CS 512 Formal Methods, Spring 2014 — Instructor: Assaf Kfoury

First Order Definability

January 21, 2014 — Rawane Issa

(These lecture notes are **not** proofread and proof-checked by the instructor.)

Notational Conventions

Suppose that we have a relational structure *(think of a relational structure as being very similar to a relational database)* $\mathcal{M} := (M, \ldots)$ where M is the domain of definition of $\mathcal{M}$'s predicates (also called universe of $\mathcal{M}$). Now we would like to interpret φ in the structure $\mathcal{M}$. As mentioned in Handout 17, φ may contain free variables that we can interpret via an environment/look-up table *(the later naming convention explains this parallel we mentioned with relational databases)* l that would assign an of the domain of interpretation M to every free variable. If φ is closed (all its variables are bounded by quantifiers), we may omit l as all of our variables in this case are interpreted: remember that the quantifiers assign meaning to these variables by "searching" for interpretation in M.
For an arbitrary wff φ we may therfore write:

$$\mathcal{M}, l \models \varphi \tag{1}$$

if and only if φ is semantically valid in $\mathcal{M}$.
Now this notation is a little cumbersome as it does not explicitly show how we are assigning values to these free variables. Instead we will stick to one of the following conventions:

Conventions. Assuming φ has 3 free variables *(you may generalize this to any constant number of variables)*, in order to simplify our understanding of this look up table we may write:

$$\text{Convention 1. } M, a_1, a_2, a_3 \models \varphi(x_1, x_2, x_3) \tag{2}$$

Which means that l assigns to each free variable x_i an assignment a_i, i.e. $l(x_i) := a_i$.

We can further write the following if we so wish to do so:

$$\text{Convention 2. } M \models \varphi[a_1, a_2, a_3] \tag{3}$$

to say the same thing.

Note. You should keep in mind that these assignments a_i could very well be numerical assignments depending on M: if $M := \mathbb{N}$ then we could pick a_i to be say 10.

Definable Problems in FOL

Now that we have formally defined the syntax, the semantics and the overall conventions of FOL we would like to see what kind of meaningful problems we can express in this language. We will mainly be dealing in this section with notions such as "$<$" that you have probably used all your life. Of course we can generally express much more complicated things in FOL. But as we will soon see, even with such simple problems, the expressiveness of First Order Logic will draw us to the limits of not only this order of logic but also of what it is that we can answer in computer science and mathematics general.

In this section we will stick to dealing with the structure :

$$\mathcal{N} := (\mathbb{N}, \doteq, S^{\mathcal{N}}, O^{\mathcal{N}}) \tag{4}$$

where $S^{\mathcal{N}}$ is a unary successor function, $0^{\mathcal{N}}$ is a constant *(in terms of algebraic structures,* $O^{\mathcal{N}}$ *is the additive identity element of* $\mathcal{N}$*)* and $\doteq$ is the equality symbol.

Case of a finite subset of naturals

We can define any finite subset of naturals in FOL. Take $A = \{2, 3, 5\}$ for example:

$$\{2, 3, 5\} = \{n \in \mathbb{N} \mid \mathcal{N}, n \models (x = SS0 \vee x = SSS0 \vee x = SSSSS0)\} \tag{5}$$

Case of predecessor function

Take the following unary function:

$$pred(n) = \begin{cases} n-1 & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

which basically defines for any natural number n its predecessor $n - 1$ *(By convention we will take the predecessor of* 0 *to be* 0*.)*.

Notice that we can think of this function as a set of pairs $(n, pred(n))$ of inputs and outputs: in our case $\{(0, 0), (1, 0), (2, 1), (3, 2), (4, 3), \ldots\}$. The set formed by these pairs will represent what we call the graph of that function.

Rather than directly defining $pred(n)$, we will define $graph(pred(n))$ as this will be prove to be much easier to define in FOL (i.e. reduce the problem of defining $pred(n)$ in FOL to the problem of defining $graph(pred(n))$ in FOL). An important assumption that we are making is that we can fully and uniquely specify a function in terms of its graph. A proof of this is however not too difficult.The graph of $pred(n)$ can be defined as :

$$graph(pred) = \{(m, n) \in \mathbb{N} \times \mathbb{N} \mid \mathcal{N}, m, n \models (x = 0 \wedge y = 0) \vee (x = Sy)\} \tag{6}$$

The wff defining the set is only satisfied for pairs (x, y) which satisfy the property $pred(y) = x$. This function is therefore first order definable. (*Nomenclature: We call* m, n *witnesses.*)

Case of the "$<^{\mathcal{N}}$" operator

Now take the less than operator defined by the set

$$<^{\mathcal{N}} = \{(m,n) \in \mathbb{N} \times \mathbb{N} | m <^{\mathcal{N}} n\} \tag{7}$$

Is this is first order definable ?

Note that the expression written in (7) is not the one we are looking for: we are looking for a wff which only uses the operator defined for the structure $\mathcal{N}$ (4). In some sense, the structure $\mathcal{N}$ set the rules of the blocks which we can use in the game of finding a wff which defines what we'd like to express.

