Jeffrey Shallit  
School of Computer Science  
University of Waterloo  
shallit@uwaterloo.ca  
https://cs.uwaterloo.ca/~shallit
This lecture is about an amazing theorem first proved by Presburger in 1929.

Mojżesz Presburger (1904–1943)
(died in the Holocaust)

It concerns the decidability of a certain logical theory.
Logical theories and formulas

But first, let’s recall some things you learned in CS 245.

First, what a first-order *logical structure* is. We will be discussing logical structures where the objects are the natural numbers \( \mathbb{N} = \{0, 1, 2, \ldots\} \).

For our purposes it is a language of formulas like

1. \( \forall x \ \exists y \ y < x \)
2. \( \forall x \ \exists y \ x < y \)
3. \( \exists x \ \forall y \ y < x \)

Since we are talking about the specific model \( \mathbb{N} \), each such well-formed formula, with all variables quantified, (like the three above) has a corresponding truth value: either true or false.
First-order logical formulas

1. $\forall x \ \exists y \ y < x$

2. $\forall x \ \exists y \ x < y$

3. $\exists x \ \forall y \ y < x$

Formula #1 claims that there is always a natural number strictly less than a given natural number. This is false, because if $x = 0$ there is no such number.

Formula #2 claims that there is always a natural number strictly greater than a given natural number. This is a true formula.

Formula #3 says there is some fixed natural number strictly greater than all natural numbers. This is false.

All three formulas are in the first-order structure of the natural numbers with the operation “less than”.
The most basic operations we always allow in the first-order theory of $\mathbb{N}$ are $=$ (equality) and the logical operations $\land$ (and), $\lor$ (or), $\implies$ (logical implication), $\iff$ (iff), and $\neg$ (not). We also allow the universal quantifier ($\forall$) and the existential quantifier ($\exists$).

If we allow other kinds of operations, we can write down a wide variety of formulas.

What does the following formula assert?

$$\forall q \, \exists p > q \, \forall x \geq 2 \, \forall y \geq 2 \, p \neq x \cdot y.$$  

Exercise: try to say it concisely in English.
On the previous slide you were asked to interpret the formula

$$\forall q \exists p > q \forall x \geq 2 \forall y \geq 2 \ p \neq x \cdot y.$$ 

This is a formula in the structure of natural numbers with multiplication and $\geq$. It asserts “there exist infinitely many primes”.

Notice that here a formula like $\forall x \geq 2 \ P(x)$ is shorthand for $\forall x \ (x \geq 2) \implies P(x)$. 


Here’s another formula. What does it assert?

\[ \forall n \geq 3 \ \forall a \geq 1 \ \forall b \geq 1 \ \forall c \geq 1 \ a^n + b^n \neq c^n. \]

That is the claim of Fermat’s last theorem.
Here is one more:

$$\forall q \exists p > q \ \forall x \geq 2 \ \forall y \geq 2 \ (p \neq x \cdot y \ \land \ p + 2 \neq x \cdot y).$$

This one says that there are infinitely many “twin primes”, and nobody knows currently whether this is true or false.
David Hilbert (1862–1943) was a very influential German mathematician.

His dream was to find an algorithm that, given a formula from a certain family of number theory formulas, would decide if it is true or false.

Today this is known as Hilbert’s tenth problem; it is known there is no such algorithm.
Kurt Gödel (1906–1978) was a logician from Brno in (what is now) the Czech Republic.

One of his accomplishments was to show that even in relatively weak logical theories (like \( \text{Th}(\mathbb{N}, +, \times) \)) there are true statements with no proof in the theory.

Such a theory is called “incomplete”.
Alan Turing (1912–1954) was an English mathematician and computer scientist.

As a consequence of his results, there exists no algorithm that, given a formula in a sufficiently powerful logical structure, will decide if it is provable or unprovable.
Presburger arithmetic

Presburger arithmetic is $\text{Th}(\mathbb{N}, +)$, the first-order theory of the natural numbers with addition.

This is a relatively weak theory, so the results of Gödel and Turing don’t apply to it.

Notice that in such a structure we get inequalities like $<$ and $>$ “for free”.

For example, $x < y$ can be replaced with the assertion

$$(x \neq y) \land \exists z \ y = x + z.$$
As an example, in Presburger arithmetic we can state the “Chicken McNuggets theorem”.

This theorem answers the question, “At McDonald’s where you can only buy McNuggets in packages of 6, 9, or 20, what is the largest number \( N \) of McNuggets you cannot buy?”

The theorem says “\( N = 43 \)”. 
Here is how to state the Chicken McNuggets theorem:

\[(\forall t > 43 \exists x, y, z \ 6x + 9y + 20z = t) \land 
\neg(\exists x, y, z \ 6x + 9y + 20z = 43)\].

The first part of the formula says that every number of Chicken McNuggets greater than 43 can be purchased, and the second part says that there is no way to purchase exactly 43.

Note: although 6x, 9y, 20z, etc. are not literally expressible in Th(\(\mathbb{N}, +\)), these are actually just shorthand for x + x + x + x + x + x, y + y + y + y + y + y + y + y + y, etc.
Presburger’s theorem

Now here’s Presburger’s theorem: $\text{Th}(\mathbb{N}, +)$, the first-order theory of the natural numbers with addition, is consistent, complete, and \textit{decidable}.

Consistent means you can’t deduce both $\varphi$ and $\neg \varphi$.

Complete means that for each well-formed formula $\varphi$, either $\varphi$ or $\neg \varphi$ is deducible from the axioms.

