Detecting Patterns in Traces of Procedure Calls

Abdelwahab Hamou-Lhadj & Timothy Lethbridge

{ahamou, tcl}@site.uottawa.ca

University of Ottawa - Canada

Workshop on Dynamic Analysis (WODA) May 2003
Co-located with ICSE 2003
Portland, Oregon

Objective

- Execution traces are important to understand the *behavior* and sometimes the *structure* of a software system
- Execution patterns can bridge the gap between *low level system components* and *high level domain concepts*
  - And hence help program comprehension
- We need:
  - an efficient way for *detecting* these patterns.
  - to understand when two patterns can be considered as *equivalent*
- We focus on patterns of procedure calls

Why Traces of Procedure Calls?

- The abstraction level of traces of procedure calls seems well suited for program comprehension.
  - Applies to methods in OO as well

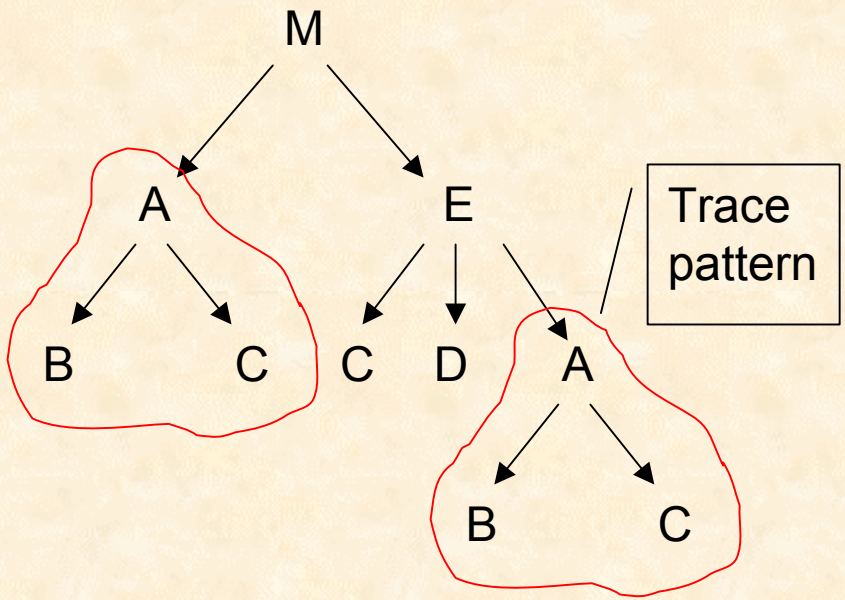
- Alternatives:
  - Inter-process messaging
    - Not a tree structure
  - Statement level
    - Vastly more detail

Definition of a Trace Pattern

- "A sequence of calls that occurs repetitively but non-contiguously in several places in the trace"
  - Zayour and Lethbridge
- We add: instances of a given pattern do not need to be identical
- Ideally, a trace pattern corresponds to an abstract domain concept.
  - E.g. a user identifiable aspect of some feature
  - But reality is far from the ideal

Trace Structure

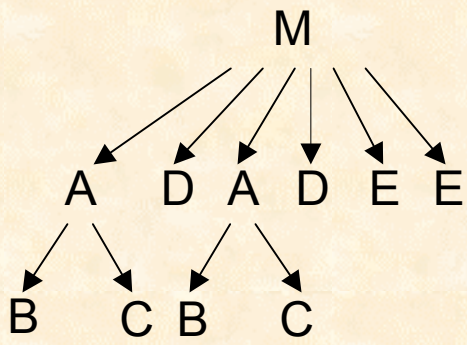
- A trace of procedure calls from a single thread is represented by a *rooted labeled ordered tree*
- A trace pattern is represented as a non-contiguous repeated subtree.



