Formal Methods for Interactive Systems © Alan Dix 1991
http://www.hiraeth.com/books/formal/

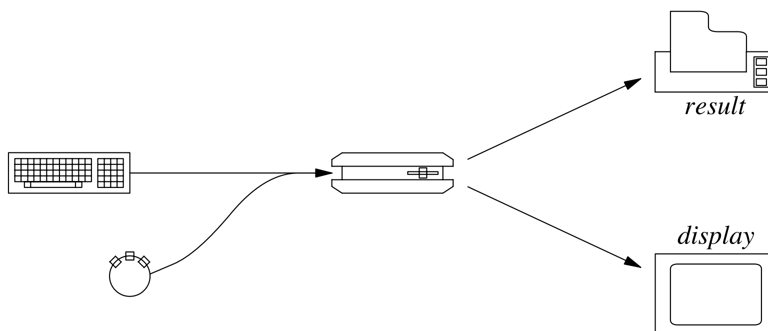
CHAPTER 3

Red-PIEs – result and display

3.1 Introduction

In the previous chapter, we considered a very simple input–output model of an interactive system. We noted that the *effect* part of the model could correspond to various levels of system responses. This rather extreme level of abstraction allowed us to discuss properties which we can then apply to various facets of an actual system.

Perhaps the most well-quoted interface property is *what you see is what you get* (WYSIWYG). In the context of a word processor this refers to the close relationship between the displayed and printed version of the document. In order to discuss properties of this nature we will introduce just such a distinction. We will distinguish those parts of the effect which are to do with the final end-product of the system, the *result*, and those that are more ephemeral, the *display.* So in the case of the word processor, the final result is the printed, formatted output, and the display is the moment-by-moment view the user has on the screen:



At a physical level, this distinction can be associated with the actual screen and printed output. However, similar distinctions appear at a more abstract level. For instance, when we think about the word processor we may be interested in the *document* as an entity in its own right, which may be formatted in different ways. It may thus be the document itself which is regarded as the result rather than a particular printing of it. Similarly, on the display side, we may want to ignore the limited size of the screen and think of the display as the entire view the user can get by scrolling up and down. Regarding the result and display in this light makes some properties (such as the exact visual appearance) less important, but may highlight other issues. For instance, spaces at the end of lines, which may be invisible on screen and in a particular printout, may cause the document to be badly formatted at some future stage. A similar problem, and now a familiar sight, is where hyphen- ation intro- duced by a word processor gets "left behind" in the docu- ment when it is re-formatted to a diff- erent line width.

In more complex systems such as databases or integrated CAD–CAM systems it is more clear that the significant result of an interaction is not a single hard-copy. On the other hand, we do not wish simply to identify parts of the system's representation and call these the result. Ideally, the result should be identified with a user-oriented model of the end-product of the system, and identifying this is a major part of the task analysis relating the user to the formal model.

To a large degree, we can regard the result as being the element of the system which is there because of the task or goal, and the display as the elements required for interaction. This can be a useful parallel to help us appropriately define the result of a system: for instance, we would not expect the cursor position in the result. Sometimes, especially when considering a system at several levels of abstraction, we may want to include interaction elements in the result. For instance, when looking at a word processor as a whole, we may only want to include the document itself in the result. However, when we look at a subdialogue, it may be that the result of that dialogue is a style sheet. The style sheet will affect the formatting of the document, but is of itself an artifact of the system, not the task.

Our main interest in this chapter will be on these *observability* properties, asking what the user can infer about the final result by examining the display. The properties will be information rather than presentation oriented. So, for example, we will not consider issues such as whether the visual appearance of text on screen and paper is identical. Instead, we would ask merely whether we can *work out* what the printed form will look like from the displayed form.

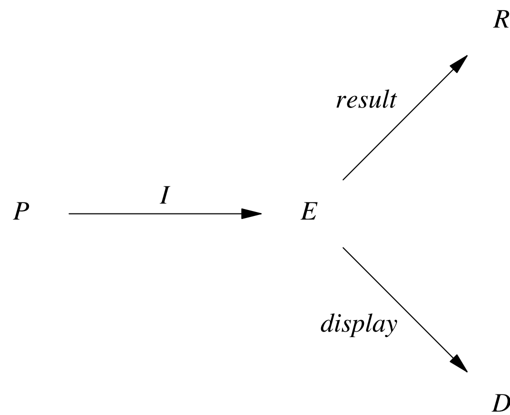
In addition, the introduction of the result and display will enable us to take a first look at what it means for a command to be *global,* that is, affecting the whole result without regard to the current display, or *local,* that is, all the

changes are visible from the display. An example of the former would be a global search/replace command and of the latter, typical character-by-character typing or direct manipulation.

