Generalizing WYSIWYT Visual Testing to Screen Transition Languages

Darren Brown Margaret Burnett Gregg Rothermel

TR#03-60-03
Oregon State University, Corvallis, OR 97331

May 27, 2003

. Abstract

How can rigorous forms of testing be supported in a way that is both compatible with the visual aspect of visual programming languages, and usable by the audiences using those languages — even when the audience has no background in software engineering? Visual programs are likely to contain at least some errors, and supporting a visual form of testing would give users a way to spot those errors early in the program's life. In previous work, we have developed a visual testing methodology known as WYSIWYT, for use in visual spreadsheet languages, and in this work, we show formally that this methodology can be generalized to screen transition diagrams. The algorithms and accompanying proof of the coverage equivalence that they ensure provide the mechanisms needed for the screen transition paradigm to incorporate WYSIWYT testing for both professional and end-user programming audiences.

1. Introduction

Visual programming languages (VPLs) are becoming increasingly common in several domains. For example, visual programming languages or sublanguages are becoming the most common way to do some kinds of GUI programming, the most common way of specifying visualization graphics depicting scientific data, and a common vehicle for macro generation for end-user applications. However, despite the increase in the use of VPLs for these and other programming tasks, there has been almost no attention to providing software engineering support mechanisms to programmers working in these languages.

Two issues relevant to VPLs have particular implications for software engineering in VPLs. The first is diversity of audience: while some users of VPLs are professional programmers, some are end users with no training in professional software engineering notions and methods. The second is the need to develop rigorous testing approaches that are fully compatible with the non-traditional paradigms and mechanisms used in VPLs, such as the specification of program semantics by directly manipulating objects or demonstrating with concrete examples.

We have previously worked to bring some of the benefits of applying formalized notions of testing to the informal, incremental, development world of spreadsheet-like VPLs through a highly interactive visual testing mechanism known as the ”What You See Is What You Test” (WYSIWYT) methodology [4, 15]. The methodology is completely visual, and is designed to support end users as well as more sophisticated programmers.

WYSIWYT has mainly been explored in the spreadsheet paradigm. There has also been work to adapt it to the dataflow paradigm [7], but since “underneath the hood” the spreadsheet paradigm uses a dataflow evaluation engine, this adaptation does not prove very much about the potential generality of WYSIWYT. In this paper, we consider whether WYSIWYT can be used for another visual paradigm, namely, the screen transition paradigm.

The screen transition paradigm uses screen transition diagrams to specify program behavior. These are an adaptation of state transition diagrams in which output states are represented via screen contents (and report contents and database contents), and transitions among these output states specify the conditions under which state changes occur. Screen transition diagrams are the primary communication device by which customer requirements are entered into the Lyee methodology [10, 17], a program generation facility used by a major Japanese software corporation[1] for commercial software development. The real-world needs of the Lyee methodology provide the context for the work reported here. In addition, the screen transition paradigm has been prototyped in a system called SILK for GUI development, and SILK empirical work shows that it can be effectively used by end-user programmers [9]. Finally, as a screenshot of SILK will demonstrate, the screen transition paradigm bears a visual similarity to another important VPL paradigm, namely, the visual rule-based paradigm. The similarity between these two paradigms, which has been exploited in part by Altaira [13] and Kara [5], implies that our findings for the screen transition paradigm may well extend to visual rule-based languages as well.

We begin our consideration of generalizing WYSIWYT to screen transition languages by summarizing WYSIWYT and the screen transition paradigm. We then define the properties of screen transition diagrams necessary for WYSIWYT, and sketch WYWISYT's visual aspects in this paradigm as an example. Finally, we explore the viability of this approach through translation to a formalism and ver-

[1]Catena Corporation, Tokyo

ification of that translation from a testing perspective.

2. Background: The WYSIWYT Testing Methodology

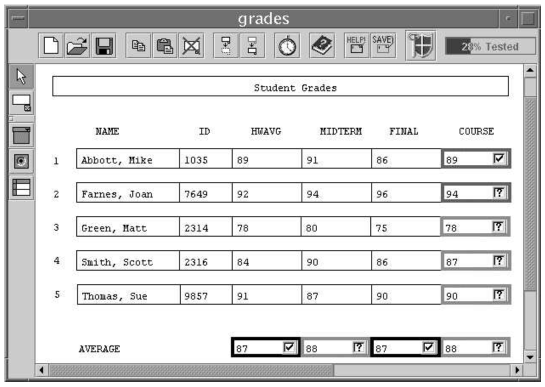
In previous work [4, 15], we presented the WYSIWYT methodology for testing spreadsheets. The WYSIWYT methodology provides feedback about the "testedness" of cells in spreadsheets in a manner that is incremental, responsive, and entirely visual, and its effectiveness has been demonstrated empirically [8, 15, 16].

The underlying assumption behind the WYSIWYT methodology has been that, as the user develops a spreadsheet incrementally, he or she could also be testing incrementally. We have integrated a prototype of WYSIWYT into our research spreadsheet language Forms/3 [2]. In our prototype, each cell in the spreadsheet is considered to be untested when it is first created, except input cells. Testedness is reflected via border colors on a continuum from untested (red, or light gray in this paper) to tested (blue, or black in this paper).

With WYSIWYT, the process of testing spreadsheets such as the one in Figure 1 is as follows. During the user's spreadsheet development, whenever the user notices a correct value, he or she lets the system know of this decision by validating the correct cell (clicking in the decision check box in its right corner), which causes a check mark to appear, as shown in Figure 1. This communication lets the system track judgments of correctness, propagate the implications of these judgments to cells that contributed to the computation of the validated cell's value, and reflect this increase in testedness by coloring borders of the checked cell and its contributing cells more tested (more blue).

