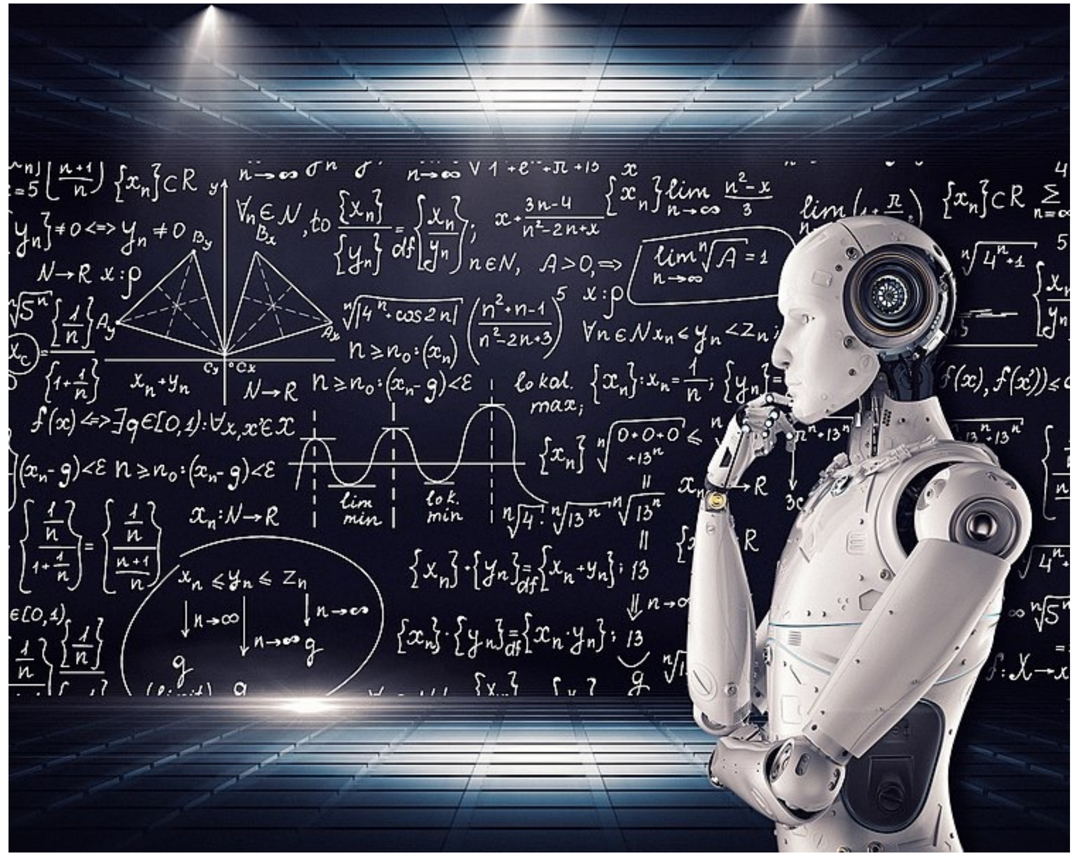
Lecture 20: Automatic Theorem-Proving

Mark Hasegawa-Johnson



Outline

- Propositional Logic
- First-Order Logic
- Proving “there exists” vs. “for all” theorems
- Variable normalization & Unification
- Search: Forward-chaining & Backward-chaining

Propositional Logic

- “Propositions” are statements that can be either True or False
  - P=“an iguana is an animal with scales”
  - Q=“an iguana is an animal that breathes air”
  - R=“an iguana is a reptile”
- Propositional logic studies the relationships among propositions.

