Math Logic, Week#2

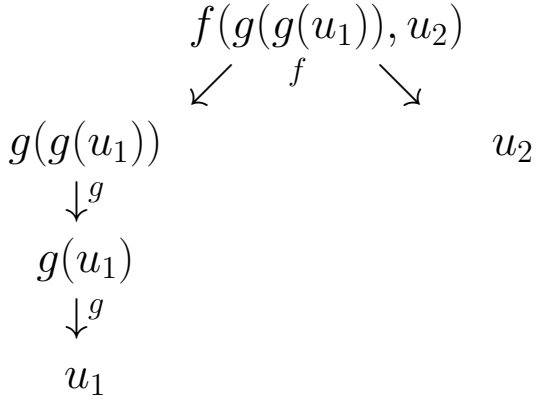
Woden's Day

Reminder: What does $\Sigma \models \varphi$ mean — for φ a formula and Σ a set of formulas?

§**1.4** is a very formal justification of what we do in defining sets and functions by induction.

Enderton builds an item inductively from a set $B \subseteq U$ with functions $f : U \times U \to U$ and $g : U \to U$.

This amounts to building up elements in a tree — essentiually, a parse tree:



The analogue of Enderton's approach to the Induction Principle is that set of well-formed-formulas is both

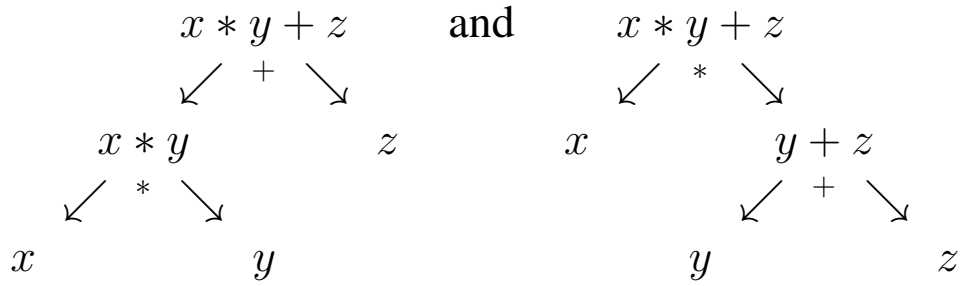
C^***:** The smallest set of strings of symbols which contains the proposition letters and is *closed* under $\varepsilon_\neg$, $\varepsilon_\wedge$, $\varepsilon_\vee$, ...

C_***:** The set of all strings of symbols with parse trees, (i) whose leaves are proposition letters, and (ii) where each non-leaf is one of $\varepsilon_\neg$, $\varepsilon_\wedge$, $\varepsilon_\vee$, ... applied to shorter strings.

To prove $C_* \subseteq C^*$ for this case**:** By induction: if it's false, pick a string violating the inclusion of minimum height, and get an immediate contradiction.

That generation is <u>free</u> if each element parsable string has a *unique* parse tree.

E.g., we can't allow both parse trees



(If we did, we'd get 2 results as we evaluate $2 * 3 + 4$.)

- **Unique Readability:** We our notation for formulas, we have enough parentheses to get unique parse trees.
- **So we can extend** a truth assignment h to the proposition letters to a truth assignment $\overline{h}$ on all formulas.

- **¿Any more questions on §1.4?**
- **¿Or do you want to take until Frea's Day (Friday) to think about it?**

¿Questions on §1.5?

For Frea's Day: Finish Chapter 1.

Note that Enderton's discussion of "*effectiveness*" — or "*computability*" — is quite informal.

- But you can fill in an intuition: this notion of *computability* is exactly that of being computable in most major computer languages (*e.g.*, C, C++, Java) on a machine with no limits on its memory and no limits on how long the program can run — just that it stop *sometime* and having used only *some finite* amount of time.

- A set is *decidable* if there is a computer program that, when its input is any possible element of the set, will run, ultimately halt, and
  1. output *true* if it's in the set, or
  2. output *false* it's not in the set.
- A set is *semi-decidable* if there is a computer program that, when its input is any possible element of the set, will run, and
  1. ultimately halt and output *true*, if it's in the set, or
  2. either output *false* or run forever, if it's not in the set.

Alan Turing proved that there is a set of integers that is semi-decidable but not decidable. (An analogous result follows for sets of strings.)

(The difference between decidability and semi-decidability is somewhat analogous to the difference between P (polynomial-time) and NP (nondeterministic polynomial-time). But nobody can prove whether or not P=NP.)

Next Homework Assignment: §1.7 ## 1,2.