3.2 The red-PIE model

3.2.1 Definition

We will model the distinction between result and display as a refinement of the PIE model. To do this we assume that from our effect space E there are two maps, *display* and *result*, to sets D and R respectively. These correspond in the obvious manner to the immediately visible display and the product that would be obtained if the interactive process were finished immediately. Diagrammatically we have the following:



For obvious reasons we refer to this structure as a red-PIE. We assume that the effect space E can be regarded as a system state and also that it is minimal in order to give the display and result the desired behaviour.

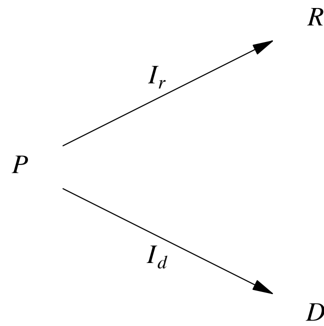
In the last chapter we expressed concern about using state representations, as these may introduce additional complexity which has no effects at the user level. However, we also saw that we can derive minimal states, using the monotone closure operator. This means that we will be able to assume that E has just, but only just, enough in it. We will make this more precise in the following sections.

3.2.2 Resolution (product) of result and display

The definition of a red-PIE given implies the existence of two additional interpretation functions:

$$I_r \quad = \quad result \circ I$$
$$I_d \quad = \quad display \circ I$$

giving rise to two PIEs sharing a common command set, $< P, I_r, R >$ and $< P, I_d, D >$. One could argue that these two PIEs are a more basic configuration than the red-PIE above, and that we should instead have ignored E and just asked for two such PIEs:



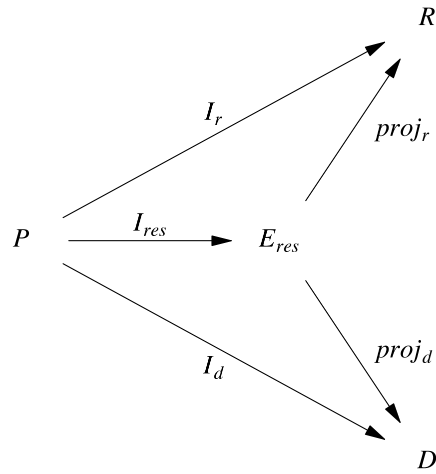
However, given any pair of PIEs with a common command set we can define a new PIE which factors both. We will call this PIE the *resolution* of the two PIEs. We define the resolution using an equivalence on P. Define the resolution equivalence $\equiv_{res}$ by:

$$p \equiv_{res} q \quad \hat{=} \quad I_r(p) = I_r(q) \quad \textbf{and} \quad I_d(p) = I_d(q)$$

That is, two command histories are resolution equivalent if they yield both the same result and display. Equivalently, we could use the equivalences defined by I_r and I_d and define $\equiv_{res}$ as:

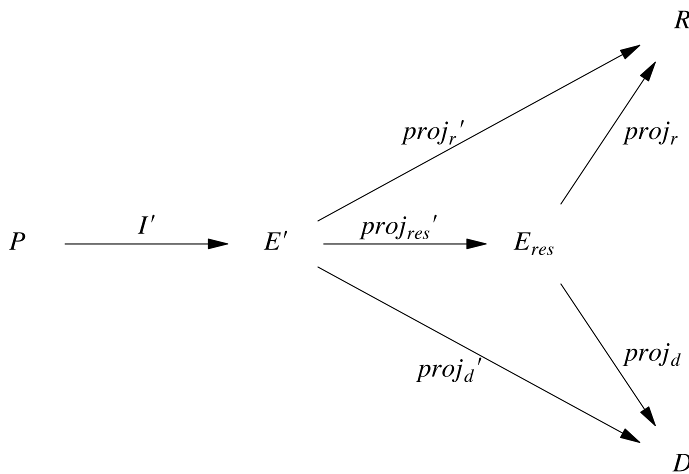
$$p \equiv_{res} q \quad \hat{=} \quad p \equiv_r q \quad \textbf{and} \quad p \equiv_d q$$

That is, $\equiv_{res}$ is the weakest equivalence stronger than both $\equiv_r$ and $\equiv_d$. We can define E_{res} as $P/\equiv_{res}$ and I_{res} as the canonical map in the standard way. Because $\equiv_{res}$ is stronger than $\equiv_r$ and $\equiv_d$, we can factor I_r and I_d with I_{res} and projection maps $proj_r$ and $proj_d$, so that the following commutes:



