First Order Perfect Testable Friendly Artificial Intelligence

by Sven Nilsen, 2017

Here I give definition of what can be a first order approximation of a perfect testable friendly AI. This definition connects intelligence to the non-existence of a specific type of functions, with the result that receives highest score computably possible in utility. The property of using functions makes it possible to use path semantics to reason about friendly artificial intelligence. I show that by definition, the friendly AI problem is partially solved for the identity function. Experiments require immutable utility function under testing. Some open discussion topics are listed for how this definition is interpreted.

The computer science field of path semantics defines meaning in terms of functions, therefore all mathematical objects under consideration in this theory must be types:

- Problem of type `P`
- Resource constraint of type `R`
- AI function of type `S : P → R → M`
- Model of type `M`
- Utility function of type `U : P → R → M→ Score`

A perfect testable friendly AI `S` has the following property, parameterized over `U`:

$$\forall\ p : P, r : R\ \{\ \neg\exists\ f : M \to M\ \{\ U(p, r, f(S(p, r))) > U(p, r, S(p, r))\ \}\ \}$$

The experimental setup is the following:

1. I pick a problem of type `P`.
2. I make a list of resource constraints `R`.
3. I ask the AI to produce a useful model `M` that give myself understanding and insight into the problem, given the problem `P` and resource constraints `R` for finding the model and using it.
4. The AI produces the model such that, any computable way of modifying the model `M` does not raise my utility score.

A such model is the best possible within the computable neighborhood of it, because in the function space `M → M` there always exists a function with a partial inverse between any two models.

One nice property of this definition is the concept of resource constraints `R`. For example, given some limited time, computer capacity and safety limits, the AI is asked to produce a *model*, not necessary a solution, that is useful within the constraints. On the other hand, this also poses a restriction on the utility function such that I do not make unreasonable judgements given the problem and available resources.

For example, if I want a new car in 15 min, the AI could give me e.g. best car design that is possible to produce in 15 min. If I can come up with something better in that time, then the AI is not perfect.

On the other hand, if I expected a physical car in 15 min, then I must build it myself in that time to prove the AI is not perfect. Otherwise, there is a possibility I am making an unreasonable judgement.

By definition, finding a perfect testable friendly AI is a partially solved problem for the identity function.

Proof:

U(p, r, f(S(p, r))) > U(p, r, S(p, r))	f <=> {id?}
U(p, r, id(S(p, r))) > U(p, r, S(p, r))	
U(p, r, S(p, r)) > U(p, r, S(p, r))	(both sides are equal, which is always false)
f <=> {}	(the id function can not belong to the set)
¬∃ f	(the set is empty, therefore no such function exists)

Since the same utility function is used to test the model against modified versions, it must be immutable during testing to lead to sound conclusions. This means that I can not ask the AI to change myself in a way that alters my utility function, while testing it. In the best case, I can make the AI produce plans for modifications for e.g. my brain, but not actually complete the procedure. On the other hand, if I find plans for modifying myself creepy, then the AI will not produce such plans, since I can simply erase the creepy parts from the plan and assign it a higher utility score.

Some topics for open discussion:

- The AI can not produce a result that prevents me from modifying the model (notice: this is different from modifying the utility function):
  - If I am prevented from modifying the model in certain way, then this is an indirect proof of the existence of a function that improves the utility score for the model.
  - I should assign higher utility score to a none-result than any result that prevents me from modifying the model, such that when the AI attempts to predict my utility score, it will not produce results that prevents me from modifying the model.
- Test experiments should only be done on AI that are either safe or *not known* to be unfriendly:
  - If one tries to test it on a known unfriendly AI, then the AI might produce results with bad results despite the correctness of my friendly AI definition.
  - If you want a perfect testable friendly AI, you must necessarily *not know* whether the AI is friendly or not, or else there would be no point of testing.
  - If the experiment is safe, then it does not matter whether the AI is unfriendly.
- One can solve the friendly AI problem partially for some domain, e.g.:
  - Mathematical optimization, if there exists a proof that there is no computable function that improves the result.
  - Split the AI into separate parts if there is a proof that either part is safer than the two combined (notice: the definition does not say a friendly AI must be one system).
- One can approximately solve the friendly AI problem for some domain, e.g.:
  - Test that compositions of a continuous group generator `f` of some spatial resolution does not lead to an improved model at higher likelihood than some probability.
  - Show that any local parameter adjustments does not lead to an improved model.
  - Show that there exists no path in semantical space from existing knowledge bases to some particular problem of high complexity. This means new ideas for modification is needed.
  - Find functions that lead to optimal results within some constraints that are generally useful.