45. Coverability for Parallel Programs

Lenka Turoňová

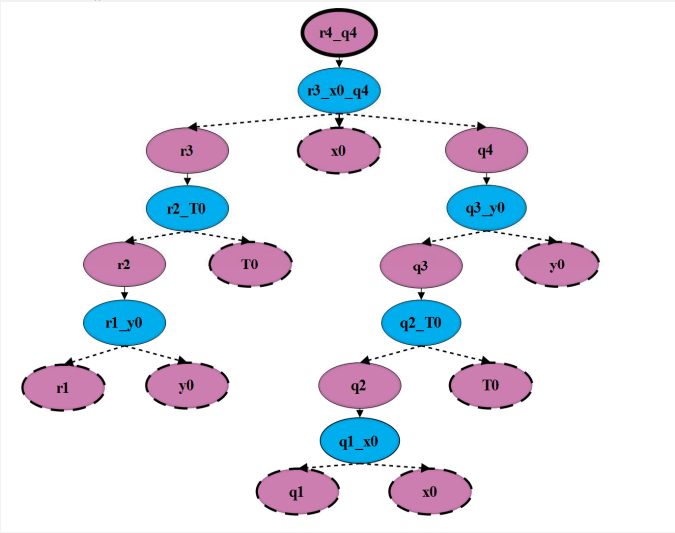
Faculty of Information Technology, Brno University of Technology

ABSTRACT

We improve existing method for the automatic verification of systems with parallel running processes. The technique is based on an effort to find an inductive invariant. We use a variant of counterexample-guided refinement (CEGAR). The effectiveness of the method depends on the size of the invariant. In this paper, we explore the possibility of improving the method by focusing on finding the smallest invariant.

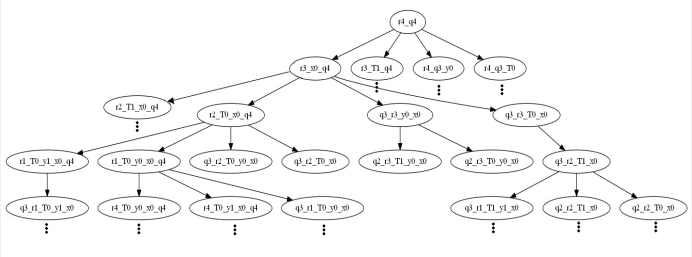
METHODOLOGY

Our method is built upon the algorithm presented in the article [1] (GBR) which is based on inspecting counterexample runs within an abstract domain. From each counterexample run, the method infers a part of an inductive invariant. This part of invariant is then used to refine the abstract domain. We represent the invariant expressed by a conjunction of clauses as a set of „words".



In each iteration of CEGAR, the algorithm performs the forward search followed by the backward analysis trying to find configurations leading to the set of incorrect configurations. Subsequently, these configurations are included in the parameter. When the domain parameter contains a whole inductive invariant, the abstraction becomes precise enough so that the system can be verified.

Starting from the incorrect configurations (*q4r4*) we explore their predecessors (*q4r3x0*). They are obtained by applying the rules in a reverse way. First, each predecessor is abstracted (a set {*q4*, *r3*, *x0*}). We expressed it with a set of minimal configurations from the parameter. We obtain the relevant predecessors of the configuration that represent preconditions necessary to reach the set of the incorrect configurations.



A set of all predecessors and their subwords is created and one of the configurations is selected. It is added to the parameter of the abstract domain as a part of the inductive invariant. The forward run is performed. If the set of incorrect configurations is reached then the configurations from the parameter form the whole invariant since it excludes all counterexample paths.

EXPERIMENTS

Based on the method, we have implemented a prototype in Python to find invariants for checking safety properties for parameterized programs communicating via shared variables and mutexes. Our tool uses the input format of *mist2*. We have implemented also a version of the algorithm (GRB) and compare the performance of the algorithm with our implementation in experiments on a set of Petri net benchmarks.

Name	GRB		My approach		Reduction
	Size	Iter.	Size	Iterar.	
basicME	45	5	22	19	44%
rw	331	8	36	35	90%
Peterson	135	8	107	21	21%
newrtp	45	9	54	53	-20%
pingpong	80	10	28	27	65%

Table 1 presents results for both algorithms. Column on the left shows the benchmark name while columns on the right show a size of invariant and the number of CEGAR iterations. The last column shows the percentage reduction in the size of the invariant.

The results demonstrate that our approach discovered the possibility of finding the smaller invariant than which was determined with the method GRB. However, the price for smaller invariant is a greater number of iterations required to find it since each iteration adds only one word to the parameter. The previous method adds many words at each iteration so that less iterations were necessary to performed, however, many of them are useless and not optimal.

CONCLUSIONS

We presented an approach which is built upon the existing method which runs a program in an abstract domain to find the invariant for verification. The domain is refined using CEGAR to obtain overapproximation which is precise enough so that the system can be verified.

While the existing method represented uses the technique BFS our method determines the invariant based on backtracking. We obtain configurations representing preconditions necessary for reaching the set of incorrect configurations. In each iteration, randomly one configuration is added to the parameter of the abstract domain.

Based on the method, the prototype in Python was implemented and tested on the set of Petri net benchmarks. The results show that our approach in some cases reduces the size of the invariant.

We proved that there is a possibility to find the smaller invariant than which was discovered using method GRB. The price for smaller invariant is an increase in the number of iterations required to find it. Since our implemented method is unable to find the invariant for all the benchmarks future research will focus on developing a method which would find always the smaller invariant and in the best case, find the smallest one.

REFERENCES

[1] GANTY, P., RASKIN, J.F., VAN BEGIN, R. *A complete abstract interpretation framework for coverability properties of wsts. Verication, Model Checking, and Abstract Interpretation*, 3855, 2006.

ACKNOWLEDGEMENT

I would like to thank my supervisor Mgr. Lukáš Holík, Ph.D. for his continuous support, teaching me how to deal with the technical difficulties and giving me valuable advice. He has always been approachable and his attitude towards my efforts has been kind, encouraging, and constructive. I am thankful to him for the numerous hours he spent on answering every single question I had.