

Lecture 5: DPLL and First-Order Logic

September 16th, 2026



Gabriel Poesia

Assistant Professor, CSE

University of Michigan, Ann Arbor

poesia@umich.edu

<https://gpoesia.com>

1. Propositional logic: recap
2. DPLL
3. First-order logic
4. Reasoning in first-order logic
5. Logic in practice: Z3

PROPOSITIONAL LOGIC:

RECAP

A language and its meaning

In propositional logic, we start with **atoms** $P, Q, R, \dots$

We build sentences using $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

$$(P \rightarrow Q) \wedge (P \vee R)$$

An **interpretation** assigns each atom a truth value in $\{\perp, \top\}$.

The connectives then determine the truth of every sentence.

DEFINITION (MODEL)

An interpretation m is a **model** of α if α is true in m , written $m \models \alpha$.

For a KB, a model must make *every* sentence in the KB true.

Entailment is about every model

DEFINITION (ENTAILMENT)

$KB \models \beta$ if every model of KB is also a model of β .

$$\forall m : m \models KB \implies m \models \beta$$

Example: $\{P \vee Q, \neg P\} \models Q$.

How can we show that an entailment does *not* hold?

Find one **countermodel**: $m \models KB$ but $m \not\models \beta$.

For example, $P \vee Q \not\models P$: take $P = \perp, Q = \top$.

Equivalence and CNF

$\alpha \equiv \beta$ means that α and β have exactly the same models.

We used equivalences to convert sentences to **conjunctive normal form** (CNF):

DEFINITION (CNF)

A **CNF sentence** is a conjunction of clauses.

A **clause** is a disjunction of literals.

A **literal** is an atom or its negation.

$$(P \vee \neg Q) \wedge (Q \vee R) \wedge \neg R$$

Satisfiable

True in at least one interpretation

Unsatisfiable

False in all interpretations

Resolution can refute an unsatisfiable set of propositional clauses.

Entailment as one satisfiability query

Write K for the conjunction of the sentences in a finite KB.

To ask whether $KB \models \beta$, try to find a countermodel:

THEOREM

$$KB \models \beta \iff K \wedge \neg\beta \text{ is unsatisfiable.}$$

A satisfying assignment for $K \wedge \neg\beta$ is exactly a world where the premises hold and the query does not.

SAT

We found a countermodel; the query is not entailed

UNSAT

No countermodel exists; the query is entailed

We saw this is decidable, via the naïve truth-table method

**Can we search for a satisfying assignment
without enumerating all 2^n worlds?**

DPLL

SAT is a Boolean CSP

DEFINITION (SAT)

The **Boolean satisfiability problem** (SAT) asks whether a propositional formula has a satisfying assignment.

For a formula in CNF, this is a CSP:

Variables

Atomic propositions

Domains

$\{\perp, \top\}$

Constraints

At least one true literal per clause

We already know how to approach this: **backtracking + propagation**.

For entailment, solve $K \wedge \neg\beta$.

If SAT, then this gives a countermodel. If UNSAT, that proves the query.

Lots of real-world applications: software and hardware verification, planning, scheduling problems, mathematics...

Simplifying a partial assignment

At the basis we'll do backtracking search. So we'll choose a variable and try assigning it to $\top$ and $\perp$
But when we assign, we can simplify the formula:

1. Delete clauses that already have a true literal
2. Delete false literals from the remaining clauses

Example: suppose we're solving $(P \vee Q) \wedge (\neg P \vee R)$ and we set $P = \top$. How can we simplify now?

$$\underbrace{(P \vee Q)}_{\text{satisfied}} \wedge \underbrace{(\neg P)}_{\text{false}} \vee R \implies R$$

This gives us two stopping conditions:

Some clause becomes empty

UNSAT branch, since we can't satisfy the empty clause.
Backtrack.

The list of clauses becomes empty

SAT: All original clauses are satisfied!

In the first case, just one branch fails. We need to try other branches. UNSAT only if we exhaust them.

In the second case, unassigned atoms can take any value. No need to assign everything!

Unit propagation

Suppose $\neg P \vee Q$ is a clause and we have assigned $P = \top$.

After simplification, only Q remains. If there is a satisfying assignment, what value will Q take?

$Q = \top$ is forced.

DEFINITION (UNIT PROPAGATION)

Assign the literal of a unit clause to true, simplify, and repeat while new unit clauses appear.

