A Logic-Based Approach to Data Flow Analysis Problems
(Preliminary Version)

S. Sagiv       N. Francez
Dept. of Computer Science, the Technion, Haifa, Israel 32000
M. Rodeh
IBM Israel Scientific Center, the Technion City, Haifa, Israel 32000
R. Wilhelm
FB 10-Informatik, Universität des Saarlandes, D-6600 Saarbrücken, F.R.G

Abstract

A new uniform formalism for tracking static properties of programs is presented. The formalism annotates each point in a program with static assertions, i.e., assertions which hold independently of the specific execution path leading to this point.

The novel idea is to use Horn clauses to specify the consistent environments and the meaning of program operations.

The abstract interpretation technique is used for finding conservative approximations to the static assertions and to analyze the accuracy of the approximation.

The formalism is used to specify and solve the problem of tracking pointer equality and allocations in Pascal-like languages. It is shown that the solutions are optimal under some natural assumptions.

1 Introduction

A standard approach to defining the meaning of a program is by using a domain called environment and associating an environment transformer with every program operation (e.g. [CC77]). The complete static information at a given point is the set of reachable environments at this point. In general, it is hard and sometimes impossible to compute the reachable environments. To overcome this problem, the technique of abstract interpretation has been introduced by P. Cousot and R. Cousot [CC77]. In this technique, an abstract domain is used for computing a conservative approximation to the reachable environments.

In this paper, we combine the abstract interpretation technique with logic machinery in a novel way to develop a syntactic formalism for specifying data flow analysis (DFA) problems. The main idea is to describe environments as sets of assertions and use logical closure operations w.r.t Horn clauses to define the environment transformers.

Our syntactic formalism enables the systematic study of DFA problems as well as the automatic derivation of DFA algorithms to solve these problems.
1.1 The Logic Based Data Flow Analysis Formalism

To discuss our logic based DFA formalism (LDFA), consider a simplified version of the pointer equality problem (PEP) in a restricted subset of Pascal, which allows only pointers to simple types (e.g., integer), no procedures, no arrays, and no records. PEP is used throughout the paper as a running example.

In LDFA, with each program we associate a finite set $C$ of objects and a finite set $P$ of predicate symbols; the predicates model the dynamic properties of the DFA problem. For a predicate $p \in P$ of arity $l$, and objects $t_1, t_2, \ldots, t_l \in C$, $p(t_1, t_2, \ldots, t_l)$ is a dynamic assertion since it may or may not hold at the end of any particular execution path in the program.

In PEP, the set $C$ of objects is the set of all the pointer variables in the program (including the constant nil) and $P = \{\text{eq}\}$. For an execution path $\pi$ in the program, the dynamic equality assertion $\text{eq}(t_1, t_2)$ holds if at the end of $\pi$ either both $t_1$ and $t_2$ are not allocated or they point to the same address. In particular, $\text{eq}(t, \text{nil})$ holds at $\pi$ if $t$ is not allocated at the end of $\pi$.

Static properties are obtained from the dynamic properties by quantifying over sets of execution paths. For each dynamic property $p \in P$, we obtain the universal property $p_v$ by universal quantification and the existential property $p_\exists$ by existential quantification.

A DFA problem is that of finding, for a given point $pt$ in the program, whether a certain universal (or existential) assertion holds at $pt$. In the existential version of PEP one needs to find whether $\text{eq}_3(t_1, t_2)$ holds. An object $t \in C$ is safe at $pt$ if $\text{eq}_3(t, \text{nil})$ does not hold at $pt$. The problem of finding the safe objects is important both from software engineering point of view and for program optimizations. The universal version of PEP may also be of interest. For example, if $\text{eq}_v(t, \text{nil})$ holds at the end of the program then there is no need to deallocate the space of $t$.

Since in general, we cannot expect algorithms to yield exact results for DFA problems, we are interested in conservative approximations, i.e., whenever the algorithm yields that $p_v(t_1, t_1, \ldots, t_n)$ holds at a point $pt$, then (dynamically) for every execution path $\pi$ to $pt$, $p(t_1, t_2, \ldots, t_n)$ holds at $\pi$. However, the converse need not hold. Similarly, if the algorithm yields that $p_\exists(t_1, t_1, \ldots, t_n)$ does not hold at $pt$, then there does not exist an execution path $\pi$ leading to $pt$ such that $p(t_1, t_2, \ldots, t_n)$ holds at $\pi$.

For example, in many cases algorithms cannot track the actual execution paths and take into account also some control flow paths which need not be executable, thereby getting conservative but not necessarily accurate results. A DFA solution is statically exact if it is exact under the assumption that every control flow path is executable, i.e., $p_v(t_1, t_1, \ldots, t_n)$ holds at a point $pt$ if and only if for every control path $\pi$ to $pt$, $p(t_1, t_2, \ldots, t_n)$ holds at $\pi$. Similarly, $p_\exists(t_1, t_1, \ldots, t_n)$ holds at a point $pt$ if and only if there exists a control path $\pi$ to $pt$, s.t. $p(t_1, t_2, \ldots, t_n)$ holds at $\pi$. Notice that by definition, any statically exact solution is a conservative approximation.

1.2 Main Results and Comparison to Previous Work

1.2.1 Application to the Pointer Equalities Problem

After developing the theory of LDFA in Sections 2–5, we apply it to PEP in Sections 6–7 for several subsets of Pascal. The usage of abstract interpretation guarantees that our DFA results are all conservative. Furthermore, it is shown that all the DFA results are the best (see [CC79]) in the sense