example 1

Forward Chaining Execution

Step 1: Initialization

- Set $new = \{\ \}$ to track any newly inferred facts.

Step 2: First Iteration - Apply Rules to Known Facts

1. **Apply the Hostility Rule (5):**
    - **Rule:** $Enemy(x, America) \Rightarrow Hostile(x)$
    - **Match:** We have $Enemy(Nono, America)$
    - **Substitute:** $x = Nono$
    - **Inference:** $Hostile(Nono)$
    - **Add:** $Hostile(Nono)$ to new

Example 1

2. **Apply the Missile-to-Weapon Rule (4):**
   - **Rule:** $Missile(x) \Rightarrow Weapon(x)$
   - **Match:** We have $Missile(M_1)$
   - **Substitute:** $x = M_1$
   - **Inference:** $Weapon(M_1)$
   - **Add:** $Weapon(M_1)$ to new

Example 1

3. **Apply the Missile Ownership Rule (3):**
   - **Rule:** $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
   - **Match:** We have both $Missile(M_1)$ and $Owns(Nono, M_1)$
   - **Substitute:** $x = M_1$
   - **Inference:** $Sells(West, M_1, Nono)$
   - **Add:** $Sells(West, M_1, Nono)$ to new

Example 1

Step 3: Add new to KB and Repeat if Necessary

- Add all sentences from new to KB $Hostile(Nono), Weapon(_1)$, and $Sells(West, M_1, Nono)$

- If new is empty, return $false$ (no conclusion). Since new is not empty, proceed to the next iteration.

Example 1

Step 4: Second Iteration - Apply Crime Rule (1)

1. **Apply the Crime Rule (1):**
   - **Rule:** $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
   - **Match:** We now have:
     - $American(West)$ (from Fact 6)
     - $Weapon(M1)$ (inferred in Step 2)
     - $Sells(West, M_1, Nono)$ (inferred in Step 2)
     - $Hostile(Nono)$ (inferred in Step 2)
   - **Substitute:** $x = West, y = M_1, z = Nono$
   - **Inference:** $Criminal(West)$
   - **Add:** $Criminal(West)$ to new

Example 1

Step 5: Conclusion

- **Check if the query is answered:** $Criminal(West)$ matches the query.
- **Return** $Criminal(West)$ as the answer.

Example 1 diagram

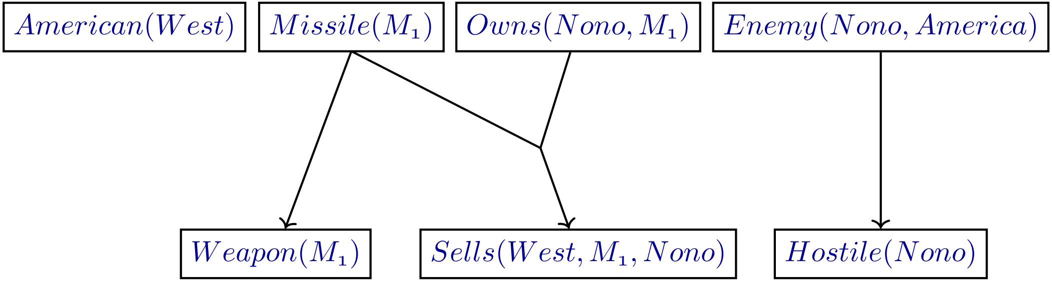
Fact:

$\boxed{American(West)}$ $\boxed{Missile(M_1)}$ $\boxed{Owns(Nono, M_1)}$ $\boxed{Enemy(Nono, America)}$

Example 1 diagram

Rule:

