ECS 20 – Fall 2021 – Phillip Rogaway Logic I

Today: Logic
Propositional logic: Boolean values, digital circuits, logic-design problems, satisfiable and tautological formulas, provability

Wikipedia page on
Propositional calculus: https://en.wikipedia.org/wiki/Propositional_calculus
and on Boolean algebra https://en.wikipedia.org/wiki/Boolean_algebra

Propositional Logic = Propositional Calculus
= Sentential Logic = Sentential Calculus ≈ Boolean algebra

Universe of two points $\mathbb{B} = \{0,1\}$ or, alternatively,
$\mathbb{B} = \{F, T\}$ or $\mathbb{B} = \{\text{False}, \text{True}\}$

That is, I will interchangeably use 0, F, False and 1, T, True. I have even seen examples where the Boolean values are regarded as {-1, 1}.
(The font for $\mathbb{B}$ is blackboard bold: \mathbb{…} in LaTeX)

The $\mathbb{B}$ stands for **Boolean**, named after George Boole, a 19th century mathematician.

It is wonderful thing having a universe of only two points—a way simpler universe than the natural numbers, the integers, or the reals, where most of you have been told to live for all of your math classes. In fact, I can't for the life of me understand why school children start their studies with the positive integers, $\mathbb{Z}^+$, which is a terribly complicated thing compared to $\mathbb{B}$. Maybe you can go back and teach all the young children in your life propositional logic, and let them not worry about those nasty integers until they are older.

Here are some sets you should know:

$\mathbb{B} = \{0,1\}$ // Booleans. Symbol is uncommon. Also written $\mathbb{Z}_2$
$\mathbb{N} = \{0,1,2,3,...\}$ // The natural numbers. I always include 0
$\mathbb{Z} = \{...,-2,-1,0,1,2,...\}$ // The integers
$\mathbb{Q}$ = the rational numbers = $\{a/b\colon a, b \in \mathbb{Z},\ b \neq 0\}$
$\mathbb{R}$ = the real numbers

Just like integers and reals have operations defined on them, like addition (+) and multiplication (·), and so do Boolean values.

If P and Q are variables represent Boolean values, then we define a sort of **multiplication** on them as

$P \cdot Q$ -- lots of alternative notations: $P \wedge Q$ P and Q P AND Q PQ
P && Q

What does it mean??

P	Q	$P \wedge Q$
F	F	F
F	T	F
T	F	F
T	T	T

A **truth table**

I’m using **infix notation**, but don’t let that confuse you as to the nature of the thing: a function from AND: $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$. We could just as well have used the more familiar prefix notation for functions, like AND(P,Q)

To make a truth table you really need to know how to count in binary. Do you all know? 0, 1, 10, 11, 100, 101, …
Or, with leading zero, then, for example, 000, 001, 010, 011, 100, 101, 110, 111. Everybody follows this convention; please don’t write a truth table any other way.

Similarly, for OR, with it’s various notations

$P \vee Q$ P or Q P OR Q P || Q

P	Q	$P \vee Q$
F	F	F
F	T	T
T	F	T
T	T	T

(As an aside, why do our truth table rows in this order? Practice **counting in binary**. 0, 1, 10, 11, 100, 101, 110, 111, 1000, …)

We need at least one more operator, a unary operator NOT

¬P not P NOT P !P $\overline{P}$

P	¬ P
F	T
T	F

We can put them together and make longer formulas
P and not(Q) or not(P) and Q

That’s actually an interesting function, we call it XOR:
P xor Q P ⊕ Q P + Q

P	Q	P ⊕ Q
F	F	F
F	T	T
T	F	T
T	T	F

Here’s another functionality that’s very useful: implies. P IMPLIES Q, or

P	Q	P → Q
F	F	T
F	T	T
T	F	F
T	T	T

And yet another. Biconditional: P ↔ Q P IFF Q P iff Q P ≡ Q

P	Q	$P \leftrightarrow Q$
F	F	T
F	T	F
T	F	F
T	T	T

You can either think of operators like $\oplus$, $\rightarrow$, $\leftrightarrow$ as "basic" operators of sentential logic or, alternatively, you can think of them as "syntactic sugar" – convenient short hand for formulas that are "actually" made of ∧, ∨, ¬ . (But, for that matter, did we really need all of ∧, ∨, ¬ ? Exercise: write OR using only AND and NOT; write AND using only OR and NOT. In fact, this is more than an exercise: it is a way to introduce **DeMorgan's Laws.)**

