Math Logic, Week#III

Moon's Day

Reminder: What does $\Sigma \models \Phi$ mean — for Φ and Σ sets of formulas?

¿Any remaining questions on chapter I (except for homework)**?**

Recall that Friday I assigned more problems: §**1.7 ## 2, 3, 4, 8.**

For #8 your proof — like Enderton's discussion — may be a bit vague. And use your intuition about computability.

¿Any questions on chapter 2 before definability in a structure?

1. Another example of getting nonsense by mixing "*formal language*" — or "*object language*" — and "*meta-language*":

   The (ancient) liar's paradox: "*This sentence is false.*"

   In this regard: the symbols (i) "$\models$" and (ii) "$\models\!\!=\!\!|$" are part of the meta-language — just abbreviations for the English expressions

   (i) "*satisfies*" or "*logically implies*" (tell those 2 usages apart by context), and

   (ii) "*is logically equivalent to*."

   In particular, when we write in the formal language, we may not include those symbols.

2. Note that, with any formal logic, we're much more precise than people usually are in English:
   - In English you can point to an athlete or a politician and say "*ugh*" and people mostly understand.

     (But f I just said "*hate them all*," you'd wonder whether I meant all politicians or all chapters of the textbook.)
   - But the more abstract the topic, the more careful we have to be. Distinguish
     - *"Everybody is not happy,"* and
     - *"Not everybody is happy."*

     And:
     - "*I only ate the cake*," and
     - "*I ate only the cake.*"

3. Enderton uses some notation I find unnatural:
   (a) Instead of writing $f(x)$ for function f applied to x, he writes fx.
   I'll try to write $f(x)$ in my notes.
   But don't be surprised if I slip back and forth between the 2 notations.
   And you may use either one in your homework (although I'd prefer $f(x)$).

   (b) I'll also normally write $\mathfrak{A} \models \varphi[s]$ instead of $\models_{\mathfrak{A}} \varphi[s]$.

4. **¿Do you understand the comment about the "commutative diagram" on page 83?**

5. Variables are rather like pronouns. Out of context, when I say "$x \geq y$" or *it is greener than I am*", you don't know what "*it*" x and y are.
   - We need something else — an interpretation of the variable symbols — to make sense of what it is.
   - In a context we may fix the meaning — "*I saw something amazing in the grocery store: it is greener than I am,*" or $\exists x \forall y (x \geq y)$.
   - In "$x \geq y$" x is *free*. But in the context $\forall x \exists y (x \geq y)$ x is *bound*.
   - Contrary to some common usage, the *binding* of x and y covers as little as makes grammatical sense.
   - Analogue in programming: a global reference in a function is a reference to something whose meaning must be specified elsewhere in the program.
   - Whenever we change the context of something, we must make sure we don't change the meaning of these global references. We'll see a notion of "*substitutability*" — just and issue of not changing the context of any name.

Definability: In human language, we describe tend to create new words as needed (as well as for other reasons).

(But you might want to look up the Sapir-Wharf hypothesis in linguistics.)

- But in mathematics and computer science, we often investigate what we can do in a fixed language. (For example, in many programming languages, there is no way to discuss the address of a particular variable.)

  If we change the language, we change the topic.
- So we ask "*what can we say in this particular language.*"

Next assignment:

- Finish reading §2.2. (And we'll skip 2.3.)
- Exercises: §2.1 ## 1,5
  and §2.2## 1, 2, 5, 6, 9, 11, 12, 14, 15
  (Warning: a few of the last ones above are challenging!)

Go through homeworks:

- Finish §1.5 #2
- §1.5 #7
- §1.5 #9
- Exercise B
- Exercise C
- §1.7 #2
- §1.7 #1

Wodin's Day

¿Questions on material through §2.2?

If you haven't seen homomorphisms before, you probably need to stare at eamples for a while to see what they are.

- (The origin of the word "*homomorphism*" is from Greek: "*homos*" (same) and "*morphe*" (shape).)
- A class of examples: Think of a vector space (over the real numbers) as having infinitely many functions: $+$ and $r\cdot$ (scalar multiplication by r) for each real number r. Then a homomorphism is just a linear transformation.

  Examples on the plane $\mathbb{R} \times \mathbb{R}$: (i) project each point onto the x axis; (ii) map each point (x, y) to $(x/2, y/2)$; (iii) rotate the entire plane 45 degrees clockwise; (iv) map every poiint to $(0, 0)$.
- Enderton has one unusual usage: most people say an *isomorphism* is a homomorphism that is 1-1 and onto. Enderton requires only that the homomophism be 1-1. (But when he says to structures are isomorphic, he does require the function be both 1-1 and onto.)

