

Satisfiability Modulo Theories

Concepts and common theories

CMPUT 664 • Formal Verification

Today

1. First-order logic semantics
2. What SMT is
3. Common theories
4. Decidability and completeness

First-order logic semantics

Structures give symbols meaning

A first-order structure

$$M = (D, I)$$

consists of a nonempty **domain** D and an **interpretation** I .

For each symbol in the signature, I chooses:

- an element of D for each constant c ,
- a total function $D^n \rightarrow D$ for each n -ary function f , and
- a relation $I(p) \subseteq D^n$ for each n -ary predicate p .

A variable assignment $\rho : \text{Var} \rightarrow D$ gives values to variables.

Evaluating terms and atoms

Terms denote elements of the domain:

$$\begin{aligned} \llbracket c \rrbracket_{M,\rho} &= I(c) \\ \llbracket x \rrbracket_{M,\rho} &= \rho(x) \\ \llbracket f(t_1, \dots, t_n) \rrbracket_{M,\rho} &= I(f)(\llbracket t_1 \rrbracket, \dots, \llbracket t_n \rrbracket) \end{aligned}$$

An atomic formula is true when the interpreted relation contains the evaluated arguments:

$$M, \rho \models p(t_1, \dots, t_n) \iff (\llbracket t_1 \rrbracket, \dots, \llbracket t_n \rrbracket) \in I(p).$$

Equality compares the denoted elements. Boolean connectives use their usual truth tables.

Quantifiers range over the domain

Let $\rho[x \mapsto d]$ agree with ρ except that it maps x to d .

$$M, \rho \models \forall x. F \iff \text{for every } d \in D, M, \rho[x \mapsto d] \models F$$

$$M, \rho \models \exists x. F \iff \text{for some } d \in D, M, \rho[x \mapsto d] \models F$$

The same formula can therefore have different truth values in structures with different domains or interpretations.

Models, satisfiability, and validity

A **sentence** has no free variables, so its truth does not depend on a variable assignment.

For a sentence F :

- M is a **model** of F when $M \models F$.
- F is **satisfiable** when some structure is a model of F .
- F is **valid** when every structure is a model of F .
- F is **unsatisfiable** when no structure is a model of F .

From SAT to SMT

SAT asks whether a propositional formula has a satisfying Boolean assignment.

$$(p \vee q) \wedge (\neg p \vee r)$$

SMT asks whether a formula is satisfiable with respect to one or more theories.

$$(x < y) \wedge (y < x + 1)$$

The atoms now have theory-specific meaning.

A theory is a signature plus axioms

Formally, a first-order theory is a pair

$$T = (\Sigma, A).$$

- The **signature** Σ is the vocabulary: its sorts, constants, functions, and predicates.
- The **axioms** A state what those symbols mean by restricting their legal interpretations.

For linear integer arithmetic, for example,

$$\Sigma_{\text{LIA}} \supseteq \{\text{Int}, 0, 1, +, -, \leq\},$$

while A_{LIA} specifies the intended behavior of integer addition and order.

Σ determines what formulas we can write. A determines which interpretations count as models.

Models of a theory

A structure M is a **model of** T when it satisfies every axiom:

$$M \models T \quad \Longleftrightarrow \quad \forall \alpha \in A. M \models \alpha.$$

The signature says which expressions are well formed. The axioms determine which interpretations are legal.

For LIA, an interpretation in which $1 + 1 = 3$ is not a model because it violates the intended integer arithmetic.

Satisfiable modulo a theory

Let T be a theory. A formula F is T -**satisfiable** if there is a model of T that satisfies F .

$$\exists M. M \models T \wedge M \models F$$

If no such model exists, F is T -**unsatisfiable**.

The answer may change when the theory changes.

Validity and equivalence modulo a theory

A formula is **valid modulo T** when every T -model satisfies it:

$$T \models F.$$

Two formulas are **equivalent modulo T** when

$$T \models F_1 \leftrightarrow F_2.$$

Ordinary first-order validity considers all interpretations. Validity modulo T considers only interpretations allowed by the axioms of T .

The theory changes the answer

Consider

$$x > 0 \wedge x < 1.$$

Over the reals, it is satisfiable; for example, $x = \frac{1}{2}$.

Over the integers, it is unsatisfiable.

The sort of x is part of the specification.