Do a truth table involving more than two variables, like

if s then x else y

Warm about the use of the word **formal** having two different meanings: rigorous and symbolic. Now we are going to treat logic formally in the *second* sense:

Definition: A **well-formed formula** (WFF) (of propositional logic) over a nonempty set of **variables** $\mathcal{P}$ (finite or countably infinite) is:
(the variables may not contain any of the symbols: ¬ ∨ ∧ ( ) F T)

- F and T are WFF
- If $X \in \mathcal{P}$ then X is a WFF
- If α and β are WFFs then so are: $(\neg\alpha)$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$,
  // stop here: let's treat the other binary operators as "syntactic sugar"

(Nothing else is a WFF.)

This is an example of a **recursive definition**.
Formulas are just **strings**: sequences of symbols from an alphabet.

There are alternative ways to give a recursive definition like the one we just gave for a WFF. Here is how you would write it using a **context-free grammar**:

$W \rightarrow \mathcal{P} \mid T \mid F \mid (W \vee W) \mid (W \wedge W) \mid (\neg W)$

$\mathcal{P} \rightarrow L \mid L_N$

$L \rightarrow P \mid Q \mid R$

$N \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid NN$

The T, F, ∨, ∧ , ¬, P, Q, R, 0, 1, …, 0 are called **terminals** while the W, S, N, $\mathcal{P}$ are called **nonterminals** or **variables.**

A different notation that captures the same thing is called **Backus-Naur Form** (more frequently called **Backus Normal Form**, or **BNF**). It’s nice for more explicitly distinguishing terminals and nonterminals.

```
<wff> ::=  <prop-symbol> | “T” | “F” |  “(“  <wff> “ ∨”  <wff>) |
                “(“ <wff>  “∧”  wff “)”    |  “(  ¬ “  <wff>  “)”
<prop-symbol> ::-  <letter>  | <letter> <number>
<letter>  ::=  “P” | “Q” | “R”
<number> → “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9” |
                   <number> <number>
```

Def: A **truth assignment** *t* over a set $\mathcal{P}$ is a map *t*: $\mathcal{P} \rightarrow \mathbb{B}$.

A t.a. is also called a **model**.

A t.a. gives a formula ϕ whose variables are in $\mathcal{P}$ a truth value in the natural way; formally, we extend *t* to a t.a. on WFFs by asserting that

$t(\mathrm{T})=\mathrm{T}$

$t(\mathrm{F})=\mathrm{F}$

$t((\alpha \vee \beta)) = t(\alpha) \vee t(\beta)$

$\mathrm{t}((\alpha \wedge \beta)) = t(\alpha) \wedge t(\beta)$

$\mathrm{t}((\neg\alpha)) = \neg\, t(\alpha)$

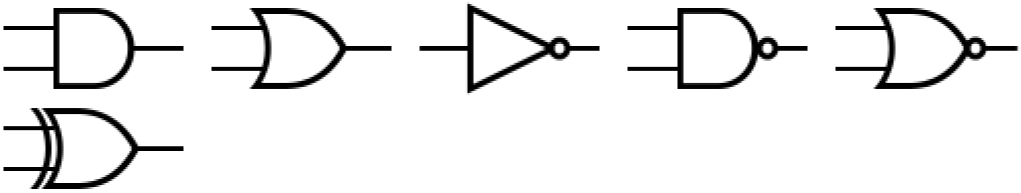
Another **recursive definition**. In this way we have **extended** the definition of t from o this larger domain of WFFs.

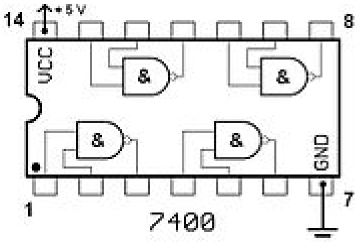
Don’t confuse strings of **symbols** with their semantics (e.g., the wedge on the LHS of the third formula has a very different meaning on the RHS; there is a big difference between a **formal symbol** and a **logical operation.**

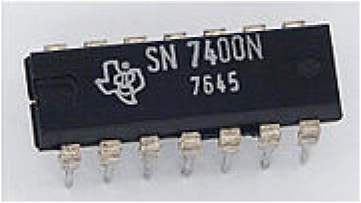
In common usage we use a **precedence order** allows us to omit many parenthesis, adopting a convention:

$\neg$
$\wedge$
$\vee$
$\rightarrow$
$\leftrightarrow$ (or some would put at same level as $\rightarrow$)

and right-to-left within a level (or some would say left-to-right; or some would say ill-defined).

