# 《Build Code Analysis with Symbolic Evaluation》Review

## Paper Info

Build Code Analysis with Symbolic Evaluation

Ahmed Tamrawi, Hoan Anh Nguyen, Hung Viet Nguyen, Tien N. Nguyen

## Major Contribution

In this paper, the authors introduce a build code analysis and refactoring tool for makefiles, named SYMake. This tool can extract the dependency relations between target files and prerequisites, and detect several code smells in makefiles, including cyclic dependency, duplicate prerequisites and rule inclusion. Besides, it can also support some refactoring operations. However, refactoring operations are not our research point in our work. Their work is important, and it is the only work aimed at code smells detection in makefiles. Hence some key insights can be referred to our own works.

In my personal view, the shining points of this work include the construction of SDG(Symbolic Dependency Graph), V model and T model. Based on these models, the tool can extract the dependency relations. The analysis is processed on these abstract models. Although the authors focus different code smells from us, their techniques are also useful in our work.

## Main Work

In this paper, the authors design a tool for code smells detection in makefiles. The tool can handle some typical bugs in makefiles, such as cyclic dependency relations, duplicate prerequisites and rule inclusion. In order to extract the dependency relation, the authors design some models to store the information in the makefiles, including SDG, V model and T model.

### Makefile Symbolic Dependency Graph(SDG)

A SDG captures the dependencies among prerequisites and targets, and respective recipes. In order to capture the dependencies in details, each SDG's node refers to a data structure, called V-model, representing a symbolic string value for a name and containing symbols to represent the inputs or data retrieved from user environment. One example of SDG and V-model is shown as follows.

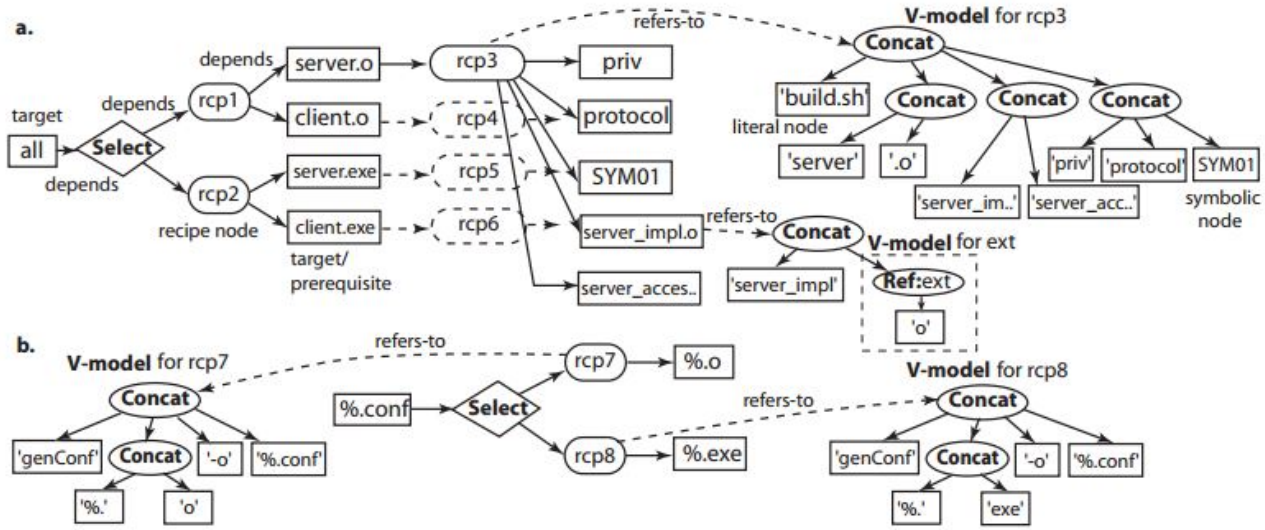
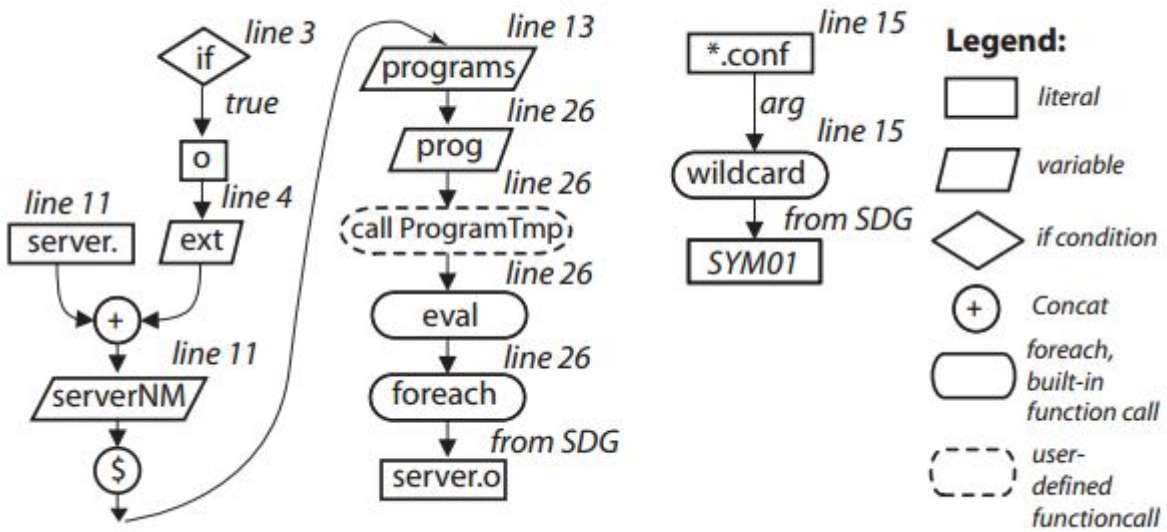



