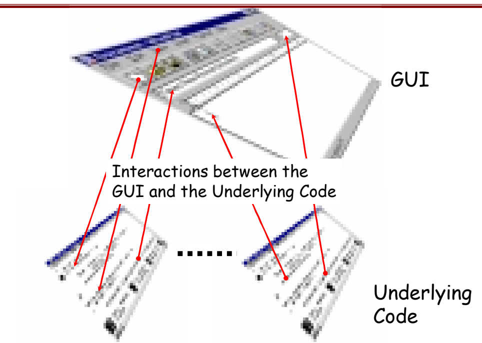
Plan Generation for GUI Testing

- *The 21st International Conference on Software Engineering*
- *The Fifth International Conference on Artificial Intelligence Planning and Scheduling*
- *IEEE Transactions on Software Engineering*

Research Focus

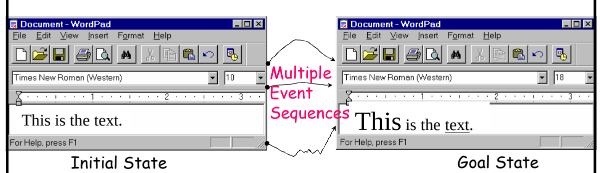


Why Planning for GUI Testing

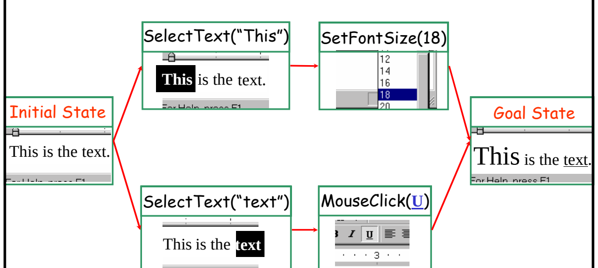
- GUIs are Event Driven
- Individual User Events
  - NOT ENOUGH !
  - Sequences of User Events lead to Different States
- Test Case: Sequence of User Events
- How to Generate Test Cases ?
- Use Planning to Select Likely Test Cases

Selecting Test Sequences

- Infinitely Many
- Randomly Choose Sequences
- Expert Chooses Sequences
- Automatically Generate Events for COMMONLY USED TASKS



A Plan for a GUI Task



Outline

- Using Planning for Test Case Generation
  - Overall Approach
  - Exploiting GUI Structure
  - Generating Alternative Test Cases
- Experimental Results
- Related Research
- Concluding Remarks

7

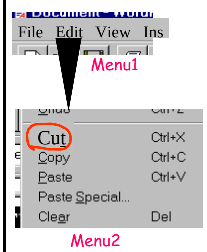
Overview of Test Generation

Phase	Step	Test Designer	Automatic Planning-based System
Setup	1		Derive Planning Operators from GUI
	2	Code Preconditions and Effects of Operators	
Test Case Generation	3	Specify a Task (Initial and Goal States)	
	4		Generate Test Cases

8

Straightforward Approach

- Define One Operator for each User Action



Operator :: *CUT*
Preconditions:
  isCurrent(Menu2).

Effects:
  FORALL Obj in Objects
    Selected(Obj) ⇒
      ADD inClipboard(Obj)
      DEL onScreen(Obj)
      DEL Selected(Obj)
  ADD isCurrent(Menu1)
  DEL isCurrent(Menu2).

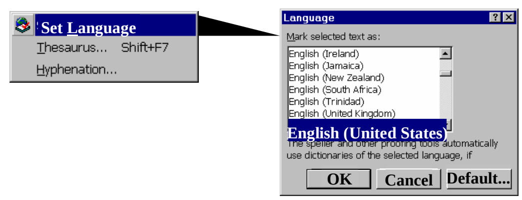
9

Exploit the GUI's Structure

- Reduce the Number of Operators
  - System more Efficient
  - Easier for the Test Designer

