Algebraic Sized Type Constructors

by Sven Nilsen, 2020

In this paper I provide a recursive way of constructing every algebraic sentence in Sized Type Theory by grammar rules such that the sentence is guaranteed to have a certain size.

|r| = n
|s| = 1
k(a) = why(∃ x { |x| == a })

nat	Constructors ∀ x, y, z	Examples
0	⊥, (r, x),	(⊥, ⊥ + ⊥), (23, ⊥), ((⊥, ⊥) + ⊥)
1	false, true, 0, 1, 2, …,	(3, ⊥), (true + ⊥), (false + (⊥, 24))
2	bool, (s + s),	bool, (bool + ⊥), (1 + 2), (bool, 5) animal ~:= dog + cat
3	-	(3 + 2 + 1), (3 + 2 + false)
4	(k(2) + k(2)), (k(2), k(2)),	(bool + bool), (bool, bool), (1 + 2, 3 + 4) (animal, animal), (animal + dog + cat)
5	-	(bool + bool + 14), ((bool, bool) + 14)
6	(k(2) + k(2) + k(2)), (k(3) + k(3)), (k(2), k(3)),	(bool + bool + bool) ((3 + 2 + 1) + (3 + 2 + 1)) (bool, (bool + 8))
7	-	(9 + bool + bool + bool) (0 + 1 + 2 + 3 + 4 + 5 + 6) ((0, 1) + (0, 2) + (0, 3) + (0, 4) + (0, 5) + (0, 6))
n	(⊥ + r), (s, r), x ~:= r defines new symbols	-
n+1	(r + s),	-

Some observations:

- Redundant rules by commutation are left out
- Notice that primes, except `2`, are covered by existing rules
- Rules for composite numbers can be derived by prime factorization and number theory
- One sentence might be constructed multiple ways since rules are not necessarily exclusive
- Recursive constructors might be thought of as a greedy algorithm given some cost of recursion

Two types are isomorphic if they have the same size, so these rules can be used to prove stuff like this:

∵	∀ x, y, i { ¬∃ f { x == (bool + 3 + z) ∧ y == bool => f : x → → y }	conjecture
∴	∀ z { |(bool + 3 + z)| >= 3 } ∧ |bool| == 2	why
∴	∀ z { |(bool + 3 + z)| ¬= |bool| }	Q.E.D.