# Knowledge: Representation and Reasoning

Yannick Pencolé
Yannick.Pencole@anu.edu.au

09 Mar 2005

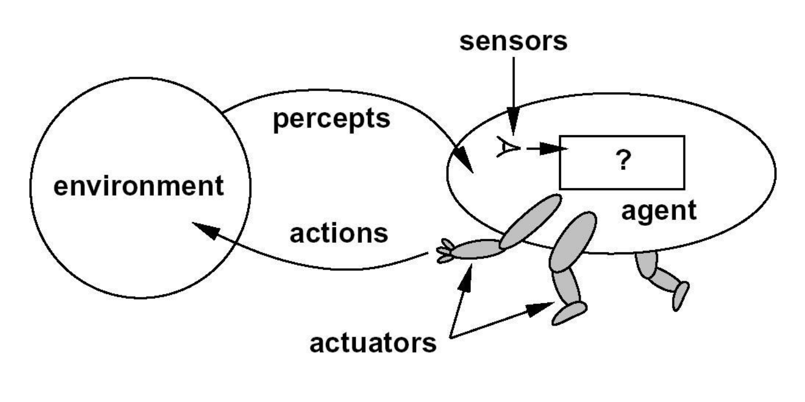
# Outline

1. Knowledge-based agents
2. Logic in general models and entailment
3. Propositional Logic: a very simple logic
4. Equivalence, validity, satisfiability
5. Inference rules and theorem proving in propositional logic

# Outline


1. Knowledge-based agents
2. Logic in general models and entailment
3. Propositional Logic: a very simple logic
4. Equivalence, validity, satisfiability
5. Inference rules and theorem proving in propositional logic

# What is an agent?




Agents interact with environments through sensors and actuators

# Knowledge-based agent

**Definition**

A Knowledge base is a set of sentences in a formal language.

**Definition**

Knowledge-based agent: Agents can be viewed at the knowledge level i.e., *what they know*, regardless of how implemented

Generic knowledge-based agent:

1. PERCEIVE an input
2. TELL the knowledge base what it perceives (same language)
3. ASK the knowledge base for an action to return (reasoning in the KB for a query, inference)
4. TELL the knowledge base about this action that has been executed (update of the KB)

A good way to represent and interact with a *KB* is Logic.

# Outline

1 Knowledge-based agents

2 Logic in general models and entailment

3 Propositional Logic: a very simple logic

4 Equivalence, validity, satisfiability

5 Inference rules and theorem proving in propositional logic

# Logic? But what is it?

## Principle

Logics are formal languages for representing information such that conclusions can be drawn. To define a logic, we need:

1. syntax: how a sentence of the logic looks like?
2. semantic: what is the meaning of the sentence?
    - Given a world, is the sentence true or false?

## Example

The language of arithmetic
Syntax:
$x + 2 \geq y$ is a sentence;
$x2 + y \geq y$ is not a sentence
Semantic:
$x + 2 \geq y$ is `true` in a world where $x = 7, y = 1$
$x + 2 \geq y$ is `false` in a world where $x = 0, y = 6$

# Entailment

Entailment means that one sentence ($\alpha$) follows from other sentences (*KB*) and is denoted:

$$KB \models \alpha$$

We say that the Knowledge Base *KB* entails $\alpha$ if and only if $\alpha$ is `true` in all worlds where *KB* is `true`. Entailment is a relationship between sentences (i.e. syntax) that is based on semantics.

## Example

Knowledge Base = { "The car is blue" "The bicycle is green or yellow" }
*KB* entails sentences $\alpha$ like:

- "The car is blue"
- `true`
- "The car is blue or the bicycle is yellow"

The sentence "The car is blue and the bicycle is yellow" is not entailed by *KB*.

