# Problem Set 6

Advanced Logic
23rd October 2022

In the following problems, Arith stands for the ``signature of the language of arithmetic'': recall that this has one individual constant `0`, one singulary function symbol `suc`, two binary function symbols `+` and `×`, and one binary predicate `≤`. See below for a reminder of the relevant definitions of $FV$ and $[t/v]$.

Note: problems 2--4 ask you to prove something about all terms and formulae of an arbitrary first-order signature $\Sigma$. You may, if you wish, just prove these claims for the special case where $\Sigma$ is Arith. This will lengthen the proofs a bit, but you may find it helpful if you're getting tripped up by notation like `$f(t_1,\ldots,t_n)$' and `$R(t_1,\ldots,t_n)$'.

1. (a) (30%) Prove that no *term* of the language of arithmetic (i.e., no member of Terms(Arith)) contains the character `a`.

   (b) (30%) Using the result of part (a), prove that no *formula* of the language of arithmetic (i.e., no member of $\mathcal{L}$(Arith)) contains the character `a`.

   a. By induction on the construction of terms.

   Base case: no variable is allowed to contain `a`, and the only individual constant of Arith, namely `0`, also does not contain `a`.

   Induction steps: if a term $t$ doesn't contain `a`, then also `suc(`$t$`)` doesn't contain `a` (since its elements are just those of $t$ together with `s`, `u`, `c`, `(`, and `)`). Similarly, if neither $t_1$ nor $t_2$ contains `a`, then nor do `+(`$t_1$`,`$t_2$`)` and `×(`$t_1$`,`$t_2$`)`, since their elements are just those of $t_1$ and $t_2$ together with `(`, `,`, `)`, and `+` or `×`.

   b. By induction on the construction of formulae.

   Base case: since `≤` is the only nonlogical predicate of Arith, the atomic formulae of Arith are all either of the form `=(`$t_1$`,`$t_2$`)` or `≤(`$t_1$`,`$t_2$`)` where $t_1$ and $t_2$ are terms of Arith. By the result of part (a), no such string contains the character `a`.

   Induction steps: if formulae $P$ and $Q$ do not contain `a`, then nor do `¬`$P$, `(`$P$`∧`$Q$`)`, `(`$P$`∨`$Q$`)`, `(`$P$`→`$Q$`)`, `∀`$v$ $P$, or `∃`$v$ $P$ (for any variable $v$).

2. (a) (15%) Prove that whenever $s,t \in$ Terms($\Sigma$) and $v \neq FV(s)$, $s[t/v] = s$.

   (b) (10%) Using the result of part (a), prove that whenever $P \in \mathcal{L}(\Sigma)$, $t \in$ Terms($\Sigma$), and $v \neq FV(P)$, $P[t/v] = P$.

   a. By induction on the construction of $s$.

   Base case: if $s$ is a variable, $FV(s) = \{s\}$, so if $v \neq FV(s)$, $v \neq s$ and so $s[t/v] = s$. If $s$ is an individual constant, we have $s[t/v] = s$ by definition.

   Induction step: suppose $s = f(s_1,\ldots,s_n)$ where each $s_i$ is such that if $v \neq FV(s_i)$, $s_i[t/v] = s_i$; and suppose $v \neq FV(s)$. Since $FV(s) = FV(s_1) \cup \cdots \cup FV(s_n)$, $v \neq$

$FV[s_i]$ for each $s_i$, so by the induction hypothesis $s_i[t/v] = s_i$. But then $s[t/v] = f(s_1[t/v],\ldots,s_n[t/v]) = f(s_1,\ldots,s_n) = s$.

b. By induction on the construction of $P$.

Base case: if $P$ is an atomic sentence $F(s_1,\ldots,s_n)$, then $FV(P) = FV(s_1) \cup \cdots \cup FV(s_n)$ and $P[t/v] = F(s_1[t/v],\ldots,s_n[t/v])$. So if $v \notin FV(P)$, $v \notin FV(s_i)$ for each $s_i$, so using the result of part a, we have $P[t/v] = F(s_1,\ldots,s_n) = P$.