Common theories

Expressiveness and intended meaning

Equality

SMT uses logical equality between terms of the same sort.

$$\forall x. x = x$$

reflexivity

$$\forall x, y. x = y \implies y = x$$

symmetry

$$\forall x, y, z. x = y \wedge y = z \implies x = z$$

transitivity

These properties make equality an **equivalence relation**. Equalities partition terms into equivalence classes.

For example, $x = y$ and $y = z$ place x , y , and z in the same class.

Uninterpreted functions

An uninterpreted function has a name and a signature, but no fixed definition.

$$f : A \rightarrow B$$

A model may choose any total function of that signature that satisfies the constraints.

For example,

$$f(a) = b \wedge f(c) = d$$

constrains two applications of f without specifying how f computes its results.

Why use uninterpreted functions?

Use an uninterpreted function when the internal computation is irrelevant or unavailable, but relationships between inputs and outputs still matter.

Examples:

- Abstract a library function while verifying its caller
- Represent a deterministic API or hardware component
- Delay implementation details while specifying required relationships

If the actual behavior matters, add axioms or choose a more specific theory.

Equality with uninterpreted functions (EUF)

EUF combines equality with uninterpreted functions. **Congruence** requires equal inputs to produce equal outputs:

$$x = y \implies f(x) = f(y).$$

Therefore,

$$x = y \wedge f(x) \neq f(y)$$

is unsatisfiable. The converse need not hold: $f(x) = f(y)$ does not imply $x = y$.

Congruence is an axiom schema

For every n -ary function symbol f , equality must propagate through f :

$$\bigwedge_{i=1}^n x_i = y_i \implies f(x_1, \dots, x_n) = f(y_1, \dots, y_n).$$

There is one instance of this schema for each function symbol and arity.

Predicates obey the corresponding rule:

$$\bigwedge_{i=1}^n x_i = y_i \implies \left(p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n) \right).$$

Equality reasoning example

Suppose

$$a = b \wedge b = c.$$

Then

$$a = c$$

transitivity

$$f(a) = f(c)$$

congruence

$$b = a$$

symmetry

$$g(f(a), b) = g(f(c), a)$$

congruence

The proof alternates between equivalence-class reasoning and propagation through function applications.

Peano arithmetic

Representative recursive axioms define addition and multiplication:

$$\begin{array}{ll} x + 0 = x & x + (y + 1) = (x + y) + 1 \\ x \cdot 0 = 0 & x \cdot (y + 1) = x \cdot y + x \end{array}$$

Induction is an axiom schema, with one instance for each formula F :

$$\left(F(0) \wedge \forall x. (F(x) \implies F(x + 1)) \right) \implies \forall x. F(x).$$

Presburger arithmetic and LIA

Presburger arithmetic uses $\mathbb{N}$ with 0, 1, addition, and equality. It excludes multiplication between variables.

Its axioms recursively define addition:

$$x + 0 = x, \quad x + (y + 1) = (x + y) + 1.$$

They also include induction for every formula F in the addition-only language:

$$\left(F(0) \wedge \forall x. (F(x) \implies F(x + 1)) \right) \implies \forall x. F(x).$$

Linear integer arithmetic (LIA) is the convenient $\mathbb{Z}$ presentation with subtraction, order, and integer coefficients. Its quantifier-free fragment does not require induction.

Linear arithmetic over rationals and reals

Both $\mathbb{Q}$ and $\mathbb{R}$ are dense. Between two distinct values, their midpoint is another value:

$$x < y \implies x < \frac{x + y}{2} < y.$$

For every finite system of linear constraints with rational coefficients,

$\text{satisfiable over } \mathbb{R} \iff \text{satisfiable over } \mathbb{Q}$
--

Choosing $\mathbb{Q}$ or $\mathbb{R}$ therefore does not change the satisfiability answer for this fragment.

Nonlinear arithmetic

An arithmetic expression is nonlinear when it contains products of variables or powers such as x^2 .

$$xy = 10, \quad x^2 + y^2 < 4.$$

The domain changes the meaning. For example,

$$x^2 = 2$$

has solutions over $\mathbb{R}$, but none over $\mathbb{Z}$.

Arithmetic theory landscape

Theory	Domain	Allowed arithmetic
Peano arithmetic	$\mathbb{N}$	Addition, multiplication, and quantifiers
Presburger arithmetic	$\mathbb{N}$	Addition, but no products of variables
Linear integer arithmetic	$\mathbb{Z}$	Linear expressions
Linear real arithmetic	$\mathbb{R}$	Linear expressions
Real closed fields	Real-closed ordered fields	Polynomial expressions and order

What is a bit-vector?

