The 12 Labors of Heracles

Jack Lucas

March 3, 2019

1 Overview

We imagine a stateless composition of data able to express a determinable AI, which can not only explain it's reasoning but also make mistakes and operate off of false information. Actor's reason only from what they know, and are able to "gossip" information among themselves. The prime goal of all actors is to unify their state with their desired goal using the same abilities at the player's disposal.

On the satisfaction of a goal we express the following simple rule:

> *A goal is satisfied if that goal is included in the state of the actor asking if the goal is satisfied.*

We imagine a set S which defines an abstract data structure for storing unique game data. An example of S is the set of sets

$$S\{QRE\}$$

for which the sets QRE hold a "type" of game data; *i.e. places, item locations, npcs, etc.*

For a set of sets we may imagine a standard mapping function which takes a functor 'f' and applies it to each value of the set. (FIX: Collasal mistake of using notation like it's for application when it's for composition)

$$f \cdot S\{QRE\} = S\{f(Q) \dots f(R) \dots f(E)\}$$

or in standard notation we might say:

$$\forall s \in S, \quad f(s)$$

We further define that the functor concatenated to the negation of the functor is equivalent to the original set.

$$f \cdot \neg f(S_1) = S_1$$

or,

$$\forall X \in S_1, \quad \forall x \in X, \quad f \cdot \neg f(x) = S_1$$

and,

$$S_1 - f \cdot \neg f(S_1) = \varnothing$$

However, this is nonsense and you've been lied to. We cannot propose a function 'f' which would operate on all unique game data in a sane way, although it would be technically possible if 'f' could determine the types of it's own arguments. In this regard, we solve the problem by proposing a 'typed relational functor' in place of 'f' such that an 'f' exists for Q, a 'g' exists for R, and so on, dependant on their types. (The relational will come into play shortly).

For this a new set P is determined holding our typed functors,

$$P\{f \dots g\}$$

These new functors retain the property of creating a 1:1 mapping between value and result. In other

words, they are referentially transparent (*pure*) functions.

Since we have the concept of types we can now express an interleaving between two sets, in an unorthodox notation where a set is treated as a functor itself.

$$P\{fgh\} \cdot S_1 = S_1\{f(Q) \dots g(R) \dots h(E)\}$$

Or in simpler terms,

$$P \cdot S_1$$

We remind that the following truth holds where,

$$S_1 - P \cdot \neg P(S_1) = \varnothing$$

For a last point we relate that P is a relational functor in that any logic retained within P can operate with knowledge of any other value within S, such that a subset of S is always potentially within the scope of any of the functors *f, g, h, etc* mapping over the sets of S. It is not allowed however, for one subset to change the values within another subset.

If organized under this manner we can now view our game world as not one world, but as the relations between two worlds. Or in another manner of speaking since two worlds exist and since our functions are referentially transparent, we can potentially say

$$S_1 = \neg P \cdot S_2$$

Another way of viewing this is to say that all values contained within the set S are now viewed as constants, and since S is the value of itself according to time, and a constant is not effected by time, S is equal to nothing. This is of course not true within the computer, but can be viewed as such when we realize that the equation

$$S_1 - P \cdot S_1 = P$$

holds if both worlds exist. In reality we would not end up with a set P, but rather the difference between sets which would encode the actual value of the mapping between worlds.

Since the difference between the worlds is contained within our Functor set, we can view the entire world as an application of functions rather than a change in state. This is as far as I can tell, a shoddy but primitive attempt at describing the lens through which we must view a purely functional game engine.

The real question is how can this property hold true with an AI?

2 A.I.

We imagine again that the set E contains within it an opaque structure describing *1-n* NPC's. Concurrently, we also describe the set Q as a structure containing "facts" about the game world; a database in a vague sense. This database should remain as static as possible.

Every NPC is given partial information of this database within itself. Such that on creation we might imagine for an NPC (N ...Nn) and a database D we end up with the equation. Where "+" stands for the addition of state to the original N.

$$Q\{N_0 + D/n ... N_n + D/n\}$$

That is each NPC has a different view of the world. Later we will discuss the components of NPCs, but for now we must touch on one quick point.

Another important aspect of the system is that the actions a player can take are unified to the npc's decision system. That is, every action a player can take, an npc makes in the same way. From a programming point of view this means every function for actions that describes a change between World 1 and World 2 must be the same functions called by

the input code handling player choices. This is important to retain the consistency of the systems ability to plan decisions, as the player will also take part in influencing the AI's ability to learn new facts.

The majority of the rest of the explanations will be examples on how we handle decision making.

3 An Emotion Engine for Finite State-Spaces

Classic texts on Artificial Intelligence describe a basic theory for understanding the computation of simulated reasoning in general domains. They also adequately bemoan with rightful agony; the lack of true reasoning in any domain the theory is actualized on due to the inability to think outside the system itself. Instances of this arise in "*Escher, Godel, Bach*" with the Mu Problem and in deliberately unsolveable puzzles. The lack found in both, is the inability to reach a level of thought that exists outside the system. "*Escher, Godel, Bach*" described this as a "strange loop", and I propose it be included in a more general system of meta-circular thought. Meta-Circular thought is a specilization on the general concept of Meta-Circular philosophy, and describes the process by which we trampoline between radically different contexts in order to gain supplementary perspective.[1]

Consider the following case where I ask you to take the word "*Bacon*" and rearrange it into the word "*Cabron*". Our rule list for making a move is as follows.