Decidable means there is an algorithm to decide, given a well-formed formula $\varphi$ in the theory, whether $\varphi$ is a theorem or not.
Presburger’s theorem

Presburger’s original proof used quantifier elimination and is complicated.

But Büchi found a much simpler proof based on finite automata!

This is the proof of Presburger’s theorem I will present in this lecture.

Furthermore, the algorithm has been implemented in a system called Walnut and is available for you to play with.

The algorithm gives even more, as we will see.
Presburger’s theorem: the proof

To understand the proof, recall the notion of bound and unbound (aka “free”) variables in a formula.

A variable is *bound* if there is a quantifier associated with it.

A variable is *unbound* or *free* otherwise.

For example, in the formula $\exists y \; y < x$, the variable $y$ is bound and $x$ is free.
Idea of the proof

Here’s the key proof idea: for each formula $\varphi$, we *construct a DFA that recognizes the language $L(\varphi)$ of base-2 representations of numbers corresponding to the values of the free variables that make $\varphi$ true*. In other words, the language $L(\varphi)$ is (constructively) regular.

We do this by parsing the logical formula and creating algorithms to combine parts of a logical formula:

- $\land$ (and) corresponds to intersection of languages
- $\lor$ (or) corresponds to union of languages.
- $\neg$ (not) corresponds to the complement of a language.
- $A \implies B$ corresponds to $(\neg A) \lor B$, so we already have that.
- $A \iff B$ corresponds to $(A \implies B) \land (B \implies A)$, so we already have that.
So the only thing left is to handle addition, equality, and the two quantifiers.

If we are creating a DFA to recognize those values of the free variables that make $\varphi$ true, then we need to deal with automata that take tuples of natural numbers as input.

How do we represent such tuples?

The idea is that we can represent pairs like $(43, 26)$ as strings of pairs of symbols. When you read the first bit of each pair, you get 43 in base 2. When you read the second bit of each pair, you get 26 in base 2.

This means that if one number has a shorter base-2 representation than the other, it needs to be “padded” on the left with leading zeros.
Representing tuples of natural numbers

For example, one representation of \((43, 26)\) is

\[
[1, 0][0, 1][1, 1][0, 0][1, 1][1, 0].
\]

If you read the first bits, you get 101011, or 43 in base 2. If you read the second bits, you get 011010, or 26 in base 2.

Our automata will always accept all representations of a given integer, even those with leading zeros.
Let's start with addition: we need a DFA that can recognize those triples $x, y, z$ such that $z = x + y$.

The basic idea is simple: maintain a state for equality, and another state for “expecting to see a carry”.

This gives us the following DFA:

All transitions not shown go to a nonaccepting “dead state”.

```
no
carry

looking
for
carry
```
Implementing equality

Equality is very easy: with two states we can accept if pairs of inputs agree everywhere, and reject otherwise:
Next, let’s handle the existential quantifier.

The basic idea is that we can handle a formula like $\exists x \ P(x, y, z)$ by taking the DFA for $P$ and changing all transitions on the triple $[x_i, y_i, z_i]$ representing some bit of $x, y$ and $z$ to a transition on just $[y_i, z_i]$.

This is called “projection”.

Do you see why this works?

One problem: projection might create an NFA.
Implementing $\exists$ and $\forall$

Solution: use the subset construction to convert NFA to DFA.

It does raise one very minor point: if automata take tuples as input, corresponding to free variables, what if the formula has no free variables?

In this case there is just one possible input, the “empty” tuple, and the automaton has just one state. Either it is accepting for all inputs, or rejecting for all inputs. This means the final result is either “true” or “false”.

Finally, to handle the universal quantifier, we just use the observation that $\forall x \; P(x)$ is just the same as $\neg \exists x \; \neg P(x)$. So we can always convert universal quantifiers to existential.
The algorithm summarized

So this gives our algorithm: take the formula $\varphi$, parse it, convert each piece to the corresponding DFA and combine them using the algorithms discussed above.

The “leaves” of the parse tree of the formula correspond to formulas like $x = 43$, which we can easily construct a DFA for.

If there are no free variables in $\varphi$, the resulting automaton has one state and either accepts everything (so $\varphi$ is true) or rejects everything (so $\varphi$ is false).

If there are free variables in $\varphi$, then the resulting automaton accepts all the base-2 representations of values of the free variables making $\varphi$ true.

The correctness of our algorithm now proves Presburger’s theorem.
The Chicken McNuggets theorem revisited

As an example, let’s rewrite our Chicken McNuggets formula as follows:

$$(\forall s > t \exists x, y, z \ 6x + 9y + 20z = s) \land \neg(\exists x, y, z \ 6x + 9y + 20z = t).$$

In Walnut, we can do this as follows:

eval chicken "(As (s>t) => Ex,y,z 6*x+9*y+20*z=s) & ~((Ex,y,z 6*x+9*y+20*z=t))":

which produces the following automaton:

![Automaton Diagram]

which accepts 0*101011, which evaluates to 43 in base 2.
The good news is that Presburger arithmetic is decidable.

The bad news is that the worst-case of the running time to decide it can be very bad.

The algorithm we presented is not optimal, and has a running time like

\[ 2 \cdot 2^{cN} \]

where \( N \) is the size of the formula. This comes from the fact that each alternating quantifier might cause determinization of an NFA to create a DFA.

Nevertheless the algorithm “often” runs quickly.
You can play with an implementation of the algorithm for Presburger arithmetic here:

https://cs.uwaterloo.ca/~shallit/walnut.html

It actually does much more than Presburger arithmetic, but you can use it just for that.