June 18, 2018

Efficient multi-graph or rooted subgraph matching via merging

Alexandre Boulgakov

Efficient multi-graph or rooted subgraph matching via merging

ABSTRACT

The problem of matching one or more graphs or subgraphs with a non-deterministic finite automaton (NFA) is of importance in various domains of computer science. An example is regular-expression matching, which can be formulated as a graph-matching problem. Current techniques of matching multiples graphs or subgraphs against NFAs have relatively high computational complexity. This disclosure presents matching techniques with complexity that is linear in the size of the graph. A single, larger, graph is constructed from the set of graphs to be matched. The larger graph is considered as an NFA. A single-NFA to single-NFA matching procedure of linear space and time complexity is employed to match the larger graph against the NFA. Overlaps between the set of graphs or subgraphs are leveraged to produce gains in computational efficiency.

KEYWORDS

- Multiple graph matching
- Non-deterministic finite automaton (NFA)
- Rooted subgraph
- Regex matching

BACKGROUND

Certain problems in computer science (e.g., regular-expression matching and similar problems) are based on matching a series of rooted, edge-labeled graphs with a non-deterministic finite automaton (NFA). Specifically, for each of given series of rooted, edge-labeled graphs, a

determination is made if any sequence of edge labels on any possible rooted path through the graph is matched by a given NFA.

A recent paper [1] describes techniques to determine a match between a single, rooted, edge-labeled graph and a given NFA. The techniques determine match between the graph and NFA by considering the graph as an NFA with all states accepting. A synchronized product of the two NFAs is defined such that the problem of determining match of single graph to NFA is equivalent to determining whether the product-NFA has a non-empty acceptance set. The problem of matching single graph to NFA is thus reduced to a reachability problem solvable in linear time and space, e.g., using a breadth-first or depth-first search.

DESCRIPTION

This disclosure presumes the existence of techniques, e.g., as described in [1], that determine match between a single graph and a given NFA, and applies such techniques to the problem of matching a multigraph or rooted subgraphs to a given NFA.

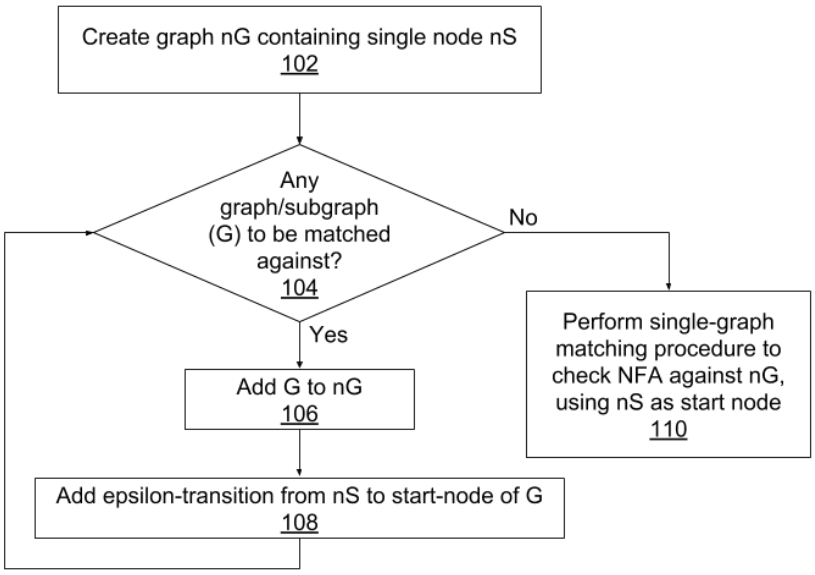

Fig. 1: Matching an NFA against multiple graphs or subgraphs

Fig. 1 illustrates matching of an NFA against multiple graphs or rooted subgraphs simultaneously. A new graph nG is created with a single node nS (102). For each graph or rooted subgraph G that the NFA is to be matched against (104), G is added to nG (106). An epsilon transition is added from nS to the start node of G (108). In this manner, a single, larger graph nG is constructed comprising the given series of graphs. A single-graph-to-NFA matching procedure (e.g., as described in [1]) is performed to match the NFA against nG using nS as the start node of nG (110).

Gains in computational efficiency accrue when the graphs comprising nG overlap. An overlap in two or more constituent graphs or subgraphs of nG causes the overlapping nodes to be processed just once. This overlap can be introduced by applying techniques like bisimulation after unifying all graphs into the larger graph nG, e.g., just before (110).

The overlap between constituent graphs often arises naturally, as illustrated by the following examples.

Example 1: A regex match is to be performed against all suffixes (`foobar`, `oobar`, `obar`, etc.) of a given string `foobar`. The traditional approach, of determining match against each suffix `foobar`, `oobar`, etc., is computationally expensive. In contrast, per the techniques of this disclosure, the string `foobar` is mapped to the graph nG, with the node nS having an epsilon transition to each node within nG. In this case, the overlap between constituent subgraphs occurs naturally, and the matching problem is solved within time and space complexity that is linear in string length.

Example 2: Given a control flow graph of a binary executable file, a syscall graph is desired together with a set of entry points into the graph. In this case, the single control flow graph is

mapped to the graph nG, and each entry point is recorded as the root of a rooted subgraph. Again, overlap naturally arises between the rooted subgraphs.

The described techniques apply to problems involving matching multiple graphs or subgraphs against another graph, or generally for matching problems in any domain where data is expressed as graphs. For example, the techniques find use in matching specifications of a particular behavior of interest against a superset of behaviors extracted from an executable file of interest using static analysis. Another application of the techniques, e.g., within the domain of computer security, is to match the same behavior specification against the observed behavior of an application when run on a test system.

CONCLUSION

This disclosure presents techniques to match one or more graphs or subgraphs with another graph or non-deterministic finite automaton (NFA). The techniques are of relatively low complexity, e.g., linear in the size of the graphs and NFA. The techniques have various applications, e.g., matching of particular behaviors of binary executables against a superset of behaviors, regular-expression (regex) matching, etc. A single, larger, graph is constructed from the set of graphs or subgraphs to be matched. The larger graph is considered as an NFA. A single-NFA to single-NFA matching procedure of linear space and time complexity is employed to match the larger graph against the NFA. Overlaps between the set of graphs or subgraphs are leveraged to produce gains in computational efficiency.

REFERENCES

[1] Boulgakov, Alexandre, “Matching a graph with a non-deterministic finite automaton,” Technical Disclosure commons, (June 14, 2018) https://www.tdcommons.org/dpubs_series/1246

[2] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, A. W. Roscoe, “FDR3: A parallel refinement checker for CSP,” *Int. J. Softw. Tools Technol. Transfer (2016) 18: 149* (https://doi.org/10.1007/s10009-015-0377-y)

[3] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, A. W. Roscoe, “Modelling and analysis of security protocols: the CSP approach,” Addison-Wesley (2001)

[4] A. Boulgakov, T. Gibson-Robinson, A. W. Roscoe, “Computing maximal weak and other bisimulations”, *Formal Aspects of Computing 28(2)* pp. 381-407

[5] A. Boulgakov, “Improving scalability of exploratory model checking.” PhD thesis, University of Oxford 2016. Available online at https://ora.ox.ac.uk/objects/uuid:76acb8bf-52e7-4078-ab4f-65f3ea07ba3d

[6] A. W. Roscoe, “The theory and practice of concurrency,” Prentice Hall NJ (1997)