

1. *Formulas*

- a. The set of *formulas* of an elementary language is defined recursively as follows.
 - i. For any positive integer n , if p is an n -ary predicate and $t_1, \dots, t_n$ are terms, then the string $pt_1 \dots t_n$ formed by concatenating p and $t_1, \dots, t_n$ is a formula.
 - ii. If f is a formula, then so is $\neg f$.
 - iii. If f and g are formulas, then so is $\Rightarrow fg$.
 - iv. If v is a variable and f is a formula, then so is $\forall v f$.
- b. For example, using the elementary language for number theory described earlier, if x, y, z are variables and $\cdot, 0$ are the multiplication operator and a constant, then $=yx$, $=x \cdot z 0$, $\neg =x \cdot z 0$, $\Rightarrow =yx \neg =x \cdot z 0$, and $\forall y \Rightarrow =yx \neg =x \cdot z 0$ are formulas.
- c. Various notations enhance readability of formulas. Binary predicates p are often expressed informally with infix notation: if t_1 and t_2 are terms, then $(t_1 p t_2)$ stands for $pt_1 t_2$. Thus, you can write the example atomic formulas $=yx$ and $=x \cdot z 0$ as $(y = x)$ and $(x = z \cdot 0)$. Negations of predicates are often written with a stroke: for example, $(x \neq z \cdot 0)$ stands for the example formula $\neg =x \cdot z 0$. Thus you can write the example formula $\Rightarrow =yx \neg =x \cdot z 0$ as $((y = x) \Rightarrow (x \neq z \cdot 0))$. Some of these parentheses are omitted according to familiar conventions: the last example is usually written $y = x \Rightarrow x \neq z \cdot 0$. Further mathematical notation will be used to stand for complicated formulas when that's convenient and the meaning is clear.

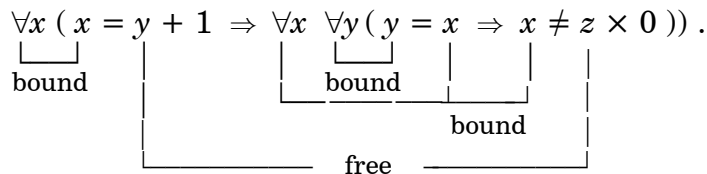
2. *Substrings and substitution*

- a. I forgot to include a definition of *substring* earlier, where it belongs. If s, t are strings and there exists a string u such that $su = t$ or $us = t$, then s is called a substring of t .
 - i. The empty string is a substring of every string.
 - ii. Every string is a substring of itself.
 - iii. If s is a substring of a string t and t is a substring of a string u , then s is a substring of u .
- b. I also forgot to introduce the idea of *substituting* a string t for specified occurrences of a symbol v in a string s . This is easy enough to visualize, but hard to describe precisely enough to avoid incorrect statements about repeated substitutions. Once the occurrences of v to be replaced are specified, the result of doing so will be denoted by s_t^v .

3. *Subformulas*

- a. If s, t are formulas and s is a substring of t , then s is called a *subformula* of t . Each of the list of formulas in the previous example, except the last, is a subformula of the next.

- b. *Theorem.* If two subformulas of the same formula f overlap in f , then one is a subformula of the other.
- i. I may need that theorem for something that follows. If so, I should prove it. For that, the notion *overlap* should be precise. I suspect it isn't, yet.
4. *Bound and free variables*
- a. In a formula like $\forall x (x = y + 1)$, the occurrences of x are said to be bound by the quantifier $\forall$ and the y is said to be free. This distinction reflects semantical practice described later: this formula is used to say something about *all* possible values of x but only about a *particular* value of y .
- b. For complicated formulas with several quantifiers and multiple occurrences of variables, these concepts are hard to define formally, but that's unfortunately necessary.
- c. An occurrence of a variable x in a formula f is *bound* by an occurrence of $\forall$ in f if this occurrence of x lies in a subformula of f that begins with that occurrence of $\forall$ followed by x , but in no shorter subformula that begins with $\forall x$. An unbound occurrence of a variable is called *free*. Consider this example, in which the bindings are indicated, as well as the two free occurrences of variables.