If the unit clause is $\neg Q$, assign $Q = \perp$.

Same idea as a CSP variable with one supported value left: we don't need to branch.

These consequences depend on the decisions made in the current branch.

Unit propagation: example

Consider the formula:

$$(P) \wedge (\neg P \vee Q) \wedge (\neg Q \vee R) \wedge (\neg R)$$

Unit clause	Forced assignment	Consequence
P	$P = \top$	$\neg P \vee Q$ becomes Q
Q	$Q = \top$	$\neg Q \vee R$ becomes R
R	$R = \top$	$\neg R$ becomes $\square$

We found a contradiction **without making any decisions**: the formula is unsatisfiable.

Could also start with $\neg R$ and propagate in the other direction.

As with AC-3, one reduction can trigger others.

Sometimes a value is not forced, but we still have a clear safe choice.

Safe here means: if the original formula is satisfiable, our choice keeps it satisfiable

Specifically, we can make a safe choice for any **pure symbol**:

DEFINITION (PURE SYMBOL)

An atom is **pure** if it occurs only positively (i.e., not negated) or only negatively (i.e., negated) in the remaining clauses.

$$(S \vee P) \wedge (\neg P \vee \neg Q) \wedge (S \vee \neg Q)$$

S only appears positively. What happens if we set $S = \top$?

Every clause containing S is satisfied, and all other clauses are unchanged.

Any other pure symbol here?

Yes, $\neg Q$. Similarly, we can set an atom that only occurs negatively to false.

⚠ CAREFUL

This choice need not be entailed! Here, $S = \perp, P = \top, Q = \top$ is also a model. We preserve the *existence* of a solution, not every solution.

An atom can become pure after we satisfy some clauses.

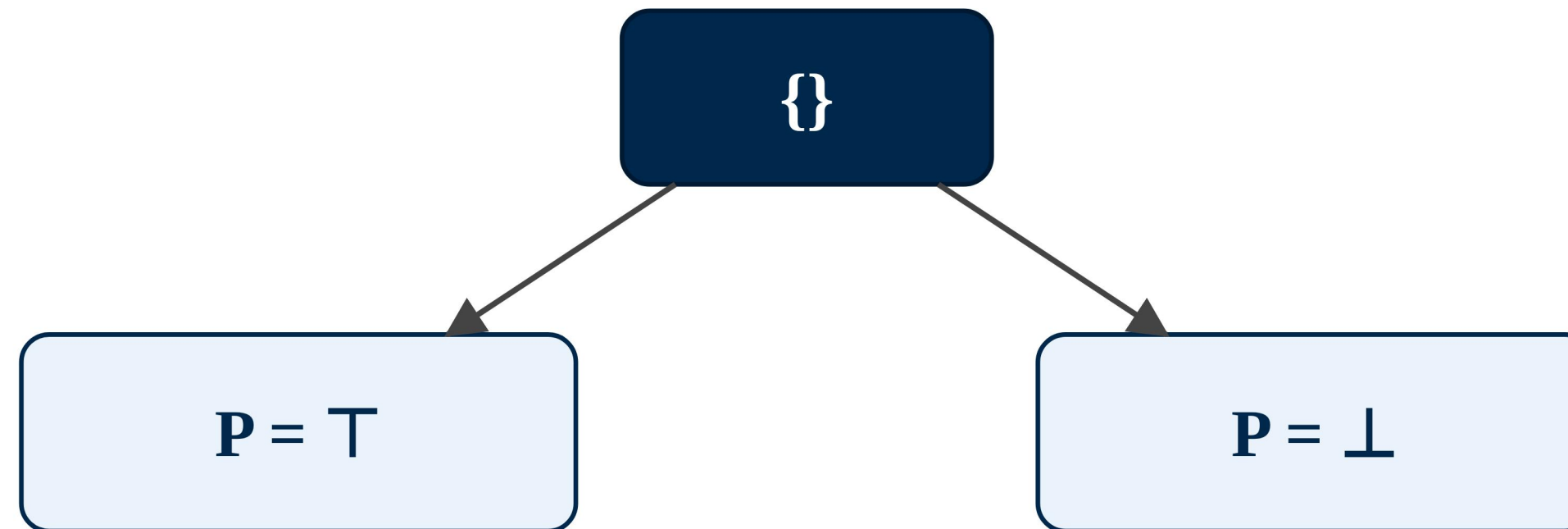
When propagation stops

Consider the formula:

$$F = (\neg P \vee Q) \wedge (\neg P \vee \neg Q) \wedge (P \vee R) \wedge (\neg R \vee \neg Q)$$

No unit clauses, and no pure symbols. What can we do?