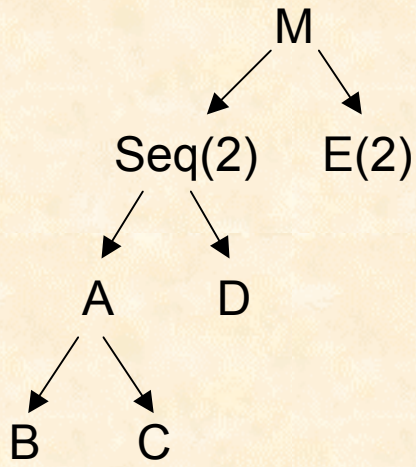
- Tree representation of the trace:
  M(A(B, C), E(C, D, A(B, C)))

Trace Preprocessing

- Sequences of calls due to loops and recursion encumber the trace
- They need to be removed
- The tree structure is maintained by adding virtual calls



- Trace with repetitions due to loops

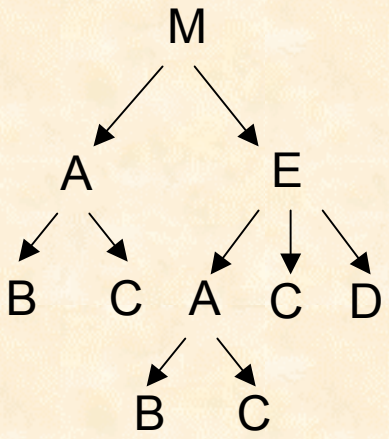


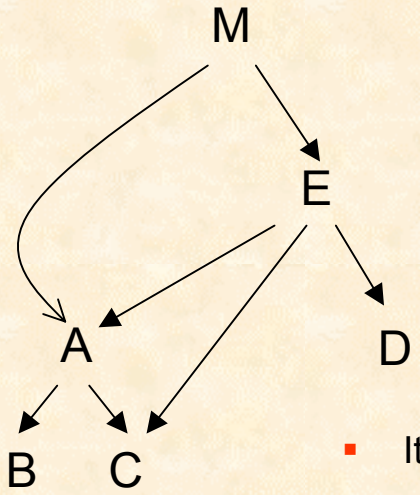
- After removing repetitions

Pattern Extraction

- Based on Valiente's algorithm for the common subexpression problem
- (The problem was introduced by J.P. Downey, R. Sethi and R.E. Tarjan)
- Any rooted tree can be transformed into its most compact form by representing common subtrees only once

- A rooted labeled ordered tree



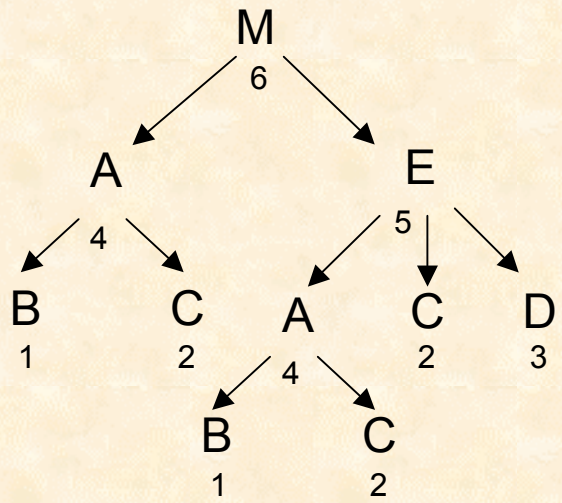


- Its compact form

Valiente's Algorithm

- Traverse the tree in a bottom-up fashion
- Assign a positive integer between 1 and n
  - n represents the size of the tree
- Two nodes *n1* and *n2* are assigned the same *certificate* if the trees rooted at them are similar according to *predefined matching criteria*
- To compute the certificate, each node is assigned a *signature*
- The signature of a node *n* consists of its label and the certificates of its direct children, if there are any.
  - This makes it unique
- A global hash table is used to store the certificates and signatures and ensure that similar subtrees will always hash to the same element.
- Our contribution: Examine the matching criteria

Example



Certificate	Frequency	Signature
1	2	B
2	3	C
3	1	D
4	2	A 1 2
5	1	E 4 2 3
6	1	M 4 5