A bit-vector of width w is a fixed sequence of Boolean bits:

$$b_{w-1} b_{w-2} \cdots b_1 b_0 \in \{0, 1\}^w.$$

The sort $\mathbf{BV}_w$ therefore contains exactly 2^w values.

The 8-bit pattern

$$11111111_2$$

represents 255 under unsigned interpretation and -1 under two's-complement signed interpretation. The stored bit pattern has no inherent signedness; the selected operation determines the interpretation.

Defining bit-vector semantics

Bit-vector operations have exact finite semantics. Representative defining equations are

[Math Processing Error]

Shifts, extraction, concatenation, multiplication, and comparisons are defined in the same bit-level manner.

Unlike arrays or EUF, bit-vectors are normally specified by the semantics of each finite-width operator rather than a small reusable list of axioms.

Why use bit-vectors?

Programs operate on fixed-width machine words, not unbounded mathematical integers.

Bit-vectors model

- overflow and wraparound exactly,
- masks, shifts, extraction, and concatenation, and
- registers, packet fields, and cryptographic operations.

For example, 8-bit addition gives

$$11111111_2 +_8 00000001_2 = 00000000_2.$$

Integers and bit-vectors are different

Question	Integer	8-bit bit-vector
Is $255 + 1 = 256$?	Yes	No
Can addition overflow?	No	Yes
Is the domain finite?	No	Yes
Signedness built into the value?	Not applicable	No; operations choose it

What is an SMT array?

An array of sort $\text{Array}(I, V)$ is a **total map** from indices of sort I to values of sort V .

- $\text{select}(a, i)$ reads the value at index i .
- $\text{store}(a, i, v)$ returns a **new array** that maps i to v .

$$a' = \text{store}(a, i, v)$$

The old array a is unchanged. SMT arrays are immutable mathematical values, not mutable program objects.

Defining array semantics

The **read-over-write** axioms say what can be observed after an update:

$$\text{select}(\text{store}(a, i, v), i) = v$$

$$i \neq j \implies \text{select}(\text{store}(a, i, v), j) = \text{select}(a, j).$$

Extensionality says arrays are equal when all their elements are equal:

$$(\forall i. \text{select}(a, i) = \text{select}(b, i)) \implies a = b.$$

Together, these laws characterize arrays through their reads and writes. Ordinary equality congruence also ensures that equal indices produce equal reads.

Why use arrays?

Arrays model structures accessed at a **symbolic index**:

- memory and heaps
- program arrays
- maps and lookup tables

For a memory write,

$$\text{mem}_1 = \text{store}(\text{mem}_0, \text{addr}, \text{value}),$$

the axioms immediately give

$$\text{select}(\text{mem}_1, \text{addr}) = \text{value},$$

while every different address keeps its old value.

Other supported theories

Other common SMT theories include:

- Floating-point arithmetic
- Strings and regular languages
- Algebraic datatypes
- Sequences and sets

Checkout [this](#) if interested.

Decidability and completeness

What automation can guarantee

What decidable means

A satisfiability problem is **decidable** if there is an algorithm that, for every input in the problem class,

- always terminates, and
- correctly returns satisfiable or unsatisfiable.

Entailment can be reduced to unsatisfiability:

$$T \models F \quad \Longleftrightarrow \quad T \cup \{\neg F\} \text{ is unsatisfiable.}$$

Decidability guarantees termination and correctness. It does not guarantee efficiency.

The full language determines decidability

Decidability depends on the complete problem definition:

- the domain, such as $\mathbb{N}$, $\mathbb{Z}$, or $\mathbb{R}$,
- the permitted operations,
- the use and ordering of quantifiers, and
- the combination of theories.

Quantifiers alone do not determine the answer. Presburger arithmetic and real closed fields allow quantifiers and remain decidable.

Conversely, multiplication over integers or natural numbers already creates undecidable reasoning problems.

A concrete terminating run

Consider this counter program P :

```
1 1: if x = 0 goto 4; otherwise goto 2
2 2: x := x - 1; goto 3
3 3: y := y + 1; goto 1
4 4: halt
```

A configuration records the program counter and the two values: (pc, x, y) .

On input $(x, y) = (2, 0)$, the complete run is

[Math Processing Error]

One program step becomes arithmetic

The effect of instruction 2 can be checked using natural-number arithmetic:

$$pc = 2 \wedge x > 0 \wedge pc' = 3 \wedge x = x' + 1 \wedge y' = y.$$

This says that (pc', x', y') is the configuration after (pc, x, y) .

