Constructive Utility Functions

by Sven Nilsen, 2017

Here I represent a fundamentally different concept than traditional outcome-based utilities. Instead of inferring utility of an action from the goal, the utility of the goal is deduced from the action. This new idea might be used for agents in environments where there are intrinsically good or bad actions.

The basic premise of constructive utility functions is that all actions have some degree of intrinsic utility. No action is considered neutral.

If an action is depending on context, then the utility of that action is a higher order function. The arguments of the higher order utility function is the part of the context that is relevant for deciding the utility.

To know how to connect context to the utility function, the arguments could be of a similar kind as current objects in Dyon. The mathematics of such functions are part of non-deterministic path semantics, but behaves deterministically in a known context.

$f : A \to B$	action \`A\` produces some change \`B\` when executed by \`f\`
$g : A \sim C \to U$	utility \`U\` of \`A\` in context \`C\`

The \`~\` symbols separates current objects (context) from arguments. It tells us that the utility function \`g\` is generally thought of as a function of type \`A → U\`, but in a non-deterministic way. The precise variation of utility depends on some unique type \`C\` or a name \`c\` of a current object, such that if there are two utility functions for two different kind of actions, they can be compared to each other to see whether they depend on the same context.

This makes it clear that no unknown action has a well defined utility. For example, if a machine generates an arbitrary random program and runs it on a computer, it is per definition unsafe behavior for an agent since it does not know what the consequences are for running the program. On the other hand, if all possible instructions in the randomly generated program is assigned a utility, one might analyze the source code of the generated program and infer the expected utility based on the state of the world.

It is worth noticing that constructive utility function are cheaper to compute than simulating the world and predict outcomes. This could explain why humans tend to divide actions into good and bad categories. For example, the ethics of killing can mean different things to people:

g_0(kill) ~ war, self_defence = if self_defence { -0.1 } else if war { -0.3 } else { -1 }
g_1(kill) = -1

A utility does not need to be a scalar. For example, to prevent murder even when the reward is high, one can use a 2D vector that is ordered by components.

Ordinary math functions appear in too many contexts to have an assigned utility. A way to work around this problem could be to guard programs with predictions of what they mean in the real world.