Application of Execution Pattern Mining and Concept Lattice Analysis on Software Structure Evaluation

Kamran Sartipi

Hossein Safyallah

{sartipi, safyalh}@mcmaster.ca

Dept. Computing and Software

McMaster University

CANADA

McMaster University

Inspiring Innovation and Discovery

SEKE' 06

July 5, 2006

Outline

- Dynamic analysis techniques in Reverse Engineering
- Proposed framework for dynamic analysis using execution pattern mining:
  - Feature-specific task scenarios
  - Program trace generation
  - Program loop elimination
  - Execution pattern generation
  - Identifying core functions using two techniques:
    - Second Pattern Generation
    - Concept Lattice Analysis
- Software structure evaluation
- Case study Xfig
- Conclusion

Application of Dynamic Analysis in Reverse Engineering

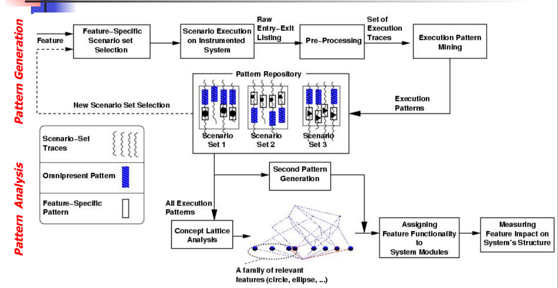
- Existing Dynamic Analysis approaches
  - Execution trace analysis: aspect mining, clustering, performance analysis, program slicing.
  - User-system interaction analysis:recovery of behavior patterns.

We use Dynamic Analysis to:

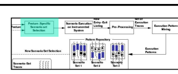
- Identify software functionality (feature) in source code
  - Traditionally, static analysis was used to locate function templates in source code.
  - We generate patterns of execution traces to identify the implementation of software features in source code by the means of task scenarios.
- Incorporate semantics into static analysis
  - Using feature-to-code assignment to find core functionality of the clustering techniques.
  - Providing metrics to evaluate structural properties of software systems.

Proposed Framework:

Dynamic Analysis using Execution Pattern Mining

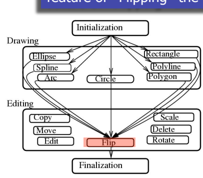


Feature-Specific Scenario Set



A feature is the unit of the system functionality (e.g., flipping a figure) A task scenario defines the user-system interaction in the form of a sequence of software system features (operations) in an informal or semi-formal manner.

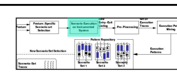
Xfig drawing tool: sample feature-specific scenario set to target Xfig feature of "Flipping" the drawn objects.



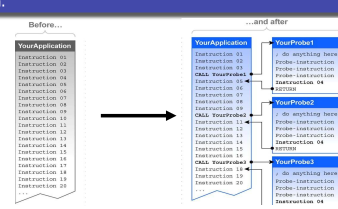
Feature-Specific Scenario Set:

- Start, Draw Ellipse, Flip, Exit
- Start, Draw Spline, Flip, Exit
- Start, Draw Arc, Flip, Exit
- Start, Draw Rectangle, Flip, Exit
- ...
- Start, Draw Polygon, Flip, Exit

Software Instrumentation



Inserting particular pieces of code (probe) into the software's source code or binary image to extract dynamic information from the running system.



We instrument the software system to generate text messages at both entrance and exit of each function, namely Entry/Exit listings.

Preprocessing (cont'd)

Extracted entry-exit listings have lots of redundancies and repetitions caused by program loops that must be eliminated.

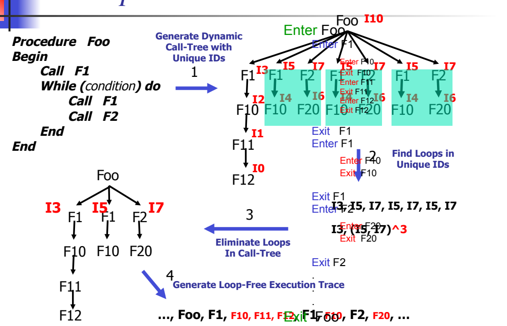
To eliminate program loops:

- Represent the Entry-Exit listing as a dynamic call tree:
  - Nodes represent functions.
  - Edges represent function calls.
  - Assign identical IDs to the nodes with identical sub-trees (nested calls).
- Prune the dynamic call tree by removing multiple instances of nodes with identical sub-trees from top to bottom.
- Generate the execution trace by a depth first traversal on the pruned tree.

