Language Diversity in Ordered Pairs

by Sven Nilsen, 2023

In this paper I discuss ordered pairs and their diverse properties from a language perspective.

Legend says that Carl Friedrich Gauss, one of the most famous mathematicians in history, found a quick formula for calculating the sum of the first `n` natural numbers:

$$\sum i\ [0, n)\ \{ n \} = n \cdot (n + 1) / 2$$

It happens that this formula also counts the number of ordered pairs `(a, b)` where `a < b` in a discrete space for ordered pairs which dimension is `n`.

A discrete space is an abstract mathematical object that can translate back and forth between some data structure and natural numbers, in a way such that all the natural numbers below some limit are "occupied". Discrete spaces are useful because they allow us to ignore the data structure when reasoning about some problem from a perspective of computational complexity. In discrete combinatorics, one composes new discrete spaces by combining simpler discrete spaces. This means, the output is a set of algorithms that "know" how to translate a seemingly difficult problem in terms of a data structure, into another problem who talks about the natural numbers.

The dimension of the discrete space is some information that the algorithm for the discrete space needs to perform a translation into natural numbers or back. However, when referring to a discrete space in general, one does not mention the dimension, e.g. "ordered pair", "unordered pair", "permutation", "power set" etc.

Therefore, although the sum of the first `n` natural numbers has a simple formula, one can change the perspective of that formula and use it in discrete combinatorics. When an idea can be translated this way using mathematics, one talks about "language diversity". You can not translate every idea the same way. There are only some particular ideas that under special circumstances that can be translated. Understanding language diversity is important because it helps to recognize how to solve seemingly difficult problems.

For example, one challenge in automated theorem proving is to collect and organize mathematical knowledge such that the solver can take shortcuts and reduce the lengths of proofs. In a such context, partial evaluation is important to avoid combinatorial explosion. The mathematical knowledge must be organized in a knowledge base to exploit common techniques that the solver can use. There is no such thing as a solver who can approach every kind of problem efficiently. Feeding a mathematical knowledge base with new knowledge as experience is gained from using the solver, is necessary because the solver itself can often not distinguish useful from useless knowledge. The knowledge format needs to reflect what is efficient common usage.

Gauss' formula for series sums is intuitive and it looks simple. Intuition has an advantage. However, in other circumstances, there are other properties than intuition that matters, e.g. in automated theorem proving. There is another formula that is useful too:

$$n \cdot (n + 1) / 2 = ((2 \cdot n - 1)^2 - 1) / 8$$

A solver, e.g. Poi, might easily prove the left side from the right side, but not vice versa.

The reason the second formula is useful is because it is easier to invert for an automated theorem prover. Some formulas are difficult to invert, typically when the same variable is used more than once. However, you can usually not program this knowledge directly into a solver's knowledge format. Instead, the knowledge format needs simple langauge rules such that it can take advantage of future exploitation of new mathematical knowledge. With other words, even though the solver does not understand why it has certain rules, it can do its job.

Humans like to think that when designing a mathematical language, there ought to be as few rules (axioms) as possible and as much as possible should follow from these few rules. All rules who assist in theorem proving ought to be provable within the theory derived from the small set of axioms. When there is a lot of such assisting rules, the language is equipped with mechanisms to organize code and share the proofs. However, this leads to a new problem, where people have to agree upon which library packages to use for theorem proving. Even worse, when there are multiple languages for theorem proving with different packages, people have to redo their own proofs when switching language. Again, this is a problem when some solvers are good at some problems, but not all. People are forced to switch language when they work on similar problems, but for different domains where different solvers are good.

The only way to satisfy this design problem is to move on to a higher level of thinking about mathematics. In this higher level of thinking, it is not the axioms which are important. You just have to accept that some solvers will be good at solving some problems without needing to be perfect on all kinds of problems. So, how can knowledge be transferred from one domain to another without losing semantics?

The solution is to fix the symbols. If two symbols are the same symbol, then its meaning is also determined from the symbol. So, the two meanings of the two symbols are the same meaning. Still, the meaning of a symbol is not "true" if a symbol is "true". It has a different truth value independent on the symbolic truth value. Only the equality of meaning is depending on the equality of the symbols. This is not a tautology in logic, so one needs to introduce it as an axiom. This is why Path Semantics has a "core axiom". In summary: The semantics of symbols is not tautological in logic and must be introduced.

Much of modern mathematics is about figuring out how to think about equality and equivalences. For different mathematical languages, there are different notions of equality or equivalences. In First Order Logic, a predicate often has the property that it has normal congruence.

This means, if `a == b`, then `p(a) == p(b)`. However, by default, First Order Logic treats `a` and `b` differently than `p(a)` and `p(b)`, because the arguments to predicates are symbols, while the output of predicates are propositions. This means that the two `==` signs are overloaded. They operate on different types, so there is actually two of them: `=sym=` and `=prop=`. Expanded, one can write: If `a =sym= b`, then `p(a) =prop= p(b)`.

The moment on introduces an operator with tautological congruence, the whole language of predicates in First Order Logic goes out the window. If `(a =sym= b)^true`, then `p(a) =prop= p(b)`. There are such mathematical languages, so First Order Logic is not suitable for all mathematics.

You can see how the pluraverse of mathematical languages makes it hard to make sure that semantics is preserved when transferring knowledge from one domain into another. Basic things, like how two objects are treated as equal or not, varies from language to language. How can one transfer knowledge given such unfavourable conditions? The answer is usually that one can not do this perfectly. However, in some cases one can and in a subset of these situations can one reason a little about it. In order to reach these situations, one has to study language diversity.