19 Expressing Mathematical Concepts (I)

Now that we have studied first-order logic at some depth we will extend our formal logic in order to be able to handle more serious mathematics with a formal proof system. For now, our approach will be conservative, that is we keep the language of first-order logic, designate a few predicates as special and provide axioms for these predicates.

The advantage of doing so is that we can use the same apparatus as before, that is first-order tableau, to prove theorems about equality, functions, and later algebraic structures, integers, and small algorithms. Instead of changing the logic we simply add axioms.

However, there is also a drawback. With an increasing number of axioms, formal reasoning becomes more and more complex, as the size of the formula to be fed to the proof system grows with the number of axioms provided, even if not all of them are actually used in the proof. Furthermore, many simple properties become astonishingly difficult to prove, as you first have to isolate and instantiate the axiom before you can use it as "reasoning rule".

At some point in the future we will, therefore, make many of the designated predicates primitives of our formal language and convert axioms into the corresponding proof rules. This will make the proof process much simpler and more elegant, but requires us to go over issues such as correctness, completeness, and compactness again.

For now, we will proceed by expressing mathematical concepts within the language of first order logic.

19.1 Equality

Equality is a binary predicate `Eq(_,_)` that comes with three axioms

```
ref:    (∀x) E(x,x)
sym:    (∀x,y) (E(x,y) ⊃ E(y,x))
trans:  (∀x,y,z) ((E(x,y) ∧ E(y,z)) ⊃ E(y,x))
```

A *model* for a set of axioms (see Smullyan page 49) is a universe U (or *domain* D) and an interpretation I of all the predicate symbols and parameters within that universe. If the set of axioms is finite, then we write this model as $\langle D,\ R^{P_1},\ldots,R^{P_n},\ c^{a_1},\ldots,c^{a_m}\rangle$, where R^{P_i} is the relation that interprets the predicate P_i and c^{a_j} is the element of D that interprets the parameter a_j. We do that only for the predicates and parameters that actually occur in the set of axioms.

Obviously there are many models of the equality predicate, not all of them being *standard models*, i.e. the models that one would conventionally have in mind. For instance, when we talk about the integers, $\langle \mathbb{Z}, x{=}y\rangle$ is the model of equality we have in mind, but $\langle \mathbb{Z}, x{=}y \texttt{ mod } 2\rangle$ is also a model. In fact, the above three axioms are insufficient to characterize an *equality*, as `(∀x,y) (E(x,y) ⊃ P(x) ⊃ P(y))` cannot be proven from the axioms.

The reason for this is that this formula isn't true in every model of the three axioms `ref`, `sym`, and `trans`, since they only describe an *equivalence relation* (just consider the model

$\langle \mathbb{Z}, x{=}y \texttt{ mod } 2\rangle$). What is missing is an axiom stating that we can replace equal for equal wherever we want. This is expressed by the so-called *substitution axiom*.

```
subst:  (∀x,y) (E(x,y) ⊃ P(..,x,..)  ⊃ P(..,y,..))
```

Actually, this axiom is not a pure axiom but an *axiom scheme* that needs to be instantiated for every predicate symbol that occurs in a formal *theory* (i.e. a set of formulas under consideration) and for every argument position in that predicate.

An important derived concept is the unique-existence operator

$(\exists!\texttt{x})P(\texttt{x}) \equiv (\exists\texttt{x})(P(\texttt{x}) \wedge (\forall\texttt{y})(P(\texttt{y}) \supset \texttt{E(x,y)}))$

where P stands for an arbitrary unary predicate. For n-ary predicates we can define this operator accordingly.

19.2 Functions

n-ary Functions can be described by (n+1)-ary predicates.

A unary function f, for instance is described by a predicate $\texttt{R}_f$, where $\texttt{R}_f\texttt{(x,y)}$ is supposed to express that $f\texttt{(x)=y}$. In order to ensure that the predicate does in fact represent a function we need to state two more axioms.

```
functionality:        (∀x)(∃!y) R_f(x,y)
functional equality:  (∀x,x',y,y')((E(x,x') ∧ R_f(x,y) ∧ R_f(x',y')) ⊃ E(y,y'))
```

These axioms have to be stated for every function symbol to be introduced. For n-ary functions, we have to state them for the appropriate (n+1)-ary predicate accordingly. Functional equality can be derived from substitution. For most function symbols we may want to give additional axioms characterizing their specific properties.

Most commonly we will deal with *binary operators*, usually written in infix format $\texttt{x}\circ\texttt{y}$. For some of these operators, we may want to require additional properties such as commutativity or associativity.

```
comm:   (∀x,y,z) (R∘(x,y,z) ⊃ R∘(y,x,z))
assoc:  (∀x,y,z,s,t,w) (R∘(x,y,s) ⊃ R∘(s,z,w) ⊃ R∘(y,z,t) ⊃ R∘(x,t,w))
```

Note that the commutativity axiom would usually be written as

```
(∀x,y,z,z) (R∘(x,y,z) ⊃ R∘(y,x,z') ⊃ E(z,z'))
```

because of `functionality`, however, that is equivalent to the shorter form given above.

Further axioms depend on what else we can state about the domain. We will revisit this issue once we have introduced axioms that describe, for instance, specific domains such as the integers or reals.

Using (n+1)-ary predicates instead of the conventional function notation makes writing formulas a bit awkward. From now on we will therefore write $f\texttt{(x)=y}$ instead of $\texttt{R}_f\texttt{(x,y)}$ and even use infix notation, where possible, but understand this as *notational abbreviation*.

19.3 Constants

Constants are best described by their effect on operators. The integer 0, for instance is known to be the neutral element of addition and the neutralizing one of multiplication.

After introducing the axioms for + and *, one could therefore characterize 0 by the axiom

```
zero:  (∀x)(x+0 = x  ∧  x*0 = 0)
```

Alternatively, if one wants to avoid designating and axiomatizing parameters, one may formulate an axiom stating the existence of a unique element with the desired properties.

```
zero-exists:  (∃!zero)(∀x)(x+zero = x  ∧  x*zero = zero)
```