Preprocessing: Tree Pruning (cont'd)

- Pruning is a 4 steps process to eliminate loop-based redundancies in a dynamic call tree.
  1. Build a string representation of the sub-tree IDs rooted at each particular node
  2. Extract repetitions from the original string (with repetitions) using a string repetition finder algorithm, *e.g., crochemore.*
  3. Represent the original string in the form of instances of repetitions and their corresponding number of repetitions.
  4. Keep sub-trees that correspond to a single instance of each repetition.

Preprocessing Example



Sequential Pattern Mining

- Given:
  - A group of items (e.g. coke, pen).
  - A group of "transaction sequences", where each transaction sequence belongs to a customer, and the transactions are ordered according to the transaction-time.

Applying a sequential pattern mining on this set of transaction-sequences reveals the common maximum sequences of items.

Interesting relationships among the items can be found.
For example in a computer bookstore, we may find that 10% of the customers first buy a C book then a C++ book and then a Java book.

Sequential Pattern Mining …

Customer Id	Customer-Sequence
1	<(30) (90)>
2	<(10 20) (30) (40 60 70)>
3	<(30, 50, 70)>
4	<(30) (40 70) (90)>
5	<(90)>

Customer-Sequence Version of the Database

Sequential Patterns
<(30) (90)> <(30) (40 70)>

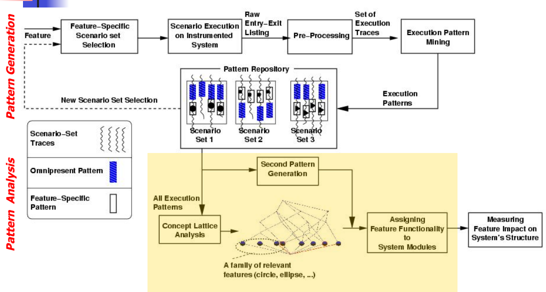
The Pattern Set (minimum support is 2)

Sequential Pattern Mining … (example)

Feature 1	Feature 2	Feature 3
F1, F4, F3, F8, F4, F15 F1, F2, F3, F8, F16, F15 F1, F5, F3, F8, F4, F10, F18, F20 F1, F7, F3, F8, F20, F13, F15 F1, F4, F3, F8, F9, F15 F1, F3, F8, F4, F10, F17, F18, F20 F1, F3, F8, F4, F10, F18, F20	F1, F4, F23, F28, F20 F1, F2, F23, F28, F15 F1, F5, F23, F28, F4, F10, F18, F20 F1, F7, F23, F28, F20, F13, F15 F1, F4, F23, F28, F9, , F4, F10, F15 F1, F23, F28, F4, F10, F17, F18, F20	F1, F4, F33, F38, F4, F15 F1, F2, F33, F38, F16, F15 F1, F5, F33, F38, F15 F1, F7, F33, F38, F20, F13, F15 F1, F4, F33, F38, F9, F15 F1, F9, F33, F38, F10, F15

feature	1	2	3	
Execution Patterns	F1 F15	F1 F15	F1 F15	← Common pattern
	F4, F10 F18, F20	F4, F10 F18, F20		← Noise pattern
	F3, F8	F23, F28	F33, F38	← Feature-specific

Proposed Framework *for*
Dynamic Analysis using Execution Pattern Mining



Identifying Features in Source Code … (Second Pattern Generation)

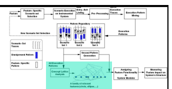
Two categories of execution patterns:

- Feature-specific patterns: core functions that implement the specific feature of a scenario-set.
- Common patterns: sequences of functions that appear in almost every executed scenario (*e.g., system initialization and termination, mouse movement, drawing canvas).*

Second sequential pattern mining is performed to separate two categories of patterns:

- **Step 1:** first sequential pattern mining with **high min-support** extracts both feature-specific and common patterns
- **Step 2:** second sequential pattern mining on the collection of all results of "Step 1" separates two patterns categories.
  - Patterns with **small supports** (e.g., less than %10) are feature-specific.
  - Patterns with **large supports** (e.g., more than %80) are common.