Each instruction gives one such case. Their disjunction defines

$$\text{Step}_P(pc, x, y, pc', x', y').$$

For example, the formula accepts $(2, 2, 0) \rightarrow (3, 1, 0)$ and rejects $(2, 2, 0) \rightarrow (3, 0, 0)$.

A halting run is a certificate

For fixed P and input w , a proposed finite trace is evidence that P halts on w .

A checker only needs to verify:

1. The first configuration is the initial state of P on input w .
2. Every adjacent pair follows one instruction of P .
3. The final configuration is at the halt instruction.

The eight-configuration trace shown earlier passes these checks. A trace containing

$$(2, 2, 0) \rightarrow (3, 0, 0)$$

fails the second check.

The non-obvious bridge to arithmetic

We now use one theorem:

Representability theorem. Every relation on natural numbers that can be checked by an always-terminating program can be described by a first-order arithmetic formula using 0, 1, +, and $\times$.

The trace checker is such a program. Therefore, for each P, w , arithmetic has a formula

$$\text{Trace}_{P,w}(r)$$

that is true exactly when r codes a finite trace accepted by the checker.

What the theorem is hiding

The number r is only a serialization of finite data, much like representing text as bytes.

The theorem guarantees that arithmetic can describe both - how to recover the configurations encoded by r , and - how to repeat the local step check across the entire finite trace.

This theorem is the substantial technical result. $\text{Trace}_{P,w}$ is an abbreviation for the arithmetic formula it provides, not a new primitive of PA.

Why this makes arithmetic undecidable

Suppose an algorithm could decide every sentence about natural numbers with addition and multiplication.

Given any program P and input w , mechanically construct

$$H_{P,w} \equiv \exists r. \text{Trace}_{P,w}(r).$$

The arithmetic algorithm would answer **yes** exactly when P halts on w . It would therefore decide the halting problem, which is impossible.

The undecidability does not come from one mysterious sentence. It comes from a mechanical translation that produces $H_{P,w}$ from the program and its input.

Why PA is undecidable

PA can verify any concrete terminating computation: a finite execution trace provides a witness that can be checked step by step.

- If P halts on w , PA can prove $H_{P,w}$.
- If P does not halt, $H_{P,w}$ is false in $\mathbb{N}$, so sound PA cannot prove it.

If an algorithm could always decide whether PA proves an input sentence, applying it to $H_{P,w}$ would decide whether any program halts.

The intuition is that multiplication over an unbounded discrete domain is expressive enough to represent arbitrary computation.

Real arithmetic can encode a fixed trace

For any fixed bound k , introduce one configuration c_i for each step and write

$$\text{Init}(c_0) \wedge \bigwedge_{i=0}^{k-1} \text{Step}(c_i, c_{i+1}) \wedge \text{Halt}(c_k).$$

Here k is chosen before constructing the formula; the large conjunction is simply expanded k times.

If the transition relation is polynomial, this can be a formula over a real closed field.

For example, choosing $k = 7$ lets us describe the seven-step trace shown earlier.

But this formula asks only for a seven-step halting trace. A different bound requires a larger formula with more variables.

The difficult part is “some finite length”

The halting problem asks

Does a valid trace exist for *some* $k \in \mathbb{N}$?

A first-order formula is finite, so it cannot literally contain a different variable c_i for every possible step.

Natural-number arithmetic gets around this by encoding an arbitrarily long finite trace as one number and using natural numbers as its positions.

Merely quantifying a real number k does not help: $0 \leq t \leq k$ describes an interval, not the finite positions $0, 1, \dots, k$.

To reproduce the natural-number encoding, the theory would need to distinguish those integer positions inside the real domain.

Real closed fields cannot define the integers

For every fixed k , a polynomial can select the finite set $\{0, 1, \dots, k\}$:

$$x(x - 1) \cdots (x - k) = 0.$$

But no first-order formula over a real closed field defines the whole set

$$\mathbb{N} = \{0, 1, 2, 3, \dots\}.$$

Quantifier elimination implies that every definable subset of the real line is a finite union of points and intervals. An infinite discrete set such as $\mathbb{N}$ has neither form.

Therefore real closed fields can describe **bounded traces**, but not arbitrary finite traces uniformly in the way required to express general program halting.

Why real closed fields are decidable

The first-order theory of real closed fields describes polynomial equations and inequalities over real-closed ordered fields.