Homework presentations:

- §1.4 #2
- §1.5 #2
- §1.5 #4
- §1.5 #7
- §1.5 # 9

Aside on Turing: — just for you to read for background information.

1. All our computer programs are long strings of characters.
   Consider the set P of all C++ (or C or Pascal or . . .) functions that
   - take just one parameter, which is a string;
   - return just a boolean — and have no side effects;
   - and access fixed set of libraries, all of which have not side effects.

   Any $p \in P$ is a string, so it makes perfect sense to compile and run p — and pass is p itself as an input string.
2. Now we ask: for $p \in P$, does p halt on input p — or does it run on forever?
   Let HALT $= \{p \in P : p \text{ halts on input } p\}$.
3. Note that HALT is semi-decidable: To determine whether $p \in$ HALT, just run p on input p and wait for something to happen.
4. But HALT is not decidable. For suppose it were. Then there would be a function $H \in P$ where, for any $p \in P$,
   - $H(p)$ returns *true* if p halts on input p, and
   - $H(p)$ returns *false* otherwise.

   Now let D be the function: . . . . . .

   ```
   bool D (string p)
   {   while (H(p)) do { }
       return true;
   }
   ```
5. Is $D \in$ HALT — *i.e.*, does execution of $D(D)$ halt?
   Check by running $H(D)$:

   If $H' \in$ HALT**,** *i.e.*, execution of $D(D)$ halts. Then $H(D) =$ *true*.
   Now execute $D(D)$. Since $H(D) =$ *true*, the while loop runs on forever — and $D(D)$ — so exedution of $D(D)$ doesn't halt after all.

   If $H' \notin$ HALT**,** *i.e.*, execution of $D(D)$ runs on forever. Then $H(D) =$ *false*.
   Again, execute $D(D)$. The while loop executes 0 times and the program returns *true* and halts, contradicting the definition of this case.
6. In either case we get a contradiction, so it must be that no such program H exists.

Another Aside: Russell's Paradox — also just for your information:

When Cantor proposed set theory he just assumed that any collection one could describe is a set. And sets may be elements of other sets.

1. Let U be the set of all sets.
2. Let $D = \{s \in U : s \notin s\}$ — the set of all sets that are not elements of themselves.
3. Now ask whether $D \in D$. Prove a contradiction as above:
   - If $D \in D$, then, by definition of D, $\notin D$.
   - And, if $D \notin D$, by definition of D, $D \in D$.

   In either case, we get a contradiction, so set D cannot exist. Modern set theory assumes that each set is composed of simpler sets, so that no set can be an element of itself, and it also says D above does not exist.

Frea's Day

Read my notes on Turing's proof and Russell's paradox above. You are not required to know this material, but it's good background

— and a more complicated version of the same sort of paradox will appear later in the class.

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.

Read: §§**2.0-2.2 (up through "Definability in a Structure"** This is using logic to model formal discussions — originally in formal mathematics, but it can be extended.

Notes:

- *Formally*, we are no using only 2 of the propositional connectives — $\rightarrow$ and $\neg$. *However*, when we're being informal, we'll continue to use $\wedge$, $\vee$, and $\leftrightarrow$. Of course, we have to be able to translate away occurrences of these connectives. *Admission:* I usually think in terms of using connectives $\wedge$, $\vee$, and $\neg$. So I have to translate too. *The point* of having only 2 propositional connectives is to keep discussions shorter.
- We'll now have two languages.

  The formal language (a.k.a. *object language*) which has the formal symbols and syntax that Enderton gives in Chapters 1-2.

  The *meta*-languaage which we use to talk about statements in the formal language. Here we use English.

  Typically, you may have used symbols such as $\wedge$, $\vee$, $\neg$, $\exists$, and $\forall$ mixed in with English. Here that will create confusion. So please don't use them in your meta-language discussions.

Note: Mixing meta-language and does create actual confusion. For example:

1. There are only finitely many words in English. So there are only finitely many phrases of ≤ 12345 words.
2. Some of these phrases, such as "*and or beautiful green if*" are nonsense.
   Some are meaningful and define specific objects, such as "*the largest satelite circling the earth*" or "*the sum of two and three*."
   And some, of course, are neither.
3. Look at those ≤ 12345-word phrases that actually specify specific natural numbers. There are, still, only a finite number of them.
   So there are some natural numbers which cannot be specified by English phrases of ≤ 12345 words.
4. By the principle of well-odering, there is a least natural number that cannot be specified by an English phrase of ≤ 12345 words.
5. But then I just specified it — and I used ≤ 12345 words. ¡A contradiction!

¿Any questions on material so far?

Go through homeworks:

- Finish §1.5 #2
- §1.5 #4
- §1.5 #7
- §1.5 #9