# Syntax of first order logic

Professor Cian Dorr

4th October 2022

New York University

# First-order languages

For the next few weeks we will be exploring the metatheory of classical first order logic. The first order of business is to give a mathematically precise definition of the *syntax* of first-order langauges: i.e. to say precisely which strings of symbols are well-formed expressions.

# Relational signatures

When we were discussing "relational structures" we used a very general notion of "relational signature" where a signature $\Sigma = \langle I_\Sigma, a_\Sigma \rangle$ is just any set $I_\Sigma$ together with an "arity" function mapping $I_\Sigma$ to $\mathbb{N}^+$.

For the purposes of defining the syntax of first-order langauges, we'll use a slightly different definition. We'll require the elements of $I_\Sigma$ (which will serve as predicates in our language) to be *strings* over our standard alphabet (e.g. Unicode), which we call Char; we have $\mathrm{Str} \coloneqq \mathrm{Char}^*$. We further require that these strings be nonempty, not include certain "special characters" (including $($, $)$, $,$, $=$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$, $\forall$, $\exists$, $($, $)$, and newline), and not be in a certain set Var to be defined below.

We'll also allow the "arity" function $a$ to map some of these strings to 0. 0-ary predicates aren't very interesting, but they can be equated with *sentence letters* familiar from propositional logic.

# Variables

**Definition**

Var, the set of *variables*, is the set of all strings whose first character is `x`, `y`, or `z`, and all of whose other characters are among `'`, `0`, `1`, . . . , `9`.

Of course we could adopt different conventions here, but it's most convenient to pick which strings count as variables once and for all. What matters is just that it be an infinite set, and not overlap our signature.

*Note:* the letters *u* and *v* are not variables. We'll use these in our metalanguage (mathematical English) as variables ranging over object-language variables: for example, we might say 'Whenever *u* is a variable, the first character of *u* is `x`, `y`, or `z`.'

# Relational atomic formulae

**Definition**

Where $\Sigma$ is a relational signature, Atoms($\Sigma$), the set of *atomic formulae* over $\Sigma$, is the set of all strings that are either in $I_\Sigma$ and mapped to 0 by $a_\Sigma$, or of the form

$$i \oplus \texttt{(} \oplus v_1 \oplus \texttt{,} \oplus \cdots \oplus v_n \oplus \texttt{)}$$

where each $v_i$ is a variable and either $i \in I_\Sigma$ and $n = a_\Sigma i$, or $i = \texttt{=}$ and $n = 2$.

Note: For single-symbol binary predicates, including $=$, we'll use 'infix' notation because it looks nicer: e.g., we will write $x = y$ and $x \leq y$ instead of `=(x,y)` and `≤(x,y)`. We will treat these as an unofficial abbreviation scheme: $x \leq y$ is just the string `≤(x,y)`.

- In Russell, $=$ is *really* infix while all other predicates are officially prefix.

## Unique readability for relational atomic formulae

Intuitively, it's clear why we needed to include the parentheses and commas in our definition of 'atomic formula'. Suppose that we left them out. The following counts as a legitimate relational signature by our definition:

$$\texttt{Fo} \mapsto 2, \texttt{Fox} \mapsto 1$$

But then the string `Foxy` would be "ambiguous", in the sense that we could consider it either as an atomic formula formed by combining the binary predicate `Fo` with the variables `x` and `y` (corresponding to `Fo(x,y)` in our notation), or as one formed by combining the singulary predicate `Fox` with the variable `y` (corresponding to `Fox(y)` in our notation).

Allowing such ambiguity would be a complete disaster in ways too numerous to mention!

# Unique readability for relational atomic formulae

So, we should check that our parentheses and commas have eliminated ambiguity, i.e.:

**Unique Readability for Relational Atomic Formulae**

For any predicates $F, G \in R_\Sigma \cup \{=\}$ such that $a_\Sigma(F) = n$ or $F = =$ and $n = 2$ and $a_\Sigma(G) = m$, or $G = =$ and $m = 2$, and any variables $v_1, \ldots, v_n, u_1, \ldots u_m$, if

$$F(v_1, \cdots, v_n) = G(u_1, \cdots, u_m)$$

then $F = G$ and $n = m$ and $v_k = u_k$ for each $k$.

