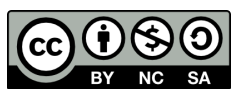
# CDM

# First-Order Logic

Klaus Sutner

Carnegie Mellon University

Spring 2024



The theory of finite state machines is heavily algorithmic and just about any reasonably decision problem involving FSMs is decidable.

> Can one use finite state machines to describe mathematical objects?
>
> If so, can one solve decision problems about these objects?

Since FSM describe recognizable languages, the first question turns into: Can we use recognizable languages to describe other mathematical objects?

# Example: Dyck Graph

Let $\Sigma = \{U, D, 0, 1, a, b\}$.

The vertices in the graph are given by the following regular expression over $\Sigma$:

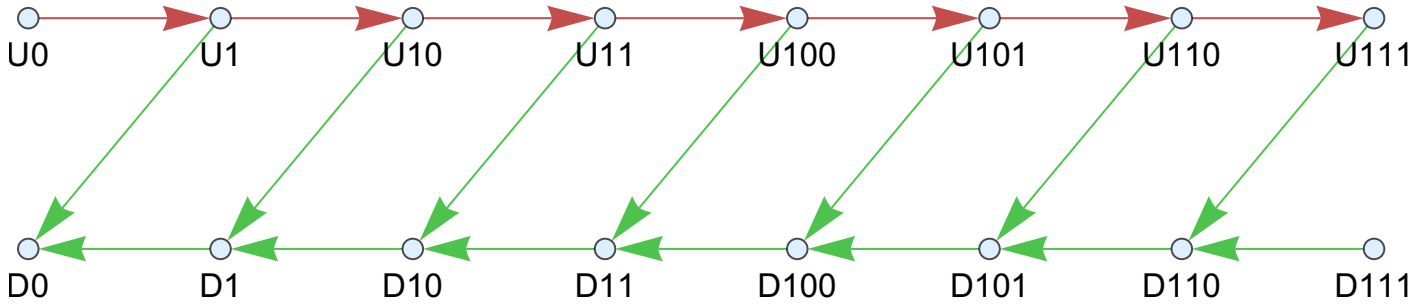
$$V = (U + D)\big(0 + 1\{0, 1\}^\star\big)$$

So there are up-vertices like $U101$ and down-vertices like $D1$.

The edges are of the form

$$Ux \xrightarrow{a} Uy \qquad Ux \xrightarrow{b} Dz \qquad Dx \xrightarrow{b} Dz$$

Here $y$ is $x + 1$ in binary, and $z$ is $x - 1$: $U101 \xrightarrow{a} U110$, $U101 \xrightarrow{b} D100$.




The Dyck graph essentially produces and infinite-state machine that recognizes expressions of the form $a^i b^i$.

We chose a slightly fancy way to represent the graph, we could have simplified matters by writing the numbers in unary. However, using binary is critical for applications to arithmetic.

The vertex set is just a recognizable language, we can use some NFA to describe it. No problem.

The edges are a bit more complicated. We can easily code them as strings

$$U101 \xrightarrow{b} D100 \quad \rightsquigarrow \quad \texttt{U101bD100}$$

but there is no way a FSM could recognize these strings.

Why?

# Workaround

But we can pick a different encoding and then FSMs will work easily:

- Instead of writing plain base 2, we use reverse base 2 (LSD first).
- Start with the label, then interleave the letters of the two vertices (perfect shuffle).

$$U011 \xrightarrow{b} D101 \quad \rightsquigarrow \quad \texttt{bUD011011}$$

With this encoding a FSM is perfectly capable of checking the arithmetic: the standard method to calculate successors/predecessors naturally translates into a FSM (with a bit of tinkering).

Let's take for granted that the Dyck graph can be expressed by two FSMs. We want a decision algorithm that takes as input the FSMs plus a query, a sentence in some formal language, and then checks if the sentence is a valid assertionn about the graph.

> What kind of queries should be allowed?

Clearly, there is a trade-off here: if we make the queries too complicated, validity may well be undecidable.

On the other hand, if the queries are too weak the whole decision algorithm will be useless.

As it turns out, a useful choice is allow queries in first-order logic.

For example, writing $x \to y$ to mean "there is an edge from $x$ to $y$," the formula

$$\exists\, x \left( \forall\, y \left( \neg (x \to y) \right) \wedge \forall\, z \left( \forall\, y \left( \neg (z \to y) \right) \Rightarrow x = z \right) \right)$$

expresses the assertion: there is exactly one vertex with out-degree 0.

Similarly we could assert that every D-vertex has indegree 2.

**Warning:** In general, FOL is too weak to express the assertions like "there is a path from $x$ to $y$." So we could not assert that the graph is strongly connected and the like.

According to the Handbook of Mathematical Logic, the field of logic is organized into 4 areas:

- set theory
- recursion theory
- proof theory
- model theory

You have seen some amount of set theory and recursion theory, we will now turn to other two areas—though only with the lightest touch.

Recall Hilbert's dream: a decision algorithm for all of mathematics.

As we now know, in its orginal form, this goal is utterly unachievable; computability theory has established hard upper bounds as to what is computable (even ignoring complexity issues).

Very well, let's scale back: let's just try to answer questions in a very limited domain. But we do want the method to be purely mechanical and not require any intuitive insights, and certainly not clairvoyance.

How should we go about this little project?

There are three major parts in the design of a logic:

- language (syntax)
- model theory (semantics)
- proof theory (deductions)

Describing a language is fairly straightforward, one has to describe the rectype of formulae in the logic. Things get somewhat interesting if one is interested in building a proof assistant or a theorem prover, but we won't.

Model theory (study of the corresponding structures) brushes up against set theory, at least for infinite models, and can be quite challenging. There will be no problem for the models we are interested in.

Proof theory is arguably the most difficult part: while proofs (deductions) are just finite data structures, analyzing and understanding them in detail is hard.

There are many examples of useful systems of logic:

> propositional, equational, first-order, second-order, higher-order, modal, linear time temporal, branching time temporal, linear, intuitionist, infinitary logics, ...

Which one is appropriate for a particular problem depends very much on the problem.

And, there is a trade-off: making the logic powerful usually results in computational complexity issues; keeping it weak limits applicability.

**Convention:** We are going to deal exclusively with classical first-order logic, aka predicate logic.

The language of first-order logic consists of the following pieces:

| | |
|---|---|
| logical symbols | |
| constants | that denote individual objects |
| variables | that range over individual objects |
| logical connectives | “and,” “or,” “not,” “implies” . . . |
| existential quantifiers | that express “there exists” |
| universal quantifiers | that express “for all” |
| non-logical symbols | |
| relation symbols | that denote relations |
| function symbols | that denote functions |

- $a$, $b$, $c$, ... for constants,
- $x$, $y$, $z$, ... for variables,
- $\vee$, $\wedge$, $\neg$, $\Rightarrow$ for the logical connectives,
- $\exists$ for the existential quantifier,
- $\forall$ for the universal quantifier,
- $f$, $g$, $h$, ... for function symbols,
- $R$, $P$, $Q$, ... for relation symbols.
- Allow $=$ for equality.

These are just conventions, there are no sacred cows here. E.g., we might also use subscripted symbols and additional connectives such as $\oplus$ or $\Leftrightarrow$.

One should distinguish more carefully between function and relation symbols of different arity: unary, binary functions and so on. If necessary, we can indicate arity like so: $R^{(3)}(x, y, c)$.

In most concrete structures only a finite number of function and relation symbols are needed.

Hence, we can convey the structure of the non-logical part of the language in a signature or similarity type: a list that indicates the arities of all the function and relation symbols.

**Example**

The standard signature for group theory is $(2, 1, 0)$: one binary function symbol for group multiplication, one unary function symbol for the inverse operation, a nullary function symbol (constant) for the neutral element.
Alternatively, we could use signature $(2)$ only—but at the cost of slightly more complicated axioms.

There are occasions when one wants to use countably many function or relation symbols. It is easy to describe languages of this form. For example, we could allow for constants $c_n$, $n \in \mathbb{N}$; together with axioms $c_i \neq c_j$ for $i < j$, this would make sure that all models are infinite.

In pure mathematical logic one also considers languages of cardinalities higher than $\aleph_0$, but this will be of no interest to us. Note that some amount of set theory is needed just to define such a language. We don't have to worry about these, everything is at most countable.