# World in Logic = Model

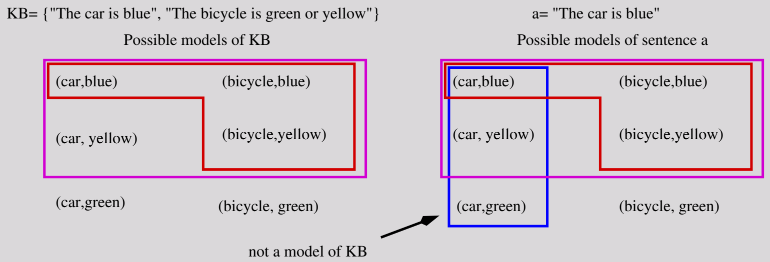
## Definition

We say *m* is a model of a sentence $\alpha$ if $\alpha$ is true in the world *m*.
We denote by $M(\alpha)$ the set of models

## Property

$KB$ entails $\alpha$ if and only if $M(KB) \subseteq M(\alpha)$.

## Example



# Inference

**Definition**

Inference: A sentence $\beta$ can be inferred from another sentence $\alpha$ by some inference algorithm *i*. This is denoted:

$$\alpha \vdash_i \beta$$

**Definition**

Soundness: An inference algorithm is sound if it produces entailed sentences

**Definition**

Completeness: An inference algorithm is complete if it can derive all the sentences which it entails.

# Well-known logics

1. Propositional logic
2. First-order logic
3. Default logic
4. Circumscription
5. Temporal logic
6. Modal logic
7. ..

Every logic has its Pros and Cons (expressivity, soundness and completeness of inference algorithm)

# Outline

1. Knowledge-based agents
2. Logic in general models and entailment
3. Propositional Logic: a very simple logic
4. Equivalence, validity, satisfiability
5. Inference rules and theorem proving in propositional logic

# Propositional Logic: a very simple logic

We consider a set of proposition symbols $\{p_1, p_2, \cdots\}$.

## Definition

Syntax: What is a sentence in the propositional logic?

1. any proposition symbol $p_i$ is a sentence (*atomic sentence*)
2. if $S$ is a sentence then $\neg S$ is a sentence
3. if $S_1$ and $S_2$ are sentences then $S_1 \wedge S_2$ is a sentence
4. if $S_1$ and $S_2$ are sentences then $S_1 \vee S_2$ is a sentence
5. if $S_1$ and $S_2$ are sentences then $S_1 \Rightarrow S_2$ is a sentence
6. if $S_1$ and $S_2$ are sentences then $S_1 \Leftrightarrow S_2$ is a sentence

## Example

$p_1$, $p_1 \wedge p_2$, $p_1 \vee (\neg p_2 \wedge p_3)$, $(p_3 \Rightarrow p_4) \wedge (p_4 \Rightarrow p_3)$, ...

# Propositional Logic: a very simple logic

## Definition

Semantics: What is the meaning of a sentence? A model $m$ is a mapping between the proposition symbols $\{p_1, p_2, \cdots\}$ and $\{true, false\}$. Given $m$, we have:

1. $\neg S$ is *true* iff $S$ is *false* ("not" $S$)
2. $S_1 \wedge S_2$ is *true* iff $S_1$ is *true* and $S_2$ is *true* ($S_1$ "and" $S_2$)
3. $S_1 \vee S_2$ is *true* iff $S_1$ is *true* or $S_2$ is *true* ($S_1$ "or" $S_2$)
4. $S_1 \Rightarrow S_2$ is *true* iff $S_1$ is *false* or $S_2$ is *true* ($S_1$ "implies" $S_2$)
   - i.e. $S_1 \Rightarrow S_2$ is *false* iff $S_1$ is *true* or $S_2$ is *false*
5. $S_1 \Leftrightarrow S_2$ is *true* iff $S_1 \Rightarrow S_2$ is *true* and $S_2 \Rightarrow S_1$ is *true* ($S_1$ "is equivalent to" $S_2$)

# Propositional Logic: a very simple logic

## Example

Symbols = $\{abcd\}$

Given the model $m = \{a = true, b = false, c = true, d = false\}$, then

- $\neg a = false$,
- $a \wedge \neg b = true$,
- $a \vee (b \wedge \neg c) = true$,
- $d \Rightarrow c = true$,
- $d \Rightarrow \neg c = true$,
- $\neg d \Rightarrow \neg c = false$,
- $\neg(a \vee b) \Leftrightarrow d = true$

# Outline

1. Knowledge-based agents
2. Logic in general models and entailment
3. Propositional Logic: a very simple logic
4. Equivalence, validity, satisfiability
5. Inference rules and theorem proving in propositional logic

# Logical equivalence

## Definition

Two sentences $\alpha$, $\beta$ are logically equivalent IF AND ONLY IF they are true in the same models. $\alpha$ entails $\beta$ and vice-versa.