# Proving Unique Readability for Relational Atomic Formulae

The proof rests on the following lemma about strings (proved in Problem Set 4):

**Concatenation Lemma**

If $s \oplus t = s' \oplus t'$ then either $s = s'$, or $s = s' \oplus u$ for some nonempty initial substring $u$ of $t'$, or $s' = s \oplus u$ for for some nonempty initial substring $u$ of $t$.

To use this to prove unique readability: suppose for contradiction that $F \neq G \in R_\Sigma$ are such that

$$F(v_1, \cdots, v_n) = G(u_1, \cdots, u_m)$$

Then by the lemma, either $F = G \oplus s$ for some nonempty initial substring $s$ of $(u_1, \cdots, u_m)$, or $G = F \oplus s'$ for some nonempty initial substring $s'$ of $(v_1, \cdots, v_n)$. But all such $s$ and $s'$ contain the character $($, whereas neither $F$ nor $G$ do (by our "no special characters" constraint on relational signatures): contradiction.

The remainder is proved along the same lines.

# Formulae

**Definition**

When $\Sigma$ is a relational signature, the set $\mathcal{L}(\Sigma)$ of *first order formulae* is the smallest set of strings such that:

1. Every element of $\mathrm{Atoms}(\Sigma)$ is in it.
2. Whenever $s$ is in it, $\neg s$ is in it.
3. Whenever $s$ and $t$ are in it, $(s \wedge t)$, $(s \vee t)$, and $(s \to t)$ are in it.
4. Whenever $s$ is in it, $\forall v\ s$ and $\exists v\ s$ are in it for every variable $v$.

I.e.: $\mathcal{L}(\Sigma)$ is the closure of $\mathrm{Atoms}(\Sigma)$ under the singulary function $[s \mapsto \neg s]$; the binary functions $[\langle s, t\rangle \mapsto (s \wedge t)]$, $[\langle s, t\rangle \mapsto (s \vee t)]$, $[\langle s, t\rangle \mapsto (s \to t)]$; and two countably infinite families of singulary functions $([s \mapsto \forall v\ s])_{v \in \mathrm{Var}}$ and $([s \mapsto \exists v\ s])_{v \in \mathrm{Var}}$.

We'll sometimes use $\mathcal{L}(\Sigma)$ to name not just the set defined above, but the *algebraic structure* with that family of functions.

# Unofficially omitting parentheses

Notice that according to the official definition, the string `x=y→y=x` is not a formula: we have to surround it with `(` and `)`.

But we'll *unofficially* omit many parentheses when we're writing formulae according to the "implicit quotation convention". We'll do so according to the following conventions:

1. Unary operations $\neg$, $\forall v$, $\exists v$ take maximum precedence; then $\wedge$ and $\vee$; and finally $\rightarrow$. (Cf. PEMDAS)
2. Multiple occurrences of the same binary connectives associate to the right: $P \wedge Q \wedge R$, $P \vee Q \vee R$, $P \rightarrow Q \rightarrow R$ are short for $P \wedge (Q \wedge R)$, $P \vee (Q \vee R)$, $P \rightarrow (Q \rightarrow R)$, respectively.

Choice point 2 rarely matters for $\wedge$ and $\vee$, but matters a lot for $\rightarrow$!

# Unique readability for formulae

The point of having all the parentheses in the official notation is to prevent "ambiguity": it would be an untold disaster if, e.g., a single string $P \wedge Q \rightarrow Q$ could be considered either as the result of combining $P$ with $Q \rightarrow Q$ using $\wedge$ or as the result of combining $P \wedge Q$ with $Q$ using $\rightarrow$.

By requiring the parentheses, we secure the

**Unique Readability Theorem for Relational Formulae**

When $\Sigma$ is a relational signature, $\mathcal{L}(\Sigma)$ (considered as an algebraic structure) has the Injective Property. That is, on the set of formulae of $\mathcal{L}(\Sigma)$, the functions $[s \mapsto \neg s]$, $[\langle s, t\rangle \mapsto (s \wedge t)]$, $[\langle s, t\rangle \mapsto (s \vee t)]$, $[\langle s, t\rangle \mapsto (s \rightarrow t)]$, $([s \mapsto \forall v\, s])_{v \in \mathrm{Var}}$ and $([s \mapsto \exists v\, s])_{v \in \mathrm{Var}}$ are all injective, and their ranges do not overlap each other or $\mathrm{Atoms}(\Sigma)$.