# Example: Arithmetic

For arithmetic it is convenient to have a relation symbol for order:

$\mathcal{L}(+, \cdot, 0, 1; <)$ of signature $(2, 2, 0, 0; 2)$

- binary function symbols $+$ and $\cdot$ (for integer addition and multiplication)
- constants $0$ and $1$ (for integers 0 and 1)
- a binary relation symbol $<$ (for the less-than relation)

Assuming we are quantifying over the natural numbers we have assertions like

$$\forall\, x\, \exists\, y\, (x < y)$$

$$\exists\, x\, \forall\, y\, (x = y \vee x < y)$$

$$\forall\, x\, (x + 0 = x)$$

Intuitively, these are all true.

In the language of arithmetic we can write a formula $\mathsf{prime}(x)$ that expresses the assertion "$x$ is prime".

$$\mathsf{prime}(x) \equiv 1 < x \wedge \forall\, u, v\, (x = u \cdot v \Rightarrow u = 1 \vee v = 1)$$

Note that $x$ here is a free variable (not quantified over) in $\mathsf{prime}(x)$. So, we can use free variables to define subsets of the domain.

We can now express the assertion "there are infinitely many primes."

$$\forall\, z\, \exists\, x\, (x > z \wedge \mathsf{prime}(x))$$

**Warning:** This is mildly misleading, FOL in general cannot express "there are infinitely many." The last formula works because we are in the context of arithmetic (where we force the domain to be a Dedekind-infinite set).

For the classical algebraic theory of fields one minimally uses a language

$$\mathcal{L}(+, \cdot, 0, 1) \text{ of signature } (2, 2, 0, 0)$$

- binary function symbols $+$ and $\cdot$ for addition and multiplication,
- constants $0$ and $1$ for the respective neutral elements.

Of course, more functions could be added: additive inverse, subtraction, multiplicative inverse, division. Introducing only addition and multiplication is the minimalist approach here.

Now consider formulae such as

$$\forall\, x, y\, (x + y = y + x)$$

$$\forall\, x\, (x \cdot 0 = 0)$$

$$\forall\, x\, \exists\, z\, ((\neg x = 0) \Rightarrow z \cdot x = 1)$$

$$1 + 1 = 0$$

It is intuitively clear what these formulae mean.

Except for the last one, they are all true in any field. In fact, the first two hold in any ring.

The third one is true in a field, but is false in an arbitrary ring.

And the last only holds in rings of characteristic 2. In fact, it defines unital rings of characteristic 2 (assuming we are not in the zero ring).

# Example: Boolean Algebra

In Boolean algebra one uses the language

$$\mathcal{L}(\sqcup, \sqcap, \overline{\ }, 0, 1) \quad \text{of signature } (2, 2, 1, 0, 0)$$

So we have

- binary function symbols $\sqcup$ and $\sqcap$ (for join and meet)
- unary function symbol $\overline{\ }$ (for complement)
- constants $0$ and $1$

Some formulae:

$$\forall\, x, y\, \exists\, z\, (x \sqcup y = \overline{z})$$

$$\exists\, x\, \forall\, y\, (x \sqcap y = 0)$$

$$\forall\, x\, \exists\, y\, (x \sqcup y = 1 \wedge x \sqcap y = 0)$$

As an example of the expressiveness of FOL consider the venerable Principle of Induction. We can express induction as a first-order formula as follows.

$$R(0) \wedge \forall\, x \left(R(x) \Rightarrow R(x+1)\right) \Rightarrow \forall\, x\, R(x)$$

Here we have added another unary relation symbol $R$ to our language. So $R(x)$ asserts that number $x$ has some unspecified property.

If we wanted to quantify over all relations $R$ we would need more horse power, say, second-order logic.

$$\forall\, R \left(R(0) \wedge \forall\, x \left(R(x) \Rightarrow R(x+1)\right) \Rightarrow \forall\, x\, R(x)\right)$$

Second-order logic is dangerously close to set theory: exactly what are we quantifying over in $\forall R$?

The standard workaround is to use an axiom schema instead:

$$\varphi(0) \wedge \forall x \left(\varphi(x) \Rightarrow \varphi(x+1)\right) \Rightarrow \forall x \, \varphi(x)$$

Here $\varphi(x)$ is any formula of arithmetic with free variable $x$.

This works fine in basic applications, say, in number theory, but is nowhere near as powerful as second-order.

While we are not interested in writing a parser or theorem prover, we still need to be a bit more careful about the syntax of our language of FOL. The components of a formula can be organized into a taxonomy like so:

- variables, constants and terms,
- equations,
- atomic formulae,
- propositional connectives, and
- quantifiers.

Every programming language has a defining report (which no one ever reads, other than perhaps compiler writers, it's the document that uses the imperative "shall" a lot), so think of this as the defining report for FOL.

The quantification part is what really matters here; propositional connectives are the same as in propositional logic while terms and equations are the same as in equational logic.

We need a supply of variables Var as well as function symbols and predicate symbols. These form a graded alphabet $\Sigma = \Sigma_0 \cup \Sigma_1$. where every function symbol and relation symbol has a fixed number of arguments, determined by a arity map:

$$\text{ar} : \Sigma \to \mathbb{N}$$

We write $\mathcal{L}(\Sigma)$ for the language constructed from signature $\Sigma$.

Function symbols of arity $0$ are constants and relation symbols of arity $0$ are Boolean values (true or false) and will always be written $\top$ and $\bot$.

In any concrete application, $\Sigma$ will be finite. However, in the literature you will often find a big system approach that introduces a countable supply of function and relation symbols for each arity. For our purposes a signature custom-designed for a particular application is more helpful.

**Definition**

The set $\mathcal{T} = \mathcal{T}(\Sigma) = \mathcal{T}(\mathsf{Var}, \Sigma)$ of all terms is defined by

- Every variable is a term.
- If $f$ is an $n$-ary function symbol, and $t_1, \ldots, t_n$ are terms, $n \geq 0$, then $f(t_1, \ldots, t_n)$ is also a term.

A ground term is a term that contains no variables.

Note that $f()$ is a term for each constant (0-ary function symbol) $f$. For clarity, we write $a$, $b$, $c$ and the like for constants.

The idea is that every ground term corresponds to a specific element in the underlying structure. For arbitrary terms we first have to replace all variables by constants. For example, in arithmetic the term $(1 + 1 + 1) \cdot (1 + 1)$ corresponds to the natural number $6$. We could introduce constants for all the natural numbers, but there is no need to do so: we can build a corresponding term from the constant $1$ and the binary operation $+$.

Given a few terms, we can apply a predicate to get a basic assertion (like $x + 4 < y$).

**Definition**

An atomic formula is an expression of the form

$$R(t_1, \ldots, t_n)$$

where $R$ is an $n$-ary relation symbol, and the $t_1, \ldots, t_n$ are terms.

These are basically the atomic assertions in propositional logic: once we have values for the variables that might appear in the terms, an atomic formula can be evaluated to true or false.

But note that we do need bindings for the variables, $R(x, y)$ per se has no truth value.

Equality plays a special role in our setup, we include the binary relation symbol $=$ in our language, with the intent that

$$s = t$$

should be interpreted as: the terms $s$ and $t$ denote the same element.

Thus, unlike with all the other relation symbols, the meaning of $=$ is fixed once and for all: it is always interpreted as equality. This is different from all other relation symbols that can have various different interpretations.

It is also interesting to consider systems without equality.

The system just described is first-order logic with equality.

One can also consider first-order logic without equality (there is no direct way of asserting equality of terms).

Lastly, one can consider the fragment of FOL that has no function symbols nor equality, only relations (the actual predicate logic).

In the presence of equality, one can still fake functions to a degree by considering relations $F$ such that

$$\forall\, x\, \exists\, y\, F(x,y) \wedge \forall\, x, u, v\, \Big(F(x,u) \wedge F(x,v) \Rightarrow u = v\Big)$$

Thus $F$ is a total and single-valued binary relation, the textbook definition of a function.