Induction step: suppose whenever $v \notin FV(P), P[t/v] = P$ and likewise for $Q$. $FV(\neg P) = FV(P)$, so if $v \notin FV(\neg P)$, $v \notin FV(P)$, so $(\neg P)[t/v] = \neg(P[t/v]) = \neg P$. $FV(P \wedge Q) = FV(P) \cup FV(Q)$, so if $v \notin FV(P \wedge Q)$, $(P \wedge Q)[t/v] = P[t/v] \wedge Q[t/v] = P \wedge Q$; similarly for $\vee$ and $\rightarrow$. $FV(\forall uP) = FV(P) \setminus \{u\}$, so if $v \notin FV(\forall uP)$, either $v = u$ or $v \in FV(P)\setminus\{u\}$ or $v = u$. In the first case, $(\forall uP)[t/v] = \forall uP$ by the definition of substitution. In the latter case, $(\forall uP)[t/v] = \forall u(P[t/v]) = \forall uP$.

3. (a) (10%) Prove that whenever $s, t \in \text{Terms}(\Sigma)$ and $v \in FV(s)$, $FV(s[t/v]) = (FV(s) \setminus \{v\}) \cup FV(t)$.

   (b) (5%) Using the result of part (a), prove that whenever $P \in \mathcal{L}(\Sigma)$, $t \in \text{Terms}(\Sigma)$, and $v \in FV(P)$, $FV(P[t/v]) = (FV(P) \setminus \{v\}) \cup FV(t)$.

a. By induction on the construction of $s$, for fixed $v$ and $t$.

Base cases: if $v \in FV(s)$ and $s$ is a variable, we must have $s = v$ and hence $s[t/v] = t$, so $FV(s)\setminus\{v\} \cup FV(t) = (\{v\}\setminus\{v\}) \cup \{v\} = \{v\} = FV(s)$. It can't happen that $v \in FV(s)$ if $s$ is an individual constant since these have no free variables.

Induction step: suppose $s = f(s_1,\ldots,s_n)$, so that $s[t/v] = f(s_1[t/v],\ldots,s_n[t/v])$, $FV(s) = FV(s_1) \cup \cdots FV(s_n)$, and $FV(s[t/v]) = FV(s_1[t/v]) \cup \cdots FV(s_n[t/v])$; and suppose for induction that for each $s_i$, if $v \in FV(s_i)$ then $FV(s_i[t/v]) = (FV(s_i)\setminus\{v\}) \cup FV(t)$. If $v \notin FV(s_i)$ then $s_i[t/v] = s_i$ so $FV(s_i[t/v]) = FV(s_i) = FV(s_i)\setminus\{v\}$. If $v \in FV(s)$ then we must have $v \in FV(s_i)$ for at least one $s_i$. So, $FV(s[t/v]) = FV(s_1[t/v]) \cup \cdots \cup FV(s_n[t/v]) = (FV(s_1)\setminus\{v\} \cup \cdots \cup FV(s_n)\setminus\{v\}) \cup FV(t) = (FV(s)\setminus\{v\}) \cup FV(t)$.

b. Base case: for an atomic formula $P = F(s_1,\ldots,s_n)$, we have $P[t/v] = F(s_1[t/v],\ldots,s_n[t/v])$, $FV(P) = FV(s_1) \cup \cdots FV(s_n)$, and $FV(P[t/v]) = FV(s_1[t/v]) \cup \cdots FV(s_n[t/v])$. So we can derive that $FV(P[t/v]) = (FV(P)\setminus\{v\}) \cup FV(P)$ when $v \in FV(P)$ using the same reasoning as with the induction step in part a.

Induction step for negation: we have $(\neg P)[t/v] = \neg(P[t/v])$, $FV(\neg P) = FV(P)$, and $FV((\neg P)[t/v]) = FV(P[t/v])$, so if $P$ has the property that $FV(P[t/v] = (FV(P)\setminus\{v\}) \cup FV(t)$, so does $\neg P$.

Induction step for conjunction, disjunction, and conditional: similar, using the same reasoning as the induction step under part a.

The induction step for $\forall uQ$ uses the same reasoning again in the case whhere

$u \neq v$. In the case of $\forall v Q$, the hypothesis holds vacuously since $v \notin FV(\forall v Q)$. Similarly for $\exists$.