Intuitively: every formula can be constructed in exactly one way.

# Unique readability for formulae

Note: the function $[\langle s, t\rangle \mapsto (s \wedge t)]$ is *not* injective on the set of all *strings*: e.g. it maps the distinct pairs $\langle ), () \wedge (\rangle$ and $\langle ) \wedge (), (\rangle$ to the same string $() \wedge () \wedge ()$.

That's OK: for the Injective Property we want, we only need that whenever $(s \wedge t) = (s' \wedge t')$ *and* $s, t, s', t'$ *are formulae*, $s = s'$ and $t = t'$. And none of the strings just mentioned are formulae.

# Proving unique readability

For the proof, define for each $c \in \mathrm{Char}$ a function $\mathrm{count}_c : \mathrm{Str} \to \mathbb{N}$ by

$$\mathrm{count}_c[] = 0 \qquad \mathrm{count}_c(a : s) = \begin{cases} \mathrm{suc}\,\mathrm{count}_c\, s & \text{if } a = c \\ \mathrm{count}_c\, s & \text{if } a \neq c \end{cases}$$

An easy inductive proof establishes $\mathrm{count}_c(s \oplus t) = \mathrm{count}_c\, s + \mathrm{count}_c\, t$.

Say that $s$ is *balanced* iff $\mathrm{count}_{(}\, s = \mathrm{count}_{)}\, s$, *left-heavy* iff $\mathrm{count}_{(}\, s > \mathrm{count}_{)}\, s$, and *parenthesis-free* iff $\mathrm{count}_{(}\, s = \mathrm{count}_{)}\, s = 0$. We begin by proving:

**Parenthesis Lemma**

Every formula is (i) balanced and (ii) such that all its proper initial substrings are either left-heavy or parenthesis-free.

Note that if a string has properties (i) and (ii), *all* its initial substrings are either balanced or left-heavy.

# Proving the parenthesis lemma

*Base step.* If $P \in \operatorname{Atoms}\Sigma$, then either $P$ is a sentential constant of $\Sigma$, in which case it and all its initial substrings are parenthesis-free, or of the form $F(v_1, \cdots, v_n)$. Since all variables and all predicates of $\Sigma$ are parenthesis-free, every such $P$ has exactly one $($ and one $)$, and so is balanced. And every proper initial substring of $P$ is either a substring of $F$, and so parenthesis-free, or of the form $F \oplus s$ where $s$ is a substring of $(v_1, \ldots, v_n$, in which case it contains one $($ and no $)$s making it left-heavy.

*Induction step for singulary connectives:* $\neg P$, $\forall v\, P$, and $\exists v\, P$ have the same number of $($s and $)$s as $P$, and so are balanced if $P$ is. And their proper initial substrings are either proper initial substrings of $\neg$, $\forall v$, or $\exists v$ and so parenthesis-free, or of the form $\neg s$, $\forall v\, s$, or $\exists v\, s$ where $s$ is a proper initial substring of $P$, and so either parenthesis-free or left-heavy since they have the same number of $($s and $)$s as $P$.

# Proving unique readability

*Induction step for binary connectives:* Suppose (i) and (ii) hold of $P$ and $Q$, and consider $R = (P\#Q)$ where $\#$ is any of $\wedge, \vee, \rightarrow$. This is balanced, since $\text{count}_( R = \text{count}_( P + \text{count}_( Q + 1$ and $\text{count}_) R = \text{count}_) P + \text{count}_) Q + 1$. And its proper initial substrings are exactly:

- []
- $(s$, where $s$ is an initial substring of $P$ (not necessarily proper)
- $(P\#t$, where $t$ is a initial substring of $Q$ (not necessarily proper)

[] is parenthesis-free. By our induction hypothesis concerning $P$, every initial substring $s$ of $P$ is either balanced or left-heavy, so $(s$ is left-heavy for all those strings. And by our induction hypotheses concerning both $P$ and $Q$, every initial substring $t$ of $Q$ is either balanced or left-heavy, $P$ is balanced, and $\#$ is parenthesis-free, so $(P\#t$ is left-heavy.

