A scheme for constructing mathematical objects

Suppose we have an idea for a new mathematical object, such as a new number system or geometry, and we want to make our idea concrete so that precise measurements or deductions can be made. How can we do it? One simple method is to build each element of our object out of *descriptions* of that element.

The general approach involves the following steps.

(1) Choose an alphabet of symbols, A, which is suitable for a language that describes the elements of the object. (A can be any set, such as the Latin alphabet I am using to type these letters, or the binary alphabet $\{0,1\}$, or the natural numbers $\mathbb{N}$, etc.)
(2) Construct a set S of 'words' in the alphabet which is sufficient to describe all elements of the object to be constructed. (Here S is a set of 'sensible' strings in the alphabet. These strings can be finite or infinite. 'Sensible' has no formal meaning, but intuitively means 'relevant to the construction'.) The strings in S represent the elements they 'describe'.
(3) Define an equivalence relation E on S, which relates two strings $s, t \in S$ if they describe the same element.
(4) Let the elements of the object be S/E.

For example, an alphabet for discussing real numbers could be the set

$$\{0,1,2,3,4,5,6,7,8,9,-,.\},$$

which consists of the 10 decimal digits along with symbols for negation and for the decimal point.

The set of 'sensible' strings in this alphabet is the set S of all infinite strings that have exactly one decimal point, and have either no negation symbols or have exactly one negation symbol at the beginning, and which do not begin with either -0 or 0. (Each such string represents a real number in the usual way.)

Now define an equivalence relation E on S which relates two strings if they are intended to represent the same real number. E should be defined more precisely, but for this example I only intend to give an example of strings that should be related: the string $.9999\ldots$ should be related by E to the string $1.000\ldots$.

Finally, $\mathbb{R}$ *could* be defined on the set S/E of equivalence classes of sensible strings. (The conventional description of $\mathbb{R}$ is a little different, but is based on the same ideas.)

The conventional construction of $\mathbb{Z}$

(1) Alphabet $= \mathbb{N} = \{0,1,2,\ldots\}$.
(2) 'Sensible' strings $= \mathbb{N} \times \mathbb{N} = \{(m,n) \mid m,n \in \mathbb{N}\}$. (Think: $(m,n) \leftrightarrow m-n$.)
(3) $E = \{((k,\ell),(m,n)) \in (\mathbb{N}\times\mathbb{N})^2 \mid k+n = m+\ell\}$.
(4) $\mathbb{Z} = (\mathbb{N}\times\mathbb{N})/E$.

Need to check: E really is an equivalence relation.

Notation: Write $m_{\mathbb{N}}$ for the element $m \in \mathbb{N}$, write $m_{\mathbb{Z}}$ for $[(m,0)]_E \in \mathbb{Z}$, write $-m_{\mathbb{Z}}$ for $[(0,m)]_E \in \mathbb{Z}$.

Need to check: the natural numbers can be identified with the nonnegative integers, and that under this identification $\mathbb{Z} = -\mathbb{N} \cup \mathbb{N} = \{\ldots, -1_{\mathbb{Z}}, 0_{\mathbb{Z}}, 1_{\mathbb{Z}}, 2_{\mathbb{Z}}, \ldots\}$.

Adding operations and relations to $\mathbb{Z}$

(1) $-[(k,\ell)]_E = [(\ell,k)]_E$.
(2) $[(k,\ell)]_E + [(m,n)]_E = [(k+m, \ell+n)]_E$.
(3) $[(k,\ell)]_E \cdot [(m,n)]_E = [(km+\ell n, kn+\ell m)]_E$.
(4) $[(k,\ell)]_E < [(m,n)]_E$ in $\mathbb{Z}$ iff $k+n < m+\ell$ in $\mathbb{N}$.

Need to check: these operations and relation are *well-defined*. (E.g., if $[(k,\ell)]_E = [(m,n)]_E$, then $-[(k,\ell)]_E = -[(m,n)]_E$. Etc.)

Need to check: the usual rules of arithmetic hold. (E.g., show that $x+(y+z) = (x+y)+z$ for all $x, y, z \in \mathbb{Z}$. Etc.)

The conventional construction of $\mathbb{Q}$

(1) Alphabet $= \mathbb{Z}$.
(2) 'Sensible' strings $= S = \{(p,q) \in \mathbb{Z} \times \mathbb{Z} \mid q \neq 0\}$. (Think: $(p,q) \leftrightarrow p/q$.)
(3) $E = \{((p,q),(r,s)) \in S \times S \mid ps = rq\}$.
(4) $\mathbb{Q} = S/E$.

Need to check: E really is an equivalence relation.

Operations are defined the way you learned in grade school. There are lots of things to check!