WYSIWYT is based on an abstract testing model we developed for spreadsheets called a cell relation graph (CRG) [15]. A CRG is a pair (V, E), where V is a set of formula graphs and E is a set of directed edges modeling dataflow relationships between pairs of elements in V. A formula graph models flow of control within a single cell's formula, and is comparable to a control flow graph. In simple spreadsheets, there is one formula graph for each cell. (See [4] for discussions of how complex spreadsheets are treated.) For example, Figure 2 shows a portion of the CRG for Figure 1's cells, represented by dotted rectangles. In the formula graphs, nodes labeled "E" and "X" are entry and exit nodes, respectively, and represent initiation and termination of evaluation of formulas. Nodes with multiple out-edges are predicate nodes (represented as rectangles). Other nodes are computation nodes. Edges within formula graphs represent flow of control between expressions, and edge labels indicate the value to which conditional expressions must evaluate for particular branches to be taken.

We used the CRG model to define a test adequacy criterion for spreadsheets [15]. (A test adequacy criterion is a definition of what it means for a program to be tested "enough.") Our *du-adequacy criterion* is a type of dataflow adequacy criterion [14]. Such criteria relate test adequacy to interactions between definitions and uses in source code (*definition-use associations*, abbreviated *du-associations*). A *definition* of cell C is a node in the formula graph for



Student Grades

	NAME	ID	HWAVG	MIDTERM	FINAL	COURSE
1	Abbott, Mike	1035	89	91	86	89 ✓
2	Farnes, Joan	7649	92	94	96	94 ?
3	Green, Matt	2314	78	80	75	78 ?
4	Smith, Scott	2316	84	90	86	87 ?
5	Thomas, Sue	9857	91	87	90	90 ?
	AVERAGE		87 ✓	88 ?	87 ✓	88 ?

Figure 1: Visual depiction of testedness of a grades spreadsheet.

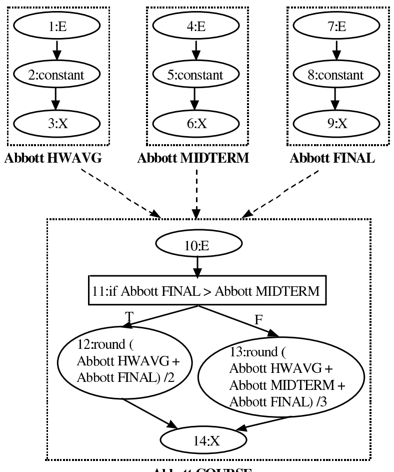

1:E
2:constant
3:X
Abbott HWAVG
4:E
5:constant
6:X
Abbott MIDTERM
7:E
8:constant
9:X
Abbott FINAL
10:E
11:if Abbott FINAL > Abbott MIDTERM
T
F
12:round (Abbott HWAVG + Abbott FINAL) /2
13:round (Abbott HWAVG + Abbott MIDTERM + Abbott FINAL) /3
14:X
Abbott COURSE

Figure 2: A partial cell relation graph for the spreadsheet of Figure 1, showing the formula graphs for the top row ("Abbott, Mike"). Dashed arrows indicate datafbw edges between cells' formula nodes.

C representing an expression that defines C's value, and a *use* of cell C is either a *computation use* (a non-predicate node that refers to C) or a *predicate use* (an out-edge from a predicate node that refers to C). For example, in Figure 2, nodes 2, 5, 8, 12, and 13 are definitions, nodes 12 and 13 are computational uses, and edges (11,12) and (11,13) are predicate uses. Under this criterion, a cell X will be said to have been tested enough when all of its du-associations have been covered (executed) by at least one test. In this model, a *test* is a user decision (which is communicated via a checkmark) that a particular cell contains the correct value given the inputs upon which it depends.

3. The Screen Transition Paradigm for WYSIWYT

In the screen transition paradigm, the general idea is that a user can visually depict input- and output-based states by explicitly sketching how the intended screens, reports, and databases appear and behave. A screenshot from the SILK system for user-interface specification [9] is shown in Figure 3. The difference from traditional state machines is that in the screen transition paradigm, the event and conditions that are required to execute the actions are only a partial specification of state.

Those familiar with visual rule-based languages will note the visual similarity between Figure 3 and rule-based languages such as KidSim/Cocoa/Stagecast [6]. The former's screen includes the latter's graphical preconditions, the former's events and conditions correspond to the latter's additional preconditions, the former's transition out-arrows are shown as "then" arrows in the latter, and the former's screen at the end of the arrow along with its transition actions represents the latter's postconditions. This similarity extends below the surface, and we will make use of this fact in Section 5.

To consider how WYSIWYT might be applied in the screen transition paradigm, we begin by providing terminology for the elements of screen transition diagrams.