The resolution is in fact minimal, in the sense that given any PIE < P, I', E' >, with projections $proj_r'$ and $proj_d'$ which factor I_r and I_d respectively, e.g. $I_d = proj_d' \circ I'$, then there exists another projection $proj_{res}'$ from E' to E_{res} such that:

$$
\begin{aligned}
I_{res} &= proj_{res}' \circ I' \\
proj_r' &= proj_r \circ proj_{res}' \\
proj_d' &= proj_d \circ proj_{res}'
\end{aligned}
$$



To prove this, we simply construct $proj_{res}'$ such that:

$$\forall e' \in E' \quad proj_{res}'(\, e'\,) \;=\; I_{res}(\, p\,)$$

where

$$I'(\, p\,) \;=\; e'$$

This is a well-defined function since:

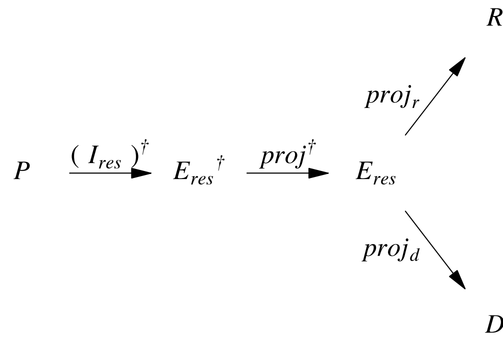
$$I'(\, p\,) \;=\; I'(\, q\,) \quad \Rightarrow \quad I_r(\, p\,) \;=\; I_r(\, q\,) \quad \textbf{and} \quad I_d(\, p\,) \;=\; I_d(\, q\,)$$

(because I' factors I_r and I_d)

$$\Rightarrow \quad I_{res}(\, p\,) \;=\; I_{res(}\, q\,)$$

and it clearly satisfies the required conditions.

We are almost at the original definition, only we also asked that the effect be monotone. We can easily achieve this by factoring I_{res} by its monotone closure, $(\, I_{res}\,)^\dagger$:



We can then define I, *result* and *display* by:

$$\begin{aligned} I &= (\, I_{res}\,)^\dagger \\ result &= proj_r \circ proj^\dagger \\ display &= proj_d \circ proj^\dagger \end{aligned}$$

Which is minimal by the minimality of the constructions for resolution and monotone closure. To be precise, any other monotone PIE with relevant projections will sit to the left of E_{res} by the minimality of resolution, and thence must also sit to the left of $(E_{res})^\dagger$ by the minimality of monotone closure.

Thus we have obtained the original definition. That is, although the definition using two PIEs appears more basic, we can, without loss of generality, assume the existence of a single monotone effect as in the first definition. This makes the statement of principles of observability and predictability much simpler.

3.2.3 Commutativity of resolution and monotone closure

Before moving on it is worthwhile noting the equivalence of an alternative derivation of this state E. Given the desire for a monotone effect factoring both the result and display, we might have chosen to move towards monotonicity first, and then later using the resolution. That is, we might have constructed $I_r^\dagger$ and $I_d^\dagger$ and then looked at their resolution, $(I^\dagger)_{res}$ say. It turns out that the resolution of two monotone PIEs is itself monotone:

LEMMA: $(I^\dagger)_{res}$ is monotone.

$$\begin{aligned} p \;(\equiv^\dagger)_{res}\; q \;&\hat{=}\; p \equiv_r^\dagger q \;\;\textbf{and}\;\; p \equiv_d^\dagger q \\ &\Rightarrow\; (\forall s \in P \;\; ps \equiv_r^\dagger qs) \\ &\qquad\quad \textbf{and}\;\; (\forall s \in P \;\; ps \equiv_d^\dagger qs) \end{aligned}$$

(by monotonicity of $I_r^\dagger$ and $I_d^\dagger$),

$$\begin{aligned} &\Rightarrow\; \forall s \in P \;\; (ps \equiv_r^\dagger qs \;\;\textbf{and}\;\; ps \equiv_d^\dagger qs) \\ &\Rightarrow\; \forall s \in P \;\; (ps \;(\equiv^\dagger)_{res}\; qs) \end{aligned}$$

Thus $(E^\dagger)_{res}$ would have been a good candidate for the state effect E. Happily these two constructions yield the same effect. That is:

THEOREM: commutativity of resolution and monotone closure.

$$(I_{res})^\dagger \;=\; (I^\dagger)_{res}$$

We can prove this by showing the equality of the two equivalences $(\equiv_{res})^\dagger$ and $(\equiv^\dagger)_{res}$. We have already shown that $(I_{res})^\dagger$ is minimal: hence we already know that:

$$p \;(\equiv_{res})^\dagger\; q \;\Rightarrow\; p \;(\equiv^\dagger)_{res}\; q$$

Thus we only need to prove the reverse implication.