**Definition**

The set of formulae of FOL is defined by

- Every atomic formula is a is a formula.
- If $\varphi$ and $\psi$ are formulae, so are $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, and $(\varphi \Rightarrow \psi)$.
- If $\varphi$ is a formula and $x$ a variable, then $(\exists\, x\, \varphi)$ and $(\forall\, x\, \varphi)$ are also formulae.

So these are compound formulae versus the atomic formulae from above.

The $\varphi$ in $(\exists\, x\, \varphi)$ and $(\forall\, x\, \varphi)$ is called the matrix of the formula.

Note that a term by itself is not a formula: it denotes an element (if it has no free variables) rather than a truth value.

Our definition establishes a strict syntax, the kind that makes it easy for a parser to analyze an expression.

Unfortunately, for human readers, this is a problem, legibility is quite poor. So, we also have a sloppy syntax that makes things easy to read and comprehend.

For example, we omit unnecessary parentheses and write formulae such as

$$\forall\, x\, (R(x) \Rightarrow \exists\, y\ S(x, y))$$

The intended meaning is: "For any $x$, if $x$ has property $R$, then there exists a $y$ such that $x$ and $y$ are related by $S$."

Binary relation and function symbols are often written in infix notation, using standard mathematical symbols. In particular in number theory and algebra one writes $x + y < z$ rather than the cumbersome $<(+(x, y), z)$.

This is mildly annoying for the parser, but a big win for humans.

One often contracts quantifiers of the same kind into one block.

$$\forall\, x \forall\, y \exists\, z\, (= (+(x, z), y))$$

Sloppy, but eminently readable, style

$$\forall\, x, y \exists\, z\, (x + z = y)$$

When it comes to rendering quantifiers in math text, some authors appear to suffer from temporary insanity and try to make things as difficult to parse as humanly possible. For example:

$$(\forall\, x\,)(\forall\, y\,)(\exists\, z\,)f(x,y) = g(z)$$

or, even worse,

$$\forall\, x\,, \forall\, y\,, \exists\, z\,, f(x,y) = g(z)$$

as opposed to

$$\forall\, x, y\, \exists\, z\, \Big(f(x,y) = g(z)\Big)$$

In many texts implication is written as $\varphi \to \psi$.

Alas, this notation becomes less than ideal when we also deal with functions or relations $f : A \to B$ or in the context of diagrams.

We will write $\varphi \Rightarrow \psi$ if we want to make sure the arrow is understood as implication.

Other old-fashioned notation: $\mathbf{C}(\varphi, \psi)$ and $\varphi \supset \psi$.

We assume that implication associates to the right, so

$$\varphi \Rightarrow \psi \Rightarrow \chi \quad \text{means} \quad \varphi \Rightarrow (\psi \Rightarrow \chi)$$

**Definition**

A variable that is not in the range of a quantifier is free or unbound (as opposed to bound). We write $\mathrm{FV}(\varphi)$ for the set of free variables in $\varphi$. A formula without free variables is closed, or a sentence.

One often indicates free variables like so:

| | |
|---|---|
| $\varphi(x, y)$ | $x$ and $y$ may be free |
| $\exists\, x\, \varphi(x, y)$ | only $y$ may be free |
| $\forall\, y \exists\, x\, \varphi(x, y)$ | closed. |

Substitutions of terms for free variables are indicated like so:

| | |
|---|---|
| $\varphi(s, t)$ | replace $x$ by $s$, $y$ by $t$ |
| $\varphi[s/x, t/y]$ | replace $x$ by $s$, $y$ by $t$ |

We will explain shortly how to compute the truth value of a sentence.

The notation

$$\varphi(x_1, \ldots, x_n)$$

only expresses the fact that $\mathsf{FV}(\varphi) \subseteq \{x_1, \ldots, x_n\}$.

Note: this does not mean that they all actually occur, some or even all of them may be missing. This turns out to be preferable over the alternative.

This is really no different from saying that $2x^2 - y$ is a polynomial in variables $x$, $y$ and $z$.

**Exercise**

*Construct an algorithm that computes free variables and determines the scope of quantifiers.*

A priori, a formula $\varphi(x)$ with a free variable $x$ has no truth value associated with it: we need to replace $x$ by a ground term to be able to evaluate.

However, taking inspiration from equational logic, it is convenient to define the truth value a formula with free variables to be the same as its universal closure: put universal quantifiers in front, one for each free variable.

$$\forall\, x, y, z \left(x * (y * z) = (x * y) * z\right)$$

is somewhat less elegant and harder to read than

$$x * (y * z) = (x * y) * z$$

In algebra, the latter form is much preferred.

Note that according to our definition it is perfectly agreeable to quantify over an already bound variable. To make the formula legible one then has to rename variables. For practical reasons it is best to simply disallow clashes between free and bound variables.

$$\forall\, x \left( R(x) \wedge \exists\, x \forall\, y\ S(x, y) \right)$$

Better: rename the $x$ inside

$$\forall\, x \left( R(x) \wedge \exists\, z \forall\, y\ S(z, y) \right)$$

These issues are very similar to problems that arise in programming languages (global and local variables, scoping issues). They need to be addressed but are not of central importance.

Our definition of a formula of FOL is sufficiently precise to reason about the logic, but it is still a bit vague if we try to actually implement an algorithm that operates on these formulae.

**Exercise**

*Explain how to implement formulae in FOL. Describe the data structure and make sure that it can be manipulated in a reasonable way.*

**Exercise**

*Give a precise definition of free and bound variables by induction on the buildup of the formula.*

**Exercise**

*Implement a renaming algorithm that removes potential scoping clashes in the variables of a formula.*

**Exercise**

*Implement a substituting algorithm that replaces a free variable in a formula by a term.*

So what exactly is the context that we need to interpret a formula in FOL, what additional information do we need to be able to check whether a formula is true of false?

Fix a language $\mathcal{L} = \mathcal{L}(\Sigma)$ of some specific signature.

**Definition**

A (first-order) structure or interpretation (of signature $\Sigma$) is a carrier set together with a collection of functions and relations on that set, one relation/function for each symbol in the signature.

In order to interpret formulae in $\mathcal{L}(\Sigma)$ over a structure, the signatures have to match: for each function symbol we have an actual function, and for each relation symbol we have an actual relation.

So a first-order structure in general looks like so:

$$\mathfrak{A} = \langle A; f_1, f_2, \ldots, R_1, R_2, \ldots \rangle$$

The set $A$ is the carrier set of the structure.

In addition, we have a list of functions on $A$, and a list of relations on $A$ (all of the right arity).

In practice, unary and binary functions and relations are by far the most important in applications, but higher arities may occur.

That point behind a FO structure is that it provides an interpretation for each of the non-logical symbols in the language $\mathcal{L}(\Sigma)$.

$$f \text{ function symbol} \quad \rightsquigarrow \quad f^{\mathfrak{A}} \text{ a function in } \mathfrak{A}$$

$$R \text{ function symbol} \quad \rightsquigarrow \quad R^{\mathfrak{A}} \text{ a relation in } \mathfrak{A}$$

Given this interpretation of function and relation symbols over $\mathfrak{A}$, we can determine whether a formula holds true over $\mathfrak{A}$.

This is very different from trying to establish universal truth. All we are doing here is to confirm that, in the context of a particular structure, a certain formula is valid. As it turns out, this is all that is really needed in the real world.

Of course, when the structure changes the formula may well become false.

# Aside: Dots

There is an alternative notation the literature that distinguishes between a formal symbol and its interpretation by placing a dot on top of the former.

For example, we express the transitivity axiom for order relations as follows:

$$\forall\, x, y, z \left( x \mathrel{\dot{<}} y \wedge y \mathrel{\dot{<}} \Rightarrow x \mathrel{\dot{<}} z \right)$$

Similarly, we can formalize the extensionality axiom of set theory like so:

$$\forall\, x, y \left( \forall\, z \left( z \mathrel{\dot{\in}} x \Leftrightarrow z \mathrel{\dot{\in}} y \right) \Rightarrow x \doteq y \right)$$

It is often not worth the effort of keeping the notation for symbols and interpretations strictly separate.

We'll quite casual about this, things are always clear from context (famous last words).