Symbolic Logic Functions

- Unary functions (map one proposition to another)
  - $\neg$ (not): $\{F,T\} \rightarrow \{T,F\}$
- Binary functions (map two propositions to one)
  - $\wedge$ (and): $\{(F,F),(F,T),(T,F),(T,T)\} \rightarrow \{F,F,F,T\}$
  - $\vee$ (or): $\{(F,F),(F,T),(T,F),(T,T)\} \rightarrow \{F,T,T,T\}$
- Rules (generate one proposition from another)
  - $P \implies Q$ (implies): if $\mathrm{P} = \mathrm{T}$ we can infer $\mathrm{Q} = \mathrm{T}$
  - $P \Leftrightarrow Q$ (equivalent): infer either P or Q to match the value of the other

Outline

- Propositional Logic
- First-Order Logic
- Proving “there exists” vs. “for all” theorems
- Variable normalization & Unification
- Search: Forward-chaining & Backward-chaining

First Order Logic

- Propositional logic says that propositions can be constructed from other propositions
- First-order logic says propositions can also be constructed by applying predicates to constants

Predicates, Constants, Variables, Propositions, and Rules

- A **predicate** is like a function, that can be applied to some **variables**.
  - $BreathesAir(x)$ is true if and only if x breathes air.
- A **constant** is a particular object in the real world, which can be the value of the argument of a function:
  - $reptiles$ is a constant
- A **proposition** is a predicate applied to a constant
  - $BreathesAir(reptiles)$ is true if and only if reptiles breathes air.
- A **rule** is an implication or equivalence that's true for all values of its variable
  - $BreathesAir(x) \wedge Scales(x) \Longrightarrow Reptile(x)$: everything that breathes air and has scales is a reptile.

Theorem Proving

An automatic theorem-prover uses a database of known facts and known rules to prove a theorem. For example, suppose we know that:

- Iguanas have scales: $Scales(iguanas)$
- Iguanas breathe air: $BreathesAir(iguanas)$
- Anything that breathes air and has scales is a reptile: $BreathesAir(x) \wedge Scales(x) \Longrightarrow Reptile(x)$

And suppose we want to prove that:

- Iguanas are reptiles: $Reptile(iguanas)$

Theorem Proving by Forward-Chaining

- Forward-chaining is the process of applying rules to facts in order to prove more facts.
- For example, let's start by combining these two facts:
$$BreathesAir(iguanas) \wedge Scales(iguanas)$$
- Now let's apply this rule:
$$BreathesAir(x) \wedge Scales(x) \Longrightarrow Reptile(x)$$
- The result: we have proven that:
$$Reptile(iguanas)$$

Theorem-Proving by Forward-Chaining

Notice that, when we're forward-chaining, each step of the process just expands the set of available facts. If we start with the following database of facts:

$$BreathesAir(iguanas) \wedge Scales(iguanas)$$

… and if we apply the rule $BreathesAir(x) \wedge Scales(x) \Longrightarrow Reptile(x)$, then the database can only get larger. It becomes this:

$$BreathesAir(iguanas) \wedge Scales(iguanas) \wedge Reptile(iguanas)$$

Forward-chaining just keeps going, until the fact we want is part of the database, or until we can't prove any more facts.

Outline

- Propositional Logic
- First-Order Logic
- Proving “there exists” vs. “for all” theorems
- Variable normalization & Unification
- Search: Forward-chaining & Backward-chaining

Quantification

- It is sometimes useful to express compound propositions that are true for some values of their variables, but not all.
- To do this, we introduce two new symbols, called quantifiers:
- ∃ (there exists)
  - Suppose P is the proposition $\mathrm{P} = \exists x\colon F(x)$
  - Then $P = T$ if and only if, for at least one value of the variable x, $F(x) = T$
- ∀ (for all)
  - Suppose P is the proposition $\mathrm{P} = \forall x\colon F(x)$
  - Then $P = T$ if and only if, for all values of the variable x, $F(x) = T$

Existence theorems

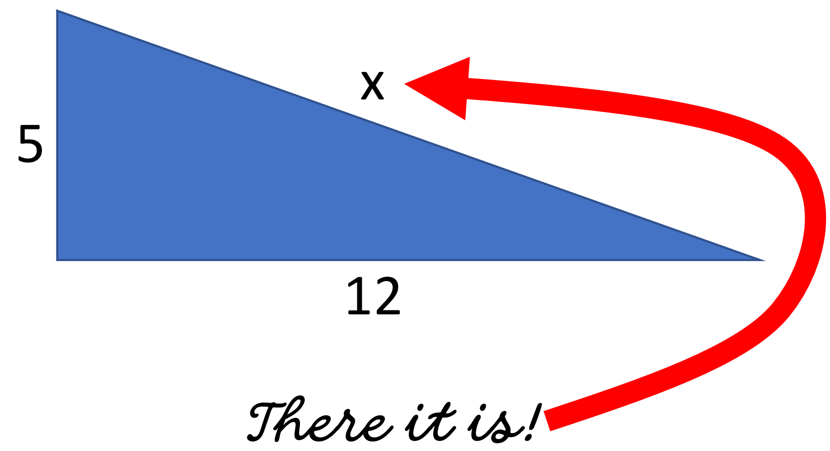
An existence theorem is a theorem of the form "there exists an x such that F(x)," which we write as

