**PIE Model**

**Notes by:** Robert Beck, revised December 7, 2000  
**Reference:** Dix and Runciman, Proc. BCS Conf. on People and Computers, 1985, Cambridge University Press

Let $C$ be the set of commands available in a system.  
*Example:* The calculator program in standard view has 27 buttons, each representing one command.

Let $P \subseteq C^*$ be the set of programs, written as strings of commands applied from left to right. The notation indicates that not every string of commands is expected to be a meaningful program.  
*Example:* Using the calculator, a program to add 23 and 41 is the sequence of 6 commands $2 \, 3 + 4 \, 1 =$

**Exercise:** Find a string of commands that is not a meaningful program.

Define the binary operation $\sim$ on $P$, formally $\sim: P \times P \rightarrow P$, by concatenation of strings of commands.  
*Example:* Let $p$ be the program $3 \, 7 +$ and $q$ be the program $4 \, 2 =$. Then $p \sim q$ is the program $3 \, 7 + 4 \, 2 =$ and $q \sim p$ is the program $4 \, 2 = 3 \, 7 +$

Let $\phi$ denote the empty program. Note that $\phi$ is the identity of $\sim$, namely for any program $p$, $p \sim \phi = p$ and $\phi \sim p = p$

Let $E$ be the set of visible effects of the programs, namely the visible changes to the window or dialog box associated with the system. Note that the set of effects $E$ is different than the set of states for the system. The states include invisible changes that occur in the process of computation.  
*Example:* The effect of the calculator program is shown in the dialog box and the memory toggle, just below it and left justified.

Define $I: P \rightarrow E$ to be the interpretation function that maps programs to effects.  
*Example:* For the calculator $I(2)$ is 2, and $I(6\ast)$ is 6.

**Exercise:** Find $I(\phi)$.

We now address the question of when programs are the “same,” and present four interpretations of “sameness.”

1. Clearly two programs $p$ and $q$ are the “same” if they are equal as strings of commands, $p = q$. We expect that starting from the same place and doing the same sequence of commands gives the same result. *Computers are deterministic systems.*
2. Define two programs $p$ and $q$ to be \textit{effect-equivalent}, denoted $p \approx q$, if $I(p) = I(q)$. Therefore, in this sense $p$ and $q$ are the “same” if they produce the same effect.

\textbf{Exercise:} Find several examples of pairs of distinct programs that are effect-equivalent.

3. Define two programs $p$ and $q$ to be \textit{mode-equivalent}, denoted $p \sim q$, if for all programs $r$, $I(p \sim r) = I(q \sim r)$. Therefore, in this sense $p$ and $q$ are the “same” if they produce the same effect \textit{and} do so when followed by any other program. This means that $p$ has not left the system in a different mode than $q$ did so that some subsequent program will cause a different output.

\textit{Example:} Compare the programs $2 * 7 +$ and $2 + 12 = $ followed by the program $3 =$.

Note that $p \sim q$ if and only if for all programs $r$, $(p \sim r) \approx (q \sim r)$

\textbf{Exercise:} Find several examples of pairs of distinct programs that are mode-equivalent.

\textbf{Exercise:} Find several examples of pairs of distinct programs that are effect-equivalent \textit{but not} mode-equivalent. Suppose that $e$ is the effect of such a pair of programs, namely $I(p) = e = I(q)$. The effect $e$ is called \textit{ambiguous} because we cannot tell which mode we are in.

4. Define two programs $p$ and $q$ to be \textit{history-equivalent}, denoted $p \approx q$, if for all programs $s$, $(s \sim p) \sim (s \sim q)$. Therefore, in this sense $p$ and $q$ are the “same” if they produce the same effect \textit{and} do so when preceded by and followed by any other program.

\textit{Example:} Compare the programs

Note that $p \approx q$ if and only if for all programs $r$ and $s$, $(s \sim p \sim r) \approx (s \sim q \sim r)$

\textbf{Exercise:} Find several examples of pairs of distinct programs that are history-equivalent.

