Expressing Hard Math With Quantifiers

Expressing Theorems: Four-Square Theorem

Four-Square Theorem Every natural number is the sum of ≤ 4 squares.

Expressing Theorems: Four-Square Theorem

Four-Square Theorem Every natural number is the sum of ≤ 4 squares.

Four-Square Theorem Every natural number is the sum of 4 squares. We allow 0.

Expressing Theorems: Four-Square Theorem

Four-Square Theorem Every natural number is the sum of ≤ 4 squares.

Four-Square Theorem Every natural number is the sum of 4 squares. We allow 0.

$$(\forall x)(\exists x_1, x_2, x_3, x_4)[x = x_1^2 + x_2^2 + x_3^2 + x_4^2]$$

Expressing Statements: Goldbach's Conjecture

Goldbach's Conjecture Every sufficiently large even number can be written as the sum of two primes.

Expressing Statements: Goldbach's Conjecture

Goldbach's Conjecture Every sufficiently large even number can be written as the sum of two primes.

$$(\exists x)(\forall y > x)$$

$$[\mathrm{EVEN}(y) \rightarrow (\exists y_1, y_2)[\mathrm{PRIME}(y_1) \wedge \mathrm{PRIME}(y_2) \wedge (y = y_1 + y_2)]]$$

Vinogradav's Theorem

Vinogradov's Theorem Every sufficiently large odd number can be written as the sum of three primes.

Vinogradav's Theorem

Vinogradov's Theorem Every sufficiently large odd number can be written as the sum of three primes.

$$(\exists x)(\forall y > x)$$

$$[ODD(y) \rightarrow$$

$$(\exists y_1, y_2, y_3)[\mathrm{PRIME}(y_1) \wedge \mathrm{PRIME}(y_2) \wedge \mathrm{PRIME}(y_3) \wedge (y = y_1 + y_2 + y_3)]]]$$

Square root of 2

Square root of 2

Thm $\sqrt{2} \notin \mathbb{Q}$. (We will prove this later in the course.)

Square root of 2

Thm $\sqrt{2} \notin \mathbb{Q}$. (We will prove this later in the course.)

We want to express this with quantifiers over $\mathbb{Z}$.
Note that if $2 = \frac{x^2}{y^2}$ then $2y^2 = x^2$.

Square root of 2

Thm $\sqrt{2} \notin \mathbb{Q}$. (We will prove this later in the course.)
We want to express this with quantifiers over $\mathbb{Z}$.
Note that if $2 = \frac{x^2}{y^2}$ then $2y^2 = x^2$.

$$\neg(\exists x, y)[2y^2 = x^2]$$

Square root of 2

Thm $\sqrt{2} \notin \mathbb{Q}$. (We will prove this later in the course.)
We want to express this with quantifiers over $\mathbb{Z}$.
Note that if $2 = \frac{x^2}{y^2}$ then $2y^2 = x^2$.

$$\neg(\exists x, y)[2y^2 = x^2]$$

$$(\forall x, y)[2y^2 \neq x^2]$$

Square root of 2

Thm $\sqrt{2} \notin \mathbb{Q}$. (We will prove this later in the course.)
We want to express this with quantifiers over $\mathbb{Z}$.
Note that if $2 = \frac{x^2}{y^2}$ then $2y^2 = x^2$.

$$\neg(\exists x, y)[2y^2 = x^2]$$

$$(\forall x, y)[2y^2 \neq x^2]$$

Note that using $\neg(\exists x, y) \equiv (\forall x, y)\neg$ ended up not having a $\neg$ in the final expression.

Order Notation

Sometimes We Don't Care About Constants

The following conversation would never happen.

Sometimes We Don't Care About Constants

The following conversation would never happen.
LEO: Bill, I have **an algorithm** that solves SAT in roughly n^2 time!

Sometimes We Don't Care About Constants

The following conversation would never happen.
LEO: Bill, I have **an algorithm** that solves SAT in roughly n^2 time!
BILL: Roughly? What do you mean?