Figure a is a SDG which models the dependency relations in the makefile. Figure b is the V model which responds to a specific node in the SDG. In fact, V model stores the string computation information of a single node in the SDG. Based on the SDG and V model, we can obtain all the dependency relations and the concrete names of the targets and prerequisites.

## Evaluation Trace Model

A T-model is similar in spirit to an execution trace. It is a labeled, directed, and acyclic graph representing how a SDG node's value is computed and manipulated through different Makefile's program elements at an evaluation point. The following figure shows an example of a T-model.




In the real world development of makefiles, developers often use scripts to generate the static makefiles. Some string values, which are the names of targets or prerequisites, are generated in the process of computation dynamically. When developers alter some values in the scripts, the results might change. Hence, it is important to capture the dynamic property of the makefiles. T model can achieve this goal.

## Application

Based on the SDG, V-model and T-model, the authors conduct several experiments and design some applications.

### Code Smell Detection

Code smells which can be detected in this work include cyclic dependency, loop of recursive variables, duplicate prerequisites and rule inclusion. They are mainly based on the SDG and V-model.

For example, to detect cyclic dependency, they use a graph algorithm to detect a directed cycle that goes through targets, prerequisites, recipes via dependency edges. This code smell is also concerned in our work.

Because these models capture the major semantic information in the makefiles, all meaningful smell detection can be processed upon these models. In our own work, we are concerned about the inconsistencies in the makefiles. Some code smells in this work, such as the rule inclusion, are not in our research domain.

### Refactoring Support

In real work application development, we often need to re factor a make system. The authors design an application to support automatic renaming, where SYMake needs to find all code locations where a variable was initialized or referenced. T-models are created and updated over time in this application. However, this application is not in our research domain.

Because T model is used to capture the dynamic generation of the static makefiles in order to support refactoring, it is useless in our work. We only need to construct a structure similar to the SDG and V model in our work.

## Future Work

This paper is the first paper we have found to detect code smells in the makefiles systematically. The techniques in this work are important and meaningful. Some insights in this paper is similar to the tool APMAKE, which is also aimed at code smells detection in makefiles.

However, this work has some limitations. The tool can not extract the actual dependency relations when the makefile is being executed. Some methods in APMAKE can be used to make up this weakness, and the dynamic SDG can be obtained by monitoring the file access when a single statement in the makefile is being executed. This part in our work has been implemented by Gang Fan. Anyway, we can also refer to the construction of SDG in this paper to obtain concrete dependency graph (CDG), and compare CDG with the actual dynamic SDG to find the inconsistencies in the makefiles.

Some techniques in this work are not needed in our work. For example, T-model is used to evaluate the node's value in V-model in order to support renaming. However, in our work, this is not necessary.