

Static Analysis of JAVA Programs in a Rule-Based Framework¹

M. Alpuente M. A. Feliú C. Joubert A. Villanueva²

*Universidad Politécnica de Valencia, DSIC / ELP
Camino de Vera s/n, 46022 Valencia, Spain*

Abstract

This paper presents a practical JAVA program analysis framework obtained by combining an extended verification toolbox with a JAVA virtual machine. In our methodology, rules are used to specify complex interprocedural program analyses involving dynamically created objects. After extracting an initial set of information about JAVA program semantics from the program Bytecode, our framework transforms the rules of a particular analysis into a Boolean Equation System (BES), whose local resolution corresponds to the demand-driven computation of program analysis results.

1 Introduction

Static program analysis extracts program semantics information from code, without running it. An example of such an analysis to study the data-flow dependencies of a program is the definition-use analysis. An abstract representation of the program containing the variable definitions and uses at each program statement is built, on which the analysis is solved.

In this paper, we focus on static *reference* analyses of JAVA programs and generally speaking, of any object-oriented programming languages, which are characterized by data abstraction, inheritance, polymorphism, dynamic binding of method calls, dynamic loading of classes, and querying of program semantics at runtime through reflection. A reference analysis, also called *points-to* analysis, determines information about the set of objects to which a reference variable or field may point during program execution. There is a real interest in using such an analysis in program understanding tools (e.g. semantics browsers or program slicers), in software maintenance tools and also in testing tools using coverage metrics.

Recently, various rule-based specifications for a large number of program analyses have been developed using a simple relational query language, called Datalog [1]. This language, based on declarative rules to both describe and query a deductive database, is rich enough to describe complex interprocedural program analyses involving dynamically created objects. This paper presents a fully automatic and efficient demand-driven evaluation framework for Datalog queries, based on a local Boolean equation system (BES) resolution [2]. The system, called DATALOG-SOLVE, has been developed within the CADP verification toolbox [4] and connected to the JOEQ virtual machine [5] in order to detect errors, like non-satisfaction of the query, in JAVA programs at compile time.

¹ This work has been partially supported by the EU (FEDER), the Spanish MEC/MICINN, under grant TIN 2007-68093-C02, and the Technical University of Valencia, under grant PAID-06-07 (TACPAS).

² Email: {alpuente,mfeliu,joubert,villanue}@dsic.upv.es

2 Datalog specification of a program analysis

The Datalog approach to static program analysis [6] can be summarized as follows. Each program element, namely variables, types, code locations, function names, are grouped in their respective *domains*. By considering only finite program domains, Datalog programs are ensured to be *safe* (query evaluation generates a finite set of facts). Each program statement is decomposed into *basic program operations*, namely load, store, assignment, and variable declarations. Each kind of basic operation is described by a relation in a Datalog program. A program operation is then described as a tuple satisfying the corresponding relation. In this framework, a *program analysis* consists in either querying extracted relations or computing new relations from existing ones.

```

### Domains
V 262144 variable.map
H 65536  heap.map
F 16384  field.map

### Relations
vP_0 (variable : V, heap : H)      inputtuples
store (base : V, field : F, source : V) inputtuples
load  (base : V, field : F, dest : V)  inputtuples
assign (dest : V, source : V)         inputtuples
vP    (variable : V, heap : H)        outputtuples
hP    (base : H, field : F, target : H) outputtuples

### Rules
vP (v, h)      :- vP_0 (v, h).
vP (v1, h)     :- assign(v1, v2), vP (v2, h).
hP (h1, f, h2) :- store(v1, f, v2), vP (v1, h1), vP (v2, h2).
vP (v2, h2)    :- load (v1, f, v2), vP (v1, h1), hP (h1, f, h2).

```

Fig. 1. Datalog specification of a context-insensitive points-to analysis

Example 2.1 Consider the Datalog program that defines context-insensitive points-to analysis (`pa.datalog` [6]) given in Fig. 1. The program consists of three parts:

- (i) A declaration of *domains* where domain names and sizes (number of elements) are specified.
- (ii) A list of *relations*, *i.e.*, atoms, specified by a predicate symbol, its arguments over specific domains and whether it is derived from an applicable Datalog rule (value `outputtuples`), or extracted from the program Bytecode (value `inputtuples`).
- (iii) A finite set of Datalog *rules*, defining the `outputtuples` relations.

The example of Datalog program analysis given in Fig. 1 consists in inferring possible points-to relations from local variables and method parameters in domain V to heap objects in domain H as well as possible points-to relations between heap objects through field identifiers in domain F .