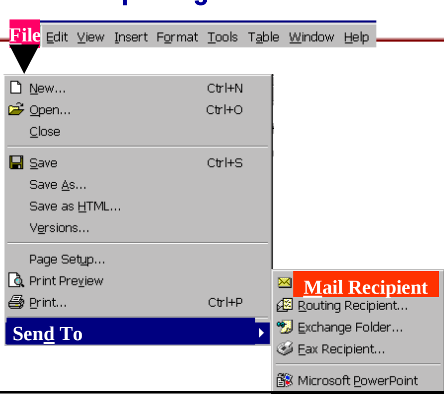
10

Opening Modal Windows



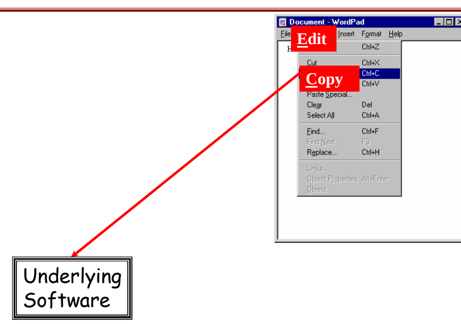
11

Opening Menus



12

Interacting with the Underlying Software



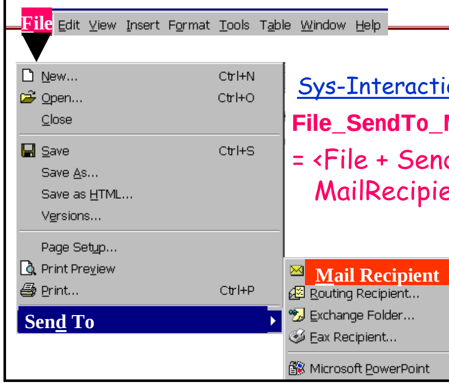
13

Create Hierarchical Operators

Two Types of Abstractions
- Combine Buttons ⇒ Create System-Interaction Operators
- Decompose GUI Hierarchically ⇒ Create Abstract Operators

14

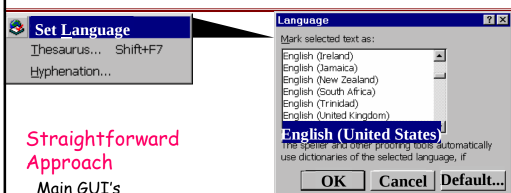
Create System-Interaction Operators



Sys-Interaction Operator:
File_SendTo_MailRecipient
= <File + SendTo + MailRecipient >

15

Create Abstract Operators



Straightforward Approach

Main GUI's Operator Set

...
Set Language
SelectFromList()
Default
OK
Cancel
...

Using Abstraction

Main GUI's Operator Set

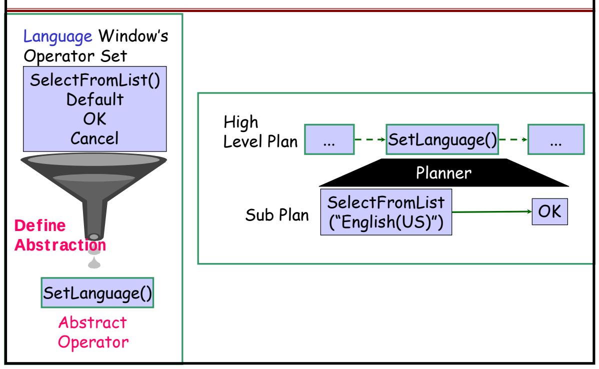
...
Set Language
...

Language Window's Operator Set

SelectFromList()
Default
OK
Cancel

16

Create Abstract Operators



17

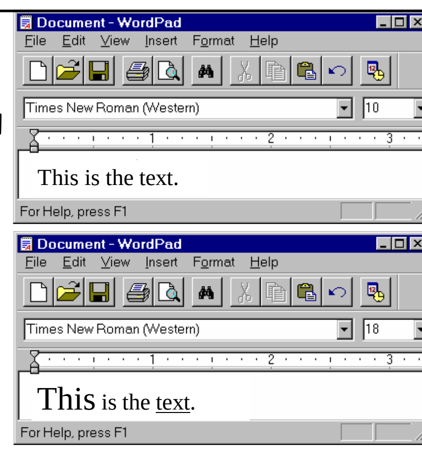
Effects of Exploiting the GUI's Structure

- Reduction in Planning Operators
  - 325 operators ⇒ 32 operators
  - Ratio 10:1 for MS WordPad
  - 20:1 for MS Word
- System Automatically Determines the System-interaction and Abstract Operators