Homeworks:
- Finish §1.5 **&** §1.5 #7
- Exercises B **&** C
- §1.7 #1 **&** #2 **&** #3 **&** #4 **&** #8
- §2.1 #1 **&** #5
- §2.2 #1 **&** #2

Aside: Intuition of first order logic

We use it for discussing objects and their properties:

1. We have a set of objects — a *universe*

   It might be that we don't *know* what the universe is,

   but we assume that there is nothing which is, say, part in and part out.

   The universe might be finite or infinite. But we do require the universe to be nonempty. (That's presumably a minor exclusion, and it makes later work easier.)

   Example in Databases and SQL: It's the set of all values (attribute values) which appear anywhere in the database at some particular time

   — possibly plus the set of all real numbers (to allow us, *e.g.*, to count things and add numbers)

   Common Mathematics Example: The universe is $\mathbb{N}$ (the set of natural numbers) or $\mathbb{R}$ (the set of real numbers).

   (Oversimplfied) **Program Verification:** The universe would include the set of all numerical variables in your program and the set of all numbers which might be values of those variables. (*Usually* we'd simplify that to $\mathbb{Z}$ (the set of all integers) or $\mathbb{R}$.

   We might also need to specify times — also as numbers (often $\mathbb{N}$).

2. Ways to identify properties of those objects and interrelationships among them.

   (named) **relations:** — named with "*relation symbols*"

   In a **relational database,** this is the basis of the representation:

student	**isTaking**
M1234	ENG-1001
M1234	PHYS-3003
M2345	PHYS-1798
M2345	PHIL-2222
⋮	⋮

   (but we'll often need to add in numbers too)

   Describing numbers, we'd have $<$.

   In **program verification** we might have a relation *hasValue* (identifier, value, time).

   (named) **elements:** — named with "*constant symbols*"

   In a *relational database* we assume that everything in the universe has a name.

   In mathematics over $\mathbb{R}$, we typically have a name for each rational number — *e.g.* $\frac{-12345}{293}$ — and also π and e.

   functions: – named with *function symbols*

   In mathematics, we might have functions $+$, $\cdot$, and exponentiaion.

   (There's a technical issue there: $0^{-1/2}$ is undefined; to fit the model, we'd have to give an arbitrary value — maybe 0 raised to any power is 0.)

   In **relational databases** function symbols aren't allowed (except on numbers).

Aside: **Examples of using first order logic:**

In mathematics: **Say a function is continuous:** (The notation comes out of the formalization of mathematics.)

1. How do you compute π^2? The decimal expansion $\pi = 3.1415\ldots$ is infinite — ¡and the way we learned to multiply starts at the right end!

   It turns out that, if we want to approximate π^2 to within 1/1000, all we need to do is to approximate π accurately enough and square that number. (That is to say, *sqr* is "continuous at" π.)

2. *(Ignore me for a moment while I do the arithmetic; you don't need to see that — and I'll tell you the number I produce is obvious.)*

   "It's obvious that" if x is within $1/10000$ of π, x^2 is within $1/1000$ of π^2 —

$$|\pi - x| < \frac{1}{10000} \quad \rightarrow \quad |\mathit{sqr}(\pi) - \mathit{sqr}(x)| < \frac{1}{1000}$$

   To be more explicit, we want to say that we mean *any* x that close to π:

$$\forall x(\ |\pi - x| < \frac{1}{10000} \quad \rightarrow \quad |\mathit{sqr}(\pi) - \mathit{sqr}(x)| < \frac{1}{1000} \ ).$$

   (And note that I put in parentheses to specify exactly what the $\forall x$ applies to.)

3. But we want to be able to approximate π^2 to within *any* restricted amount of error. Traditionally, people use the variable ε for this maximum allowable error. That hidden arithemetic showed me that

$$\forall\varepsilon\forall x(\ \ |\pi - x| < \min(\frac{\varepsilon}{10}, 1)\ \ \rightarrow\ \ |\mathit{sqr}(\pi) - \mathit{sqr}(x)| < \varepsilon\ \ ).$$

4. Now if we want to say *sqr* is continuous at an unknown value z, we can't say that approximating z within $\min(\frac{\varepsilon}{10}, 1)$ is good enough — but we can say there is some limit on $|z - x|$ that's good enough; that limit is traditionally called δ.

$$\begin{array}{l}\forall z\ \forall\varepsilon(\ \ \varepsilon > 0 \rightarrow (\ \ \exists\delta(\ \ \delta > 0\ \ \wedge \\ \qquad \forall x(\ \ |z - x| < \delta\ \ \rightarrow\ \ |\mathit{sqr}(z) - \mathit{sqr}(x)| < \varepsilon\ \ ) \\ \ \ )\ \ )\end{array}$$

(Here, since I didn't put in a "(" after the $\forall z$, our grammar says that the $\forall z$ applies to the minimum sequence of symbols that make grammatical sense.)