Tarski's **quantifier elimination** theorem gives an algorithm that removes quantifiers while preserving meaning. For example, over the reals,

$$\exists x. x^2 + bx + c = 0 \quad \Longleftrightarrow \quad b^2 - 4c \geq 0.$$

In general, elimination produces a Boolean combination of polynomial sign conditions. A sentence then reduces to a finite algebraic calculation.

Intuition for quantifier elimination

A polynomial can change sign only when it crosses one of its roots.

For finitely many one-variable polynomials, their roots split the real line into finitely many points and intervals. On each interval, every polynomial has a fixed sign.

Instead of checking infinitely many real values, an algorithm can reason about these finitely many regions.

Why the real case behaves differently

Natural-number arithmetic	Real closed fields
Has an unbounded discrete domain	Cannot define an infinite discrete subset
Encodes arbitrarily long finite traces uniformly	Encodes a trace only after a fixed bound is chosen
Addition and multiplication can simulate arbitrary computation	Quantifiers reduce to polynomial sign conditions
No general decision procedure	A decision procedure exists

The symbols $+$ and $\times$ occur in both theories. Their domains and axioms give them very different expressive power.

What about nonlinear arithmetic over $\mathbb{Q}$?

$\mathbb{Q}$ is dense, but it is **not** a real closed field. For example, $x^2 = 2$ has no rational solution.

This is different from using rational coefficients in a formula whose variables still range over $\mathbb{R}$.

More surprisingly, there is a first-order formula $\text{Int}_{\mathbb{Q}}(x)$ using rational-field operations that is true exactly when x is an integer.

Once integers can be selected inside $\mathbb{Q}$, an integer quantifier can be translated as

$$\begin{aligned}\exists n \in \mathbb{Z}. F(n) &\rightsquigarrow \exists q \in \mathbb{Q}. \text{Int}_{\mathbb{Q}}(q) \wedge F(q), \\ \forall n \in \mathbb{Z}. F(n) &\rightsquigarrow \forall q \in \mathbb{Q}. \text{Int}_{\mathbb{Q}}(q) \implies F(q).\end{aligned}$$

Thus full first-order nonlinear rational arithmetic **can** express integer program-trace reasoning and is undecidable.

An important open boundary over $\mathbb{Q}$

The following problem remains open:

Given a polynomial p with integer coefficients, is there an algorithm that always decides whether

$$\exists x_1, \dots, x_n \in \mathbb{Q}. \quad p(x_1, \dots, x_n) = 0?$$

This is **Hilbert's tenth problem** over $\mathbb{Q}$.

Representative boundaries

Satisfiability or theoremhood problem	Decidable?
Quantifier-free EUF	Yes
Quantifier-free linear integer or real arithmetic	Yes
Full Presburger arithmetic	Yes
Peano arithmetic theoremhood	No
Quantifier-free nonlinear integer arithmetic	No
Full first-order theory of $\mathbb{Q}$ with $+$ and $\times$	No
Existential polynomial equations over $\mathbb{Q}$	Open
Full theory of real closed fields	Yes
Quantifier-free fixed-width bit-vectors	Yes
Unrestricted first-order validity	No

Consistency and completeness of a theory

A theory T is **consistent** when its axioms can all hold together:

$$\exists M. M \models T.$$

A consistent theory is **complete** when, for every sentence F , exactly one holds:

$$T \models F \quad \text{or} \quad T \models \neg F.$$

Consistency rules out both. Completeness rules out neither.

Equivalently, all models of a complete theory agree on every sentence.

A small complete theory

Use a language containing only equality. Let T_2 state that exactly two objects exist:

$$T_2 = \{ \exists a. \exists b. a \neq b \wedge \forall x. (x = a \vee x = b) \} .$$

Every model of T_2 is a two-element set. Any two such models are isomorphic, so they agree on every sentence in this language.

Other complete theories include Presburger arithmetic and real closed fields.

Why completeness matters

Completeness measures how fully the axioms determine the properties expressible in the language.

If T is incomplete, there is a sentence F and two legal models such that

$$M \models T \wedge F, \quad N \models T \wedge \neg F.$$

If the intended models should agree about F , the theory may be missing an axiom.

If the theory intentionally abstracts several implementations, leaving F unsettled may be desirable.

Completeness and decidability ask different questions

Property	Question
Completeness of T	Do all models of T agree on every sentence?
Decidability of T	Can an algorithm always determine whether $T \models F$?

Completeness of a proof system

Do not confuse completeness of a theory with completeness of a proof system.

