Quantified Inference Proofs

CSE 311 Fall 25
Lecture 9

Inference Proofs Overview

A new way of thinking of proofs:

Here's one way to get an iron-clad guarantee:

1. Write down all the facts we know.
2. Combine the things we know to derive new facts.
3. Continue until what we want to show is a fact.

Inference Rules

Eliminate ∧

$$\frac{A \wedge B}{\therefore \quad A, B}$$

Eliminate ∨

$$\frac{A \vee B, \neg A}{\therefore \quad B}$$

Intro ∧

$$\frac{A, B}{\therefore \quad A \wedge B}$$

Intro ∨

$$\frac{A}{\therefore A \vee B, B \vee A}$$

Direct Proof rule

$$\frac{A \Rightarrow B}{A \rightarrow B}$$

Modus Ponens

$$\frac{P \rightarrow Q, P}{\therefore \quad Q}$$

You can still use all the propositional logic equivalences too!

Given: $((p \to q) \wedge (q \to r))$
Show: $(p \to r)$

Here's a corrected version of the proof.

1. $(p \to q) \wedge (q \to r)$	Given
2. $p \to q$	Eliminate ∧ 1
3. $q \to r$	Eliminate ∧ 1
4.1 p	Assumption
4.2 q	Modus Ponens 4.1,2
4.3 r	Modus Ponens 4.2,3
5. $p \to r$	Direct Proof Rule

When introducing an assumption to prove an implication: Indent, and change numbering.

When reached your conclusion, use the Direct Proof Rule to observe the implication is a fact.

The conclusion is an unconditional fact (doesn't depend on p) so it goes back up a level

Try it! (setup)

Given: $p \vee q$, $(r \wedge s) \rightarrow \neg q$, r.
Show: $s \rightarrow p$

Eliminate ∧

$$\frac{A \wedge B}{\therefore\ A, B}$$

Eliminate ∨

$$\frac{A \vee B, \neg A}{\therefore\ B}$$

Intro ∧

$$\frac{A; B}{\therefore\ A \wedge B}$$

Intro ∨

$$\frac{A}{\therefore A \vee B, B \vee A}$$

Direct Proof rule

$$\frac{A \Rightarrow B}{A \rightarrow B}$$

Modus Ponens

$$\frac{P \rightarrow Q; P}{\therefore\ Q}$$

You can still use all the propositional logic equivalences too!

Try it! (complete)

Given: $p \vee q$, $(r \wedge s) \rightarrow \neg q$, r.
Show: $s \rightarrow p$

1.	$p \vee q$	Given
2.	$(r \wedge s) \rightarrow \neg q$	Given
3.	r	Given
4.1	s	Assumption
4.2	$r \wedge s$	Intro ∧ (3,4.1)
4.3	$\neg q$	Modus Ponens (2, 4.2)
4.4	$q \vee p$	Commutativity (1)
4.5	p	Eliminate ∨ (4.4, 4.3)
5.	$s \rightarrow p$	Direct Proof Rule

Caution ⚠

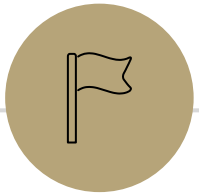
Be careful! Logical inference rules can only be applied to **entire** facts. They cannot be applied to portions of a statement (our propositional equivalences could apply to subexpressions). Why not for inference rules?

Suppose we know $p \rightarrow q$, r. Can we conclude q?

1.	$p \rightarrow q$	Given
2.	r	Given
3.	$(p \vee r) \rightarrow q$	Introduce ∨ (1)
4.	$p \vee r$	Introduce ∨ (2)
5.	q	Modus Ponens 3,4.

Intro ∨

$$\frac{A}{\therefore A \vee B, B \vee A}$$



Inference Proofs in Predicate Logic

Inference Rules for Proofs with Quantifiers

We've done symbolic proofs with propositional logic.

To include predicate logic, we'll need some rules about how to use quantifiers.

Eliminate ∀

$$\frac{\forall x\, P(x)}{\therefore\ P(a) \text{ for any } a}$$

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore\ \ \exists x\, P(x)}$$

Intro ∀

$$\frac{P(a); a \text{ is arbitrary}}{\therefore\ \ \forall x\, P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore\ \ P(c) \text{ for a fresh } c}$$

Let's see a good example, then come back to those "arbitrary" and "fresh" conditions.