With screen transition diagrams, the user specifies a program using screens, objects on those screens, and transitions. A *screen* is a window containing a formatted collection of *objects*, whose values will be used and/or produced by computations. For example, in Figure 4, a screen is the center window, containing the screen name and the objects that are on the screen. The transitions corresponding to the numbered arrows are given in Table 1.[2]

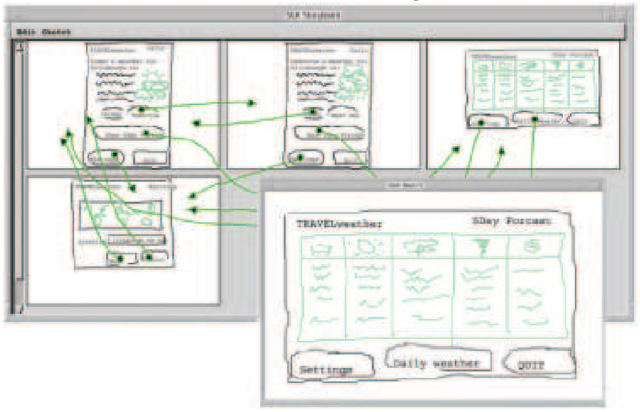
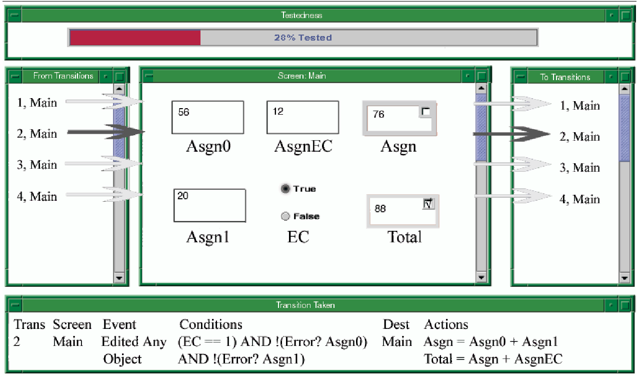


Figure 3: A SILK sketch (front) of a five-day weather forecast and storyboard (rear) [9]. An experienced user-interface designer created the sketch, including buttons, and arrows that show the screen transitions that occur when the user presses the buttons.



Trans	Screen	Event	Conditions	Dest	Actions
2	Main	Edited Any Object	(EC == 1) AND !(Error? Asgn0) AND !(Error? Asgn1)	Main	Asgn = Asgn0 + Asgn1 Total = Asgn + AsgnEC

Figure 4: An example in a (hypothetical) language of what the grades screen transition diagram might look like if the condition (bottom) is met. The Testedness indicator depicts the ratio of validated du-pairs in the spreadsheet. The center window shows the current screen. From Transitions (left) are all transitions that can reach the current screen and To Transitions (right) are all transitions that can be exercised from the current screen. Transition Taken (bottom) is the transition most recently exercised.

There are two types of objects: *input objects* and *output objects*. Input objects are objects into which the user is allowed to enter input, via the keyboard or the mouse, but they are also updatable by the program. A specialized kind of input object is an event object, which generates a user event if the user interacts with it. Output objects cannot receive user inputs. Instead, their purpose is to receive the results of computations, but they can also provide values to other computations.

Transitions can be defined by the tuple (source screen, destination screen, event, condition-action pairs). As such, transitions are a slightly more powerful form of transition than is traditional. In the tuple, *source screen* and *destination screen* enumerate the screens the transition connects. *Event* is a user or computational event under which the transition fires: the condition-action pairs are processed and the destination screen is displayed. Each *condition-action pair* contains a condition and a corresponding action to take if the condition is fulfilled. A *condition* is any arbitrary predicate, and each *action* consists of zero or more assignments to input or output objects.

4. Testing Screen Transition Diagrams

4.1. WYSIWYT Testing a Screen Transition Diagram

As explained earlier, the WYSIWYT methodology uses du-adequacy as its test adequacy criterion. The following define the definitions and uses in the screen transition paradigm in a manner that parallels those in Section 2, leading to a parallel notion of du-adequacy in this paradigm:

Definition 1: A *definition* of input object A is:

- the specification of A as an input value (including its initial value and any future values input), or
- an assignment to A in an action (presence of A in the action's left-hand side).

Definition 2: A *definition* of output object A is:

[2] We emphasize that the transition format shown throughout this article is only for precision of this discussion, and is not suitable for end users. As Pane and Myers showed empirically [11], end users are not very successful at using Boolean AND and OR, and do not tend to understand the use of parentheses as ways to specify precedence. They suggest some alternatives to these constructs, and empirically show that end users can use one set of such alternatives successfully [12].

#	Screen	Event	Predicate	Dest	Actions
1	Main	Edit any object	(EC==0 AND !(Error? Asgn0) AND !(Error? Asgn1))	Main	Asgn = Asgn0 + Asgn1 Total = Asgn
2	Main	Edit any object	(EC==1 AND !(Error? Asgn0) AND !(Error? Asgn1))	Main	Asgn = Asgn0 + Asgn1 Total = Asgn + AsgnEC
3	Main	Edit any object	(Error? Asgn0) OR OR (Error? Asgn1)	Main	Asgn = Error Total = Error
4	Main	Edit any object	Else	Main	Asgn = Undefined Total = Undefined

Table 1: Transitions for example screen transition diagram grades

- the specification of A's initial value, or
- an assignment to A in an action (presence of A in the action's left-hand side).

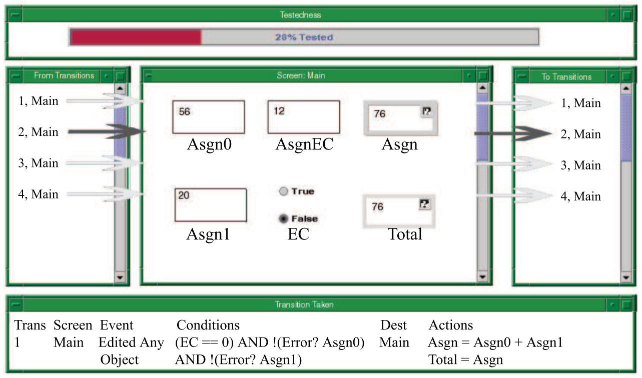
Definition 3: A *use* of object A is:

- a reference to A the right-hand side of an action (a *computational use*), or
- a reference to A in a condition (a *predicate use*).