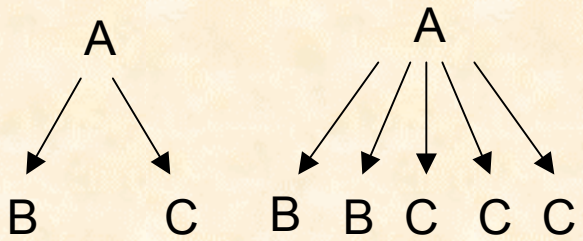
- If we consider exact match only, identical subtrees have the same certificates
- Global table that corresponds to this tree.
- We mainly traverse the table and extract the subtrees (excluding the leaves) that have more than 1 incoming edge.
- E.g. ‘A 1 2’ is a pattern

Equivalent Patterns

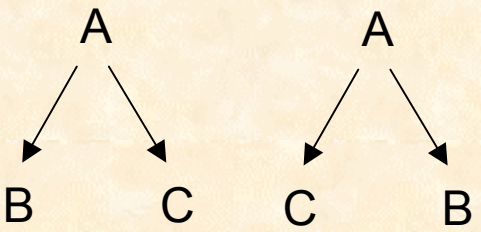
- Pattern matching criteria determine when two sequences of calls can be considered equivalent
  - Will effect usefulness and performance
- Users can select the criteria according to their knowledge of the system
  - E.g. identical patterns might be useful to novices but less useful to experts.
- De Pauw, Lorenz, Vlissides and Wegman suggested a list of matching criteria that are used for OO systems.
  - Some of them are also useful for procedural systems.

De Pauw et al. Pattern Matching Criteria Useful For Procedural Systems

- **Identity**: Two sequences of calls are similar if they have the same topology:
  - Same call structure
  - Same order of calls
  - Etc.
- **Ignoring Repetition**: repetition due to loops and recursions can be ignored when looking for patterns
- **Ignoring Ordering**: Order of calls might not be important at some levels of the call tree.
- **Depth limited**: allows comparing two subtrees up to a certain depth.
  - Deeper calls ignored



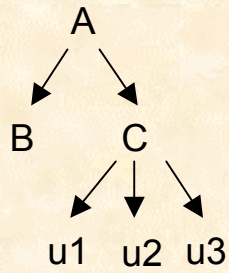
- These two sequences are similar if repetitions are ignored

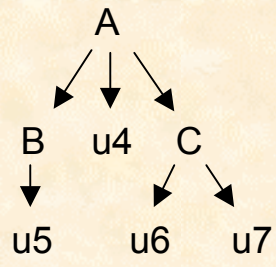


- Two sequences are similar if the order is ignored

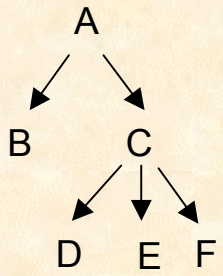
New Matching Criteria

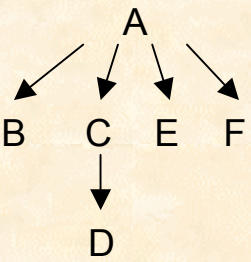
- **Removing Utilities**: Requires automatic detection of utility routines
- **Distance Limited**: Trees within a certain edit distance are considerd the same.
- **Flattening**: Ignore structure of tree
  - consider only routines encountered
  - might be useful for experts who are not interested in the call structure





- These patterns are considered equivalent if utility routines (u_i) are ignored





- The edit distance can reveal that these two sequences can be considered as the same pattern

Conclusions and Future Work

- We need to validate the *matching criteria* and analyze at which level of the tree they can be applied usefully
- We also need to study how they can be *combined*.
- It is also important to understand the relationship between the use of the matching criteria and the user's knowledge of the system
- Goal: automatic detection of patterns that most likely correspond to high level concepts.

Why this work will fail!

- We might have trouble finding ways for the criteria to work together
- There might be no intuitive meaning attributable to edit distance in the general context
- We might not find a universally appropriate definition of a ‘utility’

Why this work will succeed!

- We have already done some experiments in compressing traces
- High interest from SEs in browsing smaller traces
- We have been told by SEs that our proposed matching criteria correspond to what people want