# Proving unique readability

Next, we show that for any formulae $P, Q, P', Q'$ and binary connectives $\#$ and $\#'$, if $(P\#Q) = (P'\#'Q')$ then $P = P'$ and $Q = Q'$ and $\# = \#'$.

Suppose for contradiction that this is not the case. Then we must have $(P \neq (P'$, since otherwise we'd have $\#Q) = \#'Q')$ by the cancellativity property of $\oplus$ (if $s \oplus t = s \oplus t'$ then $t = t'$: an easy induction), and hence $\# = \#'$ and $Q = Q'$ since the connectives are just one character long.

So by the Concatenation Lemma, one of $P$ and $P'$ is a proper substring of the other. But this can't happen, since by the previous result, neither of them is parenthesis free, both are balanced, and each is such that all of its proper initial substrings are either parenthesis-free or left-heavy.

# Proving unique readability

Now we can check the Injective Property:

1. The singulary functions $[P \mapsto \neg P]$, $[P \mapsto \forall v\ P]$, $[P \mapsto \exists v\ P]$ are injective by the Cancellativity property of $\oplus$.
2. The binary functions $[\langle P, Q\rangle \mapsto (P \# Q)]$ are injective on the set of formulae by what we proved on the previous slide.
3. The atomic formulae aren't in the ranges of any of those functions since no atomic formula has $($, $\neg$, $\forall$, or $\exists$ as its first character.
4. The ranges of the singulary functions don't overlap with one another or with the ranges of the binary functions since no string has more than one of $\neg$, $\forall$, $\exists$, $($ as its first character.
5. The images of the set of formulae under the binary functions don't overlap by what we proved on the previous slide.

# Variant languages

It will sometimes be useful to think about other languages where we only use some of the logical connectives. In general, for $\Phi \subseteq \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists\}$ and a relational signature $\Sigma$, let $\mathcal{L}_\Phi(\Sigma)$ be defined as above but with the clauses for the symbols in $\Phi$ (and adding $\leftrightarrow$ to part 3 if needs be.) We leave out curly brackets in the subscript, so we have $\mathcal{L}(\Sigma) = \mathcal{L}_{\neg,\wedge,\vee,\rightarrow,\forall,\exists}(\Sigma)$.

- Russell's *The Limits of Logic* only allows languages of the form $\mathcal{L}_{\neg,\wedge,\forall}(\Sigma)$. The remaining symbols are treated as *metalinguistic abbreviations*. For example, when Russell writes $\texttt{(x=y)}\rightarrow\texttt{(y=x)}$, that is (confusingly!) just another way of referring to the very same string of symbols that according to the standard font convention would be written as $\neg(\texttt{(x=y)}\wedge\neg\texttt{(y=x)})$.
- Focusing on "smaller" languages does shorten some proofs, but it's sort of moving a bump in the carpet.

# The Recursion Theorem for (relational) formulae

Since $\mathcal{L}(\Sigma)$ has the Inductive and Injective properties, the Recursion Theorem applies. Here is a reminder of what that means:

**Recursion Theorem for Relational Formulae**

For any relational signature $\Sigma$ and functions $z : \text{Atoms}(\Sigma) \to B$; $f_\neg : B \to B$; $f_\wedge, f_\vee, f_\to : B \times B \to B$; $(u_v : B \to B)_{v \in \text{Var}}$, $(e_v : B \to B)_{v \in \text{Var}}$, there is a unique function $h : \mathcal{L}(\Sigma) \to B$ such that for all $P, Q \in \mathcal{L}(\Sigma)$ and $v \in \text{Var}$:

1. $hP = zP$ if $P \in \text{Atoms}(\Sigma)$.
2. $h(\neg P) = f_\neg(hP)$
3. $h(P \wedge Q) = f_\wedge(P, Q)$; $h(P \vee Q) = f_\vee(P, Q)$; $h(P \to Q) = f_\to(P, Q)$.
4. $h(\forall v P) = u_v P$; $h(\exists v P) = e_v P$.