In relational databases: (T. Codd based them upon first order logic)

With relations **student**(stdnt#, name1, name2, name3) and **took**(stdnt#, class, semester, year)

ask for the names of all students with first name "Adam" who *ever* took "Biol-1001":

$$\exists\, \text{st\#}\ \big(\ \mathbf{student}(\text{st\#}, \text{n1}, \text{n2}, \text{n3}) \ \wedge\ (\text{n1} = \text{Adam}) \ \wedge \exists\, \text{smstr}\ \exists\, \text{yr}(\mathbf{took}(\text{st\#}, \text{Biol1001}, \text{smstr}, \text{yr}))\ \big).$$

(Relational database syntax uses *projections* instead of $\exists$s.)

Since I haven't quantified n1, n2, or n3, all those are *free* — they're uninterpreted here. *In this context* we asking for all triples (n1, n2, n3) that satisfy the formula.

Illustrating an important point: Suppose I asked, not about a student with first name Adam, but with first name Adolph-Idi-Lizzy. Probably there has never been such a student at UC — and maybe not such a person at all. But you wouldn't want to translate the question so that it always answers something like "nobody" — it's always possible 10 such peroples will arrive in the future, and we want the translation to find them all.

This illustrates the fact that we want to saythings correctly for all contexts, not for just one particular one. More on this later.

A technicality to remember: Database langauges allow us to add new names for objects — such as Adolph-Idi-Lizzy above — at any time. Our first order languages don't.

Common Approach in Mathematics: Write down a series of *axioms* — often in first order logic — and consider anything those axioms describe.

This approach appears in CS too: For example: Suppose we have a combinatorial circuit that isn't working. A common errror is a "stuck at 0" error: one gate always outputs a 0. We

- describe the circuit in logic, allowing for the possibiility that some gate might be stuck at 0,
- describe the inputs and outputs to the circuit from a failed test, and
- and let a computer program generate all ways stuck at 0 faults could explain the failed test results.

The symbols of 1st order logic: (pages 69-70)

"Logical Symbols:"

- parentheses: "(" and ")."
- sentential connective symbols: $\rightarrow$ and $\neg$

  (Informally we shall continue to use $\wedge$, $\vee$, $\leftrightarrow$, $\top$, and $\bot$, but when we required we can formally translate them all in terms of just $\rightarrow$ and $\neg$.)
- variable symbols: "v_1," "v_2," "v_3," ...

  (Informally we shall use almost anything else that's obvious and doesn't create any ambiguity).)
- (**optional)** equality symbol "="

"Parameters:" (this is Ederton's usage)

- Quantifier symbol "$\forall$"

  (Informally we also use $\exists$ — but *formally* we define that by deMorgan's law: $\forall x \varphi$ abbreviates $\not\exists x \not\varphi$.
  That's the opposite of what happens in relational databases, where we use projection to express $\exists$ and often need to use deMorgan's law to express $\forall$ — $\forall x \varphi \models\!\!\mid\!\!= \neg\exists x \not\varphi$.)
- predicate symbols — *e.g.*, $<$ or took above
- constant symbols — *e.g.* Adam or π above.
- function symbols — *e.g.*, $-$, *sqr*, and $|\ldots|$ (absoluet value) above.

Notes: (i) Each predicate symbol and function symbol must have a specified number of argument places (parameters in CS language) — often called the "arity" of the symbol. (ii) The set of parameters is commonly called the *language* — but most authors would not include $\forall$ as part of the language. (They'd get a slightly different explanation but the same syntax and semantics.)

When we write down and expression in 1st order logic, we have several elements of context to specify before we can make sense of **what we're talking about**.

- The objects we're talking about — the universe of the structure — which Enderton specifies as the interpretation of the symbol $\forall$.

- The meaning of each of the symbols in the language — each of the predicate symbols, each of the constant symbols, and each of the function symbols.

  But if we include $=$, then the meaning of $=$ is *required* to be actual equality on the universe.
  (And yes, I'm sure you can come up with circumstances when what equality is is unclear — but please lay those aside.)

