# Graph Interpretation of Avatar Logic

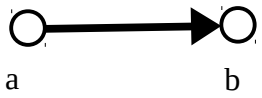
by Sven Nilsen, 2020

*In this paper I give a visual explanation of Avatar Logic using graphs.*

Avatar Logic[1] is an attempt to create a formal logic which satisfies “avatars”, in the sense of Avatar Extensions[2]. The formal theory is based on binary relations with additional axioms.

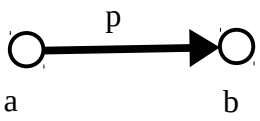
However, wrapping your head around Avatar Logic as binary relations can sometimes be difficult. Luckily, there is a trick one can use: Binary relations are isomorphic to graphs.

For example, a pair `(a, b)` might be thought of as an arrow from `a` to `b`:




(a, b)

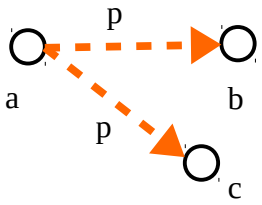
When `b : p`, one can think about it as the arrow having the label `p`:




(a, b)
b : p

Avatar Logic has no concept of `true` or `false`,
but it has `amb` (ambiguous) and `no amb` (unambiguous).
A proof is unambiguous when it terminates and no ambiguity is detected.
Unlike logic where everything is provable under a `false` assumption,
Avatar Logic can still be used under ambiguity, because not everything is provable.
One can also have non-terminating rules, but this makes only `amb` provable, but it is undecidable.

It is ambiguous to have two arrows `p`, from `a` to `b` and `c`:



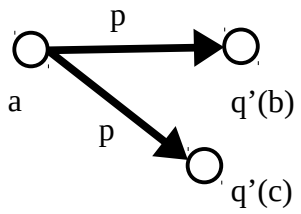

(a, b)
b : p
c : p
---
amb

Why? Well, it is just is. The axioms of Avatar Logic demands it.

However, this constraint occurs naturally, because one can think about this property as modeling a pure function. A pure function is deterministic and returns the same output for the same input.

Yet, in some cases one would like to model other stuff than pure functions.

To solve this problem, Avatar Logic introduces something called a 1-avatar:



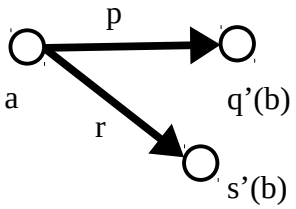

(a, b)
q'(b) : p
q'(c) : p

A 1-avatar `q'(b)` wraps `b` into a "shell" that can be reopened using `.q'(b) = b`.

When there is an arrow `p` pointing to a 1-avatar,
the axioms of Avatar Logic demands that all arrows of `p` must point to the same 1-avatar.

One can think about this property as a generic list of some type, which enforces that all elements in the list have the same type. The difference is that the type of the list is chosen by any pair, and if this constraint it violated, then ambiguity is detected.

When `uniq p` is specified, one can force a 1-avatar to behave uniquely, like a list with one element.
This trick is used when the same object `b` is related to `a` through more than one role.
It requires the 1-avatars to be different and having different roles:

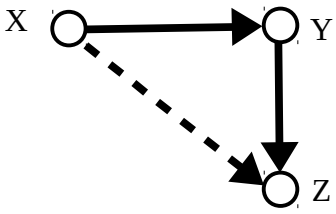



(a, b)
q'(b) : p
s'(c) : r

These rules cover everything that the axioms of Avatar Logic demands.

That's was all!

Now, one can use these rules to reason visually about what e.g. composition means:




(X, Z) :-
(X, Y),
(Y, Z).

The rule does not know the roles of `Y` or `Z`, so the arrows are unlabeled.
However, since the role of `Z` is used for both `(Y, Z)` and `(X, Z)`, it feels *wrong*.
This is not how composition is supposed to work!
A more natural rule is to use, which is used in formalization of Category Theory in Avatar Logic:

```
(X, F'(G'(Z))) :- (X, F'(Y)), (Y, G'(Z)).
```

## References:


[1] "Avatar Binary Relations"
Sven Nilsen, 2020
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/avatar-binary-relations.pdf

[2] "Avatar Extensions"
AdvancedResearch – reading sequence on Path Semantics
https://github.com/advancedresearch/path_semantics/blob/master/sequences.md#avatar-extensions

[3] "Category theory"
Wikipedia
https://en.wikipedia.org/wiki/Category_theory