When Mathematics met Programming: An introduction to Coq.

Piyush Kurur

Often it makes sense to talk about types which depend on values from another type. For example consider the type of lists parameterised by their lengths. In a dependently typed programming language, you can talk about the type of "lists of length say n" for every n of type Nat (the type of natural number). In such languages you can express constraints like the head function expects non-empty lists as its input and the compiler will ensure that every call of head is on a non-empty list (at compile time). The benefits of such checks are enormous if we care about correctness of programming. It turns out that type checking of such a language can double up as proof checking of mathematical proofs involving predicates. Thus a dependently typed language can double up a proof assistant.

In this talk I will introduce you to the proof assistant Coq. We will look at both dependently typed programming in Gallina (the dependently typed language underlying Coq) and proving theorems in Coq using the tactic language.