**Additional problems** (10% extra credit for successfully solving any one of these.)

1. Suppose that $v$ and $v'$ are distinct variables and $t$ and $t'$ are terms in which neither $v$ nor $v'$ occur free. Show that (a) for every term $s$, $s[t/v][t'/v'] = s[t'/v'][t/v]$, and (b) for every formula $P$, $P[t/v][t'/v'] = P[t'/v'][t/v]$.

a. By induction on the construction of $s$.

If $s$ is an individual constant or a variable distinct from both $v$ and $v'$, then $s[t/v] = s$ and $s[t'/v'] = s$, so $s[t/v][t'/v'] = s = s[t'/v'][t/v]$.

If $s$ is $v$, then $s[t/v] = t$, so $s[t/v][t'/v'] = t[t'/v'] = t$ (since $v'$ is not free in $t$). Also, $s[t'/v'] = s$, so $s[t'/v'][t/v] = s[t/v] = t$.

If $s$ is $v'$, the same reasoning goes through swapping $v$ and $v'$.

Induction step: suppose $s$ is $f(s_1, \ldots, s_n)$, where for each $s_i$, $s_i[t/v][t'/v'] = s_i[t'/v'][t/v]$. Then,

$$\begin{aligned} s[t/v][t'/v'] &= f(s_1[t/v], \ldots, s_n[t/v])[t'/v'] \\ &= f(s_1[t/v][t'/v'], \ldots, s_n[t/v][t'/v']) \\ &= f(s_1[t'/v'][t/v], \ldots, s_n[t'/v'][t/v]) \\ &= f(s_1[t'/v'], \ldots, s_n[t'/v'])[t/v] \\ &= s[t'/v'][t/v]. \end{aligned}$$

b. By induction on the construction of $P$.

If $P$ is an atomic formula $F(s_1, \ldots, s_n)$, then we can apply part a to get

$$\begin{aligned} P[t/v][t'/v'] &= F(s_1[t/v], \ldots, s_n[t/v])[t'/v'] \\ &= F(s_1[t/v][t'/v'], \ldots, s_n[t/v][t'/v']) \\ &= F(s_1[t'/v'][t/v], \ldots, s_n[t'/v'][t/v]) \\ &= F(s_1[t'/v'], \ldots, s_n[t'/v'])[t/v] \\ &= P[t'/v'][t/v]. \end{aligned}$$

If $P$ is $\neg Q$ where $Q[t/v][t'/v'] = Q[t'/v'][t/v]$, we have $P[t/v][t'/v'] = (\neg(Q[t/v]))[t'/v'] = \neg(Q[t/v][t'/v'] = \neg(Q[t'/v'][t/v]) = (\neg Q[t'/v')[t/v] = P[t'/v'][t/v])$.

If $P$ is $Q \wedge R$ where $Q[t/v][t'/v'] = Q[t'/v'][t/v]$ and $R[t/v][t'/v'] = R[t'/v'][t/v]$, we have $P[t/v][t'/v'] = (Q[t/v] \wedge R[t/v])[t'/v'] = Q[t/v][t'/v'] \wedge R[t/v][t'/v'] = Q[t'/v'][t/v] \wedge R[t'/v'][t/v] = (Q[t'/v'] \wedge R[t'/v'])[t/v] = P[t'/v'][t/v])$.

Similarly for $\vee$ and $\rightarrow$.

If $P$ is $\forall u Q$ where $u$ is distinct from both $v$ and $v'$ and $Q[t/v][t'/v'] = Q[t'/v'][t/v]$, we have $P[t/v][t'/v'] = \forall u Q[t/v][t'/v'] = \forall u Q[t'/v'][t/v] = P[t'/v'][t/v]$.

If $P$ is $\forall v Q$, we have $P[t/v] = P$, hence $P[t/v][t'/v'] = P[t'/v'] = \forall v(Q[t'/v'])$. Also $P[t'/v'] = \forall v(Q[t'/v'])$, so $P[t'/v'][t/v] = (\forall v(Q[t'/v']))[t/v] = \forall v(Q[t'/v']) = P[t/v][t'/v']$. The case where $P$ is $\forall v' Q$ is dealt with in the same way, swapping $v'$ for $v$.

Finally, the case where $P$ is an existential quantification is dealt with in the same way as universal quantification.