Concept Lattice Analysis



Lattice represents the structure of the relations among entities in a database.

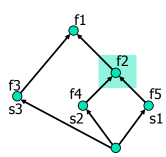
- Concept Lattice is generated from Context Table
- Each lattice node is a concept that may have objects and attributes.
- Every object has all the attributes that appear in that node or all nodes above it.
- Each attribute belongs to all objects that are in that node or every node bellow that node in the lattice.

	f1	f2	f3	f4	f5
s1	X	X			X
s2	X	X		X	
s3	X		X		

context table

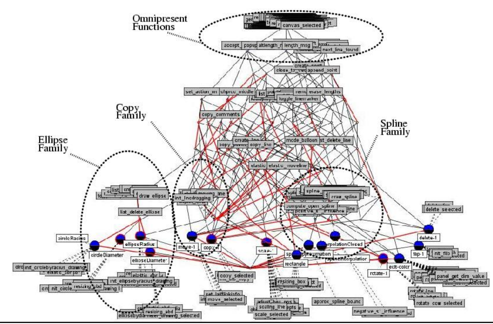
C1 = <{s1, s2, s3}, {f1}>
C2 = <{s1, s2}, {f1, f2}>
C3 = <{s1}, {f1, f2, f5}>
C4 = <{s2}, {f1, f2, f4}>
C5 = <{s3}, {f1, f3}>

concept= <{objects}, {attributes}>



concept lattice

Case Study: Xfig Drawing Tool



Experiments with Xfig Drawing Tool …

The results of execution pattern mining for a collection of 3 different Xfig features.

Feature Family	Specific Feature of Xfig	Number of Different Scenarios	Average Trace Size	Average Pruned Trace Size	Number of Extracted Patterns	Average Pattern Size
Draw Ellipse	Circle-Diameter	10	7234	2600	46	33
	Circle-Radius	10	8143	2463	48	32
	Ellipse-Diameter	10	6405	2536	41	37
	Ellipse-Radius	10	7351	2549	39	35
Copy	Move Objects	4	11887	3166	31	53
	Copy Objects	4	11460	3269	37	50
Draw Spline	Closed Interpolated	10	18635	4434	58	63
	Interpolated	10	15469	4038	66	49
	Approximated	10	15057	5362	61	47

- Characteristics of the proposed technique:
  - Prep-processing (loop elimination) drastically reduces the sizes of the execution traces.
  - Post-processing (*second pattern* or *concept lattice*) reduces the overwhelming number of execution patterns that are generated.

Experiments with Xfig Drawing Tool …

Extracted core functions for Xfig features.

Feature	Extracted Core Functions
Draw Circle	resizing_cbr, elastic_cbr, pw_curve, create_circlebyrad center_marker, create_ellipse, add_ellipse, list_add_ellipse set_lastspline, redisplay_ellipse, ellipse_bound, draw_ellipse overlapping, debug_depth, circlebyradius_drawing_selected
Draw Rectangle	resizing_box, elastic_box, boxsize_msg, create_boxobject create_point, create_line, add_line, box_drawing_selected
Draw Spline	create_spline, make_sfactor, create_sfactor, add_spline last_spline, set_latestspline, redisplay_spline, spline_bound approx_spline_bound, draw_spline, compute_closed_spline
Scale	erase_objecthighlight, init_center_scale, init_scale_line scaling_line, adjust_box_pos, elastic_scalepts, fix_scale_line rescale_points, scale_arrows, scale_arrow, scale_linewidth
Move	init_arb_move, init_move, init_line_dragging set_action_on, elastice_moveline, elastic_links, moving_line place_line, erase_lengths, place_line_x, adjust_pos set_lastposition, set_newposition, move_selected