## Logical equivalent sentences

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

# Validity and satisfiability

## Definition

A sentence is valid if it is true in ALL models:
$a \vee \neg a$, $a \Rightarrow a$, $(a \wedge (a \Rightarrow b)) \Rightarrow b$

## Deduction theorem

$KB$ entails $\alpha$ ($KB \models \alpha$) iff the sentence $KB \Rightarrow \alpha$ is valid. Validity is then connected to inference.

## Definition

1. A sentence is satisfiable if it is true in SOME models. A valid sentence is satisfiable, but a satisfiable sentence may be not valid.
2. A sentence is unsatisfiable if it is true in NO models:
   $a \wedge \neg a$, $(a \wedge b) \Leftrightarrow (\neg a \wedge c)$

## Satisfiability and inference

$KB$ entails $\alpha$ ($KB \models \alpha$) iff the sentence $KB \wedge \neg\alpha$ is unsatisfiable.

# Outline

1. Knowledge-based agents
2. Logic in general models and entailment
3. Propositional Logic: a very simple logic
4. Equivalence, validity, satisfiability
5. Inference rules and theorem proving in propositional logic

# Inference, Theorem proving

## Theorem proving

Given *KB*, can I prove the sentence $\alpha$? Is $\alpha$ satisfiable in *KB*? $KB \models \alpha$? Is $KB \wedge \neg\alpha$ unsatisfiable?

## Semantics: model-checking

Model (Truth table) enumeration, we check that $M(KB) \subseteq M(\alpha)$.
Bad news: exponential in the number of proposition symbols involved in $KB, \alpha$.
Some improved methods: Davis-Putnam-Logemann-Loveland (complete), min-conflicts-like (incomplete) hill-climbing (incomplete).

## Syntax: inference rules

- Sound generation of new sentences from old.
- Proof = a sequence of inference rules that finally generate $\alpha$

Methods: Forward-chaining, Backward chaining, Resolution

# Inference rules: examples

## Example

Modus Ponens:

$$\frac{a, a \Rightarrow b}{b}$$

And-elimination:

$$\frac{a \wedge b}{a}$$

Factoring:

$$\frac{a \vee a}{a}$$

Logical equivalences:

$$\frac{\neg a \vee \neg b}{\neg(a \wedge b)}$$

$$\frac{a \Leftrightarrow b}{a \Rightarrow b \wedge b \Rightarrow a}$$

# Forward and backward chaining methods

## Forward and backward chaining

Proof methods that are simple and efficient. They need a restriction on the form of $KB$.

$$KB = \text{conjunction of Horn clauses}$$

## Definition

A Horn clause is

- a propositional symbol $a$, or
- something like $p_1 \cdots p_n \Rightarrow c$ ($p_i$ is a premisse and $c$ the conclusion)

# Inference problem

## Knowledge base

Rule 1 $P \Rightarrow Q$

Rule 2 $L \wedge M \Rightarrow P$

Rule 3 $B \wedge L \Rightarrow M$

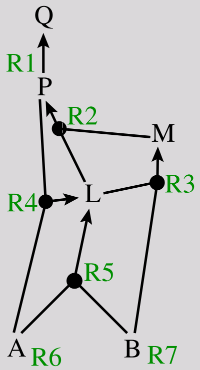
Rule 4 $A \wedge P \Rightarrow L$

Rule 5 $A \wedge B \Rightarrow L$

Rule 6 $A$

Rule 7 $B$

## Graphical Representation




## Inference problem

Is the proposition $Q$ true or not?