- d. I need the notion of free and bound occurrences of variables to describe *proper substitution* of terms for variables. For example, it's proper to substitute x or $1 + 1$ for v in the formula $0 \leq v \Rightarrow \exists w (v = w \cdot w)$: you get

$$0 \leq x \Rightarrow \exists w (x = w \cdot w) \quad 0 \leq 1 + 1 \Rightarrow \exists w (1 + 1 = w \cdot w).$$

But it's *not* proper to substitute $w \cdot w + 1$: that would yield

$$0 \leq w \cdot w + 1 \Rightarrow \exists w (w \cdot w + 1 = w \cdot w).$$

Improper substitution can change a seemingly true formula to a seemingly false one! The formal definition of this notion is also somewhat complicated: terms $u_1, \dots, u_n$ are said to be *substitutable* for variables $v_1, \dots, v_n$ in a formula f if no free occurrence of any v_k in f lies in a subformula that starts with $\forall w$ for any variable w that occurs in u_k . In the previous example, $w \cdot w + 1$ is not substitutable for v in

$$0 \leq v \Rightarrow \neg \forall w \neg (v = w \times w)$$

because the second v in this formula is free and occurs in the subformula $\forall w \neg (v = w \cdot w)$ that starts with $\forall w$, and w occurs in $w \cdot w + 1$. When $u_1, \dots, u_n$ are substitutable for $v_1, \dots, v_n$ in f , the *result* g of substituting them for all free occurrences of $v_1, \dots, v_n$ is denoted by

$$f_{u_1, \dots, u_n}^{v_1, \dots, v_n}.$$

- e. The notions *bound*, *free*, *substitutable*, and *result of substitution* really should be defined recursively. I suspect that would avoid awkward usage of the notion of subformula.
- f. A formula in which *no* variable occurs free is called a *sentence*.
- g. Some authors employ various techniques to avoid awkward formulas with nested occurrences of the same quantifier, or with free and bound occurrences of the same variable, for which it's difficult to describe and perform proper substitutions. Some don't regard strings like

$$\forall x \forall x (x = x) \qquad x = x \Rightarrow \forall x (x = x)$$

as formulas. Others don't use formulas with free variables at all. Such restrictions generally lead to more problems than they avoid, so these notes permit consideration of many awkward formulas that you'd never want to use in practice.

5. *Syntactic consequences*

- a. Like Boolean logic, first-order logic has syntactic and semantic notions of consequence. A major theorem, the Gödel completeness theorem, will show that those notions coincide.
- b. The first-order notion of syntactic consequence is the same as that for Boolean logic, except that the set of all formulas, referred to indirectly in its definition, is now the set of all formulas of a given elementary language. Moreover, the logical axioms underlying that definition have not yet been specified.
- c. Most authors permit use of a second inference rule, *generalization*: a formula p_m may also occur in a deduction $p_0, \dots, p_{n-1}$ if $p_m = \forall v p_l$ for some variable v and some $l < m$. However, I am following Enderton 1972, section 2.4, in avoiding that. This removes my presentation of first-order logic a little farther from mathematical research practice: most mathematicians, if pressed to name the inference rules they often use, might list three or four. The present discussion is not devoted to analyzing mathematical research or exposition, but to the precise description of a specific technique useful in that analysis and in certain areas of software development. Avoiding inference by generalization simplifies this technique and its description.
- d. A set T of sentences closed under c is called a *theory*. Thus, for every set S of sentences, $c(S)$ is a theory. Sentences in a theory are often called its *theorems*.

6. *Semantic consequences*

- a. Elementary languages are used to express facts about mathematical structures called relational systems. An elementary language L is described by specifying its variables, constants, operators, and predicates. A *relational system compatible with L* consists of a nonempty *underlying set* M , some associated elements of M called *constants*, and some associated operators and relations of various arities on M . The associated constants, operators, and relations, and their arities must correspond bijectively and agree with the constants,

operators and predicates of L and their arities, except that no relation on M should be specified as corresponding to the predicate $=$. (That predicate will *always* correspond to the identity relation: it need not be specified.)

- b. For example, a relational system compatible with the elementary language for number theory considered earlier would consist of a nonempty set M , constants $z, u, t \in M$, a singular and two binary operators $', +, \cdot$ on M , and a binary relation $\leq$ on M . This relational system need not have anything to do with the natural numbers—it has the same structure as the relational system $\mathbb{N}$ that the language was designed to describe, but need not agree in any other detail.
- c. As a second example, a relational system compatible with the elementary language for a real vector space considered earlier would consist of a nonempty set V , a constant 0 , a binary operator $+$, and a singular multiplication operator $\cdot_t$ for each real scalar t .
- d. Since this second language has no predicates, its compatible relational systems have no relations, only operators. Relational systems of that sort are called *algebras*; the study of their properties in general is called *universal algebra*.