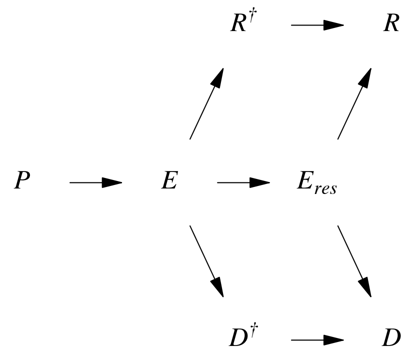
PROOF:

$$\begin{aligned} p \;(\equiv_{res})^\dagger\; q \;&\Rightarrow\; \forall s \in P \;\; (ps \equiv_{res} qs) \\ &\Rightarrow\; \forall s \in P \;\; (ps \equiv_r qs \;\;\textbf{and}\;\; ps \equiv_d qs) \\ &\Rightarrow\; (\forall s \in P \;\; ps \equiv_r qs) \\ &\qquad\quad \textbf{and}\;\; (\forall s \in P \;\; ps \equiv_d qs) \\ &\Rightarrow\; p \equiv_r^\dagger q \;\;\textbf{and}\;\; p \equiv_d^\dagger q \\ &\Rightarrow\; p \;(\equiv^\dagger)_{res}\; q \end{aligned}$$

(The astute reader might have noticed that all the implications were in fact equalities, and we needn't therefore have used the minimality of $(I_{res})^\dagger$; however, it seems nice to do so!)

Having proved the equivalence of these different definitions we can drop the distinction and refer to $E_{res}^\dagger$ as E, etc. We have, of course, from the second definition projections from E to $R^\dagger$ and $D^\dagger$ factoring $I_r^\dagger$ and $I_d^\dagger$; thus we have

the complete picture:



Note that often one will start by specifying $P \to R^\dagger \to R$ first. That is, the task or application-level objects and operations are defined giving a functional core. Later the display component may be added. The diagram above shows that it is an *abstraction* from the state of the system as whole. In Chapter 7 we will see how important it is *not* to try to build a system from this as a *component*.

3.3 Observability and predictability

When dealing with the simple PIE model, the notions of predictability and observability were slightly unclear as we were uncommitted as to what exactly the effect represented. We now have a richer effect space and we can give better definitions of these concepts. We will phrase each predictability principle as a requirement for the existence of some function between sets in the diagram, such that the resulting diagram with the function arrow added would still commute. In each case this will imply that certain arrows in the diagram must represent one-to-one mappings and thus the diagram will collapse a little.

One of the reasons for introducing strategies in §2.6 was for use at this stage; however, before wheeling in this extra complexity, we will look to see if any of the existing interpretation functions we have deserves to be monotone.

3.3.1 Simple predictability

In (almost) all editors we would expect some sort of additional editing "context" in addition to the resulting object (e.g. a cursor). Thus it will not be useful for the result interpretation to be monotone. Similarly, the display (as we've said before) contains only a part of the required information, and this is its role. So again we don't want I_d monotone. E is the result of a monotone interpretation by definition, as of course are $R^\dagger$ and $D^\dagger$, so this leaves E_{res}. What does it mean for this to be monotone?

Essentially E_{res} contains the sum of the information from R and D. (In fact, we could have defined it as the range of $I_r \times I_d$ in $R \times D$.) If I_{res} were monotone, then this would mean there was sufficient information in R and D together to predict the future behaviour of the system. For instance, if we were editing a program then it would mean that an up-to-date listing of the program together with the current display would be sufficient for all prediction. We can state this formally then as:

simple predictability:
I_{res} is monotone

In terms of the diagram, this means that E and E_{res} are no longer distinct (a monotone effect is equal to its own monotone closure).

As we promised, we can also define this in terms of a prediction function $predict_{simple}$:

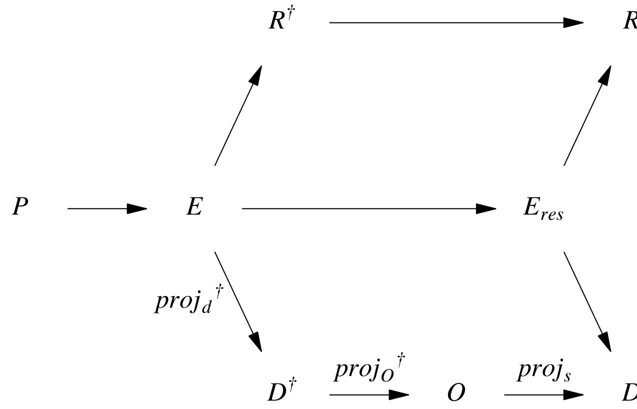
$$\exists\ predict_{simple} : R \times D \to E$$
$$\textbf{st}\ \forall p \in P \quad predict_{simple}(\ result(\ I(\ p\ )),\ display(\ I(\ p\ )\ )\ ) = I(\ p\ )$$

