Flemming Nielson - Principles of Program Analysis - 1999

Tom Rochette <tom.rochette@coreteks.org>

July 24, 2025 — daae079c

0.1 Context

0.2 Learned in this study

0.3 Things to explore

1 Overview

2 Notes

2.1 Chapter 1 - Introduction

2.2 1.1 The Nature of Program Analysis

- One common theme behind all approaches to program analysis is that in order to remain computable (tractable?) one can only provide approximate answers
- In general, we expect the program analysis to produce a possibly larger set of possibilities than what will ever happen during execution of the program
- All program analyses should be semantics based: this means that the information obtained from the analysis can be proved to be safe (or correct) with respect to a semantics of the programming language
- Program analysis should not be semantics direct: this would mean that the structure of the program analysis should reflect the structure of the semantics

2.3 1.2 Setting the Scene

2.4 Reaching Definitions Analysis

- An assignment (called a definition in the classical literature) of the form $[x := a]^\ell$ may reach a certain program point (typically the entry or exit of an elementary block) if there is an execution of the program where x was last assigned a value at ℓ when the program point is reached

2.5 Chapter 2 - Data Flow Analysis

- Classical data flow analyses:
  - Available expressions
  - Reaching definitions
  - Very busy expressions
  - Live variables

2.6 2.1 Intraprocedural Analysis

2.6.1 Initial and final labels

- $init$: **Stmt** $\rightarrow$ **Lab**
  - Returns the initial label of a statement
- $final$: **Stmt** $\rightarrow \mathcal{P}($**Lab**$)$
  - Returns the set of final labels in a statement

2.6.2 Blocks

- $blocks$: **Stmt** $\rightarrow \mathcal{P}($**Blocks**$)$
  - Blocks is the set of statements, or elementary blocks
- $labels$: **Stmt** $\rightarrow \mathcal{P}($**Lab**$)$
  - The set of labels occurring in a program
- $init(S) \in labels(S)$ and $final(S) \subseteq labels(S)$

2.6.3 Flows and reverse flows

- $flow$: **Stmt** $\rightarrow \mathcal{P}($**Lab** $\times$ **Lab**$)$
  - The set of couples representing transitions between labels
- $flow^R$: **Stmt** $\rightarrow \mathcal{P}($**Lab** $\times$ **Lab**$)$
  - The reverse flow
  - $flow^R(S) : \{(\ell, \ell')|(\ell', \ell) \in flow(S)\}$

2.6.4 The program of interest

- S_*: the program that we are analysing
- **Lab**$_*$: the labels $(labels(S_*))$ appearing in S_*
- **Var**$_*$: the variables $(FV(S_*))$ appearing in S_*
- **Blocks**$_*$: the elementary blocks $(blocks(S_*))$ occurring in S_*
- **AExp**$_*$: the set of non-trivial arithmetic subexpressions in S_*. An expression is trivial if it is a single variable or constant

2.7 2.1.1 Available Expressions Analysis

- For each program point, which expressions must have already been computed, and not later modified, on all paths to the program point
- An expression is killed in a block if any of the variables used in the expression are modified in the block
- A generated expression is an expression that is evaluated in the block and where none of the variables used in the expression are later modified in the block
- We are interested in the largest sets satisfying the equation for AE_{entry}

2.8 2.1.2 Reaching Definitions Analysis

- For each program point, which assigments may have been made and not overwritten, when program execution reaches this point along some path
- An assignment is destroyed if the block assigns a new value to the variable
- We are interested in the smallest sets satisfying the equation for RD_{entry}

2.9 2.1.3 Very Busy Expressions Analysis

- An expression is very busy at the exit from a label if, no matter what path is taken from the label, the expression must always be used before any of the variables