# Sample recursive definitions for formulae

**Example**

Suppose we have a function $z : \mathrm{Atoms}(\Sigma) \to \mathcal{L}(\Sigma')$. Then we can extend this uniquely to a function $h : \mathcal{L}(\Sigma) \to \mathcal{L}(\Sigma')$ by setting $hP \coloneqq zP$ for $P$ atomic, and

$$h(\neg P) \coloneqq \neg(hP) \qquad h(P \to Q) \coloneqq hP \to hQ$$
$$h(P \wedge Q) \coloneqq hP \wedge hQ \qquad h(P \vee Q) \coloneqq hP \vee hQ$$
$$h(\forall v P) \coloneqq \forall v (hP) \qquad h(\exists v P) \coloneqq \exists v (hP)$$

Those recursion clauses for $h$ are called "trivial": they just "push through" all the connectives.

# Sample recursive definitions for formulae

Often when we're recursively defining a function from some $\mathcal{L}(\Sigma)$ to some $\mathcal{L}(\Sigma')$, we'll want *some* of our clauses to be trivial.

**Example**

We can define a function $h : \mathcal{L}(\Sigma) \to \mathcal{L}_{\neg,\vee,\to,\forall,\exists}(\Sigma)$ by setting $hP = P$ when $P$ is atomic, $h(P \wedge Q) = \neg(\neg hP \vee \neg hQ)$, remaining recursion clauses trivial.

**Example**

The *Gödel-Gentzen translation* $g : \mathcal{L}(\Sigma) \to \mathcal{L}_{\neg,\vee,\to,\forall,\exists}(\Sigma)$ has $gP = \neg\neg P$ when $P$ is atomic, $g(\neg P) = \neg(gP)$, $g(P \wedge Q) = gP \wedge gQ$, $g(P \vee Q) = \neg(\neg gP \wedge \neg gQ)$, $g(P \to Q) = gP \to gQ$, $g(\forall vP) = \forall v(gP)$, $g(\exists vP) = \neg\forall v\neg(gP)$.

# Sample recursive definitions for formulae

**Example**

We can define a function $[y/x] : \mathcal{L}(\Sigma) \to \mathcal{L}(\Sigma)$ by setting $[y/x]R(v_1, \ldots, v_n) = R(u_1, \ldots, u_n)$ where $u_i = v_i$ if $v_i \neq x$ and $u_i = y$ if $v_i = x$, $[y/x]\forall vP = \forall v[y/x]P$ if $v \neq x$ and $\forall vP$ if $v = x$, $[y/x]\exists vP = \exists v[y/x]P$ if $v \neq x$ and $\exists vP$ if $v = x$, with remaining recursion clauses trivial.

This function is called *substitution of $y$ for $x$*, and traditionally written in postfix position: e.g. we write $\neg R(x, y)[y/x] = \neg R(y, y)$. Below we'll give a general definition of which this is a special case.

- Russell would write it as $[\mathrm{x} \mapsto \mathrm{y}]$ (also in postfix position).

# Sample recursive definitions for formulae

**Definition**

FV is the function $\mathcal{L}(\Sigma) \to \mathcal{P}(\text{Var})$ such that

1. For any atomic formula $P = R(v_1, \ldots, v_n)$, $FV(P) = \{v_1, \ldots, v_n\}$.
2. For any formula $P$, $FV(\neg P) = FV(P)$.
3. For any formulae $P, Q$, $FV(P \wedge Q) = FV(P \vee Q) = FV(P \to Q) = FV(P) \cup FV(Q)$
4. For any formula $P$ and variable $v$, $FV(\forall v P) = FV(\exists v P) = FV(P) \setminus \{v\}$.

We say that $P$ is *closed*, (a *sentence* of $\mathcal{L}(\Sigma)$) when $FV(P) = \varnothing$; otherwise *open*.

Intuitively: if we have a particular way of *understanding* $\mathcal{L}(\Sigma)$, the sentences are going to be the ones that we can use to *assert that something is the case*. *Open* formulae by contrast are more analogous to *predicates* in English; e.g. $\text{loves}(x, x) \approx$ 'loves itself'.

# First-order signatures

We are going to be interested in first-order languages that are suited for talking about *numbers* and *strings*.