Is this a reasonable demand on an interactive system? Some very simple editors would satisfy it, as would some simple graphical systems. However, it forbids any sort of off-screen memory for objects to do with the editing task that are not actually part of the finished product. So for instance find/replace strings would have to permanently visible and one would not allow any buffers for temporary pasting to and from unless these could also fit on screen. Thus although useful for simple systems it appears too restrictive for systems with greater functionality. It would, however, be sensible to require it of certain parts of the total system. For example, it would be true of a cut/paste editor when one ignored the cut/paste actions.

3.3.2 Observability from D and strategies

Simple predictability, as defined above, depends on knowing the current result. The result is of course potential, and unless one has just collected an up-to-date listing of the document (or produced an example of the product with a CAD–CAM system), this information is not available. What is available is the

display, and a primary observability aim for a system would be to view the result using the display. Clearly the current display alone will not be sufficient in general, and we will require a widening of it. That is, we want a strategy *s* on the display PIE $<P, I_d, D>$ which will give rise to an observable effect *O* which will fit into the diagram thus:

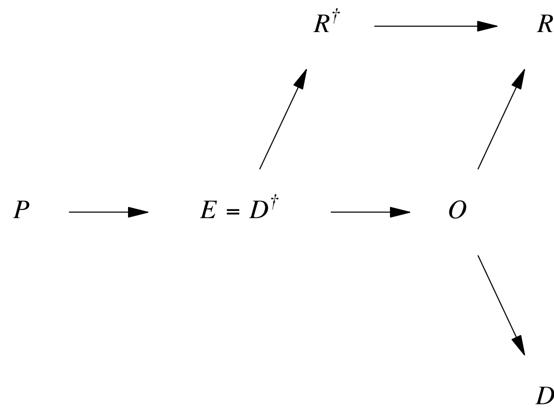


We will call the composite map from *E* to *O* *observe*, that is, $observe = proj_O{}^{\dagger} \circ proj_d{}^{\dagger}$. The rest of our observability principles will be based on maps between *O* and other sets on the diagram.

The first requirement, which we will term *result observability,* is that we should be able to observe the result via this observable effect. That is, we want a prediction function $predict_R$:

result observability:
$\exists \ predict_R : O \rightarrow R$
st $predict_R \circ observe = result$

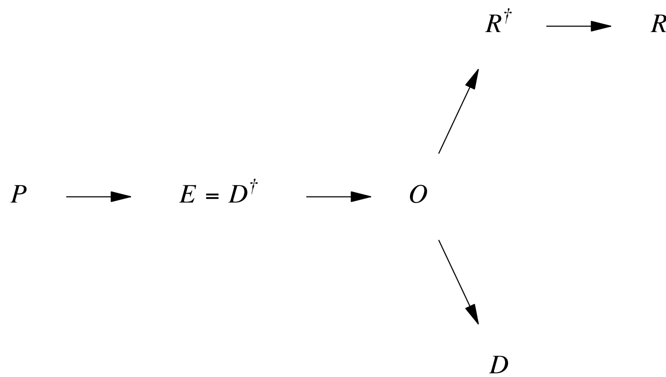
As *O* already factors the map to *D* and it also factors the map to *R*, it must be equal to E_{res} by the minimality of the resolution. Thus the diagram is simplified by identifying E_{res} and *O*. Further, since I_r is factored through *O* it is also factored through $D^{\dagger}$. Thus $D^{\dagger}$ is monotone, and factors both I_d (by definition) and I_r. It must therefore be equal to *E*, by the minimality of *E*. That is, our diagram is simplified again, by the unification of $D^{\dagger}$ and *E*:



Observing the result is, of course, useful especially if we also have simple predictability. In this case the result and display together tell us everything about the system; thus, if the observable effect can tell us everything about both the result and the display, it is sufficient for all predictions about the system. However, as we have said this applies only to simple systems. If the system is not simple predictable, we will want conditions to hold on O which are stronger than simply result observability. In particular, as well as wanting to know the current value of the result, we may well want to predict what will happen to the result in future. $R^\dagger$ contains sufficient information for this by its definition, containing all information about the system pertaining to its finite state, but ignoring purely ephemeral details. So for example, it would contain the current cursor position. If there were commands based on the screen coordinates, such as a "move to the top of the screen" command, then it would also have to include the screen framing information. It would *not* have to contain information such as what error messages were currently displayed, or, if there were no explicit use of screen coordinates, the screen framing. The ability to view this information underlies a lot of what might be termed informally the predictability of the system, and thus we define *result predictability* as the ability to observe this information from the observable effect:

result predictability:
$$\exists \; predict_{R^\dagger} : O \rightarrow R^\dagger$$
$$\textbf{st} \quad predict_{R^\dagger} \circ observe = result^\dagger$$