This question proves to be much harder to answer than the previously mentioned examples: $\mathcal{N}$ does not allow us to use the successor function an arbitrary number of times: for every specific hardwired pair of numbers, say $(4,5)$, we might be able to define $<^{\mathcal{N}} (4,5)$. But the problem now is that we will have to define this operator for each pair of numbers in $\mathbb{N}$ and there are infinitely many.

However if we extend the structure $\mathcal{N}$ with the addition operator "+", we just might be able to fix this issue. So take the structure:

$$\mathcal{N}_1 := (\mathbb{N}, =, S, +, 0) \tag{8}$$

Then:

$$\varphi_{<}(x,y) := \exists z\ (\neg(z=0) \wedge (x+z=y)) \tag{9}$$

Notice that the existential quantifier looks for a z in our domain $\mathbb{N}$ such that "$(\neg(z=0) \wedge (x+z=y))$" holds by first trying and checking if this formula holds for $S0$ (i.e. if "$(\neg(S0=0) \wedge (x+S0=y))$" holds), if not then trying it for $SS0$, if not then trying it for $SSS0$ and so on. In other words, for each pair (x,y) $\varphi_{<}(x,y)$ checks if such a z exists.

Case of the "$-^{\mathcal{N}}$" operator

Now lets see if subtraction is first order definable: We know that the minus is defined over the following domain and range:

$$- : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \tag{10}$$

i.e. the operator check if for a triple (m,n,p) the following holds "$m - n = p$". Now if we would like to look at this operator as in previous examples in terms of its graph then we will define:

$$-(m,n) = \begin{cases} m-n & \text{if } m \geq n \\ 0 & \text{if } m < n \end{cases}$$

(*Nomenclature: This is also called a monus function.*)
The graph of this function allows us to define the operator in the following manner:

$$\varphi_{-}(x,y,z) := (\varphi_{<}(x,y) \to z=0) \wedge (\neg\varphi_{<}(x,y) \to x=y+z) \tag{11}$$

Notice how we used our pre-defined wff $\varphi_{<}$: In general, its good practice to make our formulas modular as it allows what we write to become much more readable.

Further questions of Definability

Several questions arise from the examples we have seen:

Can we for example get rid of S and define it only in terms of $+$? The short answer is NO: How would we be able to generate any specific number using only addition and $0^{\mathcal{N}}$. We will need to define $1_{\mathcal{N}}$, the first non zero element (and even if we have 1, it is hard to show why this is possible).
The graph of the successor function will the look like this:

$$\varphi_s(x, y) := (x + 1 = y) \tag{12}$$

Provided that I have $1^{\mathcal{N}}$, I can get rid of S and get the following structure: $\mathcal{N}_1 := (\mathbb{N}, =, +, 0, 1)$.

can we get rid of the $1^{\mathcal{N}}$ all together? Notice that the only element for which $x + x = x$ is 0: $0 + 0 = 0$:

$$\varphi_0(x) := (x + x = x) \tag{13}$$

what about $\varphi_1(x)$? Can we design a FOL formula which defines the element 1 using only the building block found in $\mathcal{N}_1$? This questions proves to be quite the challenge. We really could not decided this question in class given the amount of time we have.

Is "+" definable from "S"? addition is defined as:

$$\forall m, n, p \in \mathbb{N}, m + n = \underbrace{S(\dots S(m) \dots)}_{n \text{ times}} = p \tag{14}$$

This is not valid syntax in FOL as we do not possess the necessary "blocks" needed in order to write down the fact that we need to apply the successor function an arbitrary number n of times (for a fixed n, say $10 + 2$, we can "hardwire" this value. But, again, we cannot simply hardwire all possible pairs values as there are infinitely many).

What happens if with extend $\mathcal{N}_1$ and add multiplication ? If we add multiplication to our structure, we gain in power and can express anything that a Turing Machine can (more formally any Turing computable function). Just remember that all what your PC needs is $+$ and $\times$.

Decidability and the limits of what is answerable

When we added multiplication into the picture, we actually reach certain limits of what we can answer: We cannot decide on whether or not certain wff are satisfiable or valid in structure which include multiplication (among other operators). The reason behind this is that we do not have a systematic way of checking if these wffs are satisfiable (and specifically checking for wffs expressing the membership of certain elements in sets). In other words we have no **algorithm** for checking if such wff are satisfiable. FOL is proving to be too powerful. Can we remove certain things from FOL and not have to encounter such problematic situations ? There are certain sets called "semi-decidable" and also called "recursively enumerable" for which we can enumerate all their members in systematic way. Now ultimately you could just enumerate all members of these sets and just output yes if you find the element you are looking for. But it is very possible for you to wait forever if the element turns out not to be in the set you are working with.

Now what if you were given a subset of the naturals and were told that both the set and its complement are semi-decidable (aka. recursively enumerable). Given these two facts, then we can actually decide membership in X by generating in parallel X and the complement of X: sooner or later our element will appear in one of the two lists in a finite amount of time.

Notation found in the Handouts

$n \uparrow p$ designates exponentiation, i.e. n^p, and $pr(x)$ designates checking if x is a prime number.