Sometimes We Don't Care About Constants

The following conversation would never happen.
LEO: Bill, I have **an algorithm** that solves SAT in roughly n^2 time!
BILL: Roughly? What do you mean?
LEO: There are constants c, d, e such that my algorithm works in time $\leq cn^2 + dn + e$. OH, the algorithm only has this runtime when the number of variables is ≥ 100.

Sometimes We Don't Care About Constants

The following conversation would never happen.
LEO: Bill, I have **an algorithm** that solves SAT in roughly n^2 time!
BILL: Roughly? What do you mean?
LEO: There are constants c, d, e such that my algorithm works in time $\le cn^2 + dn + e$. OH, the algorithm only has this runtime when the number of variables is ≥ 100.
BILL: What are c, d, e?

Sometimes We Don't Care About Constants

The following conversation would never happen.
LEO: Bill, I have **an algorithm** that solves SAT in roughly n^2 time!
BILL: Roughly? What do you mean?
LEO: There are constants c, d, e such that my algorithm works in time $\le cn^2 + dn + e$. OH, the algorithm only has this runtime when the number of variables is ≥ 100.
BILL: What are c, d, e?
LEO: Who freakin cares! I solved SAT without using brute force and you are concerned with the constants!

When Do/Don't We Care About Constants?

1) When we first look at a problem we want to just get a sense of how hard it is:

When Do/Don't We Care About Constants?

1) When we first look at a problem we want to just get a sense of how hard it is:
Exp vs Poly time?

When Do/Don't We Care About Constants?

1) When we first look at a problem we want to just get a sense of how hard it is:
Exp vs Poly time?
If poly then what degree?

When Do/Don't We Care About Constants?

1) When we first look at a problem we want to just get a sense of how hard it is:
Exp vs Poly time?
If poly then what degree?
If roughly n^2 then can we get it to roughly $n \log n$ or n?

When Do/Don't We Care About Constants?

1) When we first look at a problem we want to just get a sense of how hard it is:
Exp vs Poly time?
If poly then what degree?
If roughly n^2 then can we get it to roughly $n \log n$ or n?
Once we have exhausted all of our tricks to get it into (say) roughly n^2 time we THEN would do things to get the constant down, perhaps non-rigorous things.

We Want to Make "Roughly" Rigorous

We want to say that we don't care about constants.

We Want to Make "Roughly" Rigorous

We want to say that we don't care about constants.
We want to say that $18n^3 + 8n^2 + 12n + 1000$ is roughly n^3.

We Want to Make "Roughly" Rigorous

We want to say that we don't care about constants.
We want to say that $18n^3 + 8n^2 + 12n + 1000$ is roughly n^3.

$\boldsymbol{f \le O(n^3)}$ First attempt:

$$(\exists c)[f(n) \le cn^3].$$

We Want to Make "Roughly" Rigorous

We want to say that we don't care about constants.
We want to say that $18n^3 + 8n^2 + 12n + 1000$ is roughly n^3.

$\boldsymbol{f \le O(n^3)}$ First attempt:

$$(\exists c)[f(n) \le cn^3].$$

We do not really care what happens for small values of n. The following definition captures this:

$\boldsymbol{f \le O(n^3)}$ Second and final attempt:

We Want to Make "Roughly" Rigorous

We want to say that we don't care about constants.
We want to say that $18n^3 + 8n^2 + 12n + 1000$ is roughly n^3.

$\boldsymbol{f \le O(n^3)}$ First attempt:

$$(\exists c)[f(n) \le cn^3].$$

We do not really care what happens for small values of n. The following definition captures this:

$\boldsymbol{f \le O(n^3)}$ Second and final attempt:

$$(\exists n_0)(\exists c)(\forall n \ge n_0)[f(n) \le cn^3].$$

We Want to Make "Roughly" Rigorous

We want to say that we don't care about constants.
We want to say that $18n^3 + 8n^2 + 12n + 1000$ is roughly n^3.
$\boldsymbol{f \le O(n^3)}$ First attempt:

$$(\exists c)[f(n) \le cn^3].$$

We do not really care what happens for small values of n. The following definition captures this:
$\boldsymbol{f \le O(n^3)}$ Second and final attempt:

$$(\exists n_0)(\exists c)(\forall n \ge n_0)[f(n) \le cn^3].$$

We leave it to the reader to prove that

$$18n^3 + 8n^2 + 12n + 1000 = O(n^3)$$

by finding the values of n_0, c, d.