1. A letter can only swap spaces with another letter, and only once. (ABC -> CBA but now C or A cannot be moved).
2. An "R" can be generated for free, and only once, if a "Ca" exists in the seqeuence. It must be appended to the sequence. (Ca -> Car).

The process for attempting to solve the problem may look something like this...

1. Evaluate W ("Bacon").
2. Evaluate L ("Cabron").
3. Apply Rule 1 to "B" and "c" (Bacon -> Cabon).
4. Apply Rule 2 (Cabon -> Carbon).

At this point we are immedietely thrust outside the system of rules we have created. We can't move "b" again so we are unable to perform the last swap we need to satisfy the goal. We may imagine we can perform other swaps with Rule 1 but after a few feeble attempts we will eventually "step outside" the system and evaluate the problem from a higher level. Such that we will understand the set of rules cannot be unified to the problem to produce the desired goal.

So then we must recognize that any A.I. designed for our NPC's will be inheirently lacking in it's ability to reason outside the system. Because of this we propose a trifecta in the form of the Database, the Emotion Engine, and the state machine properties of Actors. The goal is not to produce true reasoning, but to simulate reasoning in such a way that an NPC could potentially beat the level before the player does. We also want to step away from randomization in the sense that an NPC is simply handed a random goal or property that he temporary assumes. The real magic is in creating a deterministic system that can be explained logically such that the player does not feel cheated or under the burden of ingenuity.

[1] There is of course much more to this than what has been stated but that is for another time.

3.1 State-Space Definition

We introduce the standard notation for state-space problems such that any goal in our currently desired domain can be expressed as the set M = {S, F, G}. Where:

1. S = A set containing our current state.
2. F = A set containing functions that can mutate our state.
3. G = A set containing our desired goals.

All at once we present several factors that form a ideogrammatic method for the design of our A.I. system. TODO: Expand

3.2 A Special Set F

Firstly, we restate our intended goal for NPC A.I. which is actors who can operate off of partial information in two forms.

1. For some database D that contains all facts about the world, a subset of D exists in an actor A. Such that A contains only partial information about that which can be known.
2. For some actor A that contains partial information, A can learn new information from members of D. Information from D exists in the form, *ancedent:consequent* for which it is intuitive to give examples.
   (a) John has money
   . . . . . [*f (John, Money)*]
   (b) Lucy lives at Mullvad Square
   . . . . . [*f (Lucy, Mullvad Square)*].
   (c) The area south of Mullvad Square is Presberty Lane
   . . . . [*f (Mullvad Square, South, Presberty Lane)*]

   From this it can also be reasoned that NORTH of Presberty Lane is Mullvad Square based on the negation of a function defining South.

   From this we say A can be given partial information. For instance A may know someone has money in the form: *f (X, Money)* but not know who. Note that the database D will know exactly all values for which X is true. A may know that something is south of Mullvad Square, *f (Mullvad Square, south, X)* for which D knows exactly what is south of Mullvad Square.

In this way the only requirement of our system is that all functions f express the atribute of being relational. That is, they must be able to subject themselves to partial application. This can be achieved with the sets D and A. In this manner, we say that whenever a function f is used by A it will only be for two potential reasons.

1. To satisfy a subgoal or goal
2. To determine if a subgoal or goal is satisfiable based on current knowledge

We imagine an actor A with a set M {S, F, G} where G is a goal we have just received. The actor's first step must be determining if he can even achieve the goal. To do so the actor must unify his current knowledge against the database. For instance if the goal is to reach Mullvad Square and the actor's current state exists in Penbrook Drive such that our map looks like

Mullvad Square (Goal)

↓↑

Presberty Lane

↓↑

Penbrook Drive (Current)

we may say the actor has knowledge of the location of Penbrook Drive. An actor must compute if it can find Mullvad Square. The simplest case is:

He has no knowledge of Mullvad Square.

This case will be returned to later when we discuss set G. The second case is:

The actor knows of Mullvad Square but he has no knowledge of Presberty Lane.

. In this case the Actor must prove that he knows how to get to Mullvad square. We imagine a function *g* which takes two functions *f* such that in this case we end up with the desire to ask:

g(
f(Mullvad Square, X, Y),
f(Mullvad Square, South, Presberty Lane))

g has the property that it attempts to "unify" two functions. Remember that *f* is relational in that it will return the value of uninitialized values against the database. An example of this is:

f(Mullvad Square, X, Presberty Lane)

↓

X = South

So we may define a function *g* as unifying if:

1. If *f1* and *f2* returns "true" in both cases than *f1* unifies with *f2*.
2. If either *f1* or *f2* return a value other than true, then *g* did not unify the values and returns the value of the missing value from *f1*.

We imagine a simple recursive method for this procedure. Where *f2* is a function built from the database D (In other words it always returns true. F2 is explict in the discussion but should be implemented implicitly). This will be explained after we go into the nature of the set G and S.

3.3 A Perpetual Set G

3.4 The Emotion Engine of S

4 NPC's that share, Care