Building on these definitions in the same manner as in Section 2, du-associations are interactions between definitions and uses, and the definition of du-adequacy, as in the spreadsheet paradigm, is to cover each du-association by at least one test.

How might a user pursue WYSIWYT testing in this paradigm? Figure 4 sketches a WYSIWYT interface. Note that, to be compatible with the WYSIWYT methodology, two constraints must be maintained: (1) the presentation of testedness must be integrated with the screen transition diagrams, as in the original WYSIWYT methodology, and (2) any update or test made by the user must be immediately and visually reflected in the presentation.

In Figure 4, the user has assigned values to input objects Asgn0, Asgn1, AsgnEC and set the boolean EC to true. The Transition Taken window show that transition #2 is the transition previously taken. (A complete list of transitions is given in Table 1.) The user has then validated object Total which is denoted by the checkmark in the object. In validating object Total, six of the 21 total du-associations have been covered and thus the Testedness window shows the screen transition diagram as being 28% tested. To test further, the user then sets the boolean object EC to false, thus exercising transition #1. This new situation is shown in Figure 5. Now there is a "?" in each object in which validation will increase testedness. If the user again validates Total, seven more du-associations will be covered.



Trans	Screen	Event	Conditions	Dest	Actions
1	Main	Edited Any Object	(EC == 0) AND !(Error? Asgn0) AND !(Error? Asgn1)	Main	Asgn = Asgn0 + Asgn1 Total = Asgn

Figure 5: The user has changed the boolean EC to false. The system tells the user that this situation has not been fully validated by placing a "?" in objects in which testedness can be increased through validation.

4.2. Issues Introduced by WYSIWYT for Screen Transition Diagrams

Two properties must be satisfied to apply WYSIWYT to the screen transition paradigm. First, spreadsheets are deterministic. In part, this is accomplished by nesting "if expressions". One way to ensure that this property holds in the screen transition diagram is to specify the order in which transitions' predicates are checked. (In fact, this is common in visual rule-based languages.) Second, in the spreadsheet paradigm, there is a distinguished well-defined value "undefined" that is assigned to cells in which no predicate is satisfied and no "else" clause is reachable. We introduce a similar mechanism in the screen transition paradigm for compatibility. Transition 4 in Table 1 is an example of a transition that would be built-in in some fashion for all output objects as a fall-through to ensure that all values are well defined. We refer to such transitions as *fall-through* transitions.

5. Translating Screen Transition Diagrams to CRGs

To precisely define WYSIWYT in the screen transition paradigm, a formal model of a screen transition diagram is needed. WYSIWYT for spreadsheets already has a formal model, the CRG. Can the same formal model be used to reason about WYSIWYT in screen transition diagrams? This section presents a translation method that starts with an arbitrary screen transition diagram and produces a CRG; Section 6 then considers its equivalence from a testing perspective to the original screen transition diagram. The diagram in Figure 4 and the associated Table 1 will serve as a running example.

5.1. The Translation Method

The translation method (1) translates the screen transition diagram to a set of rules, (2) translates the rules to a spreadsheet, and (3) translates the spreadsheet its CRG.

The first step of the translation method exploits the similarity between the rule-based paradigm and the screen transition paradigm. The outcome of this step is a set of rules, and the particular form of rules we choose is FAR rules. FAR (Formulas and Rules) is an end-user VPL that supports programming in both the spreadsheet and rule-based paradigms by representing the same code in both paradigms [3]. FAR rules are used because FAR's translation model provides the vehicle needed for translating FAR rules to a spreadsheet and hence to WYSIWYT's formal model, the CRG. Figure 6 presents an algorithm to translate a screen transition diagram to an ordered collection of FAR rules. Its result on our running example is shown in Figure 7.

The derived FAR rules can now be translated to a spreadsheet via the algorithm shown in Figure 8. This translation algorithm is based on the translation model presented in [3], but adds support for "else".

1. Defi ne a FAR rule of the form (*C, predicate, consequence expression*) for each transition action preserving the order per assignment action. *C* is the LHS of the assignment action, *predicate* is the predicate of the transition and *consequence expression* is the RHS of the transition action.
2. Defi ne a FAR rule of the form (*C, predicate, consequence expression*) for each input variable, where *C* is the name of the variable, *predicate* is "always" and *consequence expression* is the constant that has previously been input.

Figure 6: Translating a screen transition diagram to FAR rules.

Applying the algorithm in Figure 8 to the example's results produces a set of cells with size equal to the number of unique C's in the FAR rules created by the algorithm shown in Figure 6. Figure 9 sketches the resulting spreadsheet.

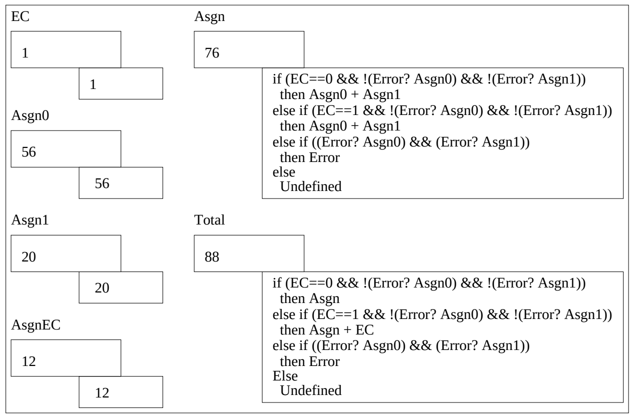
The algorithm for translating a spreadsheet to a CRG is given in [15]. The input cells' resulting formula graphs have three nodes: an entrance node, a constant node, and an exit node. The remaining cells' formula graphs have control flow between the entrance and exit nodes. See Figure 10.