$$\exists x: F(x)$$

An existence theorem:

- … can be **proven** by finding any x that satisfies the conditions.
- …but to **disprove** the statement $\exists x: F(x)$, you must prove that, for all x that can possibly exist, $\neg F(x)$

Find x:

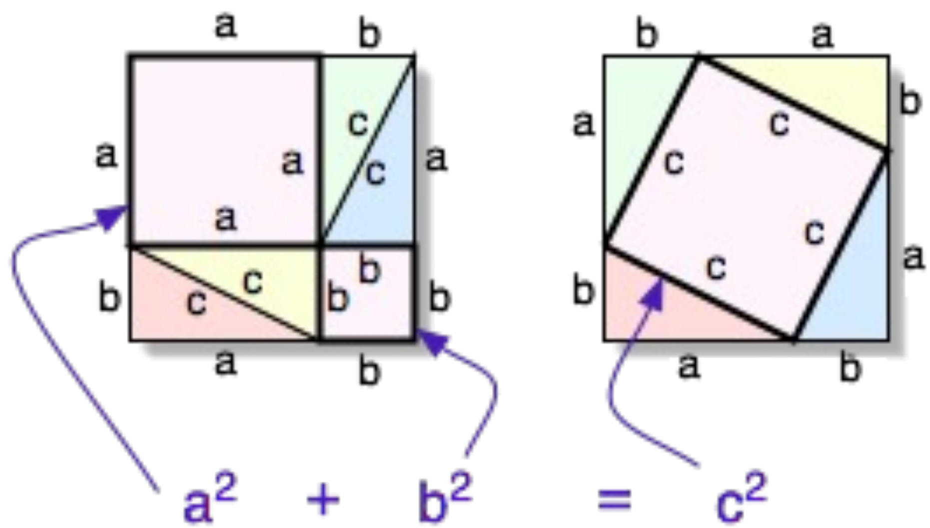


Proving and disproving theorems

A universality theorem claims that $F(x)$ is true for all x. We write it as

$$\forall x: F(x)$$

- To **disprove** the statement $\forall x: F(x)$, you just need to find a counterexample, i.e., you just need to prove that $\exists x: \neg F(x)$.
- To **prove** a universality theorem, you need to show that there could not possibly be any x that violates $F(x)$.



Proof that, for any right triangle with hypotenuse c and sides a and b, $a^2 + b^2 = c^2$. The existence of any right triangle violating this theorem would violate the proposition that the area of a rectangle with sides a and b is ab. Public domain image, https://commons.wikimedia.org/wiki/File:Pythagorean_proof.png

Two types of proofs

- To **prove an existence theorem**, or disprove a universality theorem, you just need to find an x that satisfies the statement.
  - This is done using forward-chaining or backward-chaining with ***unification***.
  - I will spend the rest of today's lecture talking about this.
- To disprove an existence theorem, or **prove a universality theorem**, you need to prove that the existence of any such x would contradict known true propositions.
  - This is done using a proof method called ***resolution***.
  - We will not cover it in this course.

Outline

- Propositional Logic
- First-Order Logic
- Proving “there exists” vs. “for all” theorems
- Variable normalization & Unification
- Search: Forward-chaining & Backward-chaining

Theorem proving

Consider the statements:

1. Chocolate is sweet
2. If something is sweet, then Jack likes it

From those, can we prove that:

3. There is somebody who likes something

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Variable Normalization

1. $Sweet(chocolate)$
2. $\forall x: Sweet(x) \Rightarrow Likes(jack, x)$
3. $\exists x, y: Likes(x, y)$

Propositions (1) and (2) prove proposition (3), but this fact is obfuscated by the different meanings of the variable x in propositions (2) versus (3).