Choose an unassigned atom, say P , and branch on its value.



Every model sets P to either true or false, so the two branches cover all possibilities.

As in CSP backtracking, each branch keeps its own assignment and propagated values.

Backtracking: first branch

$$F = (\neg P \vee Q) \wedge (\neg P \vee \neg Q) \wedge (P \vee R) \wedge (\neg R \vee \neg Q)$$

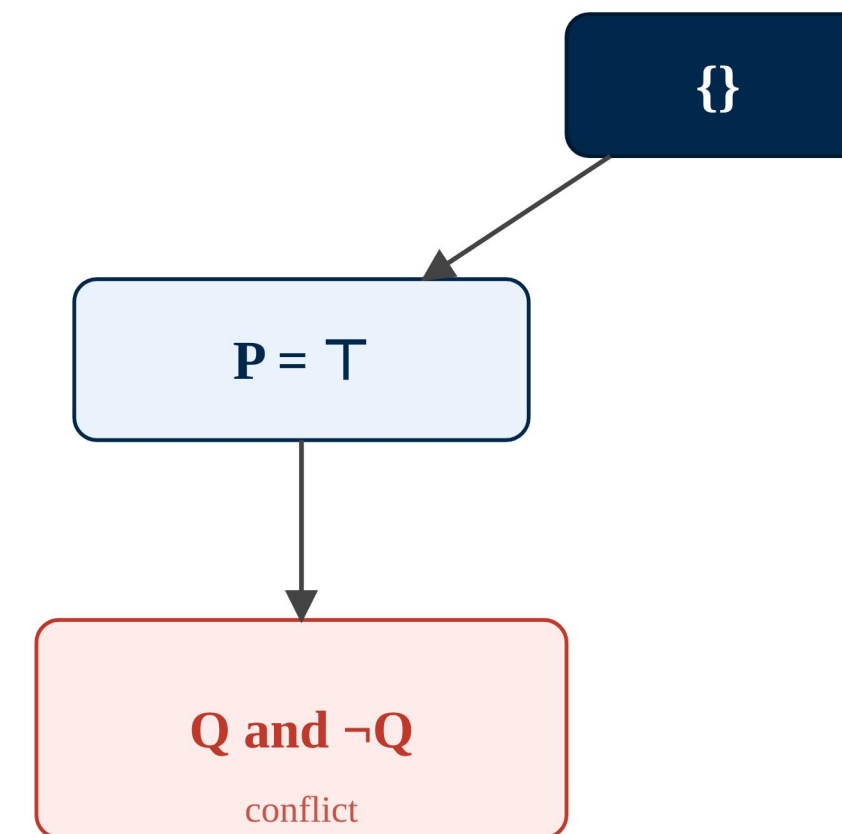
Say we try $P = \top$ first.

Clause	After $P = \top$
$\neg P \vee Q$	Q
$\neg P \vee \neg Q$	$\neg Q$
$P \vee R$	Satisfied: delete
$\neg R \vee \neg Q$	Unchanged

Propagate $Q = \top$. The clause $\neg Q$ becomes empty.

No model extends $P = \top$.

Undo the decision **and its consequences**, then try the other value.



Backtracking: second branch

$$F = (\neg P \vee Q) \wedge (\neg P \vee \neg Q) \wedge (P \vee R) \wedge (\neg R \vee \neg Q)$$

Now try $P = \perp$.