$f = O(g)$

$f \leq O(g)$ means

$$(\exists n_0)(\exists c)(\forall n \geq n_0)[f(n) \leq cg(n)].$$

$f = O(g)$

$f \le O(g)$ means

$$(\exists n_0)(\exists c)(\forall n \ge n_0)[f(n) \le cg(n)].$$

You will see $O()$ a lot in CMSC 351 and 451 when you deal with algorithms and want to bound the run time, roughly.

Other Ways to Use $O()$

$f \in n^{O(1)}$ means poly time.

Other Ways to Use $O()$

$f \in n^{O(1)}$ means poly time.

$f \in 2^{O(n)}$ means 2^{cn} for some c, and after some n_0.

Another Conversations

The following conversation would never happen.

Another Conversations

The following conversation would never happen.
BILL: Leo, I have shown that SAT **requires** roughly 2^n time!

Another Conversations

The following conversation would never happen.
BILL: Leo, I have shown that SAT **requires** roughly 2^n time!
LEO: Roughly? What do you mean?

Another Conversations

The following conversation would never happen.

BILL: Leo, I have shown that SAT **requires** roughly 2^n time!

LEO: Roughly? What do you mean?

BILL: There are constants c, d, e such that ANY algorithm for SAT takes time $\geq 2^{cn} - dn^2 - e$. OH, the algorithm only has this runtime when the number of variables is ≥ 100.

Another Conversations

The following conversation would never happen.

BILL: Leo, I have shown that SAT **requires** roughly 2^n time!

LEO: Roughly? What do you mean?

BILL: There are constants c, d, e such that ANY algorithm for SAT takes time $\geq 2^{cn} - dn^2 - e$. OH, the algorithm only has this runtime when the number of variables is ≥ 100.

LEO: What are c, d, e?

Another Conversations

The following conversation would never happen.

BILL: Leo, I have shown that SAT **requires** roughly 2^n time!

LEO: Roughly? What do you mean?

BILL: There are constants c, d, e such that ANY algorithm for SAT takes time $\geq 2^{cn} - dn^2 - e$. OH, the algorithm only has this runtime when the number of variables is ≥ 100.

LEO: What are c, d, e?

BILL: Who freakin cares! I showed SAT is not in poly time you are concerned with the constants!

$f = \Omega(g)$

$f \geq \Omega(g)$ means

$$(\exists n_0)(\exists c)(\forall n \geq n_0)[f(n) \geq cg(n)].$$

$f = \Omega(g)$

$f \geq \Omega(g)$ means

$$(\exists n_0)(\exists c)(\forall n \geq n_0)[f(n) \geq cg(n)].$$

This notation is used to express that an algorithm **requires** some amount of time.

If I proved . . .

If I proved that SAT requires $\Omega(n^3)$ time would I have solved P vs NP?

If I proved . . .

If I proved that SAT requires $\Omega(n^3)$ time would I have solved P vs NP?

No. SAT could still be in time n^4.

If I proved . . .

If I proved that SAT requires $\Omega(n^3)$ time would I have solved P vs NP?

No. SAT could still be in time n^4.

If I proved that SAT requires $n^{\Omega(\log \log \log n)}$ time would I have solved P vs NP?

If I proved ...

If I proved that SAT requires $\Omega(n^3)$ time would I have solved P vs NP?

No. SAT could still be in time n^4.

If I proved that SAT requires $n^{\Omega(\log\log\log n)}$ time would I have solved P vs NP?

Yes. That function is bigger than any poly. But result would be odd since people **really** think SAT requires $2^{\Omega(n)}$.

If I proved . . .

If I proved that SAT requires $\Omega(n^3)$ time would I have solved P vs NP?

No. SAT could still be in time n^4.

If I proved that SAT requires $n^{\Omega(\log\log\log n)}$ time would I have solved P vs NP?

Yes. That function is bigger than any poly. But result would be odd since people **really** think SAT requires $2^{\Omega(n)}$.

You would still get the \$1,000,000.