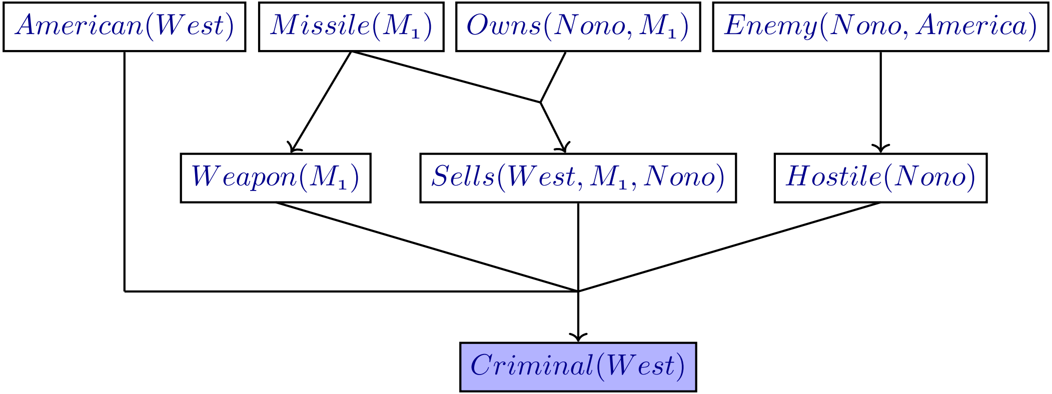
- $Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- $Missile(x) \Rightarrow Weapon(x)$
- $Enemy(x, America) \Rightarrow Hostile(x)$



Example 1 diagram

Rule:

- $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$



Example 2

e.g. **Datalog Knowledge Base (KB)**

Assume the knowledge base KB has the following rules and facts:

1. $Parent(x, y) \Rightarrow Ancestor(x, y)$

   "If x is a parent of y, then x is an ancestor of y".

2. $Parent(x, y) \land Ancestor(y, z) \Rightarrow Ancestor(x, z)$

   "If x is a parent of y, and y is an ancestor of z, then x is an ancestor of z".

3. $Parent(John, Mary)$

   "John is a parent of Mary."

4. $Parent(Mary, Sue)$

   "Mary is a parent of Sue."

Example 2

Query α

We want to know if **John is an ancestor of Sue**, so our query α is:

$$Ancestor(John, Sue)$$

1. **Initialize:**
   - Set $new = \{\ \}$ to track any newly inferred facts.
2. **First Iteration:**
   - **Rule 1:** $Parent(x, y) \Rightarrow Ancestor(x, y)$
     - **Match:** Substitute $x = John, y = Mary$ from $Parent(John, Mary)$
     - **Inference:** $Ancestor(John, Mary)$
     - **Add to** new: $Ancestor(John, Mary)$

Example 2

- **Match:** Substitute $x = Mary, y = Sue$ from $Parent(Mary, Sue)$
- **Inference:** $Ancestor(Mary, Sue)$
- **Add to** new: $Ancestor(Mary, Sue)$

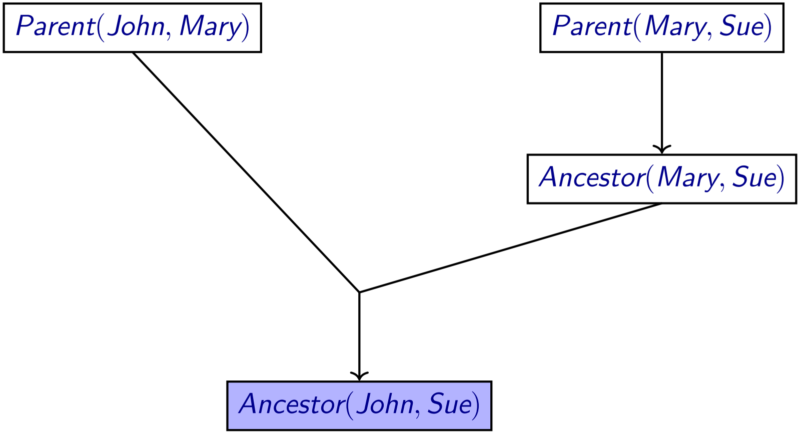
- **Rule 2:** $Parent(x, y) \wedge Parent(y, z) \Rightarrow Ancestor(x, z)$
  - **Match:** Substitute $x = John, y = Mary, z = Sue$ using $Parent(John, Mary)$ and $Ancestor(Mary, Sue)$ from new
  - **Inference:** $Ancestor(John, Sue)$
  - **Add to** new: $Ancestor(John, Sue)$

Example 2

- **Check query:**
  - Attempt to unify $Ancestor(John, Sue)$ with the query $Ancestor(John, Sue)$
  - **Unification Successful:** The algorithm returns the answer $\theta = \{\ \}$, meaning the query is true based on KB.