Automatic proof needs normalized variables.

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Variable Normalization

Variable normalization replaces the old variable names with new variable names such that:

1. If the same variable name occurs in different rules, change it so that ***<u>each rule uses a different set of variable names</u>***
2. If the same variable occurs multiple times in one rule, its multiple instances still have the same name

For example, the example on the previous page could be normalized to:

$$Sweet(chocolate)$$
$$Sweet(x_1) \Rightarrow Likes(jack, x_1)$$
$$\exists x_2, y_1 : Likes(x_2, y_1)$$

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Unification

Now we have:

$$Sweet(chocolate)$$
$$Sweet(x_1) \Rightarrow Likes(jack, x_1)$$

From these, can we prove that:

$$\exists x_2, y_1 : Likes(x_2, y_1)$$

Obviously, we somehow need to determine that $x_2 = jack$ and $y_1 = chocolate$. This is called unification.

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Unification

1. $Sweet(chocolate)$
2. $Sweet(x_1) \Rightarrow Likes(jack, x_1)$
3. $\exists x_2, y_1 : Likes(x_2, y_1)$

Define C to be the set of all constants, and define $\mathcal{V}_P$ to be the set of variables used in proposition P. Normalization guarantees that $\mathcal{V}_P \cap \mathcal{V}_Q$ is the empty set. Thus:

$$\mathcal{V}_2 = \{x_1\}$$
$$\mathcal{V}_3 = \{x_2, y_1\}$$
$$C = \{jack, chocolate\}$$

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Unification

1. $Sweet(chocolate)$
2. $Sweet(x_1) \Rightarrow Likes(jack, x_1)$
3. $\exists x_2, y_1: Likes(x_2, y_1)$

Unification finds a substitution $S: \{\mathcal{V}_P, \mathcal{V}_Q\} \rightarrow \{\mathcal{V}_Q, C\}$ that unifies the propositions P and Q, i.e., makes them into one unified proposition. For example, the substitution

$$S: \{x_1, x_2, y_1\} \rightarrow \{y_1, jack, y_1\}$$

…unifies propositions (2) and (3) to the unified proposition:

$$Sweet(y_1) \Rightarrow Likes(jack, y_1)$$

$$\exists y_1: Likes(jack, y_1)$$

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Unification in more general terms

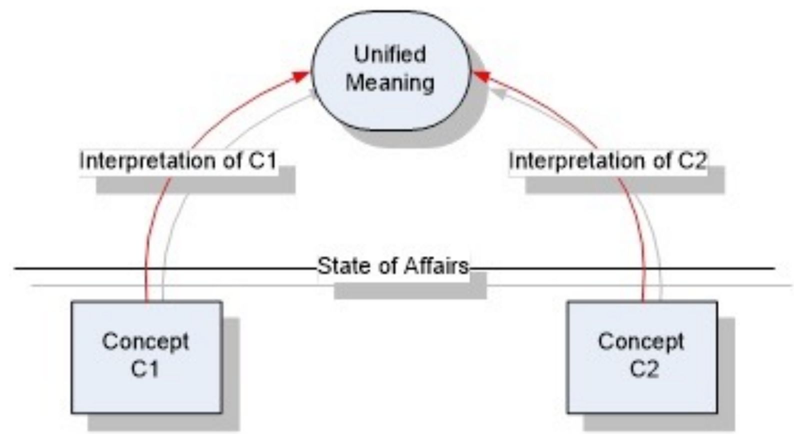
The word "unification" is more generally defined as:

- …mapping of two source expressions
- … onto a single target expression, with some standardized format, such that
- … the target expression implies both of the source expressions.

$$Sweet(y_1) \Rightarrow Likes(jack, y_1)$$
$$\exists y_1: Likes(jack, y_1)$$

…implies that…

$$Sweet(x_1) \Rightarrow Likes(jack, x_1)$$
$$\exists x_2, y_1: Likes(x_2, y_1)$$



Outline

- Propositional Logic
- First-Order Logic
- Quantification
- Proving “there exists” vs. “for all” theorems
- Variable normalization & Unification
- Search: Forward-chaining & Backward-chaining