Here is a little toy formula that we will use as an example:

$$\Phi \equiv \forall\, x \left( G(x) \Rightarrow R(f(x)) \right)$$

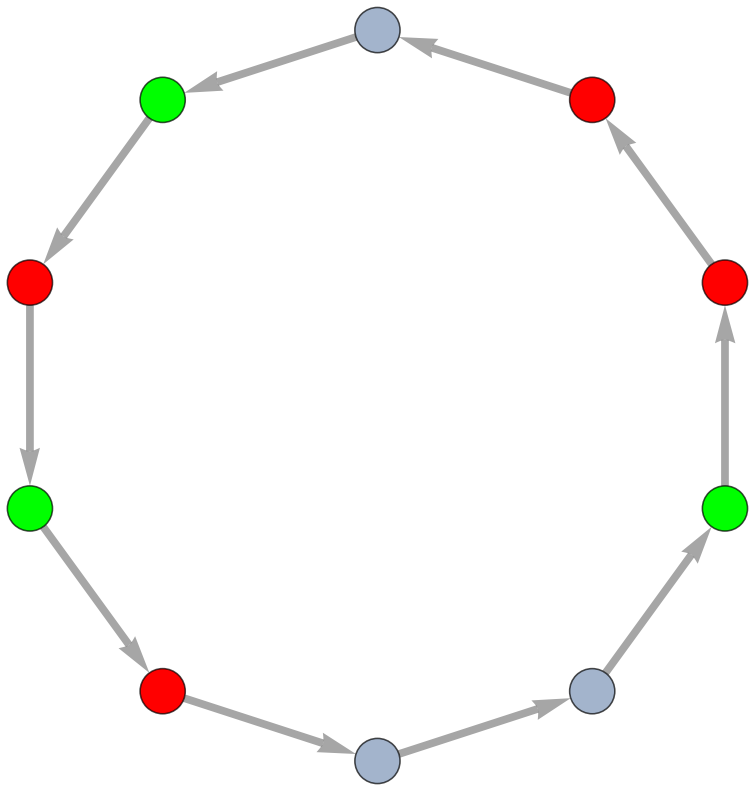
Here $f$ is a unary function symbol, and $G$ and $R$ are unary predicates (so the signature is $(1; 1, 1)$).

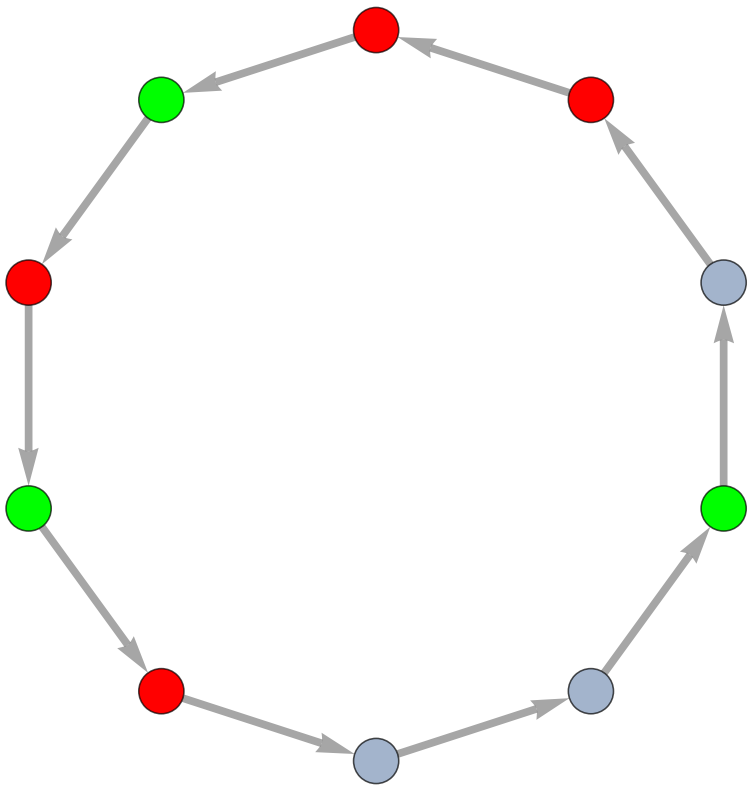
What does this formula mean intuitively? Suppose we have a suitable structure

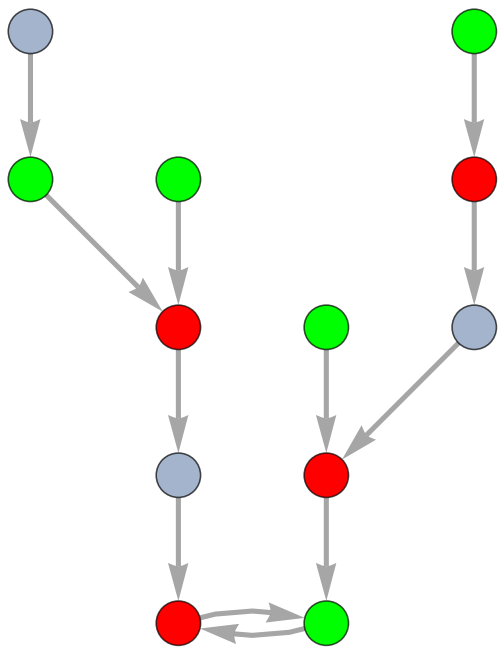
$$\mathcal{A} = \langle A; f^{\mathcal{A}}, G^{\mathcal{A}}, R^{\mathcal{A}} \rangle$$

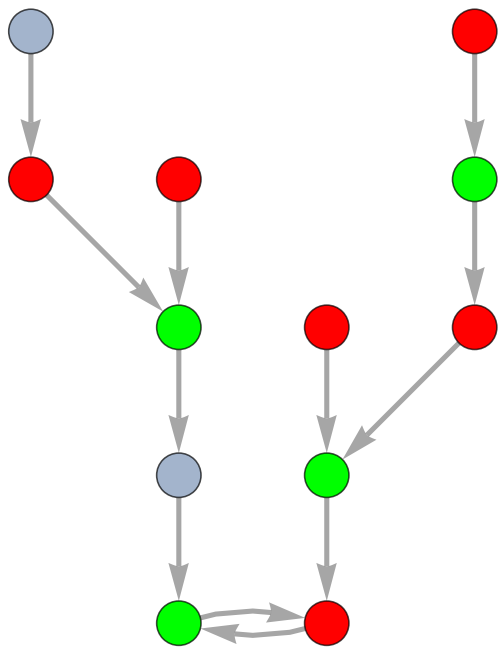
where $A$ is some set, $f^{\mathcal{A}} : A \to A$ and $G^{\mathcal{A}}, R^{\mathcal{A}} \subseteq A$.

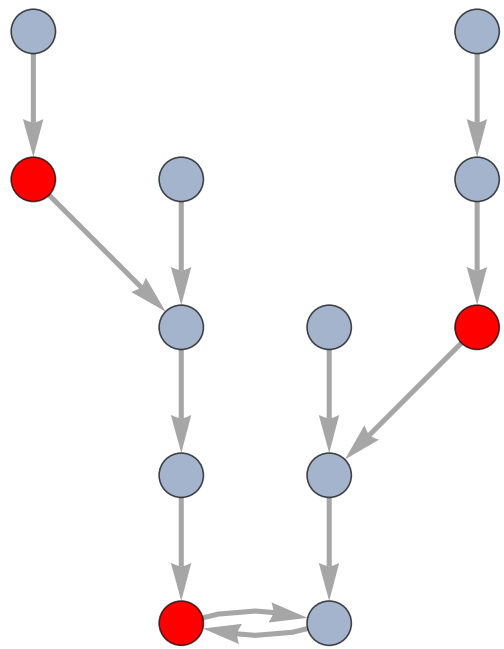
Intuitively, think of $G$ as meaning *green* and $R$ as meaning *red*. The $\Phi$ holds in such a structure if every green point maps to a red point under $f$.











How about

$$A = \mathbb{N}$$

$$f(x) = x + 2$$

$$G \quad \text{is prime}$$

$$R \quad \text{is odd}$$

This is an entirely different can of worms. Unlike the previous examples, this one is infinite and one needs to have some basic knowledge of arithmetic to check that $\Phi$ is false.

How about this formula:

$$\forall\, x\, \exists\, y \left( x < y \wedge G(y) \wedge G(f(y)) \right)$$

More generally, to deal with basic arithmetic we use the language $\mathcal{L}(+, \cdot, 0, 1; <)$ with signature $(2, 2, 0, 0; 2)$.