18

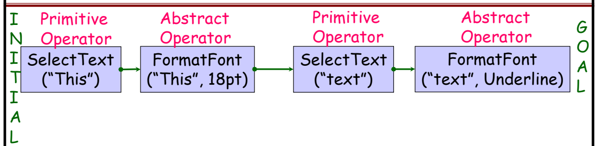
Initial State

Goal State

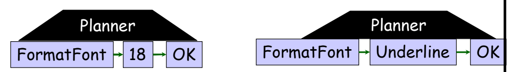


19

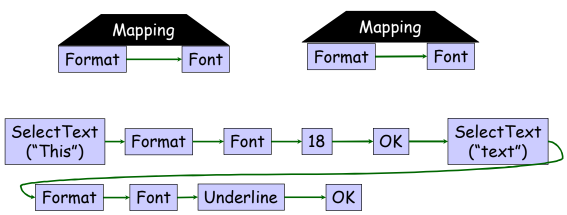
Test Case



20

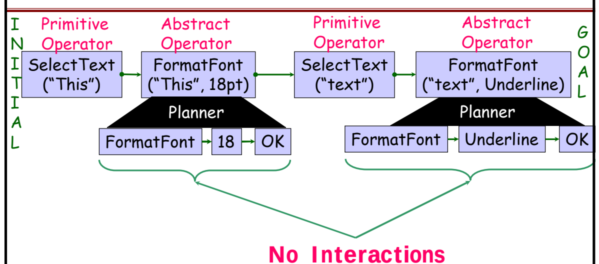


21



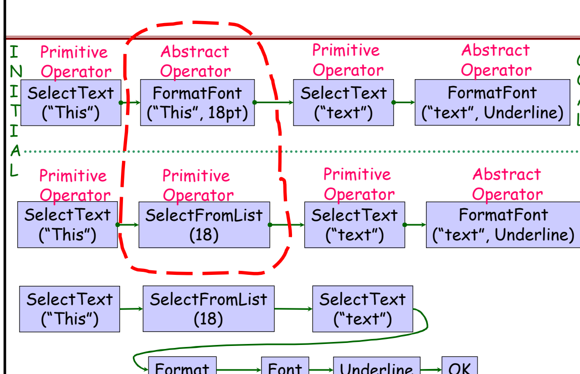
22

Different from HTN Planning



23

Alternative Test Case



24

Methods to Generate Alternative Test Cases

- Different Results from Planner
- Abstract Operator Decompositions
- Linearizations of the Partial-order Plan

25

Feasibility Study

- Purpose
  - To Determine whether Planning is a Feasible Approach for GUI Test Case Generation
    - Execution Time
    - Human Effort
- Experimental Design
  - GUI: MS WordPad
  - Planner: IPP [Koehler et al. '97]
  - Hardware Platform: 300 MHz Pentium based Machine, 200 MB RAM, Linux OS
  - 8 Tasks, Multiple Test Cases for each Task

26

Experimental Results

(Task) Plan No.	Plan Time (sec.)	Sub Plan Time (sec.)	Total Time (sec.)
1	3.16	0	3.16
2	3.17	0	3.17
3	3.2	0.01	3.21
4	3.38	0.01	3.39
5	3.44	0.02	3.46
6	4.09	0.04	4.13
7	8.88	0.02	8.9
8	40.47	0.04	40.51

27

Related Work

- GUI Testing
  - FSM [Esmelioglu and Apfelbaum] and VFSM [Shahady and Siewiorek] Models.
  - Genetic Algorithm Technique [Kasik and George]
  - Visual TDE for GUIs [Foster, Goradia, Ostrand, and Szermer]
- Planning for Testing
  - [Adele Howe, Anneliese Von Mayrhauser, Richard Mraz in ASE '97]

28

Concluding Remarks

- Automatic Planning is a Feasible Approach for GUI Test Case Generation
- Automatic Generation of Preconditions and Effects from GUI Specifications
- Generate Expected Output (Automated Verification)

29

Coverage Criteria for GUI Testing

8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9), **Vienna University of Technology, Austria, Sept. 10-14, 2001.**

30

Coverage Criteria

- Two purposes
  - Test data selection criteria
    - Rules used to select test cases
  - Test data adequacy criteria
    - Rules used to determine how much testing has been done
- Common Examples for Conventional Software
  - Statement coverage
  - Branch coverage
  - Path coverage

  } Structural Representation of the Code

Coverage Criteria for GUIs

31

- Cannot use code-based coverage
  - Source code not always available
  - Event-based input
    - Different level of abstraction
- Our Contribution
  - Hierarchical structure of the GUI in terms of events
  - Coverage criteria based on events

