# Logical Agents

## Knowledge-Based Agents

A knowledge-based agent is an agent that can infer new information or can infer what action to perform from its internal knowledge.

This process of inferring new information or what action to perform is called reasoning.

The central component of a knowledge-based agent is its **KNOWLEDGE BASE**, or **KB**. A knowledge base is a set of sentences. (Here “sentence” is used as a technical term. It is related but not identical to the sentences of English and other natural languages.) Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world. Sometimes we dignify a sentence with the name **AXIOM**, when the sentence is taken as given without being derived from other sentences.

There must be a way to add new sentences to the knowledge base and a way to query what is known. The standard names for these operations are **TELL** and **ASK**, respectively.

It is important that when a question is asked to a knowledge-based agent, the answer coming out from the agent must be inferred from its knowledge base. In other words, the answers of knowledge-based agent must FOLLOW from its knowledge base.

Each time the agent program is called, it does three things. First, it TELLs the knowledge base what it perceives. Second, it ASKs the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on. Third, the agent program TELLs the knowledge base which action was chosen, and the agent executes the action.

Naturally, a knowledge-based agent can infer what action to perform at a given time and given a certain knowledge based only if it is provided with a goal and a performance measure.

A screenshot of a cell phone

Description automatically generated

## Approaches to grow a knowledge-base

There are two approaches to grow the knowledge base of a knowledge-based agent:

1. Declarative Approach => start with an empty knowledge base and add sentences to the knowledge base by using the TELL function
2. Procedural Approach => encodes the basic sentences of the knowledge base directly as program code

Note that in each case for which the agent draws a conclusion from the available information (KB), that conclusion is guaranteed to be correct if the available information is correct. This is a fundamental property of logical reasoning.

## Logic

The inference process of a knowledge-based agent and the way the sentences of its knowledge based are organised depend on the form of LOGIC used.

Any form of LOGIC serves as a representation language of the world through which it is possible to store information and infer new information from the already-stored information.

Since any form of LOGIC serves as a representation language then it must define the following:

1. Syntax
2. Semantic

The SYNTAX defines all the sentences that are well-formed. The notion of syntax is clear enough in ordinary arithmetic: “x + y = 4” is a well-formed sentence, whereas “x4y+ =” is not.

The SEMANTIC defines the meaning of a well-formed sentence. The meaning of a well-formed sentence is either TRUE or FALSE depending on the world being considered. For example, the semantics for arithmetic specifies that the sentence “x + y = 4” is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1.

Technically speaking, a possible world is called **MODEL**. Therefore, a given well-formed sentence may be true in a certain MODEL but false in another MODEL.

As an example, the sentence “All men have a dick” is true on the planet Earth but it may be false on another planet where there exists life. The planet Earth and the other planet where there exists represent two different MODELS.

If a sentence α is true in model m, we say that **m satisfies α** or sometimes **m is a model of α**. We use the notation M(α) to mean the set of all models of α.

We use the notation α |= β to express that α semantically entails β or in other words that β follows logically from α.

The formal definition of semantic entailment is the following:

α |= β if and only if M(α) ⊆ M(β)

In other words, a sentence α semantic entails β if and only if all models that satisfy α also satisfy β.

In other words, whenever α is true, β is also true.

## Logical Inference

The process through which it is possible to infer new sentences from a set of premises (KB) is called Logical Inference.

The simplest algorithm to infer new sentences from a set of premises is called MODEL CHECKING.

The MODEL CHECKING algorithm works in the following way:

1. Given a sentence α which we want to know whether or not logically follows from a set of premises KB
2. List all models where KB is true in other words find M(KB)
3. If for all models m in M(KB), m is a model of α then α logically follows from KB and so KB |= α

If an inference algorithm m can infer a sentence α from a set of premises KB then we formally denote this this way:

KB |-m α

## Sound Inference Algorithms

A sound inference algorithm is an inference algorithm which derives sentences from a given set of premises that are entailed from the set of premises.

In other words, given an inference algorithm m, m is a sound inference algorithm if and only if given a set of premises KB, KB |-m α if and only if KB |= α.

In other words, an inference algorithm is sound if and only if all sentences that it derives from a set of premises must logically follow from the set of premises.

Soundness is a highly desirable property. An unsound inference procedure essentially makes things up as it goes along.

Model Checking is a sound inference algorithm as it exploits the definition of entailment to derive sentences from a given set of premises.

If KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world.

A screenshot of a cell phone

Description automatically generated

## Complete Inference Algorithms

An inference algorithm is COMPLETE if and only if it can derive ALL sentences which are entailed from the given set of premises.