We can interpret a formula in this language over any structure of the same type. Of course, the most important structure for arithmetic is

$$\mathfrak{N} = \langle \mathbb{N}; +, \cdot, 0, 1, < \rangle$$

the set of natural numbers together with the standard operations but we will see that there are others.

Note the slight abuse of notation here (as is standard practice). More precise would be to write: The function symbol $+$ is interpreted by $+^{\mathfrak{N}}$, the standard operation of addition of natural numbers.

For our purposes there is no gain in being quite so careful.

With this interpretation over $\mathfrak{N}$, the formula

- $0 < 1$ is true
- $\forall\, x \exists\, y\, (x < y)$ is true
- $\forall\, x, y\, (x < y \Rightarrow \exists\, z\, (x < z \wedge z < y))$ is false

How about the primality formula

$$\varphi(x) \equiv 1 < x \wedge \forall\, y, z\, (x = y \cdot z \Rightarrow y = 1 \vee z = 1)$$

This formula has a free variable $x$, so we need to bind $x$ (replace it by a term) before we can determine truth.

We use the numeral $\underline{n} = \underbrace{1 + 1 + \ldots + 1}_{n \text{ times}}$ to represent the natural number $n$ as a term. The set of primes is then

$$\{\, n \mid \varphi(\underline{n}) \text{ holds in } \mathfrak{N} \,\}.$$

Strictly speaking, $\underline{n} = \underbrace{1 + 1 + \ldots + 1}_{n \text{ times}}$ is also sloppy, we really should define

$$\begin{aligned} \underline{0} &= 0 && \text{the term, not the number} \\ \underline{n{+}1} &= +(\underline{n}, 1) \end{aligned}$$

by induction, so that $\underline{n}$ is a bonified term of our system.

One often avoids this level of precision, but for any algorithmic treatment there is no way around it.

Suppose we interpret our formulae over the structure of the real numbers instead. This is possible since we are dealing with the same signature $(2, 2, 0, 0; 2)$:

$$\mathfrak{R} = \langle \mathbb{R}; +, \cdot, 0, 1; < \rangle$$

Now $+$ refers to addition of reals, $0$ is the real number zero, and so on. These operations are much more complicated, but as far as our formula is concerned all that matters is that they have the right arity.

Over the reals the following density formula holds:

$$\theta \equiv \forall\, x, y\, (x < y \Rightarrow \exists\, z\, (x < z \wedge z < y)$$

It also holds over $\mathbb{Q}$, but over $\mathbb{Z}$ or $\mathbb{N}$ it is clearly false.

We can now give a first informal definition of truth or validity.

**Definition**

A formula of first-order logic is valid or true if it holds over any structure of the appropriate signature.

**Dire Warning:** We require to formula to hold over any first-order structure of the appropriate signature (so we can interpret the function and relation symbols).

Any structure whatsoever, not just one that you are particularly fond of.

So the formula $\forall\, x, y\, (x * y = y * x)$ is not valid, but it can be used as an axiom to filter out commutative operations.

It is standard to assume that the carrier set of a first-order structure is not empty. This has the effect that the formula

$$\exists\, x\, (x = x)$$

is valid. Experience shows that this is more convenient than allowing empty structures.

For example, $\forall\, x\, (x \neq x)$ is true in the empty structure, which looks strange.

Also, $\forall\, x\, (x \neq x) \wedge \bot$ has prenex normal form $\forall\, x\, (x \neq x \wedge \bot)$, but they are not equivalent over the empty structure: the former is false, the latter is true.

It is clear that a formula like $\varphi \Rightarrow \varphi$ is valid. In fact, if we replace the propositional variables in any tautology by arbitrary FOL sentences, we obtain a valid sentence. More precisely, let

$$\varphi(p_1, p_2, \ldots, p_n)$$

be a tautology with propositional variables $p_1, p_2, \ldots, p_n$. Let $\psi_1, \ldots, \psi_n$ be arbitrary sentences of FOL. Then

$$\varphi(\psi_1, \psi_2, \ldots, \psi_n)$$

is a valid sentence of FOL. True, but not particularly interesting.

Again in analogy to propositional logic we can define satisfiability.

**Definition**

A formula of FOL is satisfiable if it is true for some interpretation of the variables, functions and relations. It is a contradiction if it is true for no interpretation of the variables, functions and relations.

For example, in the language of binary relations the formula

$$x\ R\ x \wedge (x\ R\ y \wedge y\ R\ z \Rightarrow x\ R\ z)$$

is satisfied exactly by any structure $\mathfrak{A}$ that carries a reflexive transitive relation $R^{\mathfrak{A}}$. On the other hand,

$$\forall\, x\, (x \neq c)$$

where $c$ is a constant is a contradiction: we can interpret $x$ as the element in the structure denoted by $c$ in which case equality holds.

Slightly more complicated examples for valid formulae are

$$\forall\, x \forall\, y\, \varphi(x,y) \Rightarrow \forall\, y \forall\, x\, \varphi(x,y)$$
$$\exists\, x \exists\, y\, \varphi(x,y) \Rightarrow \exists\, y \exists\, x\, \varphi(x,y)$$
$$\exists\, x \forall\, y\, \varphi(x,y) \Rightarrow \forall\, y \exists\, x\, \varphi(x,y)$$

Note, though, that the following is not valid:

$$\forall\, x \exists\, y\, \varphi(x,y) \;\Rightarrow\; \exists\, y \forall\, x\, \varphi(x,y)$$

**Exercise**

*Verify that the first three formulae are true and come up with an example that shows that the last one is not.*

Another good source of valid formulae are assertions about the number of elements in the underlying structure. How do we say "there are exactly $n$ elements in the ground set" in FOL? First, a formula which states that there are at most $n$ elements.

$$\mathsf{Cnt}_{\leq n} \equiv \exists\, x_1, \ldots, x_n \,\forall\, y\, (y = x_1 \vee \ldots \vee y = x_n)$$

Second, a formula which states that there are at least $n$ elements.

$$\mathsf{Cnt}_{\geq n} \equiv \exists\, x_1, \ldots, x_n\, (x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \ldots \wedge x_{n-1} \neq x_n)$$

All these formulae are clearly satisfiable. The conjunction $\mathsf{Cnt}_{\leq n} \wedge \mathsf{Cnt}_{\geq n}$ pins down the cardinality to exactly $n$. Also, a formula

$$\mathsf{Cnt}_{\geq n} \Rightarrow \mathsf{Cnt}_{\geq m}$$

is valid whenever $m \leq n$.

How about a formula that states that there are infinitely many elements?

$$\mathsf{Cnt}_{\geq 1} \wedge \mathsf{Cnt}_{\geq 2} \wedge \ldots \wedge \mathsf{Cnt}_{\geq n} \wedge \ldots$$

does not work since it is not a finite formula. There are logics where this is allowed, but not in our setting. The attempt

$$\forall\, n\, \mathsf{Cnt}_{\geq n}$$

also fails; we cannot quantify over formulae in our logic.

**Exercise**

*Explain precisely why these "formulae" are not admissible in FOL.*

After some more fruitless attempts one might suspect that the statement "there are infinitely many thingies" cannot be expressed with a single formula in FOL (the $c_i \neq c_j$ trick requires infinitely many formulae).

Wrong! Let $f$ be a unary function symbol and $c$ a constant. Consider

$$\varphi \equiv \forall\, x\, (f(x) \neq c) \wedge \forall\, x, y\, (f(x) = f(y) \Rightarrow x = y).$$

So $\varphi$ states that $f$ is not surjective but injective. Hence in any interpretation that makes $\varphi$ true the carrier set must be infinite.

**Exercise**

*Use a total order to produce another formulae in FOL that forces the ground set to be infinite.*

Again, there are natural decision problems associated with this classification.

| | |
|---|---|
| Problem: | **Validity** |
| Instance: | A FOL formula $\varphi$. |
| Question: | Is $\varphi$ valid? |

| | |
|---|---|
| Problem: | **Satisfiability** |
| Instance: | A FOL formula $\varphi$. |
| Question: | Is $\varphi$ satisfiable? |

As usual, there is the search version of Satisfiability: we would like to construct a satisfying interpretation if one exists. Note that this may well entail the construction of an infinite structure.