3. **Result:**
   - Since $Ancestor(John, Sue)$ was inferred, the algorithm confirms that $John$ is indeed an ancestor of Sue, and the query is answered successfully.

Example 2 diagram



Backward Chaining Algorithm

```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
  return FOL-BC-OR(KB, query, { })

function FOL-BC-OR(KB, goal, θ) returns a substitution
  for each rule in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs ⇒ rhs) ← STANDARDIZE-VARIABLES(rule)
    for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
      yield θ′

function FOL-BC-AND(KB, goals, θ) returns a substitution
  if θ = failure then return
  else if LENGTH(goals) = 0 then yield θ
  else
    first,rest ← FIRST(goals), REST(goals)
    for each θ′ in FOL-BC-OR(KB, SUBST(θ, first), θ) do
      for each θ″ in FOL-BC-AND(KB, rest, θ′) do
        yield θ″
```

Backward Chaining Explaination

These algorithms work backward from the goal, chaining through rules to find known facts that support the proof.

1. *FOL-BC-ASK*$(KB, query)$:

Main function. From knowledge base (*KB*) and *query*, it will produce all possible substitutions that satisfy the query.

2. *FOL-BC-OR*$(KB, goal, \theta)$:

Tries to achieve the goal with a substitution θ. It fetches each rule in the knowledge base relevant to the goal (through *FETCH-RULES-FOR-GOAL*). For each *rule*, it standardizes the variables to avoid conflicts between different uses of the same *rule*. Then, it uses *FOL-BC-AND* on the left-hand side (*lhs*) of the *rule* after attempting to unify the right-hand side (*rhs*) with the *goal* using *Unify*$(rhs, goal, \theta)$. If a substitution is found, it is returned through the generator.

Backward Chaining Explaination

3. $FOL\text{-}BC\text{-}AND(KB, goals, \theta)$:

This function tries to satisfy a sequence of $goals$ with an initial substitution θ. If the substitution θ is a $failure$, it returns. If there are no more $goals$ left, it yields the current substitution. Otherwise, it takes the $first$ goal in the list and attempts to satisfy it by calling $FOL\text{-}BC\text{-}OR$ with the substituted goal ($Subst(\theta, first)$) and θ. For each substitution $(\theta)''$ generated by $FOL\text{-}BC\text{-}OR$, it recursively calls $FOL\text{-}BC\text{-}AND$ on the remaining $goals$ ($rest$) with $(\theta)''$.

Backward Chaining Explaination

- **Standardization of Variables:** Ensures that variables in each rule are unique when the rule is used, avoiding variable clashes across different applications.
- **Unification ($Unify$):** Matches the goal with the rule's right-hand side, finding substitutions to make them equivalent.
- **Substitution ($Subst$):** Applies a substitution to a goal, replacing variables with values from θ

Example 1

Datalog knowledge base (KB):

1. $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
2. $\exists x \ (Owns(Nono, x) \wedge Missile(x))$

   2.1 $Owns(Nono, M_1)$ 2.2 $Missile(M_1)$
3. $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
4. $Missile(x) \Rightarrow Weapon(x)$
5. $Enemy(x, America) \Rightarrow Hostile(x)$
6. $American(West)$
7. $Enemy(Nono, America)$

   Determine if Colonel West is a criminal: $Criminal(West)$

Example 1

Goal: Prove $Criminal(West)$

1. **Step 1: Start with the Goal** $Criminal(West)$
   - We want to prove $Criminal(West)$ According to rule (1) in the knowledge base:

   $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

   To satisfy $Criminal(West)$, we need to show that:

     - $American(West)$
     - $Weapon(y)$ for some y
     - $Sells(West, y, z)$ for some y and z
     - $Hostile(z)$ for some z

Example 1

2. **Step 2: Verify** $American(West)$

- From fact (6): $American(West)$, this is true. So, we have verified the first condition.