Proof Using Quantifiers

Suppose we know $\exists x P(x)$ and $\forall y[\, P(y) \rightarrow Q(y)]$. Conclude $\exists x Q(x)$.

Eliminate ∀

$$\frac{\forall x\ P(x)}{\therefore\ P(a) \text{ for any } a}$$

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore\quad \exists x\ P(x)}$$

Intro ∀

$$\frac{P(a); a \text{ is arbitrary}}{\therefore\qquad \forall x\ P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore\ \ P(c) \text{ for a fresh } c}$$

Example Proof Using Quantifiers (setup)

Suppose we know $\exists x P(x)$ and $\forall y[\, P(y) \rightarrow Q(y)]$. Conclude $\exists x Q(x)$.

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore \quad \exists x\, P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore \quad P(c) \text{ for a fresh } c}$$

Eliminate ∀

$$\frac{\forall x\, P(x)}{\therefore P(a) \text{ for any } a}$$

Intro ∀

$$\frac{P(a); a \text{ is arbitrary}}{\therefore \qquad \forall x\, P(x)}$$

Example Proof Using Quantifiers (Complete)

Suppose we know $\exists x P(x)$ and $\forall y[\, P(y) \rightarrow Q(y)]$. Conclude $\exists x Q(x)$.

1.	$\exists x P(x)$	Given
2.	$P(a)$	Eliminate ∃ 1
3.	$\forall y[P(y) \rightarrow Q(y)]$	Given
4.	$P(a) \rightarrow Q(a)$	Eliminate ∀ 3
5.	$Q(a)$	Modus Ponens 2,4
6.	$\exists x Q(x)$	Intro ∃ 5

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore \quad \exists x\, P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore \quad P(c) \text{ for a fresh } c}$$

Eliminate ∀

$$\frac{\forall x\, P(x)}{\therefore P(a) \text{ for any } a}$$

Intro ∀

$$\frac{P(a); a \text{ is arbitrary}}{\therefore \qquad \forall x\, P(x)}$$

Proofs with Quantifiers ("arbitrary")

We've done symbolic proofs with propositional logic.

To include predicate logic, we'll need some rules about how to use quantifiers.

Eliminate ∀

$$\frac{\forall x\, P(x)}{\therefore\ P(a) \text{ for any } a}$$

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore\ \ \exists x\, P(x)}$$

Intro ∀

$$\frac{P(a); a \text{ is arbitrary}}{\therefore\ \ \forall x\, P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore\ \ P(c) \text{ for a fresh } c}$$

"arbitrary" means a is "just" a variable in our domain.
It doesn't depend on any other variables and wasn't introduced with other information.

Proofs with Quantifiers ("fresh")

We've done symbolic proofs with propositional logic.

To include predicate logic, we'll need some rules about how to use quantifiers.

Eliminate ∀

$$\frac{\forall x\, P(x)}{\therefore\ P(a) \text{ for any } a}$$

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore\ \ \exists x\, P(x)}$$

Intro ∀

$$\frac{P(a);\ a \text{ is arbitrary}}{\therefore\ \ \forall x\, P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore\ \ P(c) \text{ for a fresh } c}$$

"fresh" means c is a new symbol (there isn't another c somewhere else in our proof).

Fresh and Arbitrary (example)

Suppose we know $\exists x P(x)$. Can we conclude $\forall x P(x)$?

1. $\exists x\, P(x)$ Given
2. $P(a)$ Eliminate ∃ (1)
3. $\forall x\, P(x)$ Intro ∀ (2)

This proof is **definitely** wrong.
(take $P(x)$ to be "is a prime number")

a wasn't **arbitrary.** We knew something about it – it's the x that exists to make $P(x)$ true.

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore\ \exists x\, P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore\ P(c) \text{ for a fresh } c}$$

Eliminate ∀

$$\frac{\forall x\, P(x)}{\therefore P(a) \text{ for any } a}$$

Intro ∀

$$\frac{P(a);\ a \text{ is arbitrary}}{\therefore\ \forall x\, P(x)}$$

Fresh and Arbitrary (notes 1)

Intro ∀

$$\frac{P(a);\ a \text{ is arbitrary}}{\therefore \quad \forall x\, P(x)}$$

Eliminate ∃

$$\frac{\exists x P(x)}{\therefore \quad P(c) \text{ for a fresh } c}$$

You can trust a variable to be **arbitrary** if you introduce it as such.