For a proof system,

$$\begin{array}{ll} \Gamma \vdash F \implies \Gamma \models F & \text{soundness,} \\ \Gamma \models F \implies \Gamma \vdash F & \text{completeness.} \end{array}$$

Proof-system completeness says every semantic consequence has a finite proof.

Theory completeness says the axioms settle every sentence.

Complete but undecidable: true arithmetic

True arithmetic is

$$\text{Th}(\mathbb{N}, +, \times),$$

the set of all first-order sentences true in the standard natural numbers with ordinary addition and multiplication.

It is **complete** because this one fixed structure gives every sentence a definite truth value.

It is **undecidable** because the computation encoding described earlier would turn a truth decider into a solution to the halting problem.

Effective axiomatization

A theory is **effectively axiomatized** when a program can enumerate its axioms:

$$A_1, A_2, A_3, \dots$$

Every axiom eventually appears. The theory may still have infinitely many axioms.

A finite axiom set is automatically effective. An infinite axiom schema is effective when its instances can be generated mechanically.

Complete plus effective gives decidable

Suppose a consistent theory T is complete and effectively axiomatized.

Given a sentence F , enumerate proofs from T while searching for

$$T \vdash F \quad \text{and} \quad T \vdash \neg F.$$

Theory completeness guarantees that one is a consequence. Completeness of first-order proof systems guarantees that the corresponding finite proof exists.

One search must eventually succeed:

complete + effective axiomatization $\implies$ decidable.

Why PA must be incomplete

Think of effective PA as a program that continually prints every PA theorem.

Suppose PA were complete. Given any arithmetic sentence F , wait until the program prints either

$$F \quad \text{or} \quad \neg F.$$

Completeness would guarantee that one appears. Soundness would guarantee that the printed answer agrees with the standard natural numbers.

This would decide true arithmetic, which is impossible. Therefore PA must leave some sentence and its negation unproved.

The Gödel–Rosser theorem strengthens the result by requiring only consistency rather than soundness.

Decidable but incomplete: pure equality

Let $T_=_$ contain only equality and no nonlogical axioms. Its models are nonempty sets of any size. Consider

$$F_{\geq 2} \equiv \exists x. \exists y. x \neq y.$$

A one-element model falsifies $F_{\geq 2}$, while a two-element model satisfies it. Therefore,

$$T_=_ \not\models F_{\geq 2} \quad \text{and} \quad T_=_ \not\models \neg F_{\geq 2}.$$

The theory is incomplete because its models disagree. Entailment in pure equality is nevertheless decidable.

Examples separate the concepts

Theory	Complete?	Decidable?
Presburger arithmetic	Yes	Yes
True arithmetic	Yes	No
Pure equality over arbitrary nonempty domains	No	Yes
Peano arithmetic, assuming consistency	No	No

Full first-order logic is semidecidable

Unrestricted first-order validity is undecidable: no algorithm can always terminate with the correct yes-or-no answer.

It is nevertheless **semidecidable** because first-order proof systems are complete.

- If F is valid, systematic proof search eventually finds a finite proof.
- If F is invalid, proof search may continue forever.

Completeness guarantees that valid formulas have proofs. It does not guarantee termination when no proof exists.

Decidable fragments of FOL

Restricting the language can restore decidability.

Fragment	Main restriction
Quantifier-free EUF	No quantifiers; equality and uninterpreted functions
Monadic FOL	Predicates are unary, with restricted function symbols
Bernays–Schönfinkel	No function symbols and prefix $\exists^* \forall^*$

The Bernays–Schönfinkel fragment is also called **effectively propositional logic**.

These restrictions make important classes of first-order formulas decidable.

Compactness of first-order logic

First-order logic is **compact**. For any set of sentences Γ ,

$$\Gamma \text{ is satisfiable} \iff \text{every finite } \Gamma_0 \subseteq \Gamma \text{ is satisfiable.}$$

The important direction says that if every finite part has a model, then the entire infinite set has a model.

Equivalently,

$$\Gamma \models F \implies \exists \text{ finite } \Gamma_0 \subseteq \Gamma. \Gamma_0 \models F.$$

A consequence of compactness

For every fixed n , FOL can express that a graph contains a path of length n .

Reachability asks whether there is a path of some finite length, with no fixed bound.

Compactness can be used to show that no first-order formula defines reachability, or equivalently transitive closure, over arbitrary graphs.

FOL can state that a relation is transitive. It cannot require that the relation be exactly the smallest transitive relation containing the graph edges.

Next time

Encoding problems as SMT