3. **Step 3: Verify** $Weapon(y)$

- Since $Sells(West, y, Nono)$ involves selling something to $Nono$, we look at fact (2.1) and (2.2):
    - $(\exists x Owns(Nono, x) \wedge Missile(x))$, where M_1 is introduced as a constant for x.
    - We know $Missile(M_1)$.
- According to rule (4): $Missile(x) \Rightarrow Weapon(x)$, so $Missile(M_1)$ implies $Weapon(M_1)$.
- Thus, $Weapon(M_1)$ is verified.

Example 1

4. **Step 4: Verify** $Sells(West, M1, Nono)$
   - From rule (3): $Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
   - Since $Missile(M_1)$ (from 2.2) and $Owns(Nono, M_1)$ (from 2.1), this implies $Sells(West, M_1, Nono)$ is true.
5. **Step 5: Verify** $Hostile(Nono)$
   - From rule (5): $Enemy(x, America) \Rightarrow Hostile(x)$
   - Since fact (7) states $Enemy(Nono, America)$, we can deduce $Hostile(Nono)$.
6. **Conclusion:**
   - We have shown that $American(West), Weapon(M1), Sells(West, M1, Nono)$, and $Hostile(Nono)$ all hold.
   - Therefore, by rule (1), $Criminal(West)$ is true.

Example 2

Let's consider a knowledge base that contains rules and facts about animals and whether they are mammals.

Knowledge Base KB

1. **Rule**:
   - **Rule 1:** If an animal has fur and gives live birth, it is a mammal.

     $HasFur(x) \land GivesLiveBirth(x) \Rightarrow Mammal(x)$
   - **Rule 2:** If an animal is a dog, it has fur and gives live birth

     $Dog(x) \Rightarrow HasFur(x) \land GivesLiveBirth(x)$
2. **Fact**:
   - $Dog(Rex)$: Rex is a dog.

Goal We want to prove $Mammal(Rex)$.

Example 2

Backward chaining algorithm:

1. **Initial Goal:** Start with $Mammal(Rex)$ as the query.
2. **Function Call:** Call $FOL\text{-}BC\text{-}ASK(KB, Mammal(Rex))$
3. **Fetch Rules for Goal:**
   - Use $FETCH\text{-}RULES\text{-}FOR\text{-}GOAL(KB, Mammal(Rex))$
   - From **Rule 1**, we have:

     $$HasFur(x) \land GivesLiveBirth(x) \Rightarrow Mammal(x)$$

   - Thus, the sub-goals are:
     - $HasFur(Rex)$
     - $GivesLiveBirth(Rex)$

Example 2

4. **Handle Sub-goals:**
   - Call $FOL\text{-}BC\text{-}AND(KB, [HasFur(Rex), GivesLiveBirth(Rex)], \{\ \})$
5. **Check First Sub-goal** $HasFur(Rex)$:
   - Use $FETCH\text{-}RULES\text{-}FOR\text{-}GOAL(KB, HasFur(Rex))$
   - No direct rules for $HasFur$, so we check if $Dog(Rex)$ implies $HasFur(Rex)$ from **Rule 2**:

$$Dog(x) \Rightarrow HasFur(x) \land GivesLiveBirth(x)$$

   - We have $Dog(Rex)$ in our facts, so we can conclude $HasFur(Rex)$ is **true**.

Example 2

6. **Check Second Sub-goal** $GivesLiveBirth(Rex)$

   - Return to $FOL\text{-}BC\text{-}AND(KB, [HasFur(Rex), GivesLiveBirth(Rex)], \{\ \})$
   - We know $HasFur(Rex)$ is true, now check $GivesLiveBirth(Rex)$
   - From **Rule 2**, we also conclude $GivesLiveBirth(Rex)$ is **true** because $Dog(Rex)$ implies both $HasFur(Rex)$ and $GivesLiveBirth(Rex)$

7. **Final Check:**

   - Since both sub-goals $HasFur(Rex)$ and $GivesLiveBirth(Rex)$ are true, we conclude that $Mammal(Rex)$ is true based on **Rule 1**

Conclusion

$Mammal(Rex)$ is true, proving that Rex is a $mammal$

Resolution

- In propositional logic: We have resolution algorithm. It is a complete inference procedure.
- Review that resolution.
- Now, we extend it to first-order logic.