2. (Optional, ungraded) Define the "standard numeral" function $\langle\cdot\rangle : \mathbb{N} \to \text{Terms(Arith)}$ recursively as follows:

$$\langle 0 \rangle = 0$$
$$\langle \text{suc}\, n \rangle = \texttt{suc(}\langle n \rangle\texttt{)}$$

Suppose we have some funtion $g : \text{Var} \to \mathbb{N}$. Recursively define a function $d : \text{Terms(Arith)} \to \mathbb{N}$ such that $d(v) = g(v)$ for every variable $v$ and $d(\langle n \rangle) = n$ for every $n \in \mathbb{N}$. Prove that the function you defined meets these requirements.

Define $d$ as follows:

$$dv = gv \text{ when } v \text{ is a variable}$$
$$d\texttt{0} = 0$$
$$d(\texttt{suc(}t\texttt{)}) = \text{suc}(dt)$$
$$d(\texttt{+(}t_1\texttt{,}t_2\texttt{)}) = dt_1 + dt_2$$
$$d(\texttt{×(}t_1\texttt{,}t_2\texttt{)}) = dt_1 \times dt_2$$

It's immediate from this that $d$ agrees with $g$ on variables, so we just need to make sure that $d\langle n \rangle = n$ for all numbers $n$. We prove this by induction on $n$.

Base case: $d\langle 0 \rangle = d\texttt{0} = 0$.

Induction step: suppose $d\langle n \rangle = n$. Then, $d\langle \text{suc}\, n \rangle = d\texttt{suc(}\langle n \rangle\texttt{)} = \text{suc}\, d\langle n \rangle = \text{suc}\, n$.

3. (Optional, ungraded) Suppose that we have a first-order signature $\Sigma$ that is "well-behaved" in the following sense: no function symbol or predicate is an initial substring of any other funtion symbol or predicate, and no function symbol or predicate has a variable as an initial substring. Let PolishTerms($\Sigma$) (the set of "terms in Polish notation" over $\Sigma$) be the closure of Var under the family of functions $\langle s_1, \ldots, s_n \rangle \mapsto f \oplus t_1 \oplus \cdots \oplus t_n$ where $f$ is an $n$-ary function symbol of $\Sigma$. (Note that unlike our usual definition of term, this one does not use parentheses and commas.)

Prove unique readability for Polish terms: i.e. that whenever $s_1, \ldots, s_m, t_1, \ldots, t_n \in \text{PolishTerms}(\Sigma)$, $f$ is an $m$-ary function symbol of $\Sigma$, and $g$ is an $n$-ary function symbol of $\Sigma$, if $f s_1 \ldots s_m = g t_1 \ldots t_n$, then $f = g$ and $m = n$ and $s_1 = t_1$ and $\ldots$ and $s_m = t_m$.

Here I'm afraid an apology is in order. With just the properties I listed, the thing I asked you to prove isn't true. For exammple, we could have a signature with just a binary function symbol `fun` and a singulary function symbol `0`. Then the string `funx0x` can be parsed in two different ways: it could be the result of applying `fun` to the variables `x0` and `x`, or the result of applying `fun` to the variable `x` and the term `0x`.

To get this way of getting rid of parentheses to work, we can simultaneously change the definition of variables such that no variable is an initial substring of any other (For example we could require every variable to end with a space and contain no other spaces.) We can also add the condition that no function symbol is an initial substring of a variable. With these changes, we can prove that every Polish term is such that no Polish term has it as a proper initial substring or is a proper initial substring of it.

Base case: a variable or initial constant can't be an initial substring of any other variable or initial constant. And it also can't be an initial substring of a complex term $ft_1 \dots t_n$, since then either the $f$ would be an initial substring of it or it would be an initial substring of $f$, both of which are ruled out. For the same reason it can't have a complex term as an initial substring.

Induction step: suppose $ft_1 \dots t_n$ is either a proper initial substring of a Polish term $s$ or has $s$ as proper iniitial substring, where $t_1 \dots t_n$ have the given property. $s$ can't be a variable or individual constant, since then either $f$ would be an initial substring of it or vice versa. So, $s$ must be a complex term $gs_1 \dots s_m$. But then one of $f$ and $g$ is an initial substring of the other, which can only happen if $f = g$; so $m = n$. Suppose for contradiction that there's some $s_i$ that's distinct from $t_i$; let $s_i$ be the least such $i$. Then since $s_i \dots s_n = t_i \dots t_n$, one of $s_i$ and $t_i$ is a proper initial substring of the other, which is ruled out by the induction hypothesis on $s_i$.