Forward-chaining

Forward-chaining is a search-based method of proving a theorem, T:

- Starting state: a database of known true propositions, $\mathcal{D} = \{P_1, P_2, \dots\}$
- Actions: unify one of the known truths, P_i, with the antecedent of a known rule of the form $P \Longrightarrow Q$.
- Neighboring states: if P_1 unifies to P creating $\mathrm{S}(\mathrm{P}) = \mathrm{S}(P_1)$, then create the new database $\mathcal{D}' = \{P_1, P_2, \dots, S(Q)\}$
- Termination: search terminates when we find a database containing T

Example of forward-chaining

Database: $\mathcal{D} = \{Sweet(chocolate)\}$

Rule: $Sweet(x_1) \Rightarrow Likes(jack, x_1)$

Theorem: $\exists x_2, y_1: Likes(x_2, y_1)$

Proof:

1. Unify $Sweet(x_1)$ to $Sweet(chocolate)$. Result:
   $\mathcal{D}' = \{Sweet(chocolate), Likes(jack, chocolate)\}$

2. Unify $Likes(x_2, y_1)$ to $Likes(jack, chocolate)$. Result:

$$\mathcal{D}'' = \left\{ \begin{matrix} Sweet(chocolate), Likes(jack, chocolate), \\ \exists jack, chocolate: Likes(jack, chocolate) \end{matrix} \right\}$$

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Forward-chaining

- **What's Special About Theorem Proving**:
  - A state, at level n, can be generated by the combination of several states at level n-1.
- **Definition: Forward Chaining** is a search algorithm in which each action
  - generates a new proposition,
  - …and adds it to the database of known propositions.

Backward-chaining

Backward-chaining is a method of proving a result, R:

- Starting state: a set of "goals" containing only one goal, the result to be proven, $\mathcal{G} = \{R\}$
- Actions: the set of possible actions is defined by
  1. A set of rules of the form $P \Longrightarrow Q$, and
  2. A set of known true propositions.
- Neighboring states: if Q unifies with some $Q' \in \mathcal{G}$ then
  - Remove Q' from $\mathcal{G}$
  - Replace it with P.
- Termination: search terminates if all propositions in the goalset are known to be true.

Example of backward-chaining

Theorem: $\mathcal{G} = \{\exists x_2, y_1 : Likes(x_2, y_1)\}$

Rules:

$$\mathbb{T} \Rightarrow Sweet(chocolate)$$
$$Sweet(x_1) \Rightarrow Likes(jack, x_1)$$

Proof step 1:

Unify $Likes(jack, x_1)$ to $Likes(x_2, y_1)$. Result:

$$\mathcal{G}' = \{Sweet(x_1)\}$$

Proof step 2:

Unify $Sweet(x_1)$ to $Sweet(chocolate)$. Result:

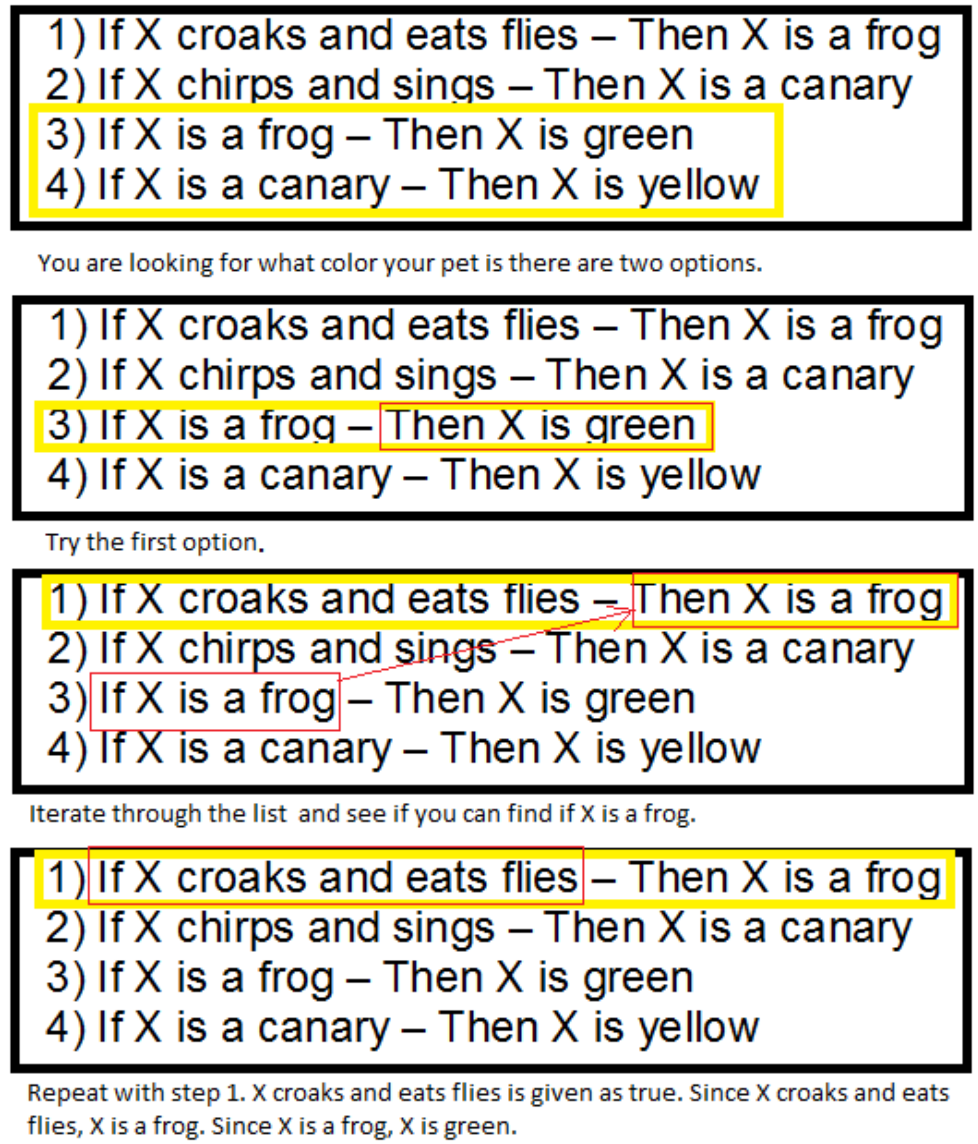
$$\mathcal{G}'' = \{\mathbb{T}\}$$

Public domain image, https://commons.wikimedia.org/wiki/File:Tomando_chocolate_1892_Gumersindo_Pardo_Reguera.jpg

Another example (from Wikipedia)

- Goal: $\{Green(fritz)\}$
- Proof step 1: $\{Frog(fritz)\}$
- Proof step 2: $\{Croaks(fritz) \wedge EatsFlies(fritz)\}$

If $Croaks(fritz)$ and $EatsFlies(fritz)$ are known to be true, then we have successfully proven that fritz is green.