If you eliminated a ∀ to create a variable, that variable is arbitrary. Otherwise it's not arbitrary – it depends on something.

You can trust a variable to be **fresh** if the variable doesn't appear anywhere else (i.e. just use a new letter)

Fresh and Arbitrary (notes 2)

Eliminate ∀

$$\frac{\forall x\, P(x)}{\therefore P(a) \text{ for any } a}$$

Intro ∃

$$\frac{P(c) \text{ for some } c}{\therefore \quad \exists x\, P(x)}$$

There are no similar concerns with these two rules.

Want to reuse a variable when you eliminate ∀? Go ahead.

Have a c that depends on many other variables, and want to intro ∃?

Also not a problem.

Arbitrary Practice (1)

In section, you said: $[\exists y \forall x\ P(x, y)] \rightarrow [\forall x \exists y\ P(x, y)]$. Let's prove it!!

Arbitrary Practice (2)

In section, you said: $[\exists y \forall x\ P(x,y)] \rightarrow [\forall x \exists y\ P(x,y)]$. Let's prove it!!

1.1 $\exists y \forall x\ P(x,y)$	Assumption
1.2 $\forall x\ P(x,c)$	Elim ∃ (1.1)
1.3 Let a be arbitrary.	--
1.4 $P(a,c)$	Elim ∀ (1.2)
1.5 $\exists y\ P(a,y)$	Intro ∃ (1.4)
1.6 $\forall x \exists y\ P(x,y)$	Intro ∀ (1.5)

2. $[\exists y \forall x\ P(x,y)] \rightarrow [\forall x \exists y\ P(x,y)]$ Direct Proof Rule

Arbitrary Practice (3)

In section, you said: $[\exists y \forall x\ P(x,y)] \rightarrow [\forall x \exists y\ P(x,y)]$. Let's prove it!!

1.1 $\exists y \forall x\ P(x,y)$	Assumption
1.2 $\forall x\ P(x,c)$	Elim ∃ (1.1)
1.4 $P(a,c)$	Elim ∀ (1.2)
1.5 $\exists y\ P(a,y)$	Intro ∃ (1.4)
1.6 $\forall x \exists y\ P(x,y)$	Intro ∀ (1.5)

2. $[\exists y \forall x\ P(x,y)] \rightarrow [\forall x \exists y\ P(x,y)]$ Direct Proof Rule

It is not required to have "variable is arbitrary" as a step before using it. But many people (including Robbie) find it helpful.

Find The Bug

Let your domain of discourse be integers.
We claim that given $\forall x \exists y$ `Greater`(y, x), we can conclude $\exists y \forall x$ `Greater`(y, x)
Where `Greater`(y, x) means $y > x$

1.	$\forall x \exists y$ `Greater`(y, x)	Given
2.	Let a be an arbitrary integer	--
3.	$\exists y$ `Greater`(y, a)	Elim ∀ (1)
4.	`Greater`(b, a)	Elim ∃ (2)
5.	$\forall x$ `Greater`(b, x)	Intro ∀ (4)
6.	$\exists y \forall x$ `Greater`(y, x)	Intro ∃ (5)

Find The Bug (with notes)

1.	$\forall x \exists y\ \texttt{Greater}(y, x)$	Given
2.	Let a be an arbitrary integer	--
3.	$\exists y\ \texttt{Greater}(y, a)$	Elim ∀ (1)
4.	$\texttt{Greater}(b, a)$	Elim ∃ (2)
5.	$\forall x\ \texttt{Greater}(b, x)$	Intro ∀ (4)
6.	$\exists y \forall x\ \texttt{Greater}(y, x)$	Intro ∃ (5)

b is not a single number! The variable b depends on a. You can't get rid of a while b is still around.
What is b? It's probably something like $a + 1$.

Bug Found

There's one other "hidden" requirement to introduce ∀.

"No other variable in the statement can depend on the variable to be generalized"

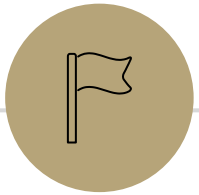
Think of it like this -- b was probably $a + 1$ in that example.