The first two clauses are satisfied. We are left with:

$$(R) \wedge (\neg R \vee \neg Q)$$

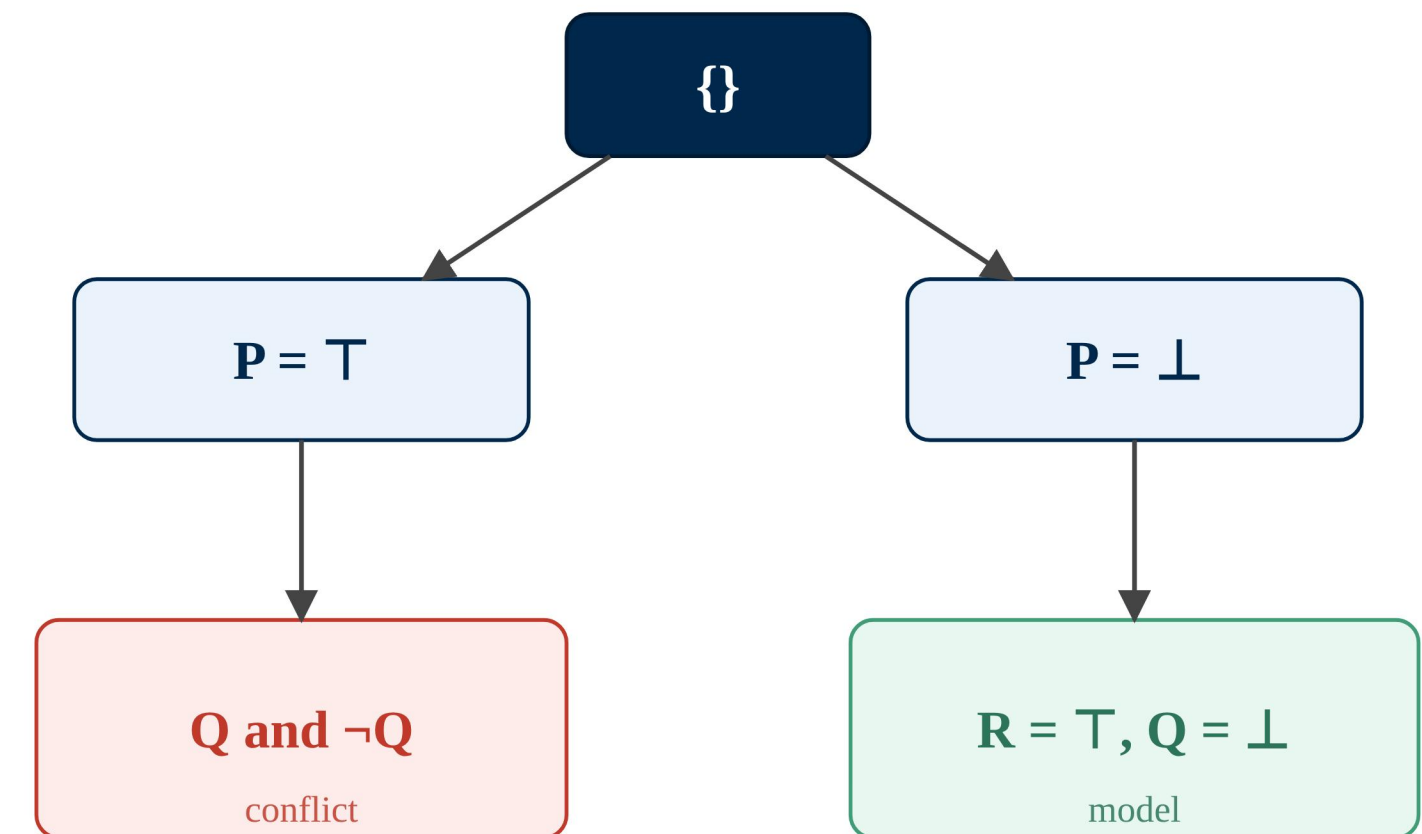
Unit R forces $R = \top$. The second clause becomes $\neg Q$.

Then $Q = \perp$ is forced. No clauses remain!

$$P = \perp, Q = \perp, R = \top$$

Check: the original clauses are satisfied by $\neg P, \neg P, R$, and $\neg Q$.

We didn't need to enumerate all eight complete assignments.



DPLL: pseudocode

Combining these ideas gives **DPLL** (Davis-Putnam–Logemann-Loveland, 1961).

Start with $DPLL(F, \{\})$:

```
1  function DPLL(F, assignment):
2      C = SIMPLIFY(F, assignment)
3      if C has an empty clause: return false
4      if C has no clauses: return true
5      if C has a unit literal L:
6          return DPLL(F, EXTEND(assignment, L=true))
7      if C has a pure literal L:
8          return DPLL(F, EXTEND(assignment, L=true))
9      P = choose an unassigned atom occurring in C
10     return DPLL(F, EXTEND(assignment, P=true))
11         or DPLL(F, EXTEND(assignment, P=false))
```

EXTEND returns a new assignment; $L = \text{true}$ means satisfy the literal (including its sign). OR short-circuits.

