A Special Case of Observational Equivalence

Theorem 0.1. *There is a Scheme expression, Obseq, such that*

$$[M \equiv N] \longleftrightarrow [\texttt{(}\textit{Obseq}\ \texttt{'}M\ \texttt{'}N\texttt{)} \equiv \texttt{(lambda (v) \#t)}].$$

Proof. Let's say that an S-expression consisting of a context, C, and a natural number, l, *does not distinguish* between Scheme expressions M and N, iff

1. neither $C[M]$ nor $C[N]$ converges within l steps, or else
2. both $C[M]$ and $C[N]$ converge to nonprintable values, or else
3. print$(C[M]) =$ print$(C[N])$.

Notice that M and N are observationally distinguishable iff there is some list, $\texttt{(}C\ \ l\texttt{)}$, that distinguishes them.

We will define a Scheme procedure *Obseq* such that $\texttt{(}\textit{Obseq}\ \texttt{'}M\ \texttt{'}N\texttt{)}$ diverges when applied to S-expressions of the form $\texttt{(}C\ \ l\texttt{)}$ that distinguish M and N, and it returns `#t` when applied to all other values. So

$$\begin{aligned}
M \equiv N &\longleftrightarrow \text{no } \texttt{(}C\ \ l\texttt{)} \text{ distinguishes } M \text{ and } N\\
&\longleftrightarrow \text{print}(\texttt{((}\textit{Obseq}\ \texttt{'}M\ \texttt{'}N\texttt{)}\ V\texttt{)}) = \texttt{\#t} \text{ for all values, } V,\\
&\longleftrightarrow \texttt{(}\textit{Obseq}\ \texttt{'}M\ \texttt{'}N\texttt{)} \equiv \texttt{(lambda (v) \#t)}.
\end{aligned}$$

To define *Obseq*, let *Not-ok?* be a Scheme procedure that returns `#t` when applied to a value that does not print in the form $\texttt{(}C\ \ l\texttt{)}$, and returns `#f` on all other values. The definition of *Not-ok?* is routine given the procedure, *Prnbl?*, from Handout 17 that detects S-expressions.

Also, let *Insert* be a procedure that inserts an expression in the hole in a context, viz.,

$$\text{print}(\texttt{(}\textit{Insert}\ \ \texttt{'}E\ \ \texttt{'}M\texttt{)}) = E[M].$$

Then *Obseq* ::=

```
(lambda (m n)
  (lambda (v)
       ;diverge if v describes a distinguishing context for M and N, and
       ;otherwise return #t.
    (or
     (Not-ok? v)                   ;return #t if v doesn't describe anything
     (let* ((C (car v))  ;C is a context
            (l (cadr v)) ;l is a natural number
            (CM (Insert C m))    ;CM is C[M]
            (CN (Insert C n)))   ;CN is C[N]
       (or
        (and              ;return #t if neither context has halted in l steps
         (not (Step CM l)) (not (Step CN l)))
                          ;if one of them has halted in l steps,
        (let ((V1 (Meval CM)) (V2 (Meval CN))) ;diverge if the other one doesn't
          (or                                  ;return #t if:
           (not (or (Prnbl? V1) (Prnbl? V2)))  ;neither has a printable value, o
           (equal? V1 V2)                      ;they have the same printable val
           Omega_0)))))))       ;the context distinguishes M and N
```

□

Theorem 0.1 would not be true if `(lambda (v) #t)` was replaced for example, by either of Ω_0 or `(lambda (v)` Ω_0`)`. This follows from the fact that the valid equations, $\mathcal{E}$, are "more undecidable[1]," than the valid equations, $\mathcal{E}_0$, of the form `(`$M = \Omega_0$`)`, though we shall not elaborate on these remarks here.

[1]Technically, $\mathcal{E}$ is Π^0_2-complete, while $\mathcal{E}_0$ is Π^0_1-complete. This implies that $\mathcal{E}$ is not half-decidable relative to $\mathcal{E}_0$. So even allowing proof systems that take all the equations in $\mathcal{E}_0$ as axioms, the Incompleteness Theorem for Scheme Equivalence still holds.