Datalog constraints are declared as sets of tuples, *i.e.*, `inputtuples` relations. For example, the relation `vP_0` consists of initial points-to relations (v, h) of a program, *i.e.*, `vP_0 (v, h)` is true if there exists a direct assignment within the program between a reference to a heap object $h \in H$ and a variable $v \in V$ (*e.g.*, `v = new`

`String()` statements in JAVA). Other Datalog constraints such as `store`, `load` and `assign` relations are calculated similarly. Each Datalog rule then models the effect of one of these input relations over the heap.

Finally, a Datalog query consists of a set of goals over the relations defined in the Datalog program, *e.g.*, $:- \text{vP}(x, y)$. where x and y are variable arguments of vP . This goal aims at computing the complete set of variables x that may point to any heap object y at any point during program execution.

3 BES evaluation of a Datalog query

Our Datalog query evaluation framework (see Fig. 2), called `DATALOG_SOLVE`, takes three inputs: a domain definition (file `.map`), a set of Datalog constraints (*i.e.*, a set of *facts*, file `.tuples`), and a Datalog query $q = \langle G, R \rangle$ (file `.datalog`), where R is a Datalog program (a finite set of Datalog rules), and G is the set of *goals* (Datalog rules with empty head). The domain definition states the possible values for each predicate’s argument in the query. Datalog constraints represent the program information relevant for the analysis. Both, domain definitions facts are automatically extracted from program Bytecode by the JOEQ compiler [5].

As in [6], we assume that Datalog programs have stratified negation (no recursion through negation), and totally-ordered finite domains, without considering comparison operators.

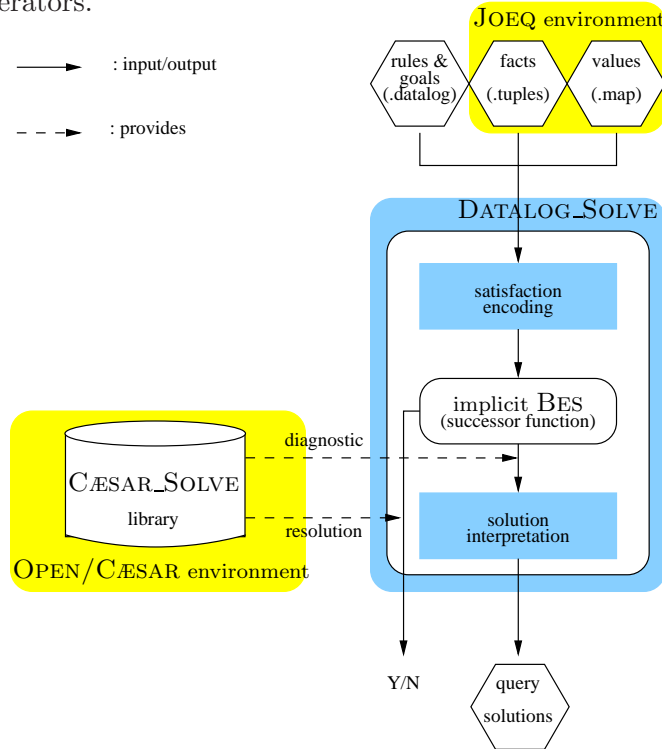


Fig. 2. JAVA program analysis using `DATALOG_SOLVE` framework

Our `DATALOG_SOLVE` system (120 lines of LEX, 380 lines of BISON and 3 500 lines of C code) proceeds in two steps: 1) translation of the Datalog query to BES, 2) generation and interpretation of the solutions to the query.

Example 3.1 Consider the Datalog program given in Fig. 1 that defines context-insensitive points-to analysis (`pa.datalog` [6]). The BES transformation of the Datalog-based program analysis for the goals $vP(x, y)$ and $hP(z1, w, z2)$ consists in the following equation system:

$$\begin{aligned}
 x_0 &\stackrel{\mu}{=} vP(x, y) \vee hP(z1, w, z2) \\
 vP(v : V, h : H) &\stackrel{\mu}{=} vP_0(v, h) \vee (assign(v, v2) \wedge vP(v2, h)) \\
 &\quad \vee (load(v1, f, v) \wedge vP(v1, h1) \wedge hP(h1, f, h)) \\
 hP(h1 : H, f : F, h2 : H) &\stackrel{\mu}{=} store(v1, f, v2) \wedge vP(v1, h1) \wedge vP(v2, h2)
 \end{aligned}$$

Boolean variable x_0 encodes the set of Datalog goals whereas (parameterised) boolean variables $vP(v : V, h : H)$ and $hP(h1 : H, f : F, h2 : H)$ represent the set of Datalog rules in the program.