That is, $I_r^\dagger$ is now factored through O:

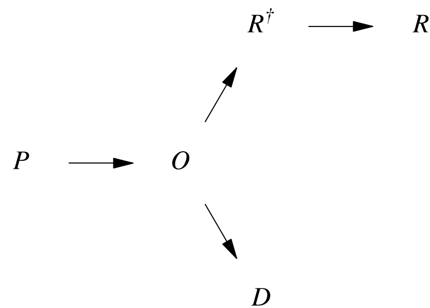


We can, of course, also define *display predictability* as the ability to observe $D^\dagger$ through O, and this is equivalent to the PIE $< P, I_d, D >$ being tameable. This together with result predictability would mean that one could predict not only the persistent effects of one's action, but also the ephemeral ones. On its own it seems less useful, and we won't dwell further on its implications.

Result and display predictability together we will call *full predictability,* and this can be given a definition in terms of a single prediction function which we will call simply *predict*:

full predictability:
$\exists$ *predict* : $O \rightarrow E^\dagger$
st *predict* $\circ$ *observe* = *identity*

In terms of the sets of the diagram, this means that O and E (and thus $D^\dagger$) will coincide, giving:



This is nearly as simple as the diagram we started with!

3.3.3 Different strategies and the limitations of observability principles

For simplicity, we have considered a single strategy. In practice, of course, the strategy used would depend on the purpose we intend. For instance, there would be a simpler strategy providing result observability than for result predictability (the latter would have to examine all buffers, search/replace strings, perhaps macro definitions, etc.)

It would be unacceptable to ask the user to use the whole strategy all of the time, and a real system would additionally need partial strategies for obtaining partial information. Further, the various observability principles state merely that the information is available, not how easy it is to use. For instance, a screen that displayed all its characters upside down would be as acceptable as a standard screen. Equally the strategy may be unusably complex.

We could address these problems of ease of use in various ways:

- We could accept their limitations and use informal reasoning to extend the formal concepts.
- We could produce richer models with principles that capture more of the notion of ease of use.
- We could produce formal measures of complexity for functions and algorithms the user is expected to use.

The last course could be followed by using programmable user models following Young *et al.* (1989) and Runciman and Hammond (1986) or using the sort of complexity analysis of Kiss and Pinder (1986). Alternatively, one could follow a naive approach such as demanding that the strategy can be described using a finite-state machine with a small (7?) number of states and simple recognition functions (like "am I at the top of the document?"). All these approaches have to be applied as they now stand during the production of a particular system and do not lend themselves well to reasoning at the abstract level of the interaction model. It is to be hoped that this will change as these methods mature.

The second option, that is, richer models and principles, will be taken up later. Chapter 5 gives specific predictability principles for temporal systems. Chapter 8 uses pointer spaces to address the issue of fidelity between display and result.

It is my belief, however, that these formal techniques will (in the foreseeable future) always require some sort of informal psychological analysis in addition to the formal statement of principles. Thus even when more specific temporal prediction principles are defined in Chapter 5, there will be an informal discussion as to the application of these.

There is one condition on the strategies that we use for observation of the general red-PIE, that can be given some formal statement, and this we examine next.

3.3.4 Passivity

The definition of a strategy that we have given so far allows some very silly ones. For instance, it would include the strategy for a simple word processor: "use the DELETE UP key until you get to the top of the document, then use DELETE DOWN till there's no document left". It is clearly sensible that any strategy designed to observe the result should not affect that result in so doing. There are several formulations of this informal principle, depending on how strict we want to be, but we shall call the general concept *passivity.*

First we will define what it means for a command or sequence of commands to be passive, then apply this to strategies. We will say a command sequence is passive, if the result before and after is the same, that is:

$$p \text{ is } passive \quad \hat{=}$$
$$\forall q \in P \qquad I_r(\, qp \,) \;=\; I_r(\, q \,)$$

This still leaves open the possibility that intermediate results are different (e.g. delete everything then type it in again!). If we don't want this we have a stronger condition, *strong passivity*:

$$p \text{ is } strong\ passive \quad \hat{=}$$
$$\forall c \in p \qquad c \text{ is } passive$$

That is, a sequence of commands is strong passive, if every command in the sequence is passive.