\textbf{Exercise:} Find several examples of pairs of distinct programs that are mode-equivalent \textit{but not} history-equivalent.

One method of error recovery is to start the calculation over or to reset the system to a fixed initial state. We assume that this initial state is represented by the effect of the empty program, namely $I(\emptyset)$. There are two approaches to returning to the initial state. One can

- back up step by step reversing each command in turn until one gets to the beginning.

This is similar to using the backspace key to delete a word from a document by deleting one letter at a time.
back up all at once by running a recovery program, consisting of a sequence of commands, or even better, by issuing the “reset” command specific to the system.

We define a system to be

- **restartable** if for every program $p$, there is a program $q$ such that $(p \leadsto q) \sim \phi$.

  *Example:* The program $23 \cdot 1$ can be restarted with the program $BS\ BS\ BS\ BS$

  *Exercise:* Give several examples programs and their corresponding restarting programs.

- **uniformly restartable** if there is some program $q$ (built into the system and independent of any other actions) which for every program $p$ gives $(p \leadsto q) \sim \phi$. The word “uniform” typically means independent of previous choices or the same for all choices.

  *Example:* Any program that is a sequence of numbers and perhaps a decimal point can be restarted with the program $CE$.

  *Exercise:* Find a program that restarts all programs, or large sets of programs (if you can’t find one that does them all).

- **quickly uniformly restartable** if it is uniformly restartable and the restarting program is actually a command.

  *Exercise:* Show that the calculator is not quickly uniformly restartable, but that there is a program of length 2 that uniformly restarts the system.

Now that we have language and symbols to describe the possibility of modes and the efforts that must be made to reset a system, we can investigate some more general questions. For example, is it possible for the system to appear to have done nothing, but to actually be in a mode other than the initial one? If so, our definition for restarting was correct: we demanded that the result of restarting be mode equivalent to $\phi$. If it is not possible, we could have just asked for effect-equivalence.

*Exercise:* Determine whether there is a program $t$ for the calculator for which $I(t) = I(\phi)$ but $t \not\sim \phi$.

Also, we can ask whether the set of effects is proper for the system. Specifically, can we obtain every possible effect by running the proper program? But more importantly, can we have done some calculations (thus establishing a history) and then do more calculations to produce a specific result? And does producing a specific result mean obtaining the effect? Or obtaining the effect in a mode equivalent manner?

*Exercise:* Determine the set of effects $E$ for the calculator. *Hint:* You may want to consider $E$ as the set of pairs $S \times M$ where $S$ is the set of strings shown in the display and $M$ is the memory indicator. You may also be able to partition the set of strings.
**Exercise:** Suppose the current effect is (415.7, blank). What program must be run to obtain the effect (-0.31, M)?

**Exercise:** Consider a system that is much different than the calculator. The new system consists of the three buttons that appear at the right end of a window title bar and the corresponding button on the desktop task bar. Find

- The set of commands $C$, the set of programs $P$, the set of effects $E$.
- The value of $I(\phi)$

**Undo.** One approach to understanding properties that might be desirable for an undo operation is to postulate (remember your geometry class) that every program $p$ in $P$ has a companion program $U(p)$ that undoes its action. But we need to be specific about how that action is undone: effect-equivalent? mode-equivalent? history-equivalent? Since we are associating an “undoing” program with each program, we are creating an undo functional. Define $U : P \rightarrow P$ to be the undo functional with the property that for all programs $p \in P$, $p \leadsto U(p) \leadsto \phi$

**Exercise:** Using the calculator system for each $c \in C$ find $U(c)$.

We also postulate that our undo functional $U$ interacts with the operation $\leadsto$ on $P$ such that for all programs $a, b$

$$U(a \leadsto b) \leadsto U(b) \leadsto U(a)$$

In many systems the undo functional is implemented as a constant functional, namely for all commands $c \in C, U(c) = undo$. However, this means that undo is a program in $P$.

**Exercise:** What is $U(undo)$?