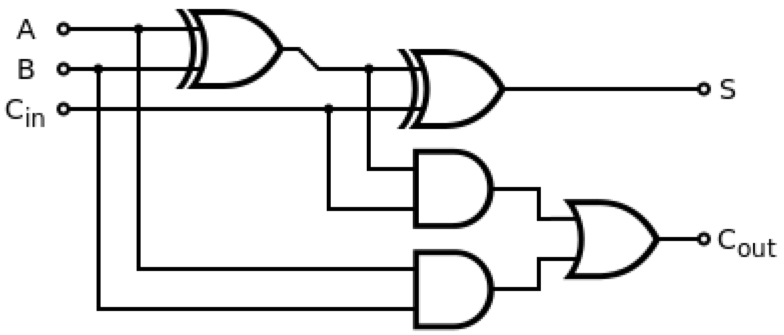


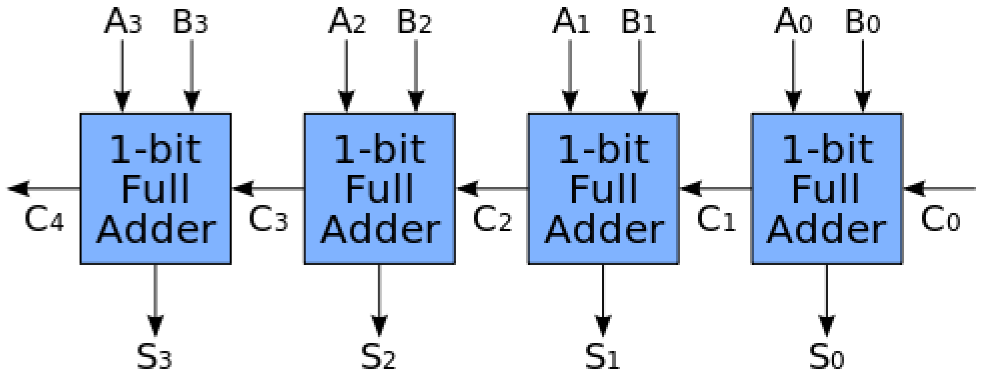




Design something, say a **majority circuit** – returns 1 if a majority of its inputs are on. Do for 3 wires.

How would you do it for 100 gates? –or an **addition circuit** using a **full adder**





Describe **Disjunctive Normal Form** (DNF).
Proposition: Each WFF is equivalent to one in DNF.
Give a proof, and give an upper bound on the number of two-input gates to realize any n-input functionality.

Define: a set of operators being **logically complete**.
Show that the following sets of operators are logically complete:
$\{\wedge, \vee, \neg\}$ $\{\wedge, \neg\}$

⊼ (write NAND as a wedge with a bar over it).
⊽ (similarly for NOR)

Definitions:

- A formula ϕ is **satisfiable** if *some* t.a. makes it true: there is a t.a. t such that $t(\phi)$=T
- A set of formula Γ is **satisfiable** if *some* t.a. makes **all** of them true.
- A formula ϕ is **tautological** (or **valid**) if it is true for **every** t.a.
- $\models \phi$ means ϕ is a tautology
- ϕ and ψ are (logically) **equivalent** , $\phi \equiv \psi$, if $\phi \leftrightarrow \psi$ is a **tautology**
- $\Gamma \models \phi$ Every t.a. that makes all formula in Γ true makes ϕ true.

Proposition: There is an algorithm (=a precisely-describable procedure, mechanism, recipe)

that, given a WFF of sentential logic, decides
- if it is a tautology.
- if it is a satisfiable.
- if it equivalent to some given, second formula.

Proof: "Truth-table algorithm"

Example:
Contrapositive: $\vDash (P \to Q) \leftrightarrow (\neg Q \to \neg P)$ ← prove

Discuss the **inefficiency** of the truth-table algorithm.

Remarkable claim: no efficient means are known for any of these problems.