One could formalize theories in arithmetic using a relational signature. For example, we could talk about addition using a three-place predicate Sum where $\mathsf{Sum}(x, y, z)$ means '$x$ and $y$ add up to $z$'. We could express the commutativity of addition using a sentence

$$\forall x_1 \forall x_2 \forall y (\mathsf{Sum}(x_1, x_2, y) \to \mathsf{Sum}(x_2, x_1, y))$$

and its associativity with

$$\forall x_1 \forall x_2 \forall x_3 \forall z (\exists y (\mathsf{Sum}(x_1, x_2, y) \wedge \mathsf{Sum}(y, x_3, z)) \to \exists y (\mathsf{Sum}(x_2, x_3, y) \wedge \mathsf{Sum}(x_1, y, z)))$$

But this gets really awkward.

# First-order signatures

For such purposes, it's a lot more convenient to use a more general kind of first-order language that contains *function symbols* as well as predicates. If we talk about addition with a binary function symbol $+$, we can express commutativity and associativity as one would expect, as:

$$\forall x \forall y (x + y = y + x)$$
$$\forall x \forall y \forall z (x + (y + z) = (x + y) + z)$$

# First-order signatures

**Definition**

A *first-order signature* is a triple $\Sigma = \langle R_\Sigma, F_\Sigma, a_\Sigma \rangle$, where $R_\Sigma$ and $F_\Sigma$ are two *non-overlapping* sets of strings that don't contain any special characters and are not in Var, and $a_\Sigma : (R \cup F) \to \mathbb{N}$.

When $\Sigma = \langle R_\Sigma, F_\Sigma, a_\Sigma \rangle$ is a first-order signature, $i$ is called a *predicate* of $\Sigma$ if $i \in R_\Sigma$ and $a_\Sigma i > 0$; a *function symbol* of $\Sigma$ if $i \in F_\Sigma$ and $a_\Sigma i > 0$; a *sentence constant* of $\Sigma$ if $i \in R_\Sigma$ and $a_\Sigma i = 0$; and an *individual constant* of $\Sigma$ if $i \in F_\Sigma$ and $a_\Sigma i = 0$.

# First order languages with function symbols

Introducing individual constants and functional symbols makes the definition of *atomic formula* more complicated: we first define a different set of strings Terms($\Sigma$), and use it to define Atoms($\Sigma$).

**Definition**

When $\Sigma$ is a first-order signature, Terms($\Sigma$) is the smallest set $X$ of strings such that:

1. $v \in X$ for all $v \in \mathrm{Var}$.
2. $c \in X$ whenever $c$ is an individual constant of $\Sigma$.
3. $f(t_1, \ldots, t_n) \in X$ whenever $f$ is a function symbol of $\Sigma$ with $a_\Sigma f = n$, and $t_1, \ldots, t_n \in X$.

# Unique readability for terms

Notice that we use parentheses and commas to demarcate a function symbol and its arguments. Again, this is to avoid ambiguity and secure

**Unique Readability for Terms**

The set Terms($\Sigma$), considered as an algebraic structure, has the Injective Property.

That is: the functions $([\langle t_1, \ldots, t_n\rangle \mapsto f(t_1, \ldots, t_n)])_{f \in F_\Sigma}$ are all injective, and their ranges do not overlap one another or Var or the set of individual constants of $\Sigma$.

This can be proved in exactly the same way as unique readability for relational formulae, by showing that every term is balanced and every initial substring of a term is either parenthesis-free or left-heavy.

# Atomic formulae

We need new definitions of 'atomic formula' and 'formula' that allows terms that aren't variables to occur as arguments of predicates:

**Definition**

Where $\Sigma$ is a first-order signature, $\mathrm{Atoms}(\Sigma)$ is the set of all strings of the form

$$i \oplus \texttt{(} \oplus t_1 \oplus \texttt{,} \oplus \cdots \oplus t_n \oplus \texttt{)}$$

where each $t_i \in \mathrm{Terms}(\Sigma)$ and either $i \in R_\Sigma$ and $n = a_\Sigma i$, or $i = \texttt{=}$ and $n = 2$.

The definition of 'formula' for first-order signatures is exactly the same as for relational signatures, just plugging in the new definition of atomic formula.