This version returns a Boolean. To get a model, also return the successful assignment.

DPLL: guarantees and cost

Why does DPLL find a model whenever one exists?

- **Unit propagation:** every model extending the branch must agree with the forced value
- **Pure symbols:** if a model exists, one exists with the chosen value
- **Branching:** the two values cover all remaining possibilities

Every recursive call assigns a new atom, so the depth is at most n .

THEOREM

DPLL terminates and returns true iff the input formula is satisfiable.

Worst case: exponentially many branches, with polynomial simplification work per node.

In practice, propagation can eliminate large subtrees before we reach complete assignments.

We can also learn a clause explaining a conflict -- Conflict-Driven Clause Learning (CDCL).

LIVE CODING!

FIRST-ORDER LOGIC

One atom for every fact

In the Wumpus world, we used a different atom for every possible fact:

$$P_{1,1}, P_{1,2}, P_{2,1}, \dots$$

We then wrote a separate breeze rule for every cell.

The subscripts help *us* read the names. Propositional logic sees each one as an unrelated symbol.

It cannot look inside $P_{1,2}$ and recognize an object, a location, or a relation.

With a finite known grid, we can enumerate the rules. But what if objects arrive later, or there are infinitely many?

Can we say “every cell has a breeze iff some adjacent cell has a pit” directly?

UNIVERSITY OF
MICHIGAN

First-order logic gives structure to the things we talk about.

Symbol	Role	Examples
Constants	Name objects	<i>alice, CSE592, zero</i>
Functions	Map objects to objects	<i>AdvisorOf(alice), Next(x)</i>
Predicates	Describe properties and relations	<i>Student(x), Takes(x, c)</i>

Variables $x, y, c, \dots$ can stand for arbitrary objects.

Functions return objects. Predicates return truth values.

The vocabulary (constants, functions, predicates) is sometimes called a **signature**.

Terms name objects

Recall that, in propositional logic, we had **sentences**: statements that could be true or false. In first-order logic, we start with **terms**, which denote objects.

DEFINITION (TERM)

Every constant and variable is a **term**. If f is a function symbol and $t_1, \dots, t_k$ are terms, then $f(t_1, \dots, t_k)$ is a term.

Examples: *alice*, *x*, *AdvisorOf(x)*, *Next(Next(zero))*.

We build terms recursively, so they can be nested.

Is *Student(alice)* a term? No: it is true or false (a sentence), rather than naming an object.

Is *AdvisorOf(alice)* a sentence? No: it names an object, rather than making a claim.

Given terms, we can now talk about (possibly) true or false statements, which here are the **formulas**.

If P is a k -ary predicate and $t_1, \dots, t_k$ are terms, then $P(t_1, \dots, t_k)$ is an **atomic formula**.

Equality $t_1 = t_2$ is also atomic.

DEFINITION (FORMULA)

Atomic formulas are formulas. If α and β are formulas, so are

$$\neg\alpha, \quad \alpha \wedge \beta, \quad \alpha \vee \beta, \quad \alpha \rightarrow \beta,$$

$$\forall x \alpha \quad \text{and} \quad \exists x \alpha.$$

As before, $\alpha \leftrightarrow \beta$ abbreviates the two implications.

$\forall$ means “for all; $\exists$ means “there exists.”

A variable occurrence inside its quantifier's scope is **bound**. Other occurrences are **free**.

A **sentence** is a formula with no free variables.

Is $x = y$ a sentence? No, x and y are free variables. Truth depends on them.

Is $\forall x \exists y (x = y)$ a sentence? Yes!

What do the quantifiers say?

Universal

$\forall x \textit{Student}(x)$

Every object is a student.

Existential

$\exists x \textit{Student}(x)$

At least one object is a student.

Usually our domain contains many kinds of objects. So we guard a universal statement with implication:

$$\forall x (\textit{Student}(x) \rightarrow \textit{Person}(x))$$

For existence, we normally use conjunction:

$$\exists x (\textit{Student}(x) \wedge \textit{Person}(x))$$

CAREFUL

$\exists x (\textit{Student}(x) \rightarrow \textit{Person}(x))$ can be true just because some non-student exists! (some x with $\textit{Student}(x)$ being $\perp$)

One breeze rule

Now we can answer the question that motivated the new language.