Discuss the difference in meaning between:

$(P \to Q) \leftrightarrow (\neg Q \to \neg P)$ // this is a statement, might be true or false
$\vDash (P \to Q) \leftrightarrow (\neg Q \to \neg P)$ // an assertion that it's always true

Some simple tautologies Velleman, *How to Prove It*, p. 21, 23, 47, 49 .
You can check any of these with a truth table.

Associative: $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$ //Mention the similarities to arithmetic
$P \vee (Q \vee R) \equiv (P \vee Q) \vee R$ //laws with $\vee$ corresponding to **addition**
//and $\wedge$ corresponding to **multiplication**

De Morgan's: $\neg (P \wedge Q) \equiv \neg P \vee \neg Q$
$\neg (P \vee Q) \equiv \neg P \wedge \neg Q$

Idempotent: $P \wedge P \equiv P$
$P \vee P \equiv P$

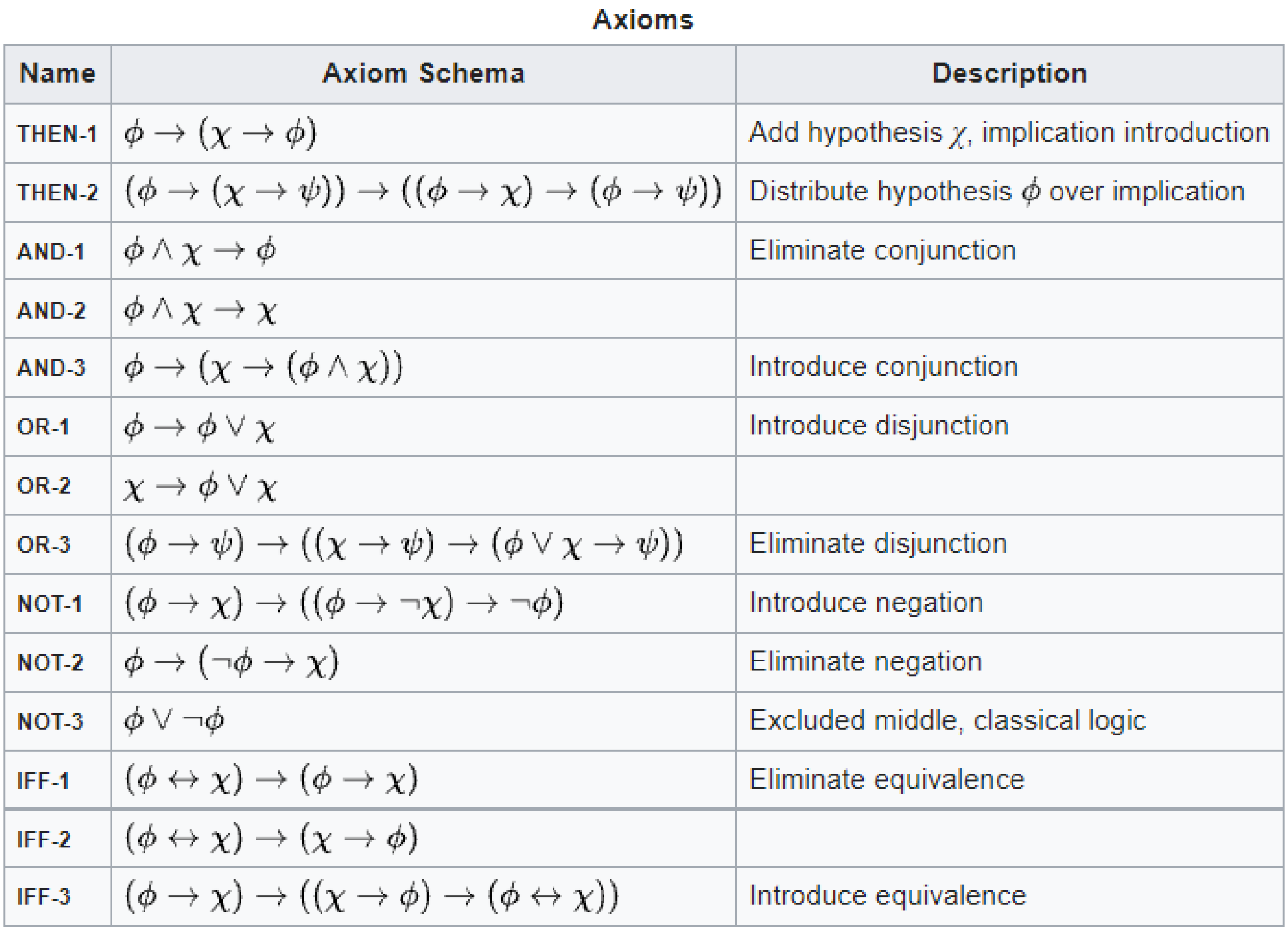
Contradiction $P \to Q \equiv \neg P \vee Q$
$P \to Q \equiv \neg (P \wedge \neg Q)$

Formal Proofs

Discuss conventional proofs vs. formal proofs.