- And, when we're looking a part of a formula, the meaning we might get by from the context and other parts of the formula. Consider, for example, the C++ function

```
bool twiddle(int x)
 { if (g(x) > f(3)) return true;
   else if (f(x) > g(3)) return false;
   else return (g(f(x)) = f(g(x));
 }
```

  What does this function accomplish? Good question. To make any sense of it we must first know what `f` and `g` are. So, in this context, we can only say something like "it does . . . , whatever `g` and `f` are."

  In programming language lingo, we're giving the *environment* of this function definition.

Syntax: Terms & Formulas: Intuitively, terms are intended to refer to objects, and formulas are intended to refer to truth values.

Yes, it's legal to consider circumstances where the objects are the truth values, but please don't confuse the intuition now!

The following are terms of a language:

- Any constant symbol of the language.
- Any variable symbol v_i.
- For any k-ary function symbol of the language, and any terms $t_1, \ldots, t_k$ of the language, $f(t_1, \ldots, t_k)$.
  - **Enderton** uses fewer symbols — he'd write $ft_1 \ldots t_k$. I find that hard to read and write, and I'll treat my preferred notation as an informal writing of Enderton's.
  - (¡So you'll be stuck reading both my notation an Enderton's!)
  - (I'll ignore the parsing issue that every term with Enderton's notation has a unique parse tree.)

And the following are formulas of the language:

- **Atomic formulas:** For each k-ary relation symbol p of the language, and for each k terms $t_1, \ldots, t_k$ of the language, $p(t_1, \ldots, t_k)$. And, if $=$ is part of the language, for each two terms t_1, t_2 of the language, $t_1 = t_2$.
- **Non-atomic formulas:** For any formulas φ and ψ of the language, and for any variable symbol v_i,

$$(\neg\varphi), \quad (\varphi \to \psi), \quad \text{and} \quad (\forall v_i\, \varphi).$$

(Note that $(\forall v_i\, \varphi)$ is legal even if v_i doesn't appear anywhere in φ. Then it will just turn out that φ and $(\forall v_i\, \varphi)$ are logically equivalent.)

Semantics: 1. We specify a *structure* that interprets each symbol of the language.

Usually we use old upper-case German-style letters for structures. In print,

$$\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \ldots, \mathfrak{M}, \ldots, \mathfrak{N}, \ldots, \mathfrak{Z}.$$

(Mostly, we use just $\mathfrak{A}$.)

In handwriting we use old German script — google Sütterlin.)

$\mathfrak{A}$ interprets each symbol of the language:

$\mathfrak{A}(\forall)$ specifies what the universe is. It's written $|\mathfrak{A}|$.

For each constant symbol c**,** $\mathfrak{A}(c)$ is the element of $|\mathfrak{A}|$ it names. It's normally denoted $c^{\mathfrak{A}}$.

For each relation symbol r — say r is k-ary — $\mathfrak{A}(r)$ is a subset of $|\mathfrak{A}|^k$ – the set of all k-tuples for which the relation is true. We write $r^{\mathfrak{A}}$.

For each function symbol f — say f is k-ary — $\mathfrak{A}(f)$ is a function from $|\mathfrak{A}|^k$ to $|\mathfrak{A}|$ (and thus is a subset of $|\mathfrak{A}|^{k+1}$. Write $f^{\mathfrak{A}}$.

2. **Aside: Scopes of Quantifiers:** We take a quantifier — just $\forall$ in Enderton's formalism — as both declaring a new variable and saying something about it's meaning. Figuring out which variable x was declared by which quantifier follow sthe common scope rules of comuter languages.

   Example: $(\forall x(\forall y(\forall x(x = y) \to x > 3)) \to (x < 19))$.

   Express that showing the nesting:

	Scope	Declares
$($ $\to (x < 19))$	0	?
$\forall x($ $)$	1	new x
$\forall y($ $\to x > 3)$	2	new y
$\forall x(x = y)$	3	new x

   - Each occurrence of variable is ***bound*** by the innermost quantifier for that variable.
     In scope 3, the occurrence of x is bound by the $\forall s$ in scope 3.
     In scope 2, the occurrence of x is bound by the $\forall x$ in scope 1
     — since the new binding in scope 2 was for y.
   - In scope 0, the x is not bound by any quantifier (in this subformula) — so this occurrence of x is ***free*** (in programmgin language, *global*)

   In creating the formalism, Tarski started interpreting the subformulas from the inside out.

   - In the exampe above, the innermost subformula is $x = y$, and the next innermost is $\forall x(x = y)$.
   - So we have ot make sense of both of them in a context where we don't know what y refers to
     — and $x = y$ in the context where we don't know what either of them is.