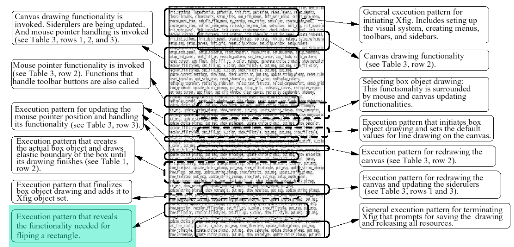
Experiments with Xfig Drawing Tool …

Less visible common Xfig features and their functions

Xfig Functionality	Extracted Core Functions
Side-Ruler Management	set_rulermark, set_siderulermark set_toprulermark, null_proc
Canvase Updating	canvas_exposed, clear_canvas canvase_selected
Mouse Pointer Handling	draw_mousefun_canvas, draw_mousefun clear_mousefun, draw_mousefn2 draw_mousefun_msg, mouse_title
Draw Line	set_line_stuff, x_color, shzoomy, shzoomx

Case Study: Xfig

*The Execution Trace for scenario "Drawing and **Flipping** Rectangle" is Annotated with Descriptions of Execution Patterns.*



Experiments with Xfig Drawing Tool …

Structural cohesion and Functional scattering measures for Xfig & Pine.

Feature Family Φ_ϕ	Contributed File (m)	$\|F_m\|$	$\|F_m \cap F_{\Phi_\phi}\|$	Structural Cohesion $SC_{\Phi_\phi}(m)$	Functional Scattering $FS(\Phi_\phi)$
Ellipse	d_ellipse.c	16	12	75%	57%
	u_elastic.c	67	8	12%	
Copy	e_copy.c	5	3	60%	32%
	e_move.c	4	3	75%	
Spline	d_line.c	9	2	22%	66%
	d_spline.c	6	5	83%	
	u_bound.c	19	2	11%	
	u_draw.c	75	14	19%	

Conclusion

We proposed:

- A pattern based approach to dynamic analysis of software systems that employs data mining techniques to extract functional information out of noisy execution traces.
- A measure of functionality scattering of a feature among structural modules as well as a measure of cohesion for each structural module.
- A method of visualizing the functional distribution of specific features on a lattice using concept lattice analysis.
- The technique deals with scalability, through:
  - Reducing the size of execution traces by eliminating the loop-based repetitions.
  - Reducing large sizes of the loop-free traces using data mining techniques.
- A method for assigning semantics to the static analysis of a software

References

- *"Dynamic Analysis of Software Systems using Execution Pattern Mining"*, H. Safyallah and K. Sartipi, Proceedings of the IEEE International Conference on Program Comprehension (ICPC 2006), pages 84-88. Athens, Greece.
- *"Alborz: An Interactive Toolkit to Extract Static and Dynamic Views of a Software System"*, K. Sartipi and L. Ye and H. Safyallah, Proceedings of the IEEE International Conference on Program Comprehension (ICPC 2006), pages 256-259. Athens, Greece.
- *"Application of Execution Pattern Mining and Concept Lattice Analysis on Software Structure Evaluation"*, K. Sartipi and H. Safyallah, International Conference on Software Engineering and Knowledge engineering (SEKE 2006). San Francisco Bay
- *"An Orchestrated Multi-view Software Architecture Reconstruction Environment"*, K. Sartipi and N. Dezhkam and H. Safyallah. IEEE International Working Conference on Reverse Engineering (WCRE 2006), Benevinto, Italy.

Application of Execution Pattern Mining and Concept Lattice Analysis on Software Structure Evaluation

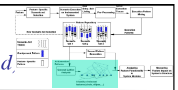
Kaman Sartipi
Hossein Safyallah
{sartipi, safyalh}@mcmaster.ca

Dept. Computing and Software
McMaster University
CANADA

McMaster University
Inspiring Innovation and Discovery

SEKE' 06
July 5, 2006

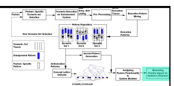
Pattern Analysis:
Concept Lattice Analysis (cont'd)