I now discuss formal proofs, although what mathematicians – and you – will mostly be producing conventional (informal) ones.

Following from Wikipedia, *Propositional Calculus*. Following 14 rules



Axioms

Name	Axiom Schema	Description
THEN-1	$\phi \to (\chi \to \phi)$	Add hypothesis χ, implication introduction
THEN-2	$(\phi \to (\chi \to \psi)) \to ((\phi \to \chi) \to (\phi \to \psi))$	Distribute hypothesis ϕ over implication
AND-1	$\phi \land \chi \to \phi$	Eliminate conjunction
AND-2	$\phi \land \chi \to \chi$	
AND-3	$\phi \to (\chi \to (\phi \land \chi))$	Introduce conjunction
OR-1	$\phi \to \phi \lor \chi$	Introduce disjunction
OR-2	$\chi \to \phi \lor \chi$	
OR-3	$(\phi \to \psi) \to ((\chi \to \psi) \to (\phi \lor \chi \to \psi))$	Eliminate disjunction
NOT-1	$(\phi \to \chi) \to ((\phi \to \neg\chi) \to \neg\phi)$	Introduce negation
NOT-2	$\phi \to (\neg\phi \to \chi)$	Eliminate negation
NOT-3	$\phi \lor \neg\phi$	Excluded middle, classical logic
IFF-1	$(\phi \leftrightarrow \chi) \to (\phi \to \chi)$	Eliminate equivalence
IFF-2	$(\phi \leftrightarrow \chi) \to (\chi \to \phi)$	
IFF-3	$(\phi \to \chi) \to ((\chi \to \phi) \to (\phi \leftrightarrow \chi))$	Introduce equivalence

One of the reasons to have axioms like the list just given is to develop a notion of “what is provable"” We will write $\Gamma \vdash \phi$ if statement ϕ follows from Γ. Read: **ϕ is provable from** Γ. **Turnstyle** is the name of the symbol.
List of logical symbols: https://en.wikipedia.org/wiki/List_of_logic_symbols

Formal proofs are quite different from **conventional proofs**, but a **thesis** in mathematics is that conventional proofs can be recast as formal ones. What are formal proof? They are syntactic objects in some formalized system. There are many choices one has in how to do the formulation, but here is what we would typically have: that a formal proof is a sequence of formula: $\phi_1, \ldots, \phi_n$ where each ϕ_i is either

1. an **assumption** or
2. an **axiom** (it appear on a list like Axiom List W) or
3. it follows from a previous set of lines in the proof by one of a number of enumerated rules – indeed we can make do with **one** rule, *modes ponens*,

i) $(A \rightarrow B)$

...

j) A

...

k) B *modes ponens*

Example:

$\vdash (PQ)(P \vee R \rightarrow S)(SQ \rightarrow U) \rightarrow U$

1. PQ	assumption
2. $P \vee R \rightarrow S$	assumption
3. $SQ \rightarrow U$	assumption
4. $PQ \rightarrow P$	AND-1 (eliminate conjunction)
5. P	*modus ponens* on (1), (4)
6. $PQ \rightarrow Q$	AND-2 (eliminate conjunction)
7. Q	*modus ponens* on (1), (6)
8. $P \rightarrow P \vee R$	OR-1 (introduce disjunction)
9. $P \vee R$	*modus ponens* on (5), (8)
10. S	*modus ponens* on (2) and (9)
11. $S \rightarrow (Q \rightarrow SQ)$	AND-3 (introduce conjunction)
12. $Q \rightarrow SQ$	*modus ponens* on (10), (11)
13. SQ	*modus ponens* on (7), (12)
14. U	*modus ponens* on (3) and (13)

Therefore

$\{PQ,\ P \vee R \rightarrow S,\ SQ \rightarrow U\} \vdash U$ or

$\vdash (PQ)(P \vee R \rightarrow S)(SQ \rightarrow U) \rightarrow U$ //The given statement is provable

$\vdash \phi$ can derive (prove) ϕ (from the $\varnothing$ — no assumptions)
$\Gamma \vdash \phi$ can derive ϕ from Γ
Discuss proofs as (1) formal objects; (2) convincing arguments in a community. Mathematicians don't seem to appreciate the extent to which those standards are socially constructed. See: *Proofs and Refutations* by Imre Lakatos. (Cf: *Genesis and Development of a Scientific Fact*, by Ludwik Fleck, about Euler's theorem on convect polyhedral: v-e+f=2).