Conjunctive Normal Form (CNF)

Sentence of FOL can be converted into an CNF sentence

e.g.

$$\forall x, y, z\ American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

becomes, in CNF,

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$$

Eliminate existential quantifiers

e.g.

"Everyone who loves all animals is loved by someone":

$$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\ Loves(y, x)]$$

The steps to eliminate in the next slide:

Conjunctive Normal Form (CNF)

- **Eliminate implications**: Replace $P \Rightarrow Q$ with $\neg P \lor Q$

  $$\forall x\ \neg[\forall y\ Animal(y) \Rightarrow Loves(x, y)] \lor [\exists y\ Loves(y, x)]$$

  $$\forall x\ \neg[\forall y\ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y\ Loves(y, x)]$$

- **Move $\neg$ inwards**:

  $\neg\forall x\ p$ becomes $\exists x\ \neg p$

  $\neg\exists x\ p$ becomes $\forall x\ \neg p$

Our sentence goes through the following transformations:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \lor Loves(x, y))] \lor [\exists y\ Loves(y, x)]$$

$$\forall x\ [\exists y\ \neg\neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ Loves(y, x)]$$

$$\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ Loves(y, x)]$$

Conjunctive Normal Form (CNF)

- **Standardize variables**: For sentences like $(\exists x\ P(x)) \lor (\exists x\ Q(x))$ that use the same variable name twice, change the name of one of the variables.

  $$\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists z\ Loves(z, x)]$$

- **Skolemize**: Using **Skolem functions** for $\exists x\ P(x)$

  $\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ Loves(y, x)]$ become:

  $\forall x\ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor [Loves(G(x), x)]$ F, G are Skolem functions

- **Drop universal quantifiers**:

  $$[Animal(F(x)) \land \neg Loves(x, F(x))] \lor [Loves(G(x), x)]$$

- **Distribute $\lor$ over $\land$**:

  $$Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)]$$

Resolution inference rule

$$\frac{l_1 \vee \ldots \vee l_k, \quad m_1 \vee \ldots \vee m_n}{Subst(\theta, l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

Where $Unify(l_i, \neg m_j) = \theta$

e.g.

$[Animal(F(x)) \vee Loves(G(x), x)]$ and $[\neg Loves(u, v) \vee \neg Kills(u, v)]$

by eliminating the complementary literals:

$Loves(G(x), x)$ and $\neg Loves(u, v)$, with the unifier $\theta = \{u/G(x), v/x\}$,

to produce the **resolvent** clause:

$[Animal(F(x)) \vee \neg Kills(G(x), x)]$

Example 1

Resolution proves that $KB \models \alpha$ by proving that $KB \wedge \neg\alpha$ **unsatisfiable**

that is, by deriving the **empty clause**.

e.g. Crime example to the sentences in CNF are:

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$$

$$\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$$

$$\neg Enemy(x, America) \vee Hostile(x)$$

$$\neg Missile(x) \vee Weapon(x)$$

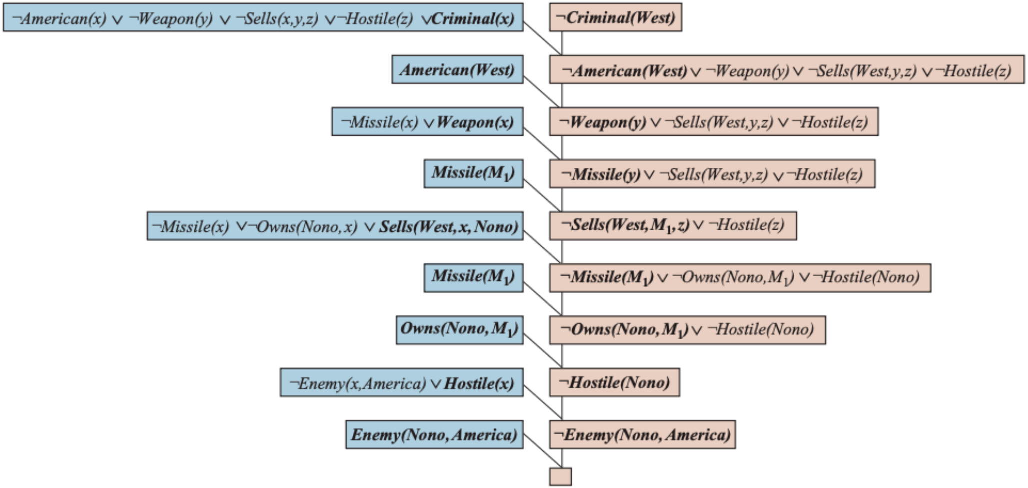
$$Owns(Nono, M_1) \qquad Missile(M_1)$$

$$American(West) \qquad Enemy(Nono, America)$$

Goal: $Criminal(West)$

We include the negated goal: $KB \wedge \neg Criminal(West)$

Example 1 diagram



Example 2

Everyone who loves all animals is loved by someone.

Anyone who kills an animal is loved by no one.

Jack loves all animals.

Either Jack or Curiosity killed the cat, who is named Tuna.

Did Curiosity kill the cat?

Example 2

First, we express the original sentences, some background knowledge, and the negated goal G in FOL:

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\ Loves(y, x)]$

B. $\forall x\ [\exists z\ Animal(z) \wedge Kills(x, z)] \Rightarrow [\forall y \neg\ Loves(y, x)]$

C. $\forall x\ Animal(x) \Rightarrow Loves(Jack, x)$

D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

F. $\forall x\ Cat(x) \Rightarrow Animal(x)$

$\neg$G. $\neg Kills(Curiosity, Tuna)$

Example 2

Now we apply the conversion procedure to convert each sentence to CNF:

A1. $Animal(F(x)) \lor Loves(G(x), x)$

A2. $\neg Loves(x, F(x)) \lor Loves(G(x), x)$

B. $\neg Loves(y, x) \lor \neg Animal(z) \lor \neg Kills(x, z)$

C. $\neg Animal(x) \lor Loves(Jack, x)$

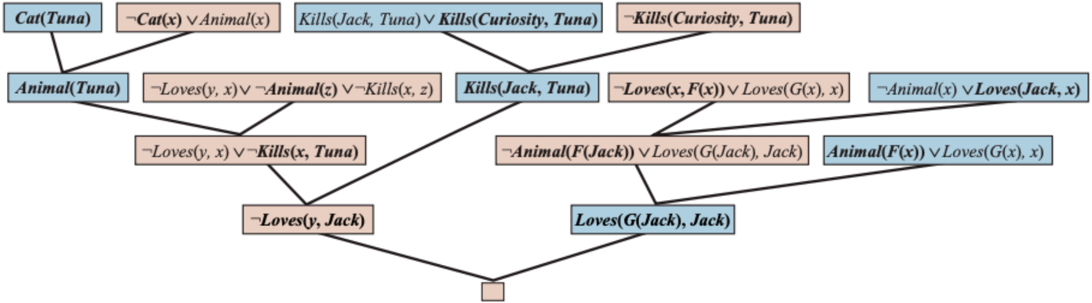
D. $Kills(Jack, Tuna) \lor Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

F. $\neg Cat(x) \lor Animal(x)$

¬G. $\neg Kills(Curiosity, Tuna)$

Example 2 diagram



Definition

Planning Domain Definition Language (**PDDL**):

1. State Planners decompose the world into logical conditions and represent a state as a conjunction of **ground atomic fluents**.

- "**ground**" means no variables
- "**fluent**" means an aspect of the world that changes over time
- "**ground atomic**" means there is a single predicate
- if there are any arguments, they must be constants

e.g.

$Poor \wedge Unknown$ represent the state of a hapless agent

$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$ represent a state in a package delivery problem

Definition

- The following fluents are not allowed in a state:
  - $At(x, y)$ because it has variables
  - $\neg Poor$ because it is a negation
  - $At(Spouse(Ali), Sydney)$ because it uses a function symbo Spouse

3. Database Semantics is a conjunction of ground fluents

- Closed-world
- Any fluents that are not mentioned are $false$
- Unique names are distinct. e.g. $Truck_1$ and $Truck_2$

Definition

3. Action schema represents a family of ground actions:

- Consists of the action name,
- A list of all the variables used in the schema,
- A precondition (**Precond**) and an effect (**Effect**).
- **Precond** and **Effect** are each conjunctions of literals Precondition (positive or negated atomic sentences)
- A set of action schemas serves as a definition of a **planning domain**

e.g. An action schema for flying a plane from one location to another:

$Action(Fly(p, from, to)$,

Precond: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$

Effect: $\neg At(p, from) \land At(p, to))$

Definition

We can choose constants to instantiate the variables, Effect yielding a ground (variable-free) action:

$Action(Fly(P_1, SFO, JFK),$

Precond: $At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$

Effect: $\neg At(P_1, SFO) \wedge At(P_1, JFK))$

- **Applicable**: A ground action a is applicable in state s if s entails the precondition of a; that is, if every positive literal in the precondition is in s and every negated literal is not.

Definition

- **Result**: The result of executing applicable action a in state s is defined as a state s' which is represented by the set of fluents formed by starting with s, removing the fluents that appear as negative literals in the action's effects (what we call the delete list or $Del(a)$), and adding the fluents that are positive literals in the action's effects (what we call the add list or $Add(a)$):

$$Result(s, a) = (s - Del(a)) \cup Add(a)$$

e.g. with the action $Fly(P_1, SFO, JFK)$, we would remove the fluent $At(P_1, SFO)$ and add the fluent $At(P_1, JFK)$

Definition

4. Initial State is a conjunction of ground fluents

5. Goal is just like a precondition: a conjunction of literals (positive or negative) that may contain variables

e.g.

$At(C_1, SFO) \wedge \neg At(C_2, SFO) \wedge At(p, SFO)$, refers to any state in which cargo C_1 is at SFO but C_2 is not, and in which there is a plane at SFO

Air cargo transport

Air cargo transport problem involving loading and unloading cargo and flying it from place to place. The problem can be defined with three actions: Load, Unload, and Fly. The actions affect two predicates: $In(c, p)$ means that cargo c is inside plane p, and $At(x, a)$ means that object x (either plane or cargo) is at airport a.

When a plane flies from one airport to another, all the cargo inside the plane goes with it. In first-order logic it would be easy to quantify over all objects that are inside the plane. But PDDL does not have a universal quantifier, so we need a different solution. The approach we use is to say that a piece of cargo ceases to be At anywhere when it is In a plane; the cargo only becomes At the new airport when it is unloaded. So At really means "available for use at a given location."

Air cargo transport

A PDDL description of an air cargo transportation planning problem:

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\quad \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\quad \wedge Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
  Precond: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
  Effect: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a),$
  Precond: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
  Effect: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to),$
  Precond: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
  Effect: $\neg At(p, from) \wedge At(p, to))$

Air cargo transport

The approach we use is to say that a piece of cargo ceases to be At anywhere when it is In a plane; the cargo only becomes At the new airport when it is unloaded. So At really means "available for use at a given location."

The following plan is a solution to the problem:

$$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$$
$$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]$$

Spare tire problem

- Consider the problem of changing a flat tire.
- The goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk.
- There are just four actions:
  - Removing the spare from the trunk
  - Removing the flat tire from the axle
  - Putting the spare on the axle
  - Leaving the car unattended overnight
- We assume that the car is parked in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.

A solution to the problem:
$Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle)]$

Spare tire problem

$Init(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk))$

$Goal(At(Spare, Axle))$

$Action(Remove(obj, loc),$

Precond: $At(obj, loc)$

Effect: $\neg At(obj, loc) \wedge At(obj, Ground))$

$Action(PutOn(t, Axle),$

Precond: $Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Spare, Axle)$

Effect: $\neg At(t, Ground) \square At(t, Axle))$

$Action(LeaveOvernight),$

Precond:

Effect: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$

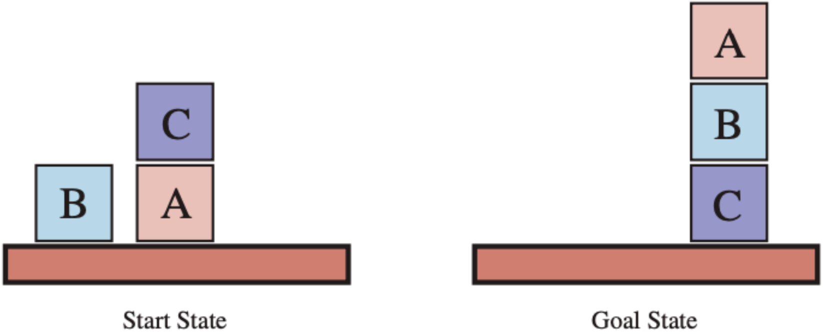
Blocks world

Blocks world One of the most famous planning domains:

- Set of cube-shaped blocks sitting on a table
- The blocks can be
- Only one block can fit directly on top of another
- A robot arm can pick up a block and move it to another position, either on the table or on top of another block
- The arm can pick up only one block at a time, so it cannot pick up a block that has another one on top of it

Blocks world

A typical goal to get block A on B and block B on C



Blocks world

A PDDL description of Block world problem:

$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{Precond: } On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{Effect: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{Precond: } On(b, x) \land Clear(b) \land Block(b) \land Block(x), \\
&\quad \text{Effect: } On(b, Table) \land Clear(x) \land \neg On(b, x))
\end{aligned}
$$

One solution is the sequence:

$$[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$$

Blocks world

$On(b,x)$ predicate block b is on x, where x is either another block or the table

$Move(b,x,y)$ action for moving block b from the top of x to the top of y

$\neg\exists x\ On(x,b)$ or $\forall x\ \neg On(x,b)$ preconditions on moving b is that no other block be on it. Because PDDL does not allow quantifiers, so we have

$Clear(x)$ predicate is true when nothing is on x

The action Move moves a block b from x to y if both b and y are clear. After the move is made, b is still clear but y is not. A first attempt at the Move schema is:

$Action(Move(b,x,y),$
    Precond: $On(b,x) \land Clear(b) \land Clear(y),$
    Effect: $On(b,y) \land Clear(x) \land \neg On(b,x) \land \neg Clear(y))$

Blocks world

Unfortunately, when x is the Table, this action has the effect $Clear(Table)$, but the table should not become clear; and when $y = Table$, it has the precondition $Clear(Table)$, but the table does not have to be clear for us to move a block onto it. To fix this, we do two things:

- First, we introduce another action to move a block b from x to the $table$:

$$\begin{aligned}&Action(MoveToTable(b, x),\\ &\quad \text{Precond: } On(b, x) \wedge Clear(b),\\ &\quad \text{Effect: } On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))\end{aligned}$$

- Second, $Clear(Table)$ will always be true. Nothing prevents the planner from using $Move(b, x, Table)$ instead of $MoveToTable(b, x)$.

Forward state-space search

```
ForwardStateSpaceSearch(initialState, goalState, actions)
    queue = initialState # initialize a queue to store nodes to explore
    explored = set() # Keep track of explored states to avoid cycles
    while queue:
        currentState = queue.pop(0) # Remove the first node from the queue
        if goalTest(currentState): # Goal reached, trace back the path
            return reconstructPath(currentState)
        explored.add(currentState)
        for action in actions:
            if action.isApplicable(currentState):
                newState = action.apply(currentState)
                if newState not in explored:
                    queue.append(newState)
    return None # if the queue is empty, no solution was found
```

Forward state-space search

reconstructPath($currentState$):

  $path = currentState$

  while $currentState.parent$:

    $currentState = currentState.parent$

    $path.append(currentState)$

  $path.reverse()$

  return $path$

Backward state-space search

```
BackwardStateSpaceSearch(initialState, goalState, actions)
    queue = goalState # Initialize a queue to store nodes to explore
    explored = set() # Keep track of explored states to avoid cycles
    while queue:
        currentState = queue.pop(0) # Remove the first node from the queue
        if currentState = initialState: # Goal reached, trace back the path
            return reconstructPath(currentState)
        explored.add(currentState)
        for action in actions:
            if action.isApplicableBackward(currentState):
                newState = action.applyBackward(currentState)
                if newState not in explored:
                    queue.append(newState)
    return None # If the queue is empty, no solution was found
```

Backward state-space search

reconstructPath($currentState$):

$path = currentState$

while $currentState.parent$:

$currentState = currentState.parent$

$path.append(currentState)$

$path.reverse()$

return $path$