You wouldn't have generalized from $\texttt{Greater}(a + 1, a)$

To $\forall x$ $\texttt{Greater}(a + 1, x)$. There's still an a, you'd have replaced all the a's.

x depends on y if y is in a statement when x is introduced.

This issue is much clearer in English proofs, which we'll start next time.



Number Theory

Why Number Theory?

Applicable in Computer Science

“hash functions” (you’ll see them in 332) commonly use modular arithmetic
Much of classical cryptography is based on prime numbers.

More importantly, a great playground for writing English proofs.

Framing Device (keygen 1)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Key generation [edit]

The keys for the RSA algorithm are generated in the following way:

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random and should be similar in magnitude but differ in length by a few digits to make factoring harder.[2] Prime integers can be efficiently found using a primality test.
   - p and q are kept secret.
2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
   - n is released as part of the public key.
3. Compute $\lambda(n)$, where λ is Carmichael's totient function. Since $n = pq$, $\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$, and since p and q are prime, $\lambda(p) = \varphi(p) = p - 1$, and likewise $\lambda(q) = q - 1$. Hence $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$.
   - $\lambda(n)$ is kept secret.
   - The lcm may be calculated through the Euclidean algorithm, since $\operatorname{lcm}(a, b) = |ab|/\gcd(a, b)$.
4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are coprime.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – the most commonly chosen value for e is $2^{16} + 1 = 65\,537$. The smallest (and fastest) possible value for e is 3, but such a small value for e has been shown to be less secure in some settings.[15]
   - e is released as part of the public key.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; that is, d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $d \cdot e \equiv 1 \pmod{\lambda(n)}$; d can be computed efficiently by using the extended Euclidean algorithm, since, thanks to e and $\lambda(n)$ being coprime, said equation is a form of Bézout's identity, where d is one of the coefficients.
   - d is kept secret as the *private key exponent*.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d. In fact, they can all be discarded after d has been computed.[16]

Framing Device (keygen 2)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Key generation [edit]

Prime Numbers

The keys for the RSA algorithm are genera

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random and should be similar in magnitude but differ in length by a few digits to make factoring harder.[2] Prime integers can be efficiently found using a primality test.
   - p and q are kept secret.

Modular Arithmetic

2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
   - n is released as part of the public key.
3. Compute $\lambda(n)$, where λ is Carmichael's totient function. Since $n = pq$, $\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$, and since p and q are prime, $\lambda(p) = \varphi(p) = p - 1$, and likewise $\lambda(q) = q - 1$. Hence $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$.
   - $\lambda(n)$ is kept secret.
   - The lcm may be calculated through the Euclidean algorithm, since $\operatorname{lcm}(a, b)$

Modular Multiplicative Inverse

4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are cop
   - e having a short bit-length and small Hamming weight results in more efficient enc … the most commonly chosen value for e is $2^{16} + 1 = 65\,537$. The smallest (and fastest) possible value for e is 3, but such a small value for e has been shown to be less secure in some settings.[15]
   - e is released as part of the public key.

Bezout's Theorem

5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; that is, d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $d \cdot e \equiv 1 \pmod{\lambda(n)}$; d can be computed efficiently by using the extended Euclidean algorithm, since, thanks to e and $\lambda(n)$ being coprime, said equation is a form of Bézout's identity, where d is one of the coefficients.
   - d is kept secret as the *private key exponent*.

Extended Euclidian Algorithm

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the private (or decryp … t also be kept secret because they can be used to calculate d. In fact, they can all be discarded after d has been computed.[16]

Framing Device (en/decryption 1)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Encryption [edit]

After Bob obtains Alice's public key, he can send a message M to Alice.

To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}.$$

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits c to Alice. Note that at least nine values of m will yield a ciphertext c equal to m,[22] but this is very unlikely to occur in practice.

Decryption [edit]

Alice can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Given m, she can recover the original message M by reversing the padding scheme.

Framing Device (en/decryption 2)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Encryption [edit]

After Bob obtains Alice's public key, he can send a message M to Alice.

To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}.$$

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits c to Alice. Note that at least nine values of m will yield a ciphertext c equal to m,[22] but this is very unlikely to occur in practice.

Modular Exponentiation

Decryption [edit]

Alice can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Given m, she can recover the original message M by reversing the padding scheme.

Divides Definition

Divides

For integers x, y we say $x|y$ ("x divides y") iff there is an integer z such that $xz = y$.

