

## Example Workflow

Here is an example of the flow of information through the project.

### Graph meta-language

We begin with the graph meta-language. This describes what kind of graph we're dealing with. This syntax is something I made up on the spot, but it conveys the idea. I like the idea of using algebraic types to allow for multiple kinds of edges and nodes within the graph, but this isn't expressed below.

An example for why different kinds of nodes would be important is if you wanted to add nodes that represented subgraphs. My convention below is that  $X \{A \ B \ C\}$  means  $X$  has  $A \ B$  and  $C$ , and  $X [A \ B \ C]$  means  $X$  has  $A \ B$  or  $C$ .

```
Edge {
    source      : Node
    destination : Node
    read       : Symbol
    write      : Symbol
    move       : enum [L, R, S]
}
```

```
Node {
    name : String
    [
        {accept : Boolean}
        {sub-graph : String}
    ]
}
```

### Graph

Here is an example of a graph that conforms to the meta-language above (note that this language describes turing machines, so the entity below is a turing machine).

### Interpreter

Assuming that you've already in some way modeled a "tape" and imported a machine, the following code interprets that machine. Most of the complexity comes from non-determinism.

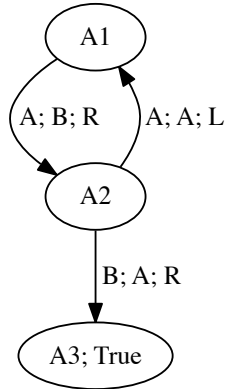


Figure 1: An example graph in the editor

```

def simulateTuring(transitions, state, accepts, tape):
    # if we're currently in an accept state, then we're done:
    # the string is in the language
    if state in accepts:
        return True

    # search through the list of all transitions and retain a list
    # of the ones that are valid for the current state
    # (read matches the tape head and the source node is the current node)
    validTransitions = (t for t in transitions if
                        t.src == state and t.read == tape.head())

    # make a copy of "the universe" for every transition and assume
    # that this is the correct transisiton, if any of the paths make
    # it to an accept state, return True.
    for t in validTransitions:
        tape_ = tape.copy()
        tape_.update(t.write, t.move)
        if simulateTuring(transitions, t.dst, accepts, tape_):
            return True

    # in no universe can this state lead to an accept
    return False

```

## Output

I'm not 100% sure that this output conforms with the above turing machine...

"AB" -> True

"AA" -> False