- An ordered set of transition action rules (one per action in Table 1):
  - (Asgn, (EC == 0 AND !(Error? Asgn0) AND !(Error? Asgn1)), Asgn0 + Asgn1)
  - (Asgn, (EC == 1 AND !(Error? Asgn0) AND !(Error? Asgn1)), Asgn0 + Asgn1)
  - (Asgn, ((Error? Asgn0) OR (Error? Asgn1)), Error)
  - (Asgn, Else, Undefi ned)
  - (Total, (EC == 0, !(Error? Asgn0) AND !(Error? Asgn1)), Asgn)
  - (Total, (EC == 1, !(Error? Asgn0) AND !(Error? Asgn1)), Asgn + AsgnEC)
  - (Total, ((Error? Asgn0) OR (Error? Asgn1)), Error)
  - (Total, Else, Undefi ned)
- Input rules (one for each input in Figure 4):
  - (EC, Always, 1)
  - (Asgn0, Always, 56)
  - (Asgn1, Always, 20)
  - (AsgnEC, Always, 12)

Figure 7: FAR rules derived by applying the algorithm in Figure 6 to the screen transition diagram of Figure 4 and Table 1.

1. For each FAR rule of the form (*C, "always", consequence expression*), translate to cell C with formula "consequence expression".
2. For each collection of two or more FAR rules containing the same LHS C:
   (a) Create a cell C with formula empty string.
   (b) Preserving the order of Figure 6's algorithm:
      i. For each FAR rule of the form (*C, predicate, consequence expression*) except for the fall-through, append to C's formula "if *predicate* then *consequence expression* else".
      ii. For the fall-through FAR rule of the form (*C, "else", consequence expression*), append to C's formula "*consequence expression*".

Figure 8: Translating FAR rules to a spreadsheet.



```
if (EC==0 && !(Error? Asgn0) && !(Error? Asgn1))
  then Asgn0 + Asgn1
else if (EC==1 && !(Error? Asgn0) && !(Error? Asgn1))
  then Asgn0 + Asgn1
else if ((Error? Asgn0) && (Error? Asgn1))
  then Error
else
  Undefined
```

```
if (EC==0 && !(Error? Asgn0) && !(Error? Asgn1))
  then Asgn
else if (EC==1 && !(Error? Asgn0) && !(Error? Asgn1))
  then Asgn + EC
else if ((Error? Asgn0) && (Error? Asgn1))
  then Error
Else
  Undefined
```

Figure 9: Sketch of Grades translated to a spreadsheet via FAR rules. Cells show values, each cell's name is above the cell, and each formula is depicted at the cell's lower right.

5.2. Comparing Du-Associations

The set of du-associations in a screen transition diagram was defined in Section 4. The methodology presented in [15] defines the set of du-associations in a CRG. A surprising outcome of the translation is that these sets for a state transition diagram and its CRG produced by the translation do not match! There can be more du-associations in the new CRG than in the original diagram.

Definitions are mapped 1:1 from screen transition diagrams to CRGs, as are computational du-associations. The difference is in the predicate du-associations. In our running example, there are 8 predicate uses (p-uses) in the screen transition diagram and 32 p-uses in the CRG. There are two reasons for this difference. First, the CRG *repeats* predicates for each cell affected. For example, each transition in Table 1 affects two cells. Thus, the predicate that appears only once in the screen transition diagram is repeated for both Asgn and Total in the CRG (as pointed out in the caption of Figure 10). Second, screen transition diagrams have only true du-associations in their conditions, whereas both true and false du-associations are explicit in CRGs, as Figure 10 shows. (Complete lists of du-associations in this example are given in [1].)

6. Coverage Equivalence of Translated Programs

The translation algorithms in Section 5 allow for an increase in du-associations when translating from the screen transition paradigm to the spreadsheet paradigm. Thus it must be shown that the set of du-associations in the screen transition paradigm subsumes the set of du-associations in the spreadsheet paradigm in order to equate all du-associations

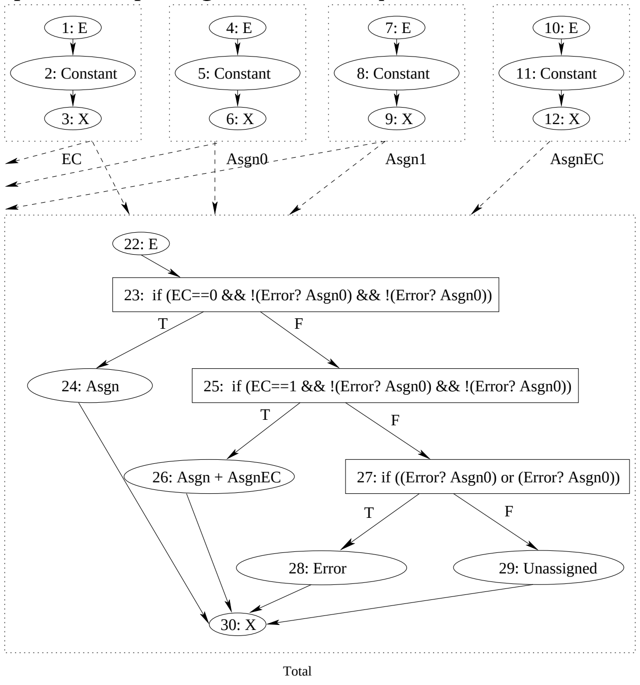


Figure 10: Grades CRG. Arrows pointing off the diagram point to cell Asgn, whose formula graph (not shown) is identical to Total's except for the contents of nodes 24 and 26.

coverage across the two paradigms. The proof requires a definition of subsuming:

Definition 4: Let S be a set of du-associations in program P. Let S′ be a subset of S. We say S′ is a *subsuming set* of S if a set of tests that is adequate for S′ is also adequate for S, and that S′ *subsumes* S.