# Forward chaining method

## Idea

Forward chaining:

1. Fire any rule whose premises are satisfied in the Knowledge Base
2. Add its conclusion to the Knowledge Base
   - Management of a *Working Memory* (an Agenda)
3. Repeat 1 and 2 until the proposition $Q$ is true or no more conclusion can be derived

## Inference rule

$$\frac{a_1, \cdot, a_n \quad a_1 \wedge \cdots \wedge a_n \Rightarrow b}{b}$$

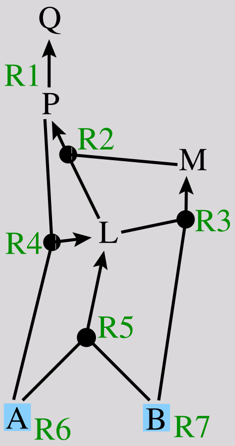
# Forward chaining: example

## Forward chaining

*A* and *B* are known in the knowledge base (Rules 6 and 7).

$$Agenda = \{A, B\}$$

## Graphical Representation



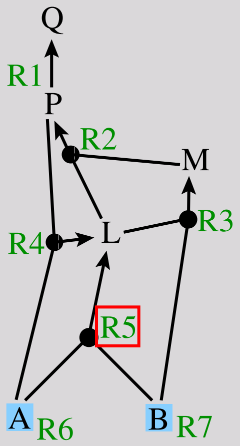
# Forward chaining: example

## Forward chaining

Rule 5 can be fired since *A* and *B* are known.

$$A \wedge B \Rightarrow L$$

## Graphical Representation



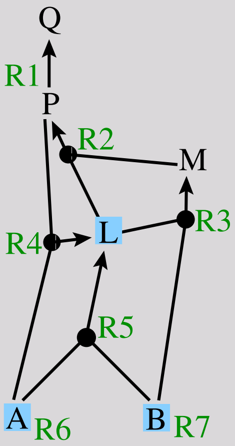
# Forward chaining: example

## Forward chaining

The agenda is updated with the conclusion (head) of Rule 5:

$$Agenda = \{A, B, L\}$$

## Graphical Representation



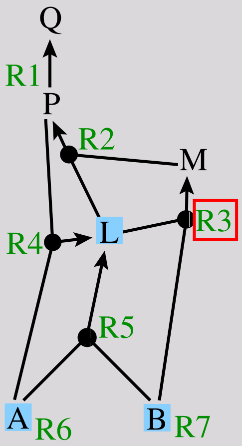
# Forward chaining: example

## Forward chaining

Rule 3 can be fired since $B$ and $L$ are known.

$$B \wedge L \Rightarrow M$$

## Graphical Representation



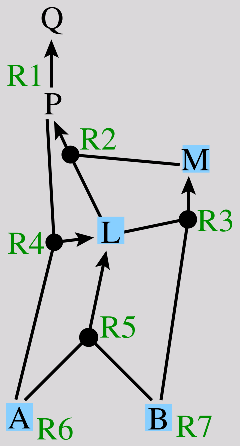
# Forward chaining: example

## Forward chaining

The agenda is updated with the conclusion (head) of Rule 3:

$$Agenda = \{A, B, L, M\}$$

## Graphical Representation



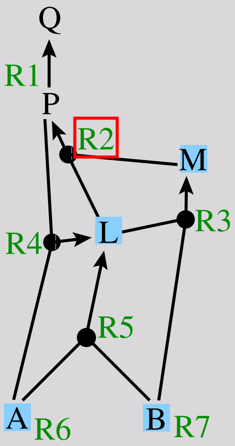
# Forward chaining: example

## Forward chaining

Rule 2 can be fired since $M$ and $L$ are known.

$$M \wedge L \Rightarrow P$$

## Graphical Representation



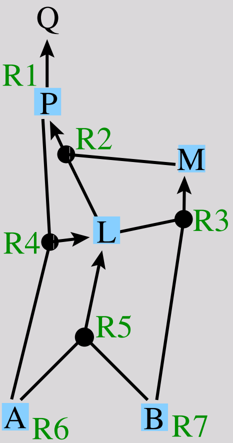
# Forward chaining: example

## Forward chaining

The agenda is updated with the conclusion (head) of Rule 2:

$$Agenda = \{A, B, L, M, P\}$$

## Graphical Representation



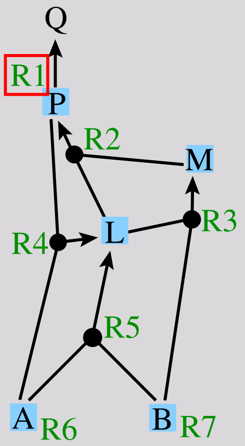
# Forward chaining: example

## Forward chaining

Rule 1 can be fired since $P$ is known.

