A New Perspective for Understanding Data Flow Analysis

Hai Lin[1a]

[1]College of Software, Shenyang Normal University

253 Huanghe North Street, Shenyang, 110034, China

[a]jlu_linhai@163.com

Keywords: data flow analysis, model checking

Abstract. Data flow analysis is a technique that is widely used in compilers and program analysis tools. A lot of practical problems use data flow analysis, e.g. live variable analysis. The usual way of solving data flow analysis problem is to solve for the fix point of some simultaneous equations. This is done via an iterative process. In this paper, we argue that data flow analysis can be understood as a temporal logic model checking problem. We illustrate this idea using the problem of live variable analysis. This provides a new perspective for understanding data flow analysis.

Introduction

Data flow analysis is a commonly used technique [1,2]. Many interesting problems in compilers and program analysis can be solved using this technique. For example, the problem of live variable analysis can be solved this way.

Here is a very simple example, illustrating the idea of live variable analysis. Consider the following C code.

```
1    int main()
2    {
3          int x;
4          x = 2;
5          x= 200;
6          printf("%d", x);
7          return 0;
8    }
```

The assignment statement on line 4 can be eliminated for the following reason. Variable x is about to be updated to 200 on line 5. So no matter what value x gets on line 4, it has no effect on the statement on line 6 at all. Hence the statement on line 4 can be safely removed. More formally, this is done using data flow analysis. We do this analysis backwards starting from line 8. We carefully analyze the set of live variables on each line. In particular, the set of live variables on line 4 is the empty set. This means that no variable is useful at all after executing the statement on line 4. So we can get rid of that statement on line 4.

In the next section, we will briefly review data flow analysis. And then we will summary the idea of model checking. After that we will present the idea of viewing data flow analysis as model checking. This is done using the example of live variable analysis. Finally we conclude in the last section.

Data Flow Analysis

Data flow analysis is a technique that can be used to obtain liveness information throughout the program [3,4]. Here is the basic idea. A variable is live means that the variable may potentially be used later on in the program. Each statement in the program may generate new live variables or remove existing live variables. We associate the following sets with each statement s in the

program.

Definition 1. Given a statement s, we define the following sets.
kill(b) = {v |v is a variable, and v is defined in s}
generate(s) = {v |v is a variable, and v is used in s}
in(s)={v |v is a variable, and v is live upon entering s}
out(s)={v |v is a variable, and v is live when exiting s}
succ(s) = {c |c is a statement, and c is a successor of s in control flow graph}
pred(s) = {c |c is a statement, and c is a predecessor of s in control flow graph}

Suppose that P is some program, we can compute generate(s) and kill(s), for each statement s in program P. After that we can compute in(s) and out(s) according to the following set of rule.

in(s) = out(s) + generate(s) – kill(s) (Rule 1)

$$\mathrm{out}(s) = \bigcup_{c \in succ(s)} \mathrm{in}(c)$$ (Rule 2)

Model Checking

Given a Kripke structure M = (S, R, L) that describes a finite state system and a temporal logic formula f describing some specification. The model checking problem is to check if M satisfies f [5,6]. Usually this is done by finding the set of all states in S that satisfy f. Here are the semantics of the temporal logic formulas.

Given an infinite sequence of states $s_0s_1s_2\ldots$, we write S^k for the suffix of $S = s_0s_1s_2\ldots$ starting at sk.

$S^K \models f$, when $s_k \models f$
$S^K \models \sim f$, when not $s_k \models f$
$S^K \models p$ or q, when $s_k \models p$ or $s_k \models q$
$S^K \models p$ and q, when $s_k \models p$ and $s_k \models q$
$S^K \models p$ U q, when there exists some i>k s.t. $s_i \models q$ and for all j s.t. k<j<i, $s_j \models p$
$S^K \models p$ R q, when either for all i>k, $s_i \models q$, or for some j>k, $s_j \models p$ and for all i s.t. k<i<j, $s_i \models q$

Many efficient implementations of model checking tools are based on the fixpoint characterization of the temporal logic operators. And the process of solving this problem is an iterative process.

Data Flow Analysis as Model Checking

In this section, we argue that we can view data flow analysis problem as a model checking problem. We illustrate this idea by using live variable analysis as an example.

Let us go back to the example from the introduction. Suppose that we want to get the liveness information for all program points. One way of doing this is to use the usual way of data flow analysis as is described in the previous section. Here is another way to think about the problem. If a variable is live at some point, that means that the variable must be used before it is redefined. In other words, it means that we have two events. a) The variable is used. b) The variable is redefined. If we use the idea of model checking, this can be rephrased as follows. A variable is live at some point in the program means that the variable is used “releases” the obligation that it cannot be redefined. So, using the “release” operator, this means “used R ~defined”. So the live variable analysis problem can be seen as a special case of model checking.

Summary

In this paper, we propose to use another perspective to understand data flow analysis. The traditional approach for solving data flow analysis is to solve for the fixed point of some

simultaneous equations. This is done through an iterative process. We argue that the same problem can be solved using model checking techniques. We illustrate the idea using live variable analysis as an example. The mean that a certain variable is live can be captured by a temporal logic formula, thus the problem can be seen as a special form of model checking.

Acknowledgement

This work is supported by Liaoning Provincial Natural Science Foundation under grant 201202202, Scientific Research Foundation of Liaoning Provincial Education Department under grant L2012388.

References

[1] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. P. O'Boyle, J. Thomson, M. Toussaint, and C. K. I. Williams. Using machine learning to focus iterative optimization. In Proceedings of the International Symposium on Code Generation and Optimization, Mar. 2004.

[2] L.-N. Pouchet, C. Bastoul, A. Cohen, and J. Cavazos. Iterative optimization in the polyhedral model: Part II, multidimensional time. In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'08), pages 90-100, Tucson, Arizona, June 2008. ACM Press.

[3] David Callahan, Keith D. Cooper, Ken Kennedy. Interprocedural constant propagation - In Proceedings of the SIGPLAN '86 Symposium on Compiler Construction , 1986.

[4] Morten Kühnrich, Stefan Schwoon, Stefan Kiefer, Technische Universität München. Interprocedural Dataflow Analysis over Weight Domains with Infinite Descending Chains - in Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures.

[5] Aske Simon Christensen, Anders Møller, Michael I. Schwartzbach. Precise analysis of string expressions - In Proc. 10th International Static Analysis Symposium, SAS ’03, volume 2694 of LNCS , 2003.

[6] Edmund Clarke, Armin Biere, Richard Raimi, Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving - Formal Methods in System Design, 2001.