COMPLETENESS becomes an issue when the premises that can be derived from a given set of premises is INFINITE. In other words, if a given set of premises entails an infinite set of sentences then the algorithm cannot derive all of them.

Fortunately, there are complete inference algorithms for logics that are sufficiently expressive to handle many knowledge bases.

We have said that if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world. However, how can we be sure that the KB of a knowledge-based agent reflects the reality? In other words, how do we know that KB is true in the real world?

Such an issue is called GROUNDING which is defined as the connection between the agent and the real world.

Grounding depends on the quality of the sensors and the learning ability of the agent.

For example, if a learning agent watches a magic show and sees that a person is able to fly, it is very likely that the agent adds this information in its knowledge base. However, such an information is wrong and so its KB does not reflect the reality. In conclusion, the learning ability of such an agent is fallacious.

Thus, one of the problems that AI researchers have when dealing with learning agent is how to make an agent learn new information from the real world which is SURELY TRUE. An agent that learnt wrong information would infer wrong information or would perform wrong actions and so it would not be RATIONAL.

## Propositional Logic

## Syntax

A screenshot of a cell phone

Description automatically generated

Given an implication P => Q, P is called the PREMISE and Q is called the conclusion.

A LITERAL is either a proposition symbol or a negated proposition symbol. In other words, P and ⌐P are both literals.

## Semantics

The semantics of propositional logic fixes the rules to determine the meaning of a sentence based on a particular model. In propositional logic, a model just fixes the truth values of the proposition symbols.

The following is a model-checking algorithm for propositional logic that checks whether a KB entails a sentence α or not .

A screenshot of text

Description automatically generatedA screenshot of a cell phone

Description automatically generatedThe time complexity of the algorithm TT-Entails is O(2n).

Propositional entailment is co-NP-complete (i.e., probably no easier than NP-complete), so every known inference algorithm for propositional logic has a worst-case complexity that is exponential in the size of the input.

## Propositional Theorem Proving

So far, we have shown how to determine entailment by model checking: enumerating models and showing that the sentence must hold in all models. In this section, we show how entailment can be done by **THEOREM PROVING**—applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models. If the number of models is large but the length of the proof is short, then theorem proving can be more efficient than model checking.

Two sentences α and β are **LOGICAL EQUIVALENT** if they are true in the same models. If α and β are logical equivalent then it is denoted as α ≡ β.

An alternative definition of logical equivalence is the following:

α ≡ β if and only if α |= β and β |= α

Such a definition implies that M(α) = M(β).

A sentence is **VALID** (aka **TAUTOLOGY**) if it is true in all models. Because the sentence True is true in all models, every tautology is logically equivalent to True.

From our definition of entailment, we can derive the **DEDUCTION THEOREM**, which was known to the ancient Greeks:

For any sentences α and β, α |= β if and only if the sentence (α ⇒ β) is a tautology

Hence, we can decide if α |= β by checking that (α ⇒ β) is true in every model.

The DEDUCTION THEOREM says two thing:

1. If α |= β then (α ⇒ β) is a tautology
2. If (α ⇒ β) is a tautology then α |= β

A sentence α in propositional logic is **SATISFIABLE** if there exists a model m which satisfies α.

Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence. The problem of determining the satisfiability of sentences SAT in propositional logic—the SAT problem—was the first problem proved to be NP-complete. Many problems in computer science are really satisfiability problems. For example, all the constraint satisfaction problems in Chapter 6 ask whether the constraints are satisfiable by some assignment.

Validity and satisfiability are of course connected:

1. α is valid if and only if ¬α is unsatisfiable
2. α is satisfiable if and only if ¬α is not valid

From the sentences above we can derive the following useful result:

α |= β if and only if the sentence (α ∧ ¬β) is unsatisfiable

We already know that α |= β if and only if the sentence (α => β) is a valid sentence. Since (α => β) is logically equivalent to (¬α ∨ β) then we can conclude the following:

α |= β if and only if the sentence (¬α ∨ β) is a valid sentence

Since α is valid if and only if ¬α is unsatisfiable then we conclude the following:

α |= β if and only if the sentence (α ∧ ¬β) is unsatisfiable

Notice that ⌐(¬α ∨ β) ≡ (α ∧ ¬β).

Proving β from α by checking the unsatisfiability of (α ∧ ¬β) corresponds exactly to the standard mathematical proof technique of reductio ad absurdum (literally, “reduction to an absurd thing”). It is also called **PROOF BY REFUTATION** or **PROOF BY CONTRADICTION**. One assumes a sentence β to be false and shows that this leads to a contradiction with known axioms α. This contradiction is exactly what is meant by saying that the sentence (α ∧ ¬β) is unsatisfiable.