$$P \Rightarrow Q$$

## Graphical Representation



# Forward chaining: example

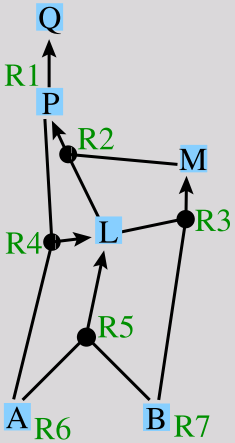
## Forward chaining

The agenda is updated with the conclusion (head) of Rule 1:

$$Agenda = \{A, B, L, M, P, Q\}$$

Q has been derived.
STOP

## Graphical Representation



# Properties of the FC algorithm

**Theorem**

*The forward chaining algorithm is complete.*

- Every atomic proposition that is provable in the knowledge base can be derived thanks to the algorithm.
- The result is due to the fact that:
  1. The knowledge base is a conjunction of Horn clauses.
  2. In the worst case of FC, every clause is fired.

**Theorem**

*The forward chaining algorithm runs in linear time.*

- Every rule is fired at most once.

# Backward chaining method

## Problem with FC

At a given step, the choice between the firable rules is random.
FC may apply rules that are *useless* for the proof of $Q$!
Data-driven algorithm

## Idea

Backward chaining: Goal-driven algorithm

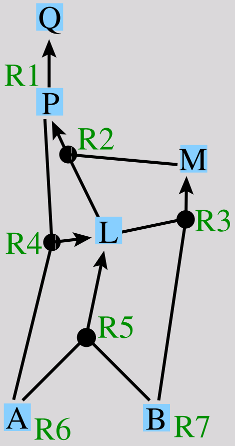
- To prove $Q$ every premise of one rule which concludes $Q$ has to be at least proved.
- Basically, BackwardChaining($Q$) =
  1. If $Q$ is known in $KB$ then $Q$ is proved
  2. Otherwise, select a rule $P_1 \wedge \cdots \wedge P_n \Rightarrow Q$ in KB
  3. Recusirvely apply BackwardChaining($P_1$), ... , BackwardChaining($P_n$) to prove the premises.
  4. Repeat 2-3 until $Q$ is proved or no more rules can be selected.

# Backward chaining: example

## Backward chaining

To prove $Q$, we need to prove $P$. To prove $P$ we need to prove $L$ and $M$ etc. So it goes until $A$ and $B$ are reached. $A$ and $B$ are known in KB so $Q$ is proved.

## Graphical Representation



# Backward chaining method (2)

**Theorem**

*The backward chaining algorithm is complete.*

**Theorem**

*The backward chaining algorithm runs in linear time.*

- Every rule is fired at most once
- In practice, it is much less than linear in size of KB: it is linear in the size of the set of rules that are involved in the proof of the premises of $Q$.

# Propositional logic inference: resolution algorithm

## FC and BC

FC and BC are efficient algorithms but they make the asssumption that KB is a set of Horn clauses.

## What about the general case?

Given a KB, is there an algorithm which is sound and complete?

## Answer

YES, this is called a resolution algorithm based on the resolution inference rule.

# Resolution inference rule

## Resolution inference rule

$$\frac{\ell_1 \vee \cdots \vee \boxed{\ell_i} \vee \cdots \vee \ell_k, \qquad \ell'_1 \vee \cdots \vee \boxed{\ell'_j} \vee \cdots \vee \ell'_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee \ell'_1 \vee \cdots \vee \ell'_{j-1} \vee \ell'_{j+1} \vee \cdots \vee \ell'_n}$$

with $\boxed{\ell_i = a}$ and $\boxed{\ell_j = \neg a}$ or $\boxed{\ell_i = \neg a}$ and $\boxed{\ell_j = a}$
($a$ is an atomic sentence)

## Example

$$\frac{\boxed{e}, \boxed{\neg e}}{\emptyset}$$

$$\frac{v \vee \boxed{\neg w} \vee \neg x, \boxed{w} \vee \neg x \vee y \vee z}{v \vee \neg x \vee y \vee z}$$

# Resolution inference rule (2): CNF

## Application of the rule

The rule is applied on sentences like $(\ell_1 \vee \cdots\cdots \vee \ell_k) \wedge \cdots \wedge (\ell_m \vee \cdots\cdots \vee \ell_n)$ where $\ell_i$ are positive/negative literals.
This form is called Conjunctive Normal Form (CNF for short).

