Operational Semantics For DefineGlobal

$$\frac{l \notin \text{dom } \sigma \quad \langle e, \rho, \sigma\{l \mapsto \text{unspecified}\}\rangle \Downarrow \langle v, \sigma'\rangle}{\langle \textsc{Val}(x, e), \rho, \sigma\rangle \to \langle \rho\{x \mapsto l\}, \sigma'\{l \mapsto v\}\rangle} \ (\textsc{DefineGlobal})$$

Detecting New Semantics

The following code detects if we are using our operational semantics:

```
(val x #f)
(val is-scheme
    (lambda () x))
(val x #t)
(is-scheme)
```

The function will return true if it is in the uscheme/scheme val function, or false if it is in the above semantics; This is because lambda's closure only encloses the ρ environment. This means that in scheme and uscheme, the value that ρ maps to in σ is unchanged, but the value in σ is changed. However, in this version, it is the value that ρ maps to which is changed, and σ is unchanged, still keeping the previous mapping. Since the lambda keeps the ρ which maps to the old σ (which still has the value of #f), it will use the value which is located there, where as in scheme, the ρ will be pointing to the value in σ which had changed (which now has the value of #t).

Comparison Of Defining Val

In Scheme's version, the underlying value at the location in σ is changed, whereas in the version we created, it is which location in σ that the variable points to which changes. Theoretically, I prefer our version, as it makes it so that you aren't redefining variables, and if you want to change a global variable, it has to be explicit. If you try to, there won't be any errors, but it will prevent older lambda functions from having this mistake propagated forward; the lambda uses the values that it had when it was called. On the other hand, it makes it less transparent, and easy to lose memory, as there isn't a way to reaccess the reassigned memory other than functions that have that memory stored. This makes it hard in practicality, as there isn't a good way of checking if you can get rid of the memory, as someone may be holding on to the older location. However, in scheme's version since the value is changed, there isn't a need for multiple locations for one variable, so it makes it easier to understand the memory structure.