We can apply any of the search algorithms in Chapter 3 to find a sequence of steps that constitutes a proof. We just need to define a 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.

Thus, searching for proofs is an alternative to enumerating models. In many practical cases finding a proof can be more efficient because the proof can ignore irrelevant propositions, no matter how many of them there are. For example, the proof given earlier leading to ¬P1,2 ∧ ¬P2,1 does not mention the propositions B2,1, P1,1, P2,2, or P3,1. They can be ignored because the goal proposition, P1,2, appears only in sentence R2; the other propositions in R2 appear only in R4 and R2; so R1, R3, and R5 have no bearing on the proof. The same would hold even if we added a million more sentences to the knowledge base; the simple truth-table algorithm, on the other hand, would be overwhelmed by the exponential explosion of models.

Therefore, when applying any search algorithm to find a proof for a sentence α given a certain KB, the propositions of the KB which must be taken into account are those that contain the proposition symbols of the sentence α and if these sentences contain other proposition symbols then also the sentences that contain these other proposition symbols must be taken into account. All other sentences can be temporarily deleted from the KB when a search algorithm is used to prove α.

One final property of logical systems is **MONOTONICITY**, which says that as information is added to the knowledge base, the set of entailed sentences can only increase but not decrease.

Then given that a logic system is MONOTONIC we can formally state the following:

For any sentence α and β, if KB |= α then KB ∧ β |= α

Nonmonotonic logics, which violate the monotonicity property, capture a common property of human reasoning: changing one’s mind.

Therefore, if a logic system is NONMONOTONIC if if KB |= α then it is not guaranteed that KB ∧ β |= α.

## Proof Monotonicity of Propositional Logic

A close up of a logo

Description automatically generated

Monotonicity can be interpreted in another way.

Given that KB |= α then by definition we know that M(KB) ⊆ M(α). If we add a sentence β then

KB ∧β|= α because M(KB ∧β) ⊆ M(KB) ⊆ M(α).

The monotonicity of propositional logic proves that the algorithm described above to search for a proof given a set of premises is sound.

Indeed, if to prove α we only need a set of sentences Q such that Q ⊆ KB then if Q |= α then

Q ∧ (KB - Q) |= α

Notice that Q ∧ (KB - Q) = KB

Monotonicity means that inference rules can be applied whenever suitable premises are found in the knowledge base—the conclusion of the rule must follow regardless of what else is in the knowledge base.

## Complete Inference Algorithms

An inference algorithm is complete when the search algorithm is complete and the proof system being used is complete as well.

## Proof by resolution

Proof by resolution is a sound and complete proof system which when coupled with a complete search algorithm leads to a complete inference algorithm.

The proof by resolution system includes only one inference rule which is called the RESOLUTION RULE.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

It is very easy to prove that the resolution rule is sound as given two disjunctions of literals that have a complementary literal lead to the fact that one of the two disjunctions without the complementary literal is true nevertheless. Therefore, the disjunction of the two disjunctions without the complementary literal is true as well.

What is surprising about the proof by resolution (which consists of only one inference rule) is that given any sentence α and β in propositional logic, a proof by resolution solver can always determine whether α |= β or not.

## Conjunctive normal form

A disjunction of literals is called CLAUSE.

In propositional logic, every sentence is logically equivalent to a conjunction of clauses.

A sentence expressed as a conjunction of clauses is in the conjunctive normal form.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

## Inference using the resolution proof system

By using the resolution proof system, it is possible to infer that KB |= α by showing that t (KB ∧ ¬α) is unsatisfiable. In other words, we prove that KB |= α by using a proof by contradiction.

In order to prove that KB |= α, the resolution-based inference algorithm does the following:

1. Converts (KB ∧ ¬α) into CNF
2. For every two clauses applies the resolution rule
3. If the resolution rule can be performed then the inferred sentence is added to the set of clauses
4. Continues doing so until one of the two following scenarios occurs
   1. there are no new clauses that can be added, in which case KB does not entail α
   2. two clauses resolve to yield the empty clause, in which case KB entails α.

Indeed, if the empty clause is inferred then the two clauses had to be α and ⌐α. However, this is a contradiction and so (KB ∧ ¬α) is unsatisfiable which means that KB |= α.

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generatedIn other words, the algorithm based on the resolution proof system is complete because either the empty clause is added to the resolution closure or the whole resolution closure is generated

As soon as the empty clause is added then KB |= α, if the whole resolution closure is generated without adding the empty clause then KB does not entail α.