"x is a divisor of y" or "x is a factor of y" means (essentially) the same thing as x divides y.

("essentially" because of edge cases like when a number is negative or $y = 0$)

"The small number goes first*" *when both are positive integers

Divides Examples

Divides

For integers x, y we say $x|y$ ("x divides y") iff there is an integer z such that $xz = y$.

Which of these are true?

$2|4$ $\quad$ $4|2$ $\quad$ $2|-2$

$5|0$ $\quad$ $0|5$ $\quad$ $1|5$

Divides Examples (with answers)

Divides

For integers x, y we say $x|y$ ("x divides y") iff there is an integer z such that $xz = y$.

Which of these are true?

$2|4$ True

$4|2$ False

$2|-2$ True

$5|0$ True

$0|5$ False

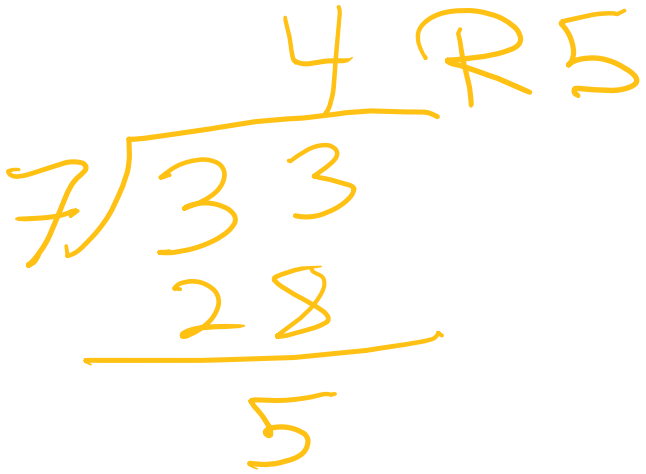
$1|5$ True

A useful theorem (intuition)

The Division Theorem

For every $a \in \mathbb{Z}$, $\boldsymbol{d} \in \mathbb{Z}$ with $\boldsymbol{d > 0}$
There exist *unique* integers q, r with $0 \leq r < d$
Such that $a = dq + r$

Remember when non integers were still secret, you did division like this?



q is the "quotient"
r is the "remainder"

Unique

The Division Theorem

For every $a \in \mathbb{Z}$, $\boldsymbol{d} \in \mathbb{Z}$ with $\boldsymbol{d > 0}$
There exist *unique* integers q, r with $0 \leq r < d$
Such that $a = dq + r$

"unique" means "only one"….but be careful with how this word is used.

r is unique, **given** a, d. – it still depends on a, d but once you've chosen a and d

"unique" is not saying $\exists r \forall a, d \quad P(a, d, r)$
It's saying $\forall a, d \exists r [P(a, d, r) \wedge [P(a, d, x) \rightarrow x = r]]$

A useful theorem (compared to Java)

The Division Theorem

For every $a \in \mathbb{Z}$, $\boldsymbol{d} \in \mathbb{Z}$ with $\boldsymbol{d > 0}$
There exist *unique* integers q, r with $0 \leq r < d$
Such that $a = dq + r$

The q is the result of `a/d` (integer division) in Java

The r is the result of `a%d` in Java

That's slightly a lie, r is always non-negative, Java's % operator sometimes gives a negative number.

Terminology

You might have called the % operator in Java “mod”

We’re going to use the word “mod” to mean a closely related, but different thing.

Java’s % is an operator (like + or ·) you give it two numbers, it produces a number.

The word “mod” in this class, refers to a set of rules

Modular Arithmetic Intuition

"arithmetic mod 12" is familiar to you. You do it with clocks.

What's 3 hours after 10 o'clock?

1 o'clock. You hit 12 and then "wrapped around"

"13 and 1 are the same, mod 12" "-11 and 1 are the same, mod 12"

We don't just want to do math for clocks – what about if we need to talk about parity (even vs. odd) or ignore higher-order-bits (mod by 16, for example)

Modular Arithmetic Notation

To say "the same" we don't want to use $=$ … that means the normal $=$

We'll write $13 \equiv 1 (\mathbf{mod}\ 12)$

$\equiv$ because "equivalent" is "like equal," and the "modulus" we're using in parentheses at the end so we don't forget it.

(we'll also say "congruent mod 12")

The notation here is bad. We all agree it's bad. Most people still use it.