As one might suspect from the few examples, these problems are much harder in FOL than in propositional/equational logic and will turn out to be highly undecidable in general.

In computer science one is interested in the following version of the Entscheidungsproblem, usually called the model checking problem. We are dealing with FOL here, other logics are more important in typical applications.

| | |
|---|---|
| Problem: | **Model Checking** |
| Instance: | A FO formula $\varphi$, a FO structure $\mathfrak{A}$. |
| Question: | Is $\varphi$ true over $\mathfrak{A}$? |

Ideally we would like to have an algorithm $\mathrm{ValidQ}(\mathfrak{A}, \varphi)$ that solves the problem. Alas, there are two major issues:

- In general, model checking must be undecidable.
- $\varphi$ is easy to specify as a data structure, but it is entirely unclear how we could deal with $\mathfrak{A}$. As written, the problem makes no sense.

Time to give a precise definition of validity and satisfiability. We begin by defining assignments in the context of FOL.

**Definition**

An assignment or valuation (over a structure $\mathfrak{A}$) associates variables of the language with elements in the ground set $A$.

Given an assignment $\sigma : \text{Var} \to A$, we can associate an element $\sigma(t)$ in $A$ with each term $t$.

- $t = x$: then $\sigma(t) = \sigma(x)$
- $t = f(r_1, \ldots, r_n)$: then $\sigma(t) = f^{\mathfrak{A}}(\sigma(r_1), \ldots, \sigma(r_n))$

**Example**

Over $\mathfrak{N}$ let $\sigma(x) = 3$. Then $\sigma(x \cdot (1 + 1))) = 6$ whereas $\sigma(x) = 0$ produces $\sigma(x \cdot (1 + 1))) = 0$.

Once we have an assignment for all the free variables in an atomic formula we can determine a truth value for it.

**Definition**

Let $\sigma$ be an assignment over a structure $\mathfrak{A}$ and $\varphi = R(t_1, \ldots, t_n)$ an atomic formula. Define the truth value of $\varphi$ (under $\sigma$ over $\mathfrak{A}$) to be

$$\mathfrak{A}_\sigma(\varphi) = \begin{cases} \text{tt} & \text{if } R^{\mathfrak{A}}(\sigma(t_1), \ldots, \sigma(t_n)) \text{ holds,} \\ \text{ff} & \text{otherwise.} \end{cases}$$

**Example**

Over the natural numbers $\mathfrak{N}$ suppose $\sigma(x) = 0$ and $\sigma(y) = 1$. Then

$$\mathfrak{N}_\sigma(x + y < 1 + 1) = \mathfrak{N}_\sigma(0 + 1 < 1 + 1) = \text{tt}$$

but for $\sigma(x) = \sigma(y) = 1$ we get

$$\mathfrak{N}_\sigma(x + y < 1 + 1) = \mathfrak{N}_\sigma(1 + 1 < 1 + 1) = \text{ff}$$

Once we have a truth value for atomic formulae, we can extend this evaluation to compound formulae without quantifiers.

**Definition (Propositional Connectives)**

| $\varphi$ | $\mathfrak{A}_\sigma(\varphi)$ |
|---|---|
| $\psi \wedge \chi$ | $H_{\mathsf{and}}(\mathfrak{A}_\sigma(\psi), \mathfrak{A}_\sigma(\chi))$ |
| $\psi \vee \chi$ | $H_{\mathsf{or}}(\mathfrak{A}_\sigma(\psi), \mathfrak{A}_\sigma(\chi))$ |
| $\neg\psi$ | $H_{\mathsf{not}}(\mathfrak{A}_\sigma(\psi))$ |

Here the functions $H_{\mathsf{and}}$, $H_{\mathsf{or}}$ and $H_{\mathsf{not}}$ are Boolean functions of arity 2, and $H_{\mathsf{not}}$ is Boolean function of arity 1 (expressing logical *and*, *or* and *not*).

We are using the fact that formulae form a recursive datatype, so we can define things by induction.

# Example

Suppose $\sigma(x) = 0$ and $\sigma(y) = 1$. Then

$$\begin{aligned}
\mathfrak{N}_\sigma(x < y \wedge y < x) &= H_{\mathsf{and}}(\mathfrak{N}_\sigma(x < y), \mathfrak{N}_\sigma(y < x)) \\
&= H_{\mathsf{and}}(\mathfrak{N}(0 < 1), \mathfrak{N}(1 < 0)) \\
&= H_{\mathsf{and}}(\mathsf{tt}, \mathsf{ff}) \\
&= \mathsf{ff}
\end{aligned}$$

This is just a simple recursive computation using a bit of table lookup.

For an assignment $\sigma$, let us write $\sigma[a/x]$ for the assignment that is the same as $\sigma$ everywhere, except that $\sigma[a/x](x) = a$. Think of this as substituting "$a$ for $x$" in $\sigma$.

**Definition (Quantifiers)**

- $\varphi = \exists\, x\, \psi$:

  Then $\mathfrak{A}_\sigma(\varphi) = \mathsf{tt}$ if there is an $a$ in $A$ such that $\mathfrak{A}_{\sigma[a/x]}(\psi) = \mathsf{tt}$.

- $\varphi = \forall\, x\, \psi$:

  Then $\mathfrak{A}_\sigma(\varphi) = \mathsf{tt}$ if for all $a$ in $A$ $\mathfrak{A}_{\sigma[a/x]}(\psi) = \mathsf{tt}$.

Note that $\sigma$ only needs to be defined on the free variables of $\varphi$ to produce a truth value for $\varphi$, the values anywhere else do not matter. If $\varphi$ is a sentence, $\sigma$ can be totally undefined.

**Definition (Validity)**

A formula $\varphi$ is valid in $\mathfrak{A}$ under assignment $\sigma$ if $\mathfrak{A}_\sigma(\varphi) = \mathrm{tt}$.

A formula $\varphi$ is valid in $\mathfrak{A}$ if it is valid in $\mathfrak{A}$ for all assignments $\sigma$. The structure $\mathfrak{A}$ is then said to be a model for $\varphi$ or to satisfy $\varphi$.

A sentence is valid (or true) if it is valid over all structures (of the appropriate signature).

Notation:

$$\mathfrak{A} \models_\sigma \varphi \qquad \mathfrak{A} \models \varphi \qquad \models \varphi$$

One uses the same notation for sets of formulae $\Gamma$. So $\mathfrak{A} \models \Gamma$ means that $\mathfrak{A} \models \varphi$ for all $\varphi \in \Gamma$.

Note the condition for validity: the formula has to hold in all structures.

**Definition**

A formula is satisfiable if there is some structure $\mathfrak{A}$ and some assignment $\sigma$ for all the free variables in $\varphi$ such that $\mathfrak{A} \models_\sigma \varphi$.

In other words, the existentially quantified formula

$$\exists\, x_1, \ldots, x_n\ \varphi(x_1, \ldots, x_n)$$

has a model, where $x_1, \ldots, x_n$ are all the free variables of $\varphi$.

In shorthand: $\exists\, \boldsymbol{x}\, \varphi(\boldsymbol{x})$.

This is analogous to validity where we insist that $\forall\, x_1, \ldots, x_n\ \varphi(x_1, \ldots, x_n)$ holds, or $\forall\, \boldsymbol{x}\, \varphi(\boldsymbol{x})$ in compact notation.

It is often more convenient to leave off the universal quantifiers.

The definitions of truth given here is due to A. Tarski (two seminal papers, one in 1933 and a second one in 1956, with R. Vaught).

A frequent objection to this approach is that we are using "for all" to define what a universal quantifier means.

True, but the formulae in our logic are just syntactic objects, data structures in some rectype and can be represented, say, as strings. Their meaning is defined in terms of first-order structures, which live in the real world of mathematics and TCS.

Occasionally we can even compute the truth value of a formula: we need to be able to perform certain operations in the structure: evaluate $f^{\mathfrak{A}}$, loop over all elements, search over all elements, ... For finite structures given by, say, lookup tables, this works fine (at least if we ignore efficiency). For infinite structures, we run into major problems.