The back-end of our system carries out the demand-driven generation, resolution and interpretation of the BES by means of the generic `CESAR_SOLVE` library of `CADP` [4], devised for local BES resolution and diagnostic generation.

The tool takes as a default query the computation of the least set of facts that contains all the facts that can be inferred using the rules defining the program analysis. This represents the worst case of a demand-driven evaluation, where all the information derivable from a Datalog program is computed.

4 Experimental Results

The `DATALOG_SOLVE` framework was applied to a number of `JAVA` programs by computing the context-insensitive pointer analysis described in Fig. 1.

Table 1
Description of the `JAVA` projects used as benchmarks.

Name	Description	Classes	Methods	Bytecodes	Vars	Allocs
frets	speech synthesis system	215	723	46K	8K	3K
nfcchat	scalable, distributed chat client	283	993	61K	11K	3K
jetty	server and servlet container	309	1160	66K	12K	3K
joone	Java neural net framework	375	1531	92K	17K	4K

To test the scalability and applicability of the transformation, we applied our technique to 4 of the most popular 100% `JAVA` projects on Sourceforge that could compile directly as standalone applications. These projects were also used as benchmarks by the `BDDBDDB` system [6], one of the most efficient deductive database engine, based on binary decision diagrams (BDDs), that scales to large `JAVA` programs. The benchmarks are all real applications with tens of thousands of users each. Projects vary in the number of classes, methods, bytecodes, variables, and heap allocations. The information details, shown on Table 1, are calculated on the basis of a context-insensitive callgraph precomputed by the `JOEQ` compiler.

All experiments were conducted using `JAVA JRE 1.5`, `JOEQ` version 20030812, on a Intel Core 2 T5500 1.66GHz with 3 Gigabytes of RAM, running Linux Kubuntu 8.04. The analysis times and memory usages of our context insensitive pointer analysis, shown on Table 2, illustrate the scalability of our BES resolution on real examples. `DATALOG_SOLVE` solves the (default) query for all benchmarks in a few

Table 2
 Times (in seconds) and peak memory usages (in megabytes) for each benchmark and context-insensitive pointer analysis.

Name	time (sec.)	memory (Mb.)
freetts	10	61
nfchat	8	59
jetty	73	70
joone	4	58

seconds. The analysis results were verified by comparing them with the solutions computed by the BDBDB system on the same benchmark of JAVA programs and analysis.

5 Conclusion and Future Work

This paper described a practical JAVA program analysis framework based on rule specifications and general purpose verification engine. The system, called `DATALOG_SOLVE`, uses Datalog for encoding in a few lines complicated program analyses and BES resolution for evaluating the Datalog rules, thus taking advantage of the redundancies that occur in program analyses. The tool architecture is based on the well-established verification framework CADP, which provides a generic library for local BES resolution. The system is publicly available on http://www.dsic.upv.es/users/elp/datalog_solve.

We plan to endow `DATALOG_SOLVE` with optimized strategies for the BES evaluation of a Datalog query, following classical Datalog optimizations like rewriting of Datalog rules to allow goal-directed bottom-up evaluation, as in the *Magic sets* approach. Another interesting improvement we plan to explore is using the rewriting logic framework implemented in the functional programming language Maude [3] as a solver for Java program analyses containing reflection.

References

- [1] Aho, A. V., M. S. Lam, R. Sethi and J. D. Ullman, “Compilers: Principles, Techniques, and Tools (2nd Edition),” Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2007.
- [2] Andersen, H. R., *Model checking and boolean graphs*, Theor. Comput. Sci. **126** (1994), pp. 3–30.
- [3] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and C. Talcott, “All About Maude: A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic,” Lecture Notes in Computer Science **4350**, Springer Verlag, 2007.
- [4] Garavel, H., R. Mateescu, F. Lang and W. Serwe, *CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes*, in: *Proceedings of the 19th Int. Conference on Computer Aided Verification CAV’2007 (Berlin, Germany)*, Lecture Notes in Computer Science **4590** (2007), pp. 158–163.
- [5] Whaley, J., *Joeq: a virtual machine and compiler infrastructure*, in: *Proceedings of the 2003 workshop on Interpreters, Virtual Machines and Emulators IVME’2003 (San Diego, California, USA)* (2003), pp. 58–66.
- [6] Whaley, J., D. Avots, M. Carbin and M. S. Lam, *Using Datalog with Binary Decision Diagrams for Program Analysis*, in: *Proceedings of the Third Asian Symposium on Programming Languages and Systems APLAS’05 (Tsukuba, Japan)*, Lecture Notes in Computer Science **3780** (2005), pp. 97–118.