$13 \equiv_{12} 1$ would have been better. "mod 12" is giving you information about the $\equiv$ symbol, it's not operating on 1.

Modular Arithmetic (need a definition)

We need a definition! We can't just say "it's like a clock"

Pause what do you expect the definition to be?

Is it related to % ?

Modular Arithmetic Equivalence Definition

We need a definition! We can't just say "it's like a clock"

Pause what do you expect the definition to be?

Equivalence in modular arithmetic

Let $a \in \mathbb{Z}, b \in \mathbb{Z}, n \in \mathbb{Z}$ and $n > 0$.
We say $a \equiv b \ (mod\ n)$ if and only if $n|(b - a)$

Huh?

Long Pause

It's easy to read something with a bunch of symbols and say "yep, those are symbols." and keep going

STOP Go Back.

You have to *fight* the symbols they're probably trying to pull a fast one on you.

Same goes for when I'm presenting a proof – you shouldn't just believe me – I'm wrong all the time!

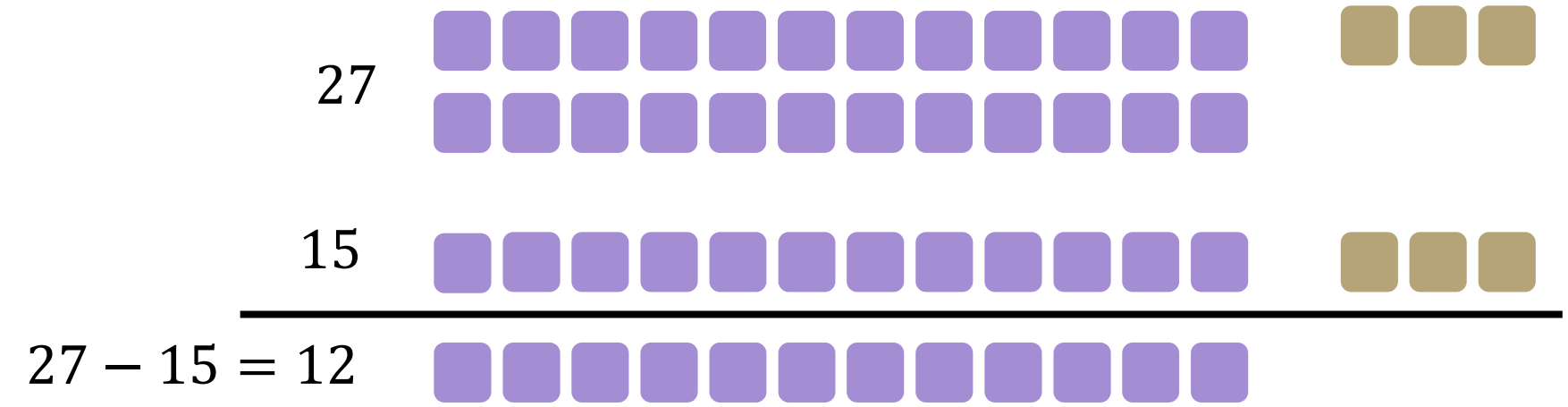
You should be *trying* to do the proof with me. Where do you think we're going next?

Why?

Your Tas will take a bit of time in section on this.

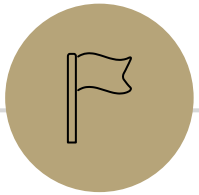
Here's the short version:

It really is equivalent to "what we expected"
`a%n=b%n` if and only if $n|(b-a)$



When you subtract, the remainders cancel. What you're left with is a multiple of 12.

The divides version is much easier to use in proofs…



Extra Practice

One more Proof (setup)

Show if we know: $p, q, [(p \wedge q) \rightarrow (r \wedge s)], r \rightarrow t$ we can conclude t.

One more Proof (complete)

Show if we know: $p, q, [(p \wedge q) \rightarrow (r \wedge s)], r \rightarrow t$ we can conclude t.

1. p	Given
2. q	Given
3. $[(p \wedge q) \rightarrow (r \wedge s)]$	Given
4. $r \rightarrow t$	Given
5. $p \wedge q$	Intro ∧ (1,2)
6. $r \wedge s$	Modus Ponens (3,5)
7. r	Eliminate ∧ (6)
8. t	Modus Ponens (4,7)