Outline

32

- GUI Definition
- Representation of GUIs
- Coverage Criteria
- Case Study
- Conclusions

GUI Definition

33

- Hierarchical
- Graphical Front-end
- Accepts User-generated and System-generated events
- Fixed sets of events
- Deterministic Output
- State of the GUI is the set of Objects and their Properties

GUI Representation

34

- Motivation
  - GUI testing needs a "Unit of Testing"
    - Manageable
    - Test the unit comprehensively
    - Test interactions among units
  - GUIs are created using library elements
    - Need to test these elements before packaging them for reuse
      - Certain level of confidence that the element has been adequately tested
    - User of these elements should be able to test the element in its context of use

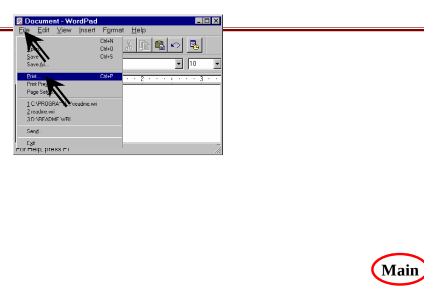
Model GUI Hierarchically

35

- Hierarchy
  - GUIs are decomposed into a hierarchy of components
  - Hierarchical decomposition makes testing intuitive and efficient
  - Several hierarchical views of GUIs
  - We examine Modal Dialogs to create the hierarchical model

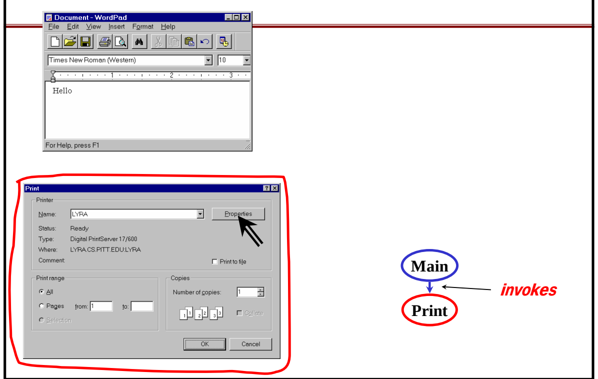
Modal Windows in GUIs

36



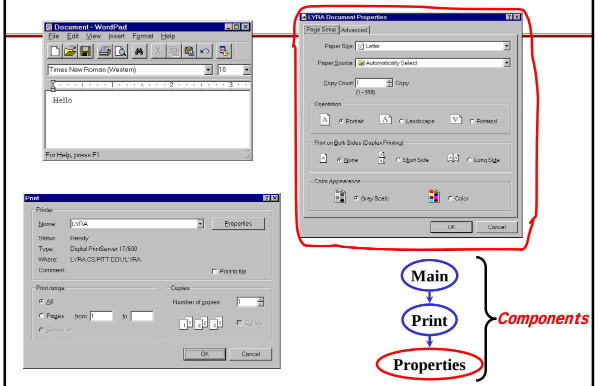
37

Modal Windows in GUIs



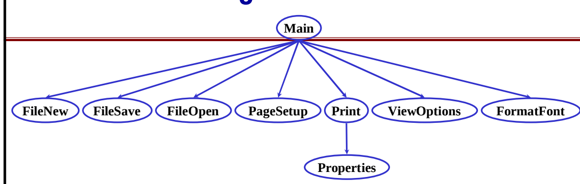
38

Modal Windows in GUIs



39

Integration Tree

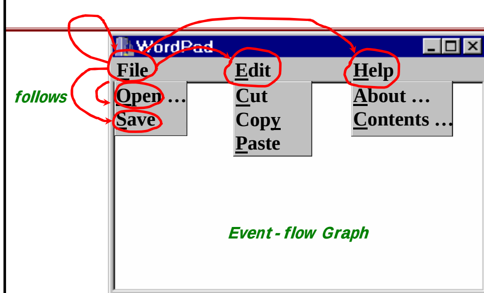


Definition: Integration tree is a triple <*N, R, B*>

- *N* is the set of components in the GUI
- *R* ∊ *N* is a designated component called the ***Main*** component
- *B* is the set of directed edges showing the invokes relation between components, i.e., $(C_x, C_y) \in B$ iff C_x invokes C_y.