Moving back to our strategies, we recall that the use of a strategy leads to a sequence of command sequences being invoked, $q_i, \cdots$ These will terminate sometime, meaning that after some n all the q_i will be null. We will call the complete sequence up to termination q:

$$q \quad = \quad q_1\, q_2 \cdots q_n$$

We will obviously want the whole sequence to be passive:

$$\{\ passive\ strategy\ \}$$
$$q \text{ is } passive$$

Further though, we want to stop silly strategies, like "delete and re-type", so we want each of the q_i to be passive and probably strong passive:

$$\{\ strong\ passive\ strategy\ \}$$
$$q \text{ is } strong\ passive$$

Even this is not enough: although we may have the same result, it is no good if the rest of the system state has been mangled. Thus we will also want the whole strategy to be passive with respect to $result^{\dagger}$, or even better to return completely to the original state:

$$\{ \textit{ strategy returns } \} \quad I(\ pq\ ) \ = \ I(\ p\ )$$

History and undo mechanisms may make this difficult to achieve, as they complicate all reachability principles, and this would probably have to be applied to the functionality disregarding such facilities.

Such a strong condition would also be unacceptable for the individual commands of the strategy, as they must change the state if they are to achieve anything. Even passivity with respect to $R^{\dagger}$ is likely to be too strong. For instance, the strategy that includes scrolling up and down in a word processor seems quite reasonable; however, this would almost certainly involve moving the cursor, which will be part of $R^{\dagger}$. To achieve strong passivity with respect to $R^{\dagger}$, we would need something like a separate browsing window in addition to the editing one.

We would like to say, for example, that the strategy may move the cursor, but doesn't mangle anything useful like buffers, even temporarily. To do this it is necessary to layer the design, and separate out those features regarded as inviolate functionality, and those that are acceptable for browsing.

3.4 Globality and locality

The observability and predictability principles are concerned with the static state of the system: although the strategy itself is dynamic, it has been reduced to a static state, the observable effect. The concepts are all to do with what can be observed of what is there, rather than acting on the objects of interest. When considering simple PIEs it was the reachability conditions that were of importance with regard to these considerations of functionality. These can be applied directly to the red-PIE.

The result space R is most critical, as this represents the final output of the system, and we would clearly want I_r to be at least strong reachable, and probably megareachable too. The display is more dodgy. It may include start-up banners and the like that are not intended to be returned to; on the other hand, it is useful if one can set up the display exactly as it is most pleasing and useful,

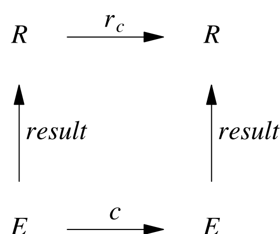
since it is through the display that one interacts with the system. In short, the reachability conditions are applied but with the willingness to make exceptions, or abstract from the full display as necessary.

If the system is highly interactive then one is also interested in the observation of the effects of commands. In particular, one wants the effects of many commands to be immediately visible from the display. Informally, it may be said that a command is *local* if its effects are contained entirely within the current display. The opposite case is a *global* command, which takes no notice of the display whatsoever. For example, typing a character would be a local command since one can see the character being inserted on the screen, and this is the only effect it has. On the other hand, "replace all words 'fred' with 'Fred'" would be a global command, as it makes no reference to the current display.

It is easier to attempt a formal definition of globality, and we can say that a command c is global if its effect can be modelled by a function on R, that is:

$$c \text{ is } global \quad \hat{=}$$
$$\exists\, r_c : R \rightarrow R \quad \textbf{st}$$
$$\forall p \in P \quad I_r(\, pc\,) = r_c(\, I_r(\, p\,)\,)$$

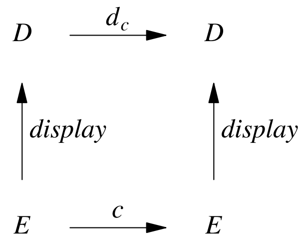
Or graphically, regarding c as the state transition function from E to E:



Notice that using this definition, a passive command is a special case of a global command! This is not exactly as intended, and we really would like to say "and the rest of the state doesn't change". Even if we did state this formally, it would not be right, as things like the cursor position and the display, would in fact change, but they would follow "in line" with the result. In Chapter 8 when we consider pointer spaces we will consider one way of expressing this requirement formally.

Locality is more complex; we could say:

$$\exists\, d_c : D \rightarrow D \quad \textbf{st}$$
$$\forall p \in P \quad I_d(\, pc\,) = d_c(\, I_d(\, p\,)\,)$$



This would make many commands on graphical systems, such as Mac-paint, local; but on text systems like word processors, many commands which are local given the informal definition, are not using this definition. For instance, "delete line" would mean that an extra line of text would be brought into the display at the top or the bottom. Where this line would appear might be predictable from the current display; what it would contain would not be.