Suppose $\mathfrak{A}$ is a structure of some signature $\Sigma$. In order to describe $\mathfrak{A}$ it is often helpful to augment the language $\mathcal{L}(\Sigma)$ by constant symbols $c_a$ for each element $a$ in the carrier set $A$ of $\mathfrak{A}$, obtaining a new signature $\Sigma_{\mathfrak{A}}$.

$\mathfrak{A}$ is naturally also a structure of signature $\Sigma_{\mathfrak{A}}$.

This step is not necessary when there already are terms in the language for all the elements of the structure. E.g., in arithmetic we can denote every natural number by a ground term

$$\underline{n} = 1 + 1 + \ldots + 1$$

This works since we are dealing with the naturals, but in general there is no reason why every element in a structure should be denoted by a term.

For example, suppose we want to deal with the reals. The standard language has constants for $0$ and $1$ but nothing else.

$$\mathfrak{R} = \langle \mathbb{R}; +, \cdot, 0, 1; < \rangle$$

Using the numeral trick, we can obtain terms for rationals (at least if we add subtraction and division to our signature), but no more.

Just to write down all available facts about $\mathbb{R}$ we need to add uncountably many constants, one for each real other than the rationals. This is no problem at all in set theory.

But it wrecks the language: the new constants are not finitary data structures. We cannot even build a parser.

**Definition**

The (atomic) diagram of a structure of some fixed signature is the set of all atomic sentences and their negations in $\mathcal{L}(\Sigma_{\mathfrak{A}})$ that are valid in $\mathfrak{A}$.

In symbols: $\operatorname{diag}\mathfrak{A}$.

The point is that the validity of any formula over $\mathfrak{A}$ is completely determined by $\operatorname{diag}\mathfrak{A}$: no other information is used in our definition of truth. Hence our putative model checking algorithm could look like this:

$$\mathsf{ValidQ}(\operatorname{diag}\mathfrak{A}, \varphi)$$

If the underlying structure $\mathfrak{A}$ is finite (and thus the diagram is essentially finite), then we can actually perform this computation: it is just recursion and copious table lookups. In fact, ValidQ will be primitive recursive given any reasonable coding.

How do we represent the atomic diagram $\operatorname{diag}\mathfrak{A}$? Given enough constants we can simply write down a table.

E.g., for a unary function symbol $f$ we can use a table with entries $c_a$ and $c_b$ provided that $b = f^{\mathfrak{A}}(a)$. Each entry corresponds to an identity $f(c_a) = c_b$ in the diagram.

For binary function symbols we get a classical Cayley style "multiplication table", and higher dimensional tables for functions of higher arity.

Relations can be handled by similar tables with entries in $\mathbb{B}$. This corresponds to the familiar interpretation of a relation $R \subseteq A^k$ as a function $R : A^k \to \mathbb{B}$.

So, the whole diagram is just a bunch of tables using special constants for all elements in the structure and Boolean values. For small finite structures this is a perfectly good representation. (Though in reality it only works when the structures are not too large).

**Theorem**

*Model checking for finite structures is decidable.*

In fact, it is easy to see that the algorithm is primitive recursive. Unfortunately, from a computational complexity perspective this is not exactly very useful.

Pushing ahead into the realm of infinite structures things become much more complicated. If the carrier set is uncountable our machinery from classical computability theory simply does not apply—the individual elements are not finitary objects and we have no handle.

However, if the carrier set is countable, computability theory does apply and we can use it to measure the complexity of the structure.

**Definition**

The theory or (complete) diagram of $\mathfrak{A}$ is the collection of all sentences in $\mathcal{L}(\Sigma_{\mathfrak{A}})$ that are valid in $\mathfrak{A}$.

$\mathfrak{A}$ is computable if its atomic diagram is decidable.

$\mathfrak{A}$ is decidable if its complete diagram is decidable.

In symbols: $\mathrm{Th}(\mathfrak{A})$ or $\mathrm{diag}^c\,\mathfrak{A}$.

The technical term **theory** in logic is quite overloaded. It can mean

- A collection of first-order sentences.
- A collection of first-order sentences that is closed under deduction.
- The collection of first-order sentences true in some structure.

This is why we mentioned the hopelessly clumsy "complete diagram" as a way to disambiguate. It's not very popular.

In the atomic diagram of $\mathfrak{N}$ we have lots of truly exciting facts such as

$\underline{0} + \underline{0} = \underline{0}$, $\underline{0} + \underline{1} = \underline{1}$, $\underline{0} + \underline{2} = \underline{2}$, ..., $\underline{1} + \underline{0} = \underline{1}$, $\underline{1} + \underline{1} = \underline{2}$, $\underline{1} + \underline{2} = \underline{3}$, ..., $\underline{17} + \underline{19} = \underline{36}$, ...

$\underline{0} \cdot \underline{0} = \underline{0}$, $\underline{0} \cdot \underline{1} = \underline{0}$, $\underline{0} \cdot \underline{2} = \underline{0}$, ..., $\underline{1} \cdot \underline{0} = \underline{0}$, $\underline{1} \cdot \underline{1} = \underline{1}$, $\underline{1} \cdot \underline{2} = \underline{2}$, ..., $\underline{17} \cdot \underline{19} = \underline{323}$, ...

$\underline{0} < \underline{1}$, $\underline{0} < \underline{2}$, $\underline{0} < \underline{3}$, ..., $\underline{1} < \underline{2}$, $\underline{1} < \underline{3}$, $\underline{1} < \underline{4}$, ..., $\underline{17} < \underline{19}$, ...

And more:

$\underline{2} + \underline{3} < \underline{2} \cdot \underline{3}$

In the complete diagram of $\mathfrak{N}$ we have the whole atomic diagram, plus formulae like

$$\forall\, x, y\, (x + y = y + x)$$
$$\forall\, x, y, z\, (x + (y + z) = (x + y) + z)$$
$$\forall\, x\, \exists\, y\, (x < y)$$
$$\exists\, x\, \forall\, y\, \neg(y < x)$$
$$\forall\, x\, \exists\, y\, \Big(x < y \wedge \mathsf{prime}(y)\Big)$$

But we do **not** know whether the next formula is in the diagram:

$$\forall\, x\, \exists\, y\, \Big(x < y \wedge \mathsf{prime}(y) \wedge \mathsf{prime}(y + \underline{2})\Big)$$

The structure $\mathfrak{N}$ of arithmetic is computable, but not decidable, even for annoyingly simple formulae involving just a single existential quantifier.

In fact, $\mathrm{diag}^c\,\mathfrak{N}$ is highly undecidable, an infinite hierarchy away from decidable.

The structure $\mathfrak{R}$ of the reals is decidable, but not computable.

This may sound mighty strange, but note that we are only dealing with first-order formulae over $\mathcal{L}(+,\cdot,0,1;<)$. Surprisingly, for these one can check validity over $\mathbb{R}$, due to a famous theorem by A. Tarski from 1951 (uses quantifier elimination).

There is a huge literature on how to make this decision algorithm reasonably efficient.

Recall our notion of validity: $\varphi$ is valid iff $\mathfrak{A} \models \varphi$ for all structures $\mathfrak{A}$ of the right signature. In other words, we can interpret the meaning of the non-logical symbols any which way we like. The meaning of the logical symbols is fixed, however, $\wedge$ means *and*, basta.

Very often, one is instead interested in truth over a single structure. For example, in arithmetic one wants to know whether $\mathfrak{N} \models \varphi$.

Is this a bug or a feature for first-order logic?

We are not going to get involved with technical details, but one of the great features of first-order logic is that it has a very nice proof theory.

One can set up a few proof rules that encapsulate the kind of reasoning that is used (in an informal manner) in all mathematical proofs. For example, here are some possible rules for propositional logic:

expansion $\dfrac{\varphi}{\psi \vee \varphi}$

contraction $\dfrac{\varphi \vee \varphi}{\varphi}$

modus ponens $\dfrac{\varphi \qquad \varphi \Rightarrow \psi}{\psi}$

Note that this is really just so much wordprocessing.

To handle quantifiers, one needs rules along the lines of

$$\frac{\phi(t)}{\exists\, x\, \phi(x)}\ (\exists i) \qquad \frac{\exists\, x\, \phi(x)}{\phi(c)}\ (\exists e)$$

$$\frac{\phi(c)}{\forall\, x\, \phi(x)}\ (\forall i) \qquad \frac{\forall\, x\, \phi(x)}{\phi(t)}\ (\forall e)$$

where $x$ is a variable, $c$ a constant, and $t$ a term.