## Property

Every sentence in proposition logic is logically equivalent to a CNF sentence.

## Example

$(a \vee b) \Leftrightarrow (c \wedge d)$ is logically equivalent to the CNF $(\neg a \vee c) \wedge (\neg a \vee d) \wedge (\neg b \vee c) \wedge (\neg b \vee d) \wedge (\neg c \vee \neg d \vee a \vee b)$.

# Conversion to a CNF

## Method

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
3. Move $\neg$ inwards using de Morgan's rules $\{\neg(a \wedge b) \equiv (\neg a \vee \neg b), \neg(a \vee b) \equiv (\neg a \wedge \neg b)\}$ and double negation rule $(\neg\neg a \equiv a)$
4. Apply distributivity law ($\vee$ over $\wedge$) and flatten $(a \wedge b) \vee c \equiv (a \vee c) \wedge (b \vee c)$

# Resolution algorithm

## Definition

Proof by contradiction: given $KB$, to prove $\alpha$, we prove that $KB \wedge \neg\alpha$ is not satisfiable.

## Example

Symbols:

- $Und$: "The students have understood this lecture"
- $Gt$: "I am a good teacher"
- $Party$: "The students went to a party last night"

Knowledge base:

$$KB = (\neg Und \Leftrightarrow (\neg Gt \vee Party)) \wedge Und$$

Query to prove: *I am a good teacher*

$$\alpha = Gt$$

# Resolution algorithm

## Example

Conversion to CNF: $KB \wedge \neg\alpha \rightarrow (KB \wedge \neg\alpha)_{CNF}$

$$
\begin{aligned}
(KB \wedge \neg\alpha)_{CNF} = & (\neg Party \vee \neg Und) \wedge \\
& (\neg Gt \vee Und \vee Party) \wedge \\
& (\neg Und \vee Gt) \wedge \\
& Und \wedge \neg Gt
\end{aligned}
$$

## Example

$\boxed{\neg\,\text{Party} \vee \neg\,\text{Und}}$ $\boxed{\neg\text{Gt} \vee \text{Und} \vee \text{Party}}$ $\boxed{\neg\text{Und} \vee \text{Gt}}$ $\boxed{\text{Und}}$ $\boxed{\neg\text{Gt}}$

# Resolution algorithm

## Example

$\neg$ Party $\vee$ $\neg$ Und | $\neg$Gt $\vee$ Und $\vee$ Party | $\neg$Und $\vee$ Gt | Und | $\neg$Gt

# Resolution algorithm

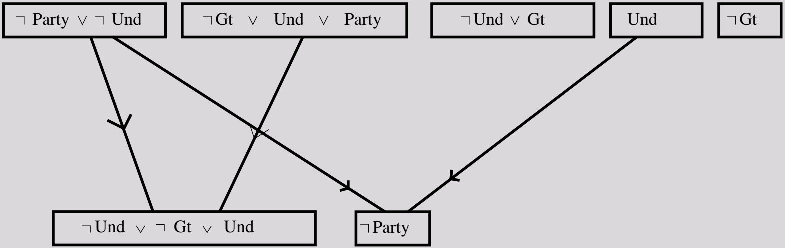
## Example

$\neg$ Party $\vee$ $\neg$ Und | $\neg$Gt $\vee$ Und $\vee$ Party | $\neg$Und $\vee$ Gt | Und | $\neg$Gt

