Formalizing the Meaning of Solving an Equation for One Variable

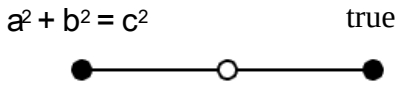
by Sven Nilsen, 2018

Since ancient times, equations have been used in mathematics to model problems, where solving it often means solving the equation for one variable. However, this approach does not automatically carry over to the semantics of automated theorem provers, because in addition to the equation, a goal must be specified such that the theorem prover can understand which variable to solve the equation for. In such systems, the meaning of solving an equation is implicitly specified in the code of the theorem prover, but not formally specified in the language of mathematics. In this paper I represent a formal semantics of solving an equation for one variable.

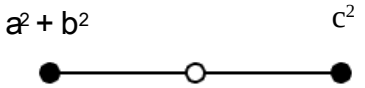
For example, consider the classical Pythagoras equation of real numbers:

$$a^2 + b^2 = c^2$$

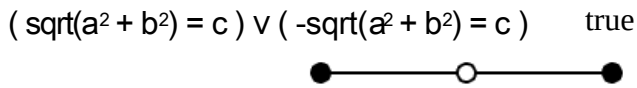
In an automatic theorem prover, the equation above can be expressed through a Havox diagram that keeps track of assumptions of equality and inequality between expressions. When the problem of solving an equation is specified, this is equal to first assuming that the equation is true:



When an equation is true, it means that one side is identical to the other:



This is how far an automated theorem prover gets without having automated tactics for this particular kind of equation. For example, when `c²` is identical to `x`, it means that `c` is identical to either `sqrt(x)` or `-sqrt(x)`. Most automated theorem provers can not work with higher order knowledge about identity from assumptions, so one must use the following expression instead:



This is expression is necessary because a constraint on `c` might added such that only one case is true. From this one can easily see that solving an equation for one variable might not be an obvious procedure. There is no guarantee that a theorem prover will stumble upon the solution by itself.

In order to specify the semantics of solving an equation for one variable, one needs a technique which allows a step directly from the equation to the goal. This knowledge is contained within the automatic theorem prover while solving the problem. Even if it fails, it is known what kind of problem it was working on.

This is where the language of path semantics comes to rescue. It turns out that only one extra criteria is necessary to specify the goal:

$a^2 + b^2 = c^2$	Assuming equation is true
$c = f(a, b)$	Assuming equation is true

The `c` is assumed to be identical to an unknown function `f` when given input to `(a, b)`.

However, for a satisfiability solver this does not work, because it is not sufficient to find some value of `a` and `b` that satisfies the two equations. There must exist a function `f` for all values of `(a, b)`! In first order logic this is written:

$$\forall\ a, b\ \{\ \exists\ f\ \{\ c = f(a, b)\ \}\ \}$$

For a satisfiability solver to prove this, one must use the following trick:

$a^2 + b^2 = c^2$	Assuming equation is true
$c = f(a, b)$	Assuming equation is false

Then, one proves non-satisfiability instead of satisfiability. This happens in addition to proving satisfiability when the second equation is assumed to be false. By proof of contradiction, there exists no values of `a` and `b` such that a function `f` does not map to `c`. Therefore, the function `f` maps to `c` for all values of `a` and `b` (that satisfies the other equations).

The problem with this definition is that a simple solver can usually not reason over higher order assumptions. For example, the identity of `f` might not be preserved between the two proofs. It might try one version that works when assuming the equation is true, and another version when assuming the equation is false. In fact, selecting an arbitrary `f` when assuming the equation is false will make this technique unsound.

Therefore, `f` must be chosen before running the solver. This means that solving an equation for one variable in general either requires reasoning about higher order assumptions, or it requires searching through the function space of the type of `f`. This puts a lower bound on the complexity of solving for one variable. This shows that solving an arbitrary equation for one variable under this definition is NP-hard.