Vocabulary: $Cell(c)$, $Breeze(c)$, $Pit(n)$, and $Adjacent(c, n)$.

$$\forall c \left(Cell(c) \rightarrow (Breeze(c) \leftrightarrow \exists n (Adjacent(c, n) \wedge Pit(n))) \right)$$

Choose any cell c . It is breezy iff at least one adjacent object n has a pit.

The same sentence applies to every cell, including cells we add later.

That is the new power we get with quantifiers: one rule covers many objects.

Every student has an advisor

Consider the vocabulary: $Student(x)$ (" x is a student") and $Advises(y, x)$ (" y advises x ").
How can we translate "Every student has an advisor"?

$$\forall x (Student(x) \rightarrow \exists y Advises(y, x))$$

1. Choose an arbitrary object x
2. If x is a student, require an object y that advises x
3. The choice of y may depend on x

Now compare:

$$\exists y \forall x (Student(x) \rightarrow Advises(y, x))$$

This says one particular person advises every student.

Changing quantifier order can change the claim.

As before, syntax alone does not say what the symbols mean.

DEFINITION (FIRST-ORDER INTERPRETATION)

An interpretation m chooses a nonempty **domain** D (a set containing all objects), maps each constant to an object in D , each function symbol to a function over D , and each predicate to a relation over D .

Part	Interpretation in a small university world
D	$\{\text{Alice, Bob, CSE592}\}$
$alice$	Alice
$Student$	$\{\text{Alice, Bob}\}$
$Course$	$\{\text{CSE592}\}$
$Takes$	$\{(\text{Alice, CSE592})\}$

Again, an interpretation is a "possible world"

Evaluating a quantified sentence

Part	Interpretation in a small university world
D	{Alice, Bob, CSE592}
$alice$	Alice
$Student$	{Alice, Bob}
$Course$	{CSE592}
$Takes$	{(Alice, CSE592)}

In this model, consider:

$$\forall x (Student(x) \rightarrow \exists c (Course(c) \wedge Takes(x, c)))$$

For $x = \text{CSE592}$, $Student(x)$ is false, so the implication is true.

For $x = \text{Alice}$, choose $c = \text{CSE592}$. The existential is true.

For $x = \text{Bob}$, no object is both a course and taken by Bob.

So the whole sentence is false in this model.

If we add (Bob, CSE592) to the interpretation of $Takes$, it becomes true.

A universal ($\forall$) needs every object to pass. An existential ($\exists$) needs just one witness.

Translating English to first-order logic

Take the vocabulary: $Student(x)$, $Course(c)$, and $Takes(x, c)$.

Some more examples:

English	First-order logic
Every student takes CSE 592	$\forall x (Student(x) \rightarrow Takes(x, CSE592))$
Some student takes no course	$\exists x (Student(x) \wedge \forall c (Course(c) \rightarrow \neg Takes(x, c)))$
Every course has some student	$\forall c (Course(c) \rightarrow \exists x (Student(x) \wedge Takes(x, c)))$
No student takes every course	$\neg \exists x (Student(x) \wedge \forall c (Course(c) \rightarrow Takes(x, c)))$

What does this sentence say?

$$\exists c (Course(c) \wedge \forall x (Student(x) \rightarrow Takes(x, c)))$$

Which English sentence has the same meaning?

A. Every student takes some course

B. Some course is taken by every student

C. Every course is taken by some student

D. Some student takes every course

The existential chooses one course first; the universal then checks every student against that same course.

Equality lets us count

First-order logic with equality can say that two terms name the same object.

Alice and Bob are different:

$$alice \neq bob \text{ (equivalent to } \neg(alice = bob)\text{)}$$

There are at least two students:

$$\exists x \exists y (Student(x) \wedge Student(y) \wedge x \neq y)$$

There is exactly one advisor:

$$\exists x (Advisor(x) \wedge \forall y (Advisor(y) \rightarrow y = x))$$

Without equality, we cannot require two witnesses to be different or one witness to be unique.

REASONING IN FIRST- ORDER LOGIC

Expressivity has a cost

For n propositional atoms, there are exactly 2^n interpretations.