Both of these functions r_c and d_c are what I will term *complementary functions* to the command c, and Chapter 9 discusses the properties of such functions and includes a more complex framework in which globality and locality can be defined. However, for the time being we will leave the above two definitions as a flavour of what the true definitions should be.

Even without an exact definition of local commands, we can state some of the properties we would expect from them, in particular the ability to spot and correct mistakes easily and using only the current and previous display. Thus we could assert the following (L being the local commands):

mistakes in local commands are obvious:

$$\forall\ c, c' \in L,\ p \in P$$
$$I(pc) \neq I(pc') \quad \Rightarrow \quad display \circ I(pc) \neq display \circ I(pc')$$
and
$$I(pc) \neq I(p) \quad \Rightarrow \quad display \circ I(pc) \neq display \circ I(p)$$

So if we try something (c) and we accidently type something else (c'), if it makes any difference at all then we can see that difference in the display. Additionally, we can tell the difference between the effect of a local command and typing nothing at all.

Note that these conditions only tell us that if there is a change (in the effect) we will be able to see some change (in the display). It does not tell us whether the changes we see are the only changes. More sophisticated approaches are required to ensure locality in this strong sense. In particular, Chapter 8 develops stronger statements of this type of property.

One stronger property that we can state using the existing model is the ability to easily work out how to reverse the effect of local commands:

local commands suggest their inverses:

$$\exists \;\; invert : \; D \times D \rightarrow P \quad \textbf{st}$$
$$\forall \;\; p \in P, c \in L \quad I(\; pc \; invert(\; d, d' \;)) \; = \; I(\; p \;)$$

where

$$d \; = \; display(\; p \;) \quad \textbf{and} \quad d' \; = \; display(\; pc \;)$$

That is, from the previous display and the current one, alone, it is possible to recover back to the previous state. This is perhaps more the sort of undo principle to which Shneiderman is referring for direct manipulation systems. (Shneiderman 1982)

3.5 Limitations of the PIE and red-PIE models

Abstract models will, by their nature, serve to specify only a subset of properties of interest. For any particular property, we may want a different model, or a refinement of the model. We cannot predict all possible such refinements and alternative models, but the succeeding chapters attempt to fill some of the more obvious holes.

The PIE model is clearly a single-user single-machine model; it does not cater well for the concept of multiple input streams. In the next chapter we consider a slightly different model that is better suited to such descriptions.

The PIE model expresses well the sequentiality of user input and machine output. It does not attempt to describe the interleaving and exact timing of these input/output events. Chapter 5 uses a model very similar to PIEs in order to describe some of the characteristics of this real-time behaviour, and to factor it out of the rest of the design process.

That leads on to the use of PIEs as the design of a system continues. The more detailed models in the succeeding chapters, can be seen as refinements of PIEs, although we will see in Chapter 6 that to do so requires a generalisation of the PIE. In Chapter 7, we consider how we can model parts of the internal specification of an interactive system using relations between PIEs. This represents, in a sense, development *within* the PIE framework.

When we considered globality and locality, the definitions were crude, because it was difficult to express the way changes in one projection of the effect should

be reflected "naturally" or "pull along" the rest of the effect. We can think of this as "the spider's legs problem."† We have a spider and pull one of its legs. What do the rest do? They must move somewhat as they are attached to the body, but where exactly? Chapters 8 and 9 deal with this problem from different perspectives. Chapter 8 on pointer spaces considers the "editing the object" perspective, whereas Chapter 9 on views takes the opposing "edit the display" perspective. In fact, these two are not as far apart as it seems, as the more natural the map between object and display the more blurred the distinction, and both approaches try to make the map as natural as possible from different directions. These two chapters are also, in a way, less abstract than the earlier models, being concerned with particular models of construction rather than entirely behavioural models. Both also, being more constructive, can be useful tools for the detailed specification and implementation of particular system. This could be seen as a useful property, or as excessive implementation bias.

We expect interaction models to be used within the framework of any formal process and of course the PIE model can be coded into whatever specification language one is using: thence the various principles become system requirements in addition to the application-specific requirements. We discuss this further in Chapter 11.

The PIE model attempts to be as abstract as possible, and one would think that this would make it unsuitable for use nearer the implementation end. However, work by Runciman and Toyn (1987) has shown that expressing PIEs in an executable functional language can be a useful implementation technique.

---

† Arachnophiles should skip the rest of this paragraph.