





Louis is really excited about the `terminate` construct. His old implementation of FLK required him to reboot any time his program ran into an infinite loop. Although he hasn't solved the halting problem, now he can guarantee not to have to reboot (excepting, of course, when his new-fangled operating system crashes) by testing his programs with `terminate` and his new `(timer N)` construct.

Louis defined the following transition rule(s) for `timer`:

$$\begin{array}{l}
 (\text{timer } N_1) \Rightarrow (\text{timer } N_2) \\
 \quad \text{where } N_1 > 1 \\
 \quad \text{and } N_2 = N_1 - 1 \\
 \hspace{15em} [\textit{timer-countdown}] \\
 \\
 (\text{timer } 1) \Rightarrow \#u \\
 \hspace{15em} [\textit{timer}]
 \end{array}$$

Louis can now use the `terminate` construct to run his program *might-go-infinite* for exactly 1000 steps (where we consider each transition to be one step). The following expression will return the result of *might-go-infinite* if it completes in under 1000 steps, otherwise it returns `#u`.

```
(terminate (timer 1000) might-go-infinite)
```

Unfortunately, Louis set off for Hawaii before he was able to extend the FL Operational Semantics to include `terminate`. In his absence, you are asked to finish it up.

- a. Give the transition rules for `terminate`.
- b. Are your rules confluent?
- c. Show how the following expression would be evaluated using the rules above:

```
(terminate (call (proc x (primop + x 2)) 5)
  (if (> 3 4)
    (rec x x)
    (proc y 1)))
```