$\neg$Und $\vee$ $\neg$ Gt $\vee$ Und

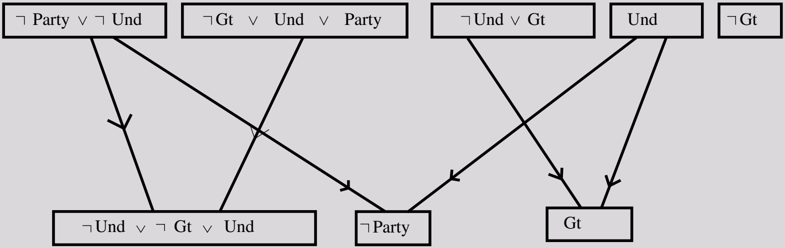
# Resolution algorithm

## Example



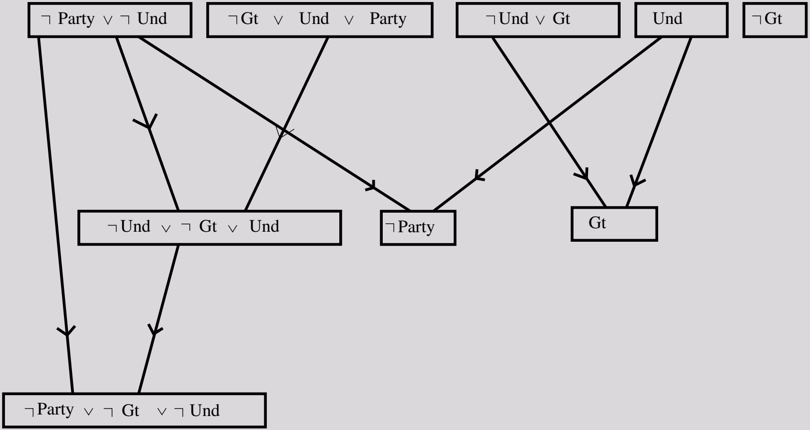
# Resolution algorithm

## Example



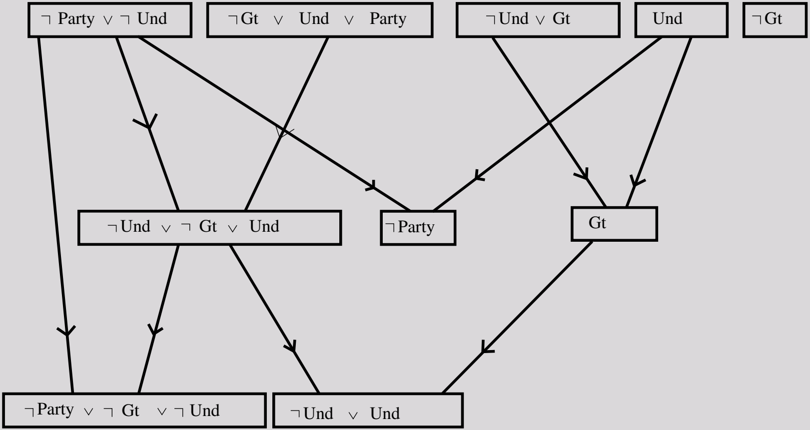
# Resolution algorithm

## Example



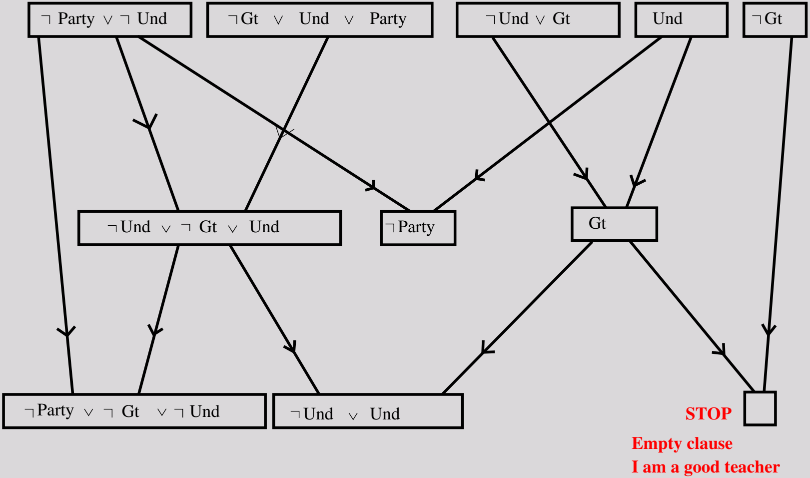
# Resolution algorithm

## Example



# Resolution algorithm

## Example



# Summary

Logical agents apply inference to a knowledge base to derive new information and make decisions
Basic concepts of logic:

- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Forward, backward chaining are linear-time, complete for Horn clauses Resolution is complete for propositional logic
Propositional logic lacks expressive power