Theorems that we won't prove in this class.

Soundness: If $\vdash \phi$ then $\vDash \phi$
(More generally, $\Gamma \vdash \phi$ implies $\Gamma \vDash \phi$)

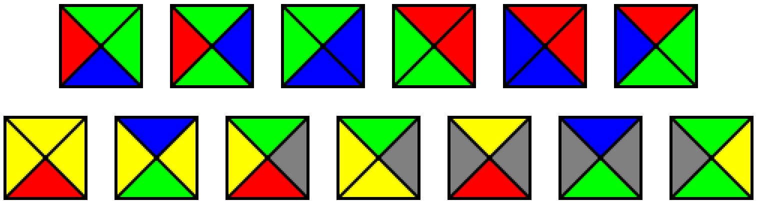
Completeness: If $\vDash \phi$ then $\vdash \phi$
(More generally, $\Gamma \vDash \phi$ implies $\Gamma \vdash \phi$)

Compactness: Let Γ be a set of WFFs.
Suppose that *every finite subset* of Γ is satisfiable.
Then Γ is satisfiable.

(Contrapositive:
Let Γ be a set of WFFs.
Suppose that Γ is **not** satisfiable.
Then *some finite subset* of Γ is already not satisfiable)

Let's use this in a fun example: **TILING** (Dominos)
Can you tile the with tiles of specified types
(adjacent edges of the same color)

Eg:



Make an example where the plane is and is not tileable. Indicate that, in ecs120, common to prove that the TILING decision question is

undecidable. But not our interest here. We are interested in whether the tileability of the plane for a given set of tile types is FALSIFIABLE -- is there a proof of untileability? There will be if the following is true:

If the plane is un-tileable for a given set of tiles, it is already un-tileable on some finite square of the plane.

Not an obvious claim -- a priori possible that plane is untileable even though every finite rectangle in it is tileable.

To prove from compactness:

Introduce a variable
P[i,j,k]: there is a tile of type *k* at position (*i, j*). Infinitely many vars.

Write a Boolean formula to capture

- At least one tile per square: for all integers i, j,

  $\bigvee_k$ P[i,j,k]

- At most one tile per square: for all i, j,
  P[i,j,k] → P[i,j,k'] for all k ≠ k'.

- Horizontal direction is good: for all i and j,

  P[i,j,k] → ($\bigvee_{k'}$ P[i+1,j,k'])

  if a tile of type k' may be put to the right a tile of type k

- Columns are good. Fall i and j, P[i,j,k] → ($\vee_{k'}$ P[i,j+1,k'])
  if a tile of type k' may be put above a tile of type k

Now: connect it to compactness theorem. The set Γ is all the formulas above. If it is unsatisfiable, then some finite subset of Γ_0 is unsatisfiable. Let *n* be the largest index used by a variable in Γ_0. Then the [-*n*..*n*] × [-*n*..*n*] subset of the plane is already untileable. That is, you can prove to someone that the plane is untileable.

Γ is satisfiable iff every the plane can be tiled with tiles
of the given types.

The plane can be tiled iff every $n \times n$ subset of it can be.

In the language you will learn in ecs120, the complement of tiling is recursively enumerable (r.e.)

A gap between Boolean expressions in modern programming languages and those in mathematics:

In most modern programming languages, like C, Java, and Python, short

```
'''
   Short-circuited evaluation in Python
'''

if (0<1 or 0/0==0): print("Hi")
if (0/0==0 or 0<1): print("there")
```

```
Hi
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    if (0/0==0 or 0<1): print("there")
ZeroDivisionError: division by zero

...Program finished with exit code 0
```

Hmm. Doesn't this mean that if we can't even reason about a program that `B1 or B2` is the same as `B2 or B1`, then it's hard to reason anything about real programs, right? Right! Reasoning about the behavior is hard and requires extreme care.