3. **Interpreting free variables:** We consider all possible *interpretations* of the free variables in $|\mathfrak{A}|$
   — all possible functions $s : \{v_1, v_2, \ldots\} \to |\mathfrak{A}|$.

   And we specify whether $\mathfrak{A}$ satisfies a formula in the context of an interpretation s of the free variables.

4. **Interpreting terms in a structure:** We extend the interpretation s to an interpretation $\overline{s}$ of all terms:
   - For v_i a variable symbol, $\overline{s}(v_i) = s(v_i)$.
   - For c a constant symbol of the language, $\overline{s}(c) = c^{\mathfrak{A}}$.
   - For term $t = f(t_1, \ldots, t_k)$,

$$\overline{s}(f(t_1, \ldots, t_k) \;=\; f^{\mathfrak{A}}(\overline{s}(t_1), \ldots, \overline{s}(t_k)).$$

5. **Defiing satisfaction of atomic formulas** — in a structure $\mathfrak{A}$ and relative to a variable interpretation s:
   - (If "=" is part of the language today) — for any terms t_1, t_2,
     $\mathfrak{A}$ *satisfies* an atomic formula $t_1 = t_2$ at s if $\overline{s}(x) = \overline{s}(y)$.
   - $\mathfrak{A}$ *satisfies* an atomic formula $r(t_1, \ldots, t_k)$ at s
     if $(\overline{s}(t_1), \ldots, \overline{s}(t_k) \;\in\; r^{\mathfrak{A}}$.

   Notation: For $\mathfrak{A}$ satisfies φ at s:

$$\mathfrak{A} \models \varphi[s] \qquad \text{or} \qquad \models_{\mathfrak{A}} \varphi[s].$$

6. Defining satisfaction for non-atomic forulas:
   - $\mathfrak{A} \models (\neg\varphi)[s]$ if $\mathfrak{A} \not\models \varphi[s]$.
   - $\mathfrak{A} \models (\varphi \to \psi)[s]$ if $\mathfrak{A} \not\models \varphi[s]$ or $\mathfrak{A} \models \psi[s]$ (or both).
   - $\mathfrak{A} \models (\forall x\varphi)[s]$ if
     for every $s' : \{v_1, v_2, \ldots \to |\mathfrak{A}|$,
     if $s'(v_i) = s(v_i)$ for every variable except (possibly) x,
     then $\mathfrak{A} \models \varphi[s']$.

Notes:

1. In our recursion on formulas above, we're using the word "not" (in $\not\models$) in defining satisfaction for $\not$ and the term "for every" in defining satisfaction for $\forall$.

   So it sounds circular. ***But*** it gives us a clear inductive definition of satisfaction to work with.

2. It's a straightforward exercise to show (by induction on formulas) that whether $\mathfrak{A} \models \varphi[s]$ depends upon only the values $s(v_i)$ where v_i actually occurs free in s.

3. A <u>sentence</u> is a formula with no free variables.

   It follows from (2) above that, for φ a sentence, whether $\mathfrak{A} \models \varphi[s]$ does not depend upon s.

   So, for φ a sentence, we normally write just $\mathfrak{A} \models \varphi$.

   And we read this as $\mathfrak{A}$ *satisfies* φ or $\mathfrak{A}$ *is a model of* φ.

Redefining logical implication for first order logic: For Σ, Φ *sets* of sentences, we say

$$\Sigma \models \Phi \tag{1}$$

if every structure (for the language)
- which is a model of every sentence in Σ
- is also a model of every sentence in Φ.

We read that as Σ *logically implies* Φ.

- Above, if $\Sigma = \{\sigma\}$ (so if Σ has only one element) we usually write just σ in notation (1).
- If $\Sigma = \emptyset$, we typically leave out Σ entirely in notation (1).
- If $\Phi = \{\varphi\}$ (so Φ has only one element, we usually write φ instead of $\{\varphi\}$ in notation (1).
- $\Sigma \models\!=\!\mid \Phi$ means $\Sigma \models \Phi$ *and* $\Phi \models \Sigma$.
  In English:: "Σ is logically equivalent to Φ."
- We reserve the term *tautologically implies* for following from the rules of propositional logic alone.

Definability in a structure $\mathfrak{A}$: In the sytle of my database example on p. 10.

- To define a relation on $|\mathfrak{A}|$ in first order logic:
  Normally we just give a defining formula.
  Of course, sometimes it's not all all obvious that the defining formul does give what we want —that step might require a proof.
- To show a relation is *not* definable in first order is harder.
  That requires a proof that considers all possible defining formulas.
  We'll see a few examples.