In first-order logic, an interpretation first needs a domain (set of all objects).

The domain can have one object, two objects, a million objects, infinitely many...

For each domain, we then need meanings for every predicate and function.

There is no finite truth table containing all first-order interpretations.

So we can't just enumerate models!

Complete, but undecidable

What about proofs?

First, some good news:

THEOREM (GÖDEL'S COMPLETENESS THEOREM)

First-order logic has sound and complete proof systems:

$$KB \models \beta \iff KB \vdash \beta.$$

If a conclusion is entailed, some finite proof exists.

But given a Turing machine M and input w , we can construct a first-order theory $T_{M,w}$ that describes its computation:

$$T_{M,w} \models Halt \iff M \text{ halts on } w$$

So no algorithm can always terminate and correctly decide every first-order entailment query.

Complete proof search eventually finds a proof when one exists; otherwise, it may run forever.

Related: Gödel's first incompleteness theorem: any consistent, effectively axiomatized theory that can express arithmetic has true arithmetic sentences it cannot prove.

LOGIC IN PRACTICE: Z3

SAT formulas contain Boolean variables. Many real problems also contain integers, arrays, bit-vectors, strings, and functions.

DEFINITION (SATISFIABILITY MODULO THEORIES (SMT))

SMT asks whether a logical formula is satisfiable when symbols have meanings from specified theories, such as integer arithmetic or fixed-width bit-vectors.

Z3 is a very popular modern SMT solver. It combines SAT with specialized reasoning for these theories.

Lots of industrial applications! Program and hardware verification, security, symbolic execution...

Let's first use it as a solver for a small scheduling CSP.

Z3 from Python: find a model

```
1  from z3 import *
2
3  alice, bob, chen = Ints('alice bob chen')
4  people = [alice, bob, chen]
5  s = Solver()
6  s.add([And(1 <= p, p <= 3) for p in people])
7  s.add(Distinct(people))
8  s.add(alice < bob, chen != 2)
9
10 print(s.check())
11 print(s.model())
```

Each integer is a presentation slot from 1 to 3.

Distinct says no two people present at the same time.

check() returns:

sat

Then model() gives one satisfying assignment.

[bob = 2, alice = 1, chen = 3]

Z3 from Python: answer an entailment query

```
1  from z3 import *
2
3  Person = DeclareSort('Person')
4  Student = Function('Student', Person, BoolSort())
5  Advises = Function('Advises', Person, Person, BoolSort())
6  alice, x, y = Consts('alice x y', Person)
7
8  kb = [
9      ForAll([x], Implies(Student(x),
10                          Exists([y], Advises(y, x)))),
11      Student(alice)
12  ]
13  query = Exists([y], Advises(y, alice))
14
15  s = Solver()
16  s.add(*kb, Not(query))
17  print(s.check())    # unsat: KB entails query
```

The code asks whether $KB \wedge \neg query$ is satisfiable.

unsat means there is no countermodel, so every model of the KB satisfies the query.

This is exactly the reduction we used for propositional entailment.

What can Z3 promise?

sat

A model was found

unsat

No model exists

unknown

The solver could not determine either

For many supported theories (e.g., "quantifier-free" fragments), Z3 is a decision procedure.

Z3 also accepts quantifiers and often handles them well using instantiation and model-based methods.

Many restricted fragments are decidable and still useful.

Of course, the general case even in first-order logic is still undecidable.

- **DPLL:** backtracking + unit propagation + pure-symbol choices for propositional SAT
- **First-order syntax:** terms, predicates, equality, and quantifiers describe objects and relations
- **First-order semantics:** a model chooses a domain and interprets the vocabulary
- **Entailment:** still means truth in every model of the KB
- **Proofs:** sound and complete calculi exist, but entailment is only semi-decidable
- **In practice:** SMT solvers decide useful fragments and often handle richer formulas

$$KB \models \beta \iff KB \wedge \neg\beta \text{ is UNSAT}$$

Next: uncertainty

Logic can represent uncertainty in one sense:

$$KB \not\models P \quad \text{and} \quad KB \not\models \neg P$$

Several worlds can remain possible, and we do not know which one we are in.

But logic alone does not say that one remaining world is more plausible than another.

It can't say "the evidence makes P twice as likely as $\neg P$."

Next lectures: probability, decisions under uncertainty... Then, learning.