We now show that the set of du-associations in the screen transition paradigm, when translated to the spreadsheet paradigm, constitutes a subsuming set in the spreadsheet as defined by the spreadsheet's CRG. This shows that the CRG, which is the formal model defining WYSIWYT for spreadsheets, generalizes to also define WYSIWYT for the screen transition paradigm. This is a necesary step in showing the feasibility of applying the WYSIWYT methodology to the screen transition paradigm.

We begin by considering the spreadsheet, as modeled by its CRG. Section 6.1 shows the construction of a subsuming set and proves that it subsumes the entire spreadsheet. We then show that the du-associations in the screen transition diagram are a subsuming set for the translated spreadsheet in Section 6.2.

6.1. Subsumption within the spreadsheet paradigm

We start by ignoring conditionals in Section 6.1.1, then we include them in Section 6.1.2.

6.1.1. Coverage Equivalence without conditionals

We begin by considering basic constructs and properties in the spreadsheet paradigm. Without control-altering constructs (conditionals are the only control altering constructs in spreadsheets), spreadsheets are composed of combinations of lines, forks, and sinks as shown in Figure 11.

Definition 5: A *line* is an ordered sequence of two or more cells. Let program (spreadsheet) P be a line. Each cell P_i in P references only cell P_{i-1}. Cell P_0 references nothing.

Definition 6: A *fork* consists of a program, two lines, and an intermediate connecting cell. Let P be a program with a sink cell P_n. Let Q and R each be line programs consisting of m and o cells respectively. Let A be a single cell that references P_n. A is the connecting cell and both Q_0 and R_0 reference A.

Definition 7: A *join* consists of two programs, a line, and an intermediate connecting cell. Let R be a line program with o cells. Let P and Q each be programs including sinks P_n and Q_m respectively. Let A be a single cell that references both P_n and Q_m. A is the connecting cell and R_0 references A.

Lemma 6.1. *For two cells C_0 and C_1 without conditionals, such that C_1 references C_0, a subsuming set of C_1 also subsumes C_0.*

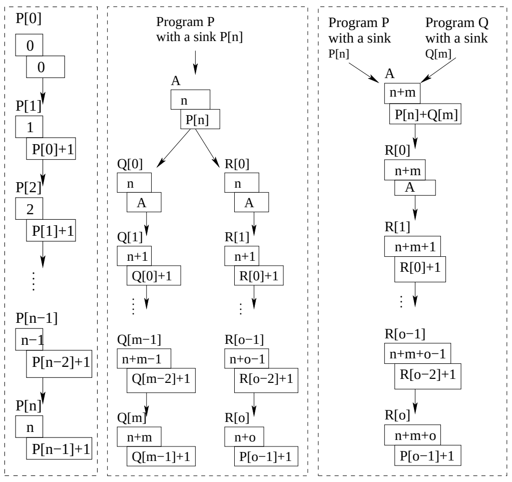
Proof. Exercising the du-associations whose uses are in C_1 guarantees the exercising of the du-associations whose uses are in C_0. This follows directly from the definition of what it is to exercise a du-association in the context of WYSIWYT [15]. Thus any set of tests that are adequate for C_1 are also adequate for C_0. By the definition of subsumption, C_0 is subsumed by any subsuming set for C_1.

Theorem 6.1. *For line programs, the subsuming set is the du-association whose use is found in the sink cell.*

Proof. The proof proceeds by induction. **Base Cases.** The one-cell case is trivial, as the set of du-associations is null. Consider a spreadsheet consisting of line M composed of two cells, M_1 and M_2. M_2 is the sink. M_2 contains a reference to M_1. Exercising the du-associations contained in M_2 guarantees the exercising of du-associations in M_1 by Lemma 6.1. **Inductive Hypothesis.** Consider a program P with only line N composed of n cells, $n > 2$. Cell N_n is the sink. The subsuming set consists of the du-association whose use is contained in cell N_n. **Inductive Step.** Consider a program Q with only line O composed of $n+1$ cells, $n > 2$. Cell O_{n+1} is the sink. If the cell O_{n+1} is removed, by the inductive hypothesis, O_n contains the subsuming set for the entire spreadsheet. Since O_{n+1} references O_n, exercising O_{n+1} guarantees exercising the subsuming set defined for O_n by Lemma 6.1. So a subsuming set for Q consists of the du-association whose use is in cell O_{n+1}.

Theorem 6.2. *For programs without conditionals, the subsuming set is the set of all du-associations whose uses are found in sink cells.*

Proof. Theorem 6.1 showed this to be true for line programs. Without conditionals there are only two other program structures that can occur in a program: a fork and a join. It is helpful to note that in a spreadsheet without control flow altering constructs, reaching a cell without error guarantees that all cells which it references directly or indirectly must be executed. A spreadsheet without conditionals can be considered a directed acyclic graph. Now the proof becomes: In a connected, directed acyclic graph,



(a) line (b) fork (c) join

Figure 11: blah Examples of a line, fork and join in the spreadsheet paradigm. Each cell is named with index notation (e.g. P[0]). Below the name is a box containing the cell's value, and below the value is a box containing the cell's formula. Arrows represent dataflow dependencies between cells.