# Unique readability

We can prove the following using the same techniques as before

**Unique Readability for Atomic Formulae**

For any signature $\Sigma$, any predicates $F, G \in R_\Sigma \cup \{=\}$ such that $a_\Sigma(F) = n$ or $F = =$ and $n = 2$ and $a_\Sigma(G) = m$, or $G = =$ and $m = 2$, and any $t_1, \ldots, t_n, t'_1, \ldots t'_m \in \mathrm{Terms}(\Sigma)$, if $F(t_1, \cdots, t_n) = G(t'_1, \cdots, t'_m)$ then $F = G$ and $n = m$ and $t_k = t_k$ for each $k$.

**Unique Readability Theorem for Relational Formulae**

When $\Sigma$ is a first-order signature, $\mathcal{L}(\Sigma)$ (considered as an algebraic structure) has the Injective Property. That is, on the set of formulae of $\mathcal{L}(\Sigma)$, the functions $[s \mapsto \neg s]$, $[\langle s, t\rangle \mapsto (s \wedge t)]$, $[\langle s, t\rangle \mapsto (s \vee t)]$, $[\langle s, t\rangle \mapsto (s \to t)]$, $([s \mapsto \forall v\ s])_{v \in \mathrm{Var}}$ and $([s \mapsto \exists v\ s])_{v \in \mathrm{Var}}$ are all injective, and their ranges do not overlap each other or $\mathrm{Atoms}(\Sigma)$.

# Recursive definitions for terms

Since Terms($\Sigma$) has the Inductive and Injective properties, we have a version of the Recursion Theorem for terms, and can use it to give recursive definitions such as the following.

**Definition**

Vars is the function Terms($\Sigma$) $\to \mathcal{P}$ Var such that:

1. For any variable $v$, $Vars(v) = \{v\}$.
2. For any individual constant $c$, $Vars(c) = \varnothing$.
3. For any $n$-ary function symbol $f$ and terms $t_1, \ldots, t_n$, $Vars(f(t_1, \ldots, t_n)) = Vars(t_1) \cup \cdots \cup Vars(t_n)$.

# Recursive definitions for terms

**Definition**

For any variable $v$ and $t \in \text{Terms}(\Sigma)$, $[t/v]$ is the unique function $\text{Terms}(\Sigma) \to \text{Terms}(\Sigma)$ such that

1. $v[t/v] = t$
2. $u[t/v] = u$ when $u$ is a variable other than $v$.
3. $c[t/v] = c$ when $c$ is an individual constant.
4. $f(t_1, \ldots, t_n)[t/v] = f(t_1[t/v], \ldots, t_n[t/v])$.

(Again we follow the tradition of writing this sort of function in postfix position.)

# Recursive definitions for formulae, redux

A common pattern for defining a function from the set of formulae of a first-order signature $\Sigma$ to some domain is to first recursively define a helper function from terms, then use it to define a function for atomic formulae, and finally recursively extend that to all formulae. For example:

**Definition**

FV is the function $\mathcal{L}(\Sigma) \to \mathcal{P}\,\mathrm{Var}$ such that

1. For an atomic formula $P = R(t_1, \ldots, t_n)$, $FV(P) = \mathrm{Vars}\, t_1 \cup \cdots \cup \mathrm{Vars}\, t_n$.

# Recursive definitions for first order formulae

**Definition**

$[t/v] : L(\Sigma) \to L(\Sigma)$ is the unique function such that

1. For an atomic formula $P = R(t_1, \ldots, t_n)$, $P[t/v] = R(t_1[t/v], \ldots, t_n[t/v])$.
2. $(\neg P)[t/v] = \neg(P[t/v])$, ...
3. $\forall u P[t/v] = \forall u(P[t/v])$ if $u \neq v$ and $\forall u P$ if $u = v$
4. $\exists u P[t/v] = \exists u(P[t/v])$ if $u \neq v$ and $\exists u P$ if $u = v$.

# Substitution

# First-order languages with definite descriptions

Note: in a couple of weeks, we will introduce one more convenient expansion of first-order syntax, namely the addition of a *definite description operator*.

This is not crucial (any more than it's crucial to have function symbols rahter than predicates), but will help simplify some definitions relevant to the "definability" and "representability" of functions.