**Definitions** The notations `FV' and `$[t/v]$' are used ambiguously for two different functions, one defined on Terms($\Sigma$) and the other on $\mathcal{L}(\Sigma)$. (Or we could think of them as standing for the union of those two functions.)

Where $\Sigma$ is a first-order signature with predicates $R_\Sigma$, function symbols $F_\Sigma$, and arity function $a_\Sigma$, the function $FV : \text{Terms}(\Sigma) \to \mathcal{P}(\text{Var})$ is defined recursively by:

$$\begin{aligned} FV(v) &= \{v\} && (\text{for } v \in \text{Var}) \\ FV(c) &= \emptyset && (\text{for } c \in F_\Sigma \text{ with } a_\Sigma(c) = 0) \\ FV(f(t_1, \dots, t_n)) &= FV(t_1) \cup \dots \cup FV(t_n) && (\text{for } f \in F_\Sigma \text{ with } a_\Sigma(f) = n > 0) \end{aligned}$$

The companion function $FV : \mathcal{L}(\Sigma) \to \mathcal{P}(\mathrm{Var})$ is defined recursively as follows:

$$\begin{aligned}
FV(F(t_1,\ldots,t_n)) &= FV(t_1) \cup \cdots \cup FV(t_n) && \text{(for } F \in R_\Sigma \text{ with } a_\Sigma(F) = n > 0\text{)}\\
FV(t_1 = t_2) &= FV(t_1) \cup FV(t_2)\\
FV(\neg P) &= FV(P)\\
FV(P \wedge Q) &= FV(P) \cup FV(Q)\\
FV(P \vee Q) &= FV(P) \cup FV(Q)\\
FV(P \to Q) &= FV(P) \cup FV(Q)\\
FV(\forall v P) &= FV(P) \setminus \{v\}\\
FV(\exists v P) &= FV(P) \setminus \{v\}
\end{aligned}$$

For any $t \in \mathrm{Terms}(\Sigma)$ and $v \in \mathrm{Var}$, the function $[t/v] : \mathrm{Terms}(\Sigma) \to \mathrm{Terms}(\Sigma)$ (written in postfix position) is defined recursively by

$$\begin{aligned}
u[t/v] &= \begin{cases} t & \text{if } u = v \\ u & \text{if } u \neq v \end{cases} && \text{(for } u \in \mathrm{Var}\text{)}\\
c[t/v] &= c && \text{(for } c \in F_\Sigma \text{ with } a_\Sigma(c) = 0\text{)}\\
f(t_1,\&,t_n)[t/v] &= f(t_1[t/v],\ldots,t_n[t/v]) && \text{(for } f \in F_\Sigma \text{ with } a_\Sigma(f) = n > 0\text{)}
\end{aligned}$$

The companion function $[t/v] : \mathcal{L}(\Sigma) \to \mathrm{Terms}(\Sigma)$ is defined recursively as follows:

$$\begin{aligned}
F(t_1,\ldots,t_n)[t/v] &= F(t_1[t/v],\ldots,t_n[t/v]) && \text{(for } F \in R_\Sigma \text{ with } a_\Sigma(F) = n > 0\text{)}\\
(\neg P)[t/v] &= \neg(P[t/v])\\
(P \wedge Q)[t/v] &= P[t/v] \wedge Q[t/v]\\
(P \vee Q)[t/v] &= P[t/v] \vee Q[t/v]\\
(P \to Q)[t/v] &= P[t/v] \to Q[t/v]\\
(\forall u P)[t/v] &= \begin{cases} \forall u P & \text{if } v = u \\ \forall u(P[t/v]) & \text{if } v \neq u \text{ and } (u \notin FV(t) \text{ or } v \notin FV(P)) \\ \text{undefined} & \text{if } v \neq u \text{ and } u \in FV(t) \text{ and } v \in FV(P) \end{cases}\\
(\exists u P)[t/v] &= \begin{cases} \exists u P & \text{if } v = u \\ \exists u(P[t/v]) & \text{if } v \neq u \text{ and } (u \notin FV(t) \text{ or } v \notin FV(P)) \\ \text{undefined} & \text{if } v \neq u \text{ and } u \in FV(t) \text{ and } v \in FV(P) \end{cases}
\end{aligned}$$