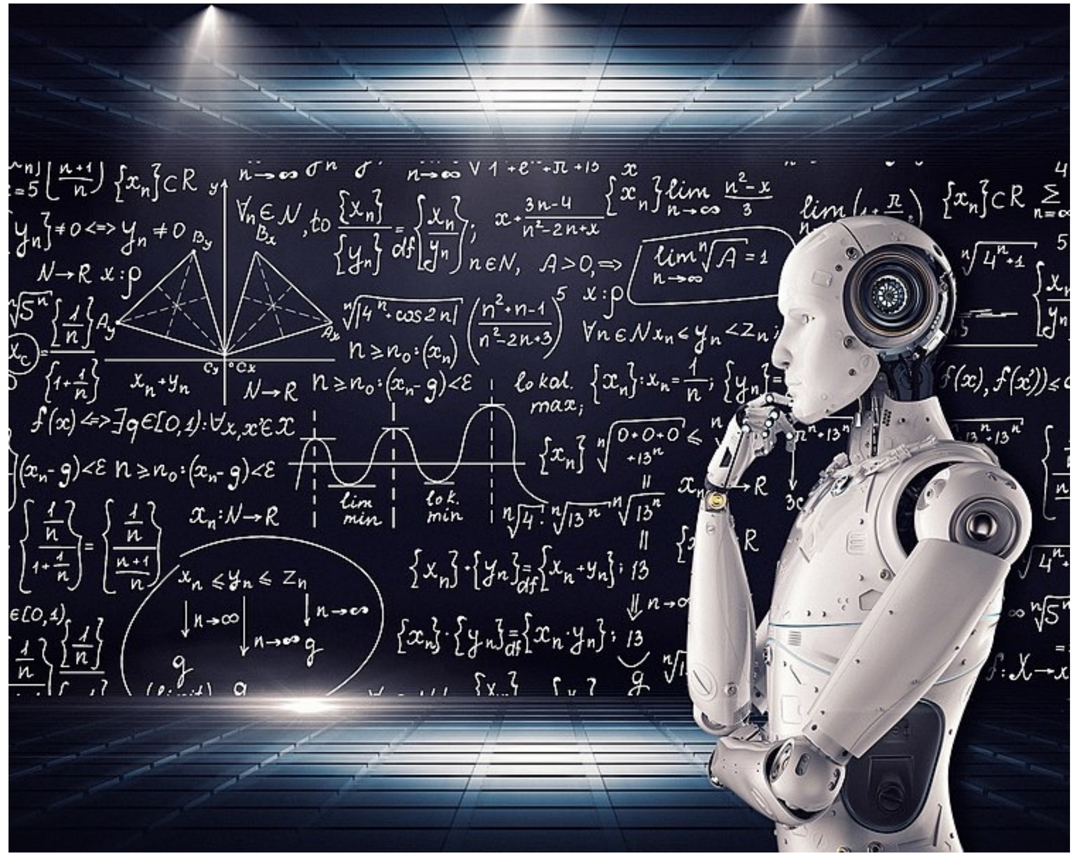


Backward-chaining

- **What Else is Special About Theorem Proving:**
  - The “goal set” is a set of propositions that need to be proven.
- **Definition: Backward Chaining** is a search algorithm in which
  - State = {goal set}
  - Action = apply a known rule, backward: replace the goal’s *consequent* (its RHS) with its *antecedents* (its LHS)
  - Termination = the goalset contains nothing but truth

Quiz

Try the quiz!



Comparison of forward-chaining and backward-chaining

Forward-chaining:

- Time complexity: $\mathcal{O}\{b^d\}$, where b is the number of rules that can be applied at any step, and d is the number of steps necessary to prove the theorem
- Space complexity: to make it easy to retrieve the database for each state, each state should save a complete copy of the database!

Backward-chaining:

- Time complexity: $\mathcal{O}\{b^d\}$, where b is the number of rules that can be applied at any step, and d is the number of steps necessary to prove the theorem
- Space complexity: each state only needs to save a copy of the goalset, which is usually much smaller than the database.

What about A*?

A* is important for any successful theorem-prover. For backward-chaining, we could use heuristics that depend on the propositions in the goalset, $\mathcal{G} = \{Q_1, Q_2, Q_3, \ldots\}$. We could use $\hat{h}(\mathcal{G}) = \hat{h}(Q_1) + \hat{h}(Q_2) + \cdots$ where:

- $\hat{h}(Q_i) = 0$ if Q_i already known to be true.
- $\hat{h}(Q_i) = 1$ if Q_i has the same form as a true proposition; maybe it is possible to unify them (1 step).
- $\hat{h}(Q_i) = 2$ if Q_i has the same form as the Q in a rule $P \Longrightarrow Q$; maybe we can unify them (1 step) and then prove P (1 more step).
- $\hat{h}(Q_i) = \infty$ otherwise, because it's unprovable.

Summary

- Proving “there exists” theorems: find an x that satisfies the statement
- Variable normalization: each rule uses a different set of variable names
- Unification: Find a substitution $S\colon\{\mathcal{V}_P, \mathcal{V}_Q\} \rightarrow \{\mathcal{V}_Q, C\}$ such that $\mathrm{S}(P) = S(Q) = U$, or prove that no such substitution exists
- Forward-chaining: Search problem in which each action is a unification, and the state is the set of all known true propositions
- Backward-chaining: Search problem in which each action is a unification, and the state is the goal (the proposition whose truth needs to be proven)