- An object is a targeted feature ϕ of a feature-specific scenario set $\mathcal{S}_\phi$.
- An attribute is a function that participates in the execution patterns within $\mathcal{S}_\phi$.
- A feature-specific concept c_ϕ is concept with a single object (feature) ϕ . We define F'_ϕ o be the set of functions that appear on c_ϕ .
- A logical module F_{Φ_ϕ} is the set of functions that implement feature family Φ_ϕ .

$$F_{\Phi_\phi} = \bigcup_{\varphi \in \Phi_\phi} F'_\varphi$$

Structural Evaluation



- Let $M_{\Phi_\phi} = \{m_1, m_2, \ldots, m_k\}$ be the set of modules where all the functions in F_{Φ_ϕ} are defined in elements of M_{Φ_ϕ}.
- Let F_m to the set of functions that are defined in modules m .

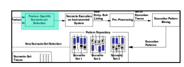
Structural cohesion of module m with respect to feature Φ_ϕ family $SC_{\Phi_\phi}(m)$,is def

$$SC_{\Phi_\phi}(m) = \frac{|F_m \cap F_{\Phi_\phi}|}{|F_m|}$$

Functional scattering of feature family Φ_ϕ , namely $FS(\Phi_\phi)$,is defined as:

$$FS(\Phi_\phi) = 1 - \frac{\sum_{m \in M_{\Phi_\phi}} SC_{\Phi_\phi}(m)}{|M_{\Phi_\phi}|}$$

Formal Definitions:
Scenario, Feature



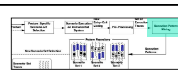
- A software *feature* ϕ (of type text) is a unit of software requirements that describes a single system functionality.
- *A scenario* is modeled as a sequence of features, as: $s = [\phi_1, \phi_2, \ldots, \phi_n]$
- A *feature-specific scenario set* $\mathcal{S}_\phi$ is a set of scenarios that all share a specific feature:

$$\mathcal{S}_\phi = \{s \mid s \in \mathcal{S} \wedge \exists \phi' \in s \bullet \phi' = \phi\}.$$

where S is the set of all system scenarios.

- A *feature family* Φ_ϕ s a set of semantically relevant features to specific feature ϕ .

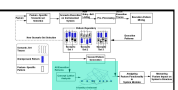
Formal Definitions:
Execution Pattern Mining



- Let $\mathcal{F}$ be the set of all function names in the subject software system.
- *Execution trace* $\mathcal{T}$ is a sequence of function names from $\mathcal{F}$.
- Let *Repository* R_{S_ϕ} be the set of all extracted traces according to the execution of task scenarios in feature-specific scenario set S_ϕ.
- An *execution pattern* $p \in \mathcal{T}$ is defined as a contiguous sequence of functions from $f \in \mathcal{F}$ that is supported by at least *MinSupport* number of the execution traces in the repository R_{S_ϕ} .
- An *execution trace* t *supports* execution pattern p iff p is a subsequence of t .

Each *execution pattern* extracts the sequence of functions that implement a common functionality within each feature-specific scenario set S_ϕ.

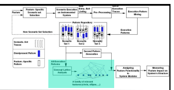
Concept Lattice Analysis



$\mathcal{R}$ $\mathcal{O}$ $\mathcal{A}$

- Provides lattice representation for the binary relation ***R*** between *objects* ***O*** and their *attribute-values* ***A*** .
- Provides a means for clustering objects based on their common attributes.
- Provides a separation method for attributes based on their sharing level.

Concept Lattice Analysis
(cont'd)



- In the binary relation $\mathcal{R}$ between Objects $\mathcal{O}$ and attributes $\mathcal{A}$:
  - The triple $\mathcal{C} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$ is called a formal context.
  - For any set of objects $O \subset \mathcal{O}$, we define $shared_A(O)$ as the set of shared attributes among objects in O.
  - For any set of attributes $A \subset \mathcal{A}$, we define $shared_O(A)$ as the set of objects whose sharing all attributes in A.
  - Concept *C* is defined as the a pair $c = < O, A >$ such that:

$$O = shared_O(A) \;\; \wedge \;\; A = shared_A(O)$$