**Warning:** While these rules are correct in spirit, as stated they are not sound. We need some additional technical conditions, take a look at any basic textbook (a fairly painless one is *A Mathematical Introduction to Logic* by H.B. Enderton).

At any rate, using a collection of proof rules we can define a notion of provability or derivability, written $\Gamma \vdash \varphi$, meaning:

> $\varphi$ can be derived from the standard axioms of logic and $\Gamma$, using only a few given proof rules.

This is also called syntactic entailment. It turns out to be an accurate representation of what mathematicians mean by "proof." Of course, this is **not** how proofs are written by the "working mathematician," but it seems that they could[†] be rephrased this way.

[†]Formalizing published proofs requires a substantial amount of qualified manpower; support systems are getting better but are still a bit raw.

A single step in a proof $\varphi \vdash_1 \psi$ is brutally simple, easily primitive recursive. Suppose we have a decidable set of axioms $\Gamma$, and define the theory of $\Gamma$ to be the set of theorems provable from $\Gamma$:

$$\mathsf{Th}(\Gamma) = \{\, \varphi \mid \Gamma \vdash \varphi \,\}$$

Then $\mathsf{Th}(\Gamma)$ is semidecidable. This is easiest to see from the perspective a recursively enumerability: we can systematically generate all possible proofs in some natural order.

Alas, that's where it ends: for many $\Gamma$ the theory is not decidable.

For example, the standard group axioms produce an undecidable theory.

On the other hand, Abelian groups produce a decidable theory.

In pure mathematical logic, it is perfectly fine to consider sets of axioms $\Gamma$ that are arbitrarily complicated: undecidable or even uncountable (using an uncountable language).

But in the computational universe we will insist that any reasonable set of axioms is decidable: otherwise we cannot check whether a given argument is a correct proof. All standard axiom systems of math are easily decidable. So the theories are semidecidable.

Note we still get a semidecidable theory even when $\Gamma$ is only semidecidable, but that's already pushing things a bit. Civilized axioms should not be based on the Halting problem.

On the side of structures and models, we have already seen a detailed definition of the notion of semantic entailment or semantic consequence, in symbols $\Gamma \models \varphi$, meaning that for all structures $\mathfrak{A}$:

$$\mathfrak{A} \models \Gamma \quad \text{implies} \quad \mathfrak{A} \models \varphi.$$

For example, if $\Gamma$ are the standard group axioms, then the formula

$$\varphi \equiv \left((x * y)^{-1} = y^{-1} * x^{-1}\right)$$

is a consequence of $\Gamma$. In this case, it is not hard to see that indeed $\Gamma \vdash \varphi$ (in fact, this does not even require FOL, a less complicated equational logic is good enough).

# Example

Let

$$\begin{aligned} (\mathsf{inj}) &\equiv \forall\, x, y \left( f(x) = f(y) \Rightarrow x = y \right) \\ (\mathsf{surj}) &\equiv \forall\, x \,\exists\, y \left( f(y) = x \right) \end{aligned}$$

Then for all $n$

$$\begin{aligned} (\mathsf{inj}), \mathsf{Cnt}_{=n} &\models (\mathsf{surj}) \\ (\mathsf{surj}), \mathsf{Cnt}_{=n} &\models (\mathsf{inj}) \\ (\mathsf{inj}), \neg(\mathsf{surj}) &\models \neg\mathsf{Cnt}_{=n} \end{aligned}$$

So proofs are relatively simple syntactic objects. Semantic entailment, on the other hand, inevitably involves set theory and seems very complicated: we have to deal with all FO structures of some signature. How are we supposed to know what these look like?

**Theorem (Completeness Theorem, Gödel 1930)**

*First-order logic is sound and complete:*

$$\Gamma \vdash \varphi \iff \Gamma \models \varphi$$

This looked like a huge step forward in Hilbert's dream: all questions about structures can be solved by looking at (strictly finitary, syntactic) proofs.

But if we consider particular structures, such as $\mathfrak{N}$, then this connection collapses.

For example, Dedekind-Peano Arithmetic, the standard axiom system for elementary arithmetic, **cannot** prove all sentences $\varphi$ such that $\mathfrak{N} \models \varphi$.

Worse, Gödel showed that in his famous Incompleteness Theorem that Dedekind and Peano did not just make a mistake: any attempt at axiomatizing arithmetic similarly fails: there are always true statements of arithmetic that are not provable in a particular FOL axiom system.

Again: we can't do

$$\Gamma = \text{ all } \varphi \text{ true in } \mathfrak{N}$$

since this set of "axioms" is highly undecidable.

Gödel's completeness theorem has a slightly unnerving consequence:

**Theorem (Compactness Theorem, Gödel 1930)**

*$\Gamma$ has a model if, and only, if every finite $\Gamma_0 \subseteq \Gamma$ has a model.*

This is positively wild, intuitively one would expect to be able to exploit infinitely many axioms (say, with infinitely many constants) to construct weird conditions, when every finite subset is perfectly harmless.

Alas, it is only the finite subsets that matter as far as the existence of a model is concerned.

**Lemma**

*The class of all finite structures (of some signature) is not axiomatizable.*

For assume that $\Gamma$ is some axiom system that characterizes finite structures. Hence $\Gamma$ has arbitrarily large finite models. But then $\Gamma$ already must have an infinite model.

To see this, add new constants $c_i$, $i \in \mathbb{N}$, to the language and add new axioms

$$c_i \neq c_j \qquad \text{for } i < j$$

to $\Gamma$ to obtain an extension $\Gamma'$.

Every finite subset of $\Gamma'$ has a model; by compactness $\Gamma'$ has a model which must be infinite by choice of the additional axioms.

**Lemma**

*The class of all infinite structures (of some signature) is not finitely axiomatizable.*

For otherwise the class of finite structures would also be axiomatizable: a finite set of axioms is like a single formula: just take the conjunction of all the axioms. That single formula can be negated.

But note that the class of infinite structures is FO definable: we can use infinitely many sentences of the form "there are at least $n$ elements".

$$\mathsf{Cnt}_{\geq n} \equiv \exists\, x_1, \ldots, x_n \, (x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \ldots \wedge x_{n-1} \neq x_n)$$

# Aside: Nonstandard Models

Gödel's completeness theorem is perfectly correct, so why does Dedekind-Peano arithmetic fail? It seems to describe the properties of $\mathfrak{N}$ very nicely.

But: we can use the construction from slide 96 to show that there are strange models of the Dedekind-Peano axioms that look very different from $\mathfrak{N}$: they contain "infinitely large natural numbers." These are called non-standard models and give rise to non-standard arithmetic.

The weird-but-true assertions that Peano arithmetic cannot prove fail in these weird, unintended models. In a sense, first-order logic simply fails at the task of pinning down arithmetic.

FOL is much better at describing whole classes of structures. For example, the class of all groups is no problem.

$$\mathbb{N} \underbrace{\ldots\mathbb{Z}\ldots\mathbb{Z}\ldots\mathbb{Z}\ldots}_{\mathbb{Q}}$$

With a little effort one can show that a nonstandard model starts with a copy of the ordinary naturals, followed by $\mathbb{Q}$-many copies of the ordinary integers.

You're welcome.

One can also produce non-standard models of real arithmetic, often written $\mathbb{R}^\star$.

Now suppose $N \in \mathbb{R}^\star$ is a "natural number" that has the property that

$$\underbrace{1+1+\ldots+1+1}_{n} < N$$

for all actual natural numbers $n$. The compactness argument used to construct $\mathbb{R}^\star$ produces these things automatically.

Then $\delta = 1/N$ is a perfectly good infinitesimal, realizing Leibniz's dream—just 300 years too late.

**Exercise**

*Try to axiomatize fields, finite fields, fields of characteristic $p$ or $0$, ordered fields, algebraically closed fields. Try the reals.*

**Exercise**

*Use similar arguments as on slide 96 to show that there is an infinite field of characteristic 2. By contrast, give an algebraic construction of such a field.*

**Exercise**

*Fill in the details in the proof on slide 97. Why is it critical that the axiom system is finite?*