40

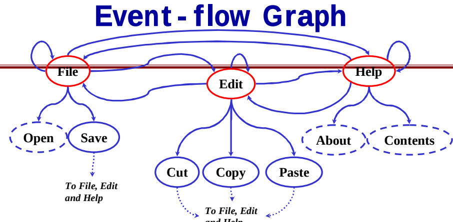
Representing a Component



Definition: Event e_x follows e_y iff e_x can be performed immediately after e_y.

41

Event-flow Graph



Definition: Event-flow graph is a 4-tuple <*V, E, B, I*>

- *V* is the set of vertices, representing events,
- *E* is the set of directed edges, showing the follows relationship,
- *B* is the set of events first available (shown in red),
- *I* is the set of events that invoke other components (dotted lines).

42

Classifying Events

- *Classification*
  - –A new classification of events aids in creating the hierarchical model of the GUI

- Opening modal windows
  - Restricted-focus events
- Closing modal windows
  - Termination events
- Opening modeless windows
  - Unrestricted-focus events
- Opening menus
  - Menu-open events
- Interacting with underlying software
  - System-interaction events

Coverage Criteria

- Intuitively
  - Each component is a unit of testing
  - Test events within each component
    - Intra-component coverage criteria
  - Test events across components
    - Inter-component coverage criteria

Coverage Criteria

- Intra-component Coverage
  - Event coverage
    - Individual events
    - Each node in the event-flow graph
  - Event-interaction coverage
    - Each pair of events
    - Each edge in the event-flow graph
  - Length-n event sequence coverage
    - Sequences of events
    - Bounded by length
      - Length-1 event sequences
      - Length-2, length-6 event sequences
    - Paths in the event-flow graph

Coverage Criteria

- Inter-component Coverage
  - Invocation coverage
    - Invoke each component
    - Each restricted-focus event
  - Invocation-termination coverage
    - Invoke each component and terminate it
    - Restricted-focus event followed by a termination event
  - Inter-component length-n coverage
    - Longer sequences from one component to another
    - Bounded by length

Case Study

- Purpose
  - To determine:
    - How many test cases do we need to test WordPad
    - Correlation between event and code-based coverage
- Experimental design
  - GUI: our version of MS WordPad (36 modal windows, 362 events)
  - Hardware platform: 350 MHz Pentium based machine, 256 MB RAM

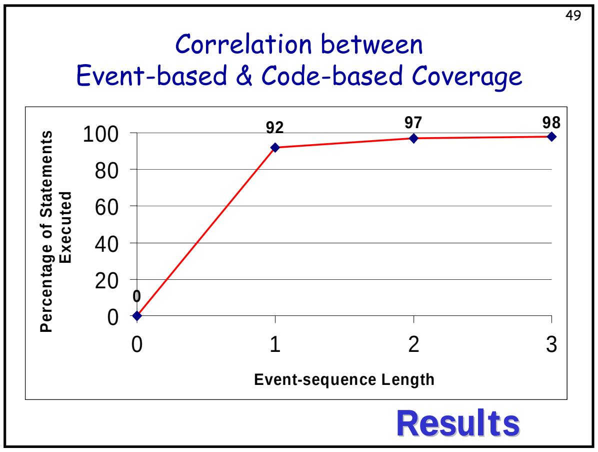
Test Cases for WordPad

	Event-sequence Length							
Component Name	**1'**	**2'**	**1**	**2**	**3**	**4**	**5**	**6**
Main			56	791	14354	255720	4490626	78385288
FileOpen			10	80	640	5120	40960	327680
FileSave			10	80	640	5120	40960	327680
Print			12	108	972	8748	78732	708588
Properties			13	143	1573	17303	190333	2093663
PageSetup			11	88	704	5632	45056	360448
FormatFont			9	63	441	3087	21609	151263
Print+Properties	1	2		13	260	3913	52520	663013
Main+FileOpen	1	2		10	100	1180	17160	278760
Main+FileSave	1	2		10	100	1180	17160	278760
Main+PageSetup	1	2		11	110	1298	18876	306636
Main+FormatFont	1	2		9	81	909	13311	220509
Main+Print+Properties				12	145	1930	28987	466578

Results

Correlation between Event-based & Code-based Coverage

- Code Instrumentation
- Generated all event sequences up to length 3. Total test cases: 21,659
- Executed all 21,659 cases and obtained execution traces
- Statement coverage


49
Correlation between
Event-based & Code-based Coverage
Percentage of Statements Executed
Event-sequence Length
100
80
60
40
20
0
0
1
2
3
0
92
97
98
Results