some sink node is reachable from each node in the graph. The proof proceeds by induction on the depth of the graph. **Base Cases: Depth = 0.** The zero-depth case is trivial. Let G(V,E) be a connected DAG with depth 0. V consists of a single node N and E is an empty set. As N is the only node in the graph and N is also a sink node, a sink node is reachable from all nodes in the graph. **Base Cases: Depth = 1.** Let G(V,E) be a connected DAG with depth 1. V can be separated into two sets: let M be the set of source nodes and N be the set of sink nodes. Obviously, as set N is the set of sink nodes, sink nodes are reachable from every member in set N. As the graph has depth 1, edges from a node in M must end in N. **Inductive Hypothesis.** Let G(V,E) be a connected DAG with depth n. Let N be the set of sink nodes in G. Some sink node in G is reachable from every node in G. **Inductive Step.** Let G$'$($V \cup N'$, $E \cup A'$) be a connected DAG with depth $n + 1$. (Note that N from the inductive hypothesis is a subset of V.) Let N' be a set of sink nodes, each connected to a node in N of graph G with an edge in A'. In removing N' and A' from the graph, the inductive hypothesis can be used to show that every node in G can reach some sink in G. Now add back N' and A'. As every edge in A' is connected from some node in N to some node in N', all nodes in N' are reachable from N. As all nodes in G can reach N, and all nodes in N which are not sinks can reach N', all nodes in G$'$ can reach some sink.

Thus, a subsuming set of a spreadsheet without control flow altering constructs is the set of du-associations whose uses are contained in the sinks of a program.

6.1.2. Coverage Equivalence with Conditionals

Now we show the construction of a subsuming set for programs with conditionals. For precision, we define the du-association types that conditionals introduce. Figure 12 illustrates these types of du-associations.

Definition 8: A *CDU-out* is a computation-use du-association whose definition is contained within a conditional cell.

Definition 9: A *CDU-in* is a computation-use du-association whose use is contained within a conditional cell.

Definition 10: A *PDU-in* is a predicate-use du-association whose use is contained within a conditional cell.

Conditionals are the only control-flow-altering constructs in the spreadsheet paradigm. If we can show that the set created by adding CDUs-out to the subsuming set subsumes all du-associations introduced with conditionals, then we can show that the set created by unioning the set of CDUs-out with the subsuming set of the spreadsheet's portions without conditionals will subsume the spreadsheet.

Theorem 6.3. *The subsuming set for spreadsheets with zero or more conditionals is the set of du-associations whose uses are in sink cells and the set of du-associations whose uses reference conditional cells.*

Proof. Proof proceeds by construction (four cases): **Du-associations not participating in conditionals**: As shown earlier, our subsuming set first consists of the du-associations with uses in the sinks. As shown in Theorem 6.2, coverage is still guaranteed for all du-associations not

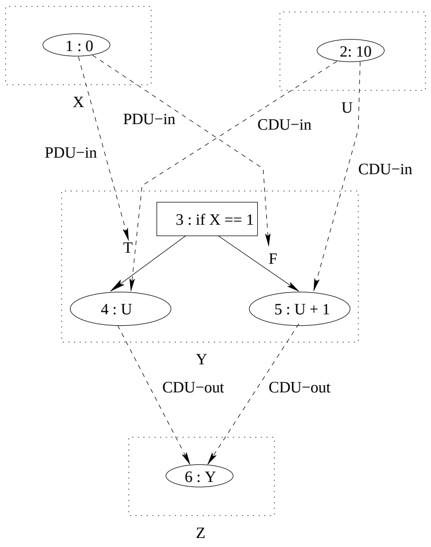


Figure 12: Definition-use association types: This is a modified CRG to highlight the definition-use types added by conditionals. We began with a cell relation graph, removing the cell dependency arrows. We then removed the entry and exit nodes. Finally, we added explicit depictions of definition-uses with dashed arrows.

involving conditionals. **CDUs-out**: We add all CDUs-out to the subsuming set. That is, we add all du-associations in which the definition is a computation use in a conditional cell. **PDUs-in**: No action is required, because of the requirement for well-defined cells that we will elaborate on in Section 4.2. By this requirement, all predicate branches are followed by CDUs-out. By including all CDUs-out in the subsuming set, both true and false PDUs-in will be reached as well. **CDUs-in**: Again, no action is required. All CDUs-in for a cell either end in sinks or are connected to the use cell's CDUs-out. This guarantees exercising the CDUs-in by the definition of what it is to exercise a du-association in the context of WYSIWYT[15]. This is also because of the requirement for well defined cells. Since all predicate branches must lead to computation nodes, and all references to CDUs-out computation nodes contained in conditional cells are included in the subsuming set, the du-associations in which the computation use resides in conditional cells are guaranteed to be subsumed as well.

6.2. Subsumption in the Screen Transition Paradigm

We have shown that there exists a subsuming set construction for the spreadsheet paradigm. Now we show that given our translation algorithms in the 5 , the du-associations in the screen transition paradigm are a superset of the subsuming set in the spreadsheet paradigm. Thus, all-du-association testing in the screen transition paradigm guarantees all-du-association testing in the spreadsheet produced by the translation, as modeled by the CRG.

Theorem 6.4. *The du-associations in the screen transition paradigm are a superset of the subsuming set in the spreadsheet paradigm.*

Proof. There are three cases. The first case is trivial in that CDUs are translated directly according to the algorithms in Section 5. The second case deals with PDUs that are not contained in sink cells. The final case discusses the PDUs that are contained in sink cells. **Case 1: Computation-use du-associations:** By Theorem 6.3, a subsuming set S for a spreadsheet consists of the conditional cells' CDUs-out and all du-associations in sink cells. Obviously, all computation-use du-associations in S are also in the screen transition diagram, since the algorithms map all computation-use du-associations 1-to-1. **Case 2: Non-sink predicate-use du-associations:** The translation algorithms preserve all predicate-use du-associations, and hence any of those in S are also in the screen transition diagram. The translation algorithms can also add new predicate-use du-associations to the spreadsheet, which can either be in non-sink conditional cells or in sink conditional cells. Let C be the set of all CDU-outs from a non-sink conditional cell. The set C together subsumes all predicate du-associations in the cell containing C's definitions, by Theorem 6.3, and we just showed that these CDU-outs are also in the screen transition diagram. **Case 3: Sink predicate-use du-associations:** Recall that one kind of these additions came from duplicating du-associations in the translation algorithms, but since S is a set, such duplicates cannot affect S's contents. The other kind of addition came from adding a false predicate-use to the spreadsheet for each (true) predicate in the screen transition paradigm. If this addition is in a sink, it will be in S. However, the final else in a cell always exists in the screen transition diagram, and since it subsumes all previous else's in the cell, the previous else's, whether additions or not, are not needed in S.

We have shown that the set of du-associations in the screen transition diagram is a subsuming set in the translated spreadsheet. Thus, all du-associations testing in the screen transition paradigm guarantees all du-associations testing in the corresponding spreadsheet.

7. Conclusion

In this paper we considered the question of whether the WYSIWYT visual testing methodology was general enough to serve as a testing methodology for the screen transition paradigm. We showed that it is by showing that there is a visual adaptation of WYSIWYT for a (hypothetical) screen transition language, and by presenting a translation from an arbitrary screen transition diagram to a coverage-equivalent CRG. Thus, the CRG formalism defining WYSIWYT for spreadsheets also generalizes to define WYSIWYT for screen transition diagrams.

This result has two practical implications. First, the screen transition paradigm is emerging for several purposes, including for teaching (as in the Kara system), for interface design by end users (as in the SILK system), and for real-world communication about software needs (as in the Lyee methodology); the ability to use WYSIWYT in this paradigm immediately provides support for testing programs in this paradigm. Second, the paradigm's strong relationship to rule-based programming and to more traditional state transition diagrams suggests that WYSIWYT may be general enough to support these paradigms as well.

. Acknowledgments

We thank the Institute of Computer Based Software Methodology and Technology, Catena Corporation, and Iwate Prefectural University for their support of this project.

References

[1] D. Brown, M. Burnett, and G. Rothermel, "End-user testing for the Lyee methodology using the screen transition paradigm and WYSIWYT," Oregon State Univ., Corvallis, OR, Tech. Rep. 03-60-01, Mar. 2003.

[2] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang, "Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm," *J. Func. Prog.*, vol. 11, no. 2, pp. 155–206, Mar. 2001.

[3] M. Burnett, S. Chekka, and R. Pandey, "FAR: An end-user language to support cottage e-services," in *IEEE Symp. Human-Centric Comp. Lang. and Env.*, Sept. 2001.

[4] M. Burnett, A. Sheretov, B. Ren, and G. Rothermel, "Testing homogeneous spreadsheet grids with the 'what you see is what you test' methodology," *IEEE Trans. Softw. Eng.*, vol. 28, June 2002.

[5] W. Hartmann, J. Nievergelt, and R. Reichert, "Kara, finite state machines, and the case for programming as part of general education," in *IEEE Symp. Human-Centric Comp. Lang. and Env.*, Sept. 2001, pp. 135–141.

[6] N. Heger, A. Cypher, and D. Smith, "Cocoa at the visual programming challenge 1997," *J. Vis. Lang. and Computing*, vol. 9, pp. 151–169, Apr. 1998.

[7] M. Karam and T. Smedley, "A testing methodology for a dataflow based visual programming language," in *IEEE Symp. Human-Centric Comp. Lang. and Env.*, Sept. 2001, pp. 280–287.

[8] V. Krishna, C. Cook, D. Keller, J. Cantrell, C. Wallace, M. Burnett, and G. Rothermel, "Incorporating incremental validation and impact analysis into spreadsheet maintenance: An empirical study," in *Int'l. Conf. Softw. Maint.*, Nov. 2001, pp. 72–81.

[9] J. Landay and B. Myers, "Sketching interfaces: Toward more human interface design," *Computer*, vol. 34, pp. 56–64, Mar. 2001.

[10] F. Negoro and I. Hamid, "A proposal for intention engineering," in *Int'l. Conf. Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, 2001.

[11] J. Pane and B. Myers, "Tabular and textual methods for selecting objects from a group," in *IEEE Symp. Vis. Lang.*, Sept. 2000, pp. 157–164.

[12] J. Pane, B. Myers, and L. Miller, "Using HCI techniques to design a more usable programming system," in *IEEE Symp. Vis. Lang.*, Sept. 2002, pp. 199–206.

[13] J. J. Pfeiffer, "Altaira: A rule-based visual language for small mobile robots," *J. Vis. Lang. and Computing*, vol. 9, no. 2, pp. 127–150, Apr. 1998.

[14] S. Rapps and E. Weyuker, "Selecting software test data using data flow information," *IEEE Trans. Softw. Eng.*, vol. 11, pp. 367–375, 1985.

[15] G. Rothermel, M. Burnett, L. Li, C. DuPuis, and A.Sheretov, "A methodology for testing spreadsheets," *ACM Trans.*

Softw. Eng. Meth., vol. 10, no. 1, Jan. 2001.

[16] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T. Green, and G. Rothermel, "An empirical evaluation of a methodology for testing spreadsheets," in *Int'l. Conf. Softw. Eng.*, Apr. 2000, pp. 198–207.

[17] C. Salinesi, C. Souveyet, and R. Kla, "Requirements modeling in Lyee," Institute of Computer Based Software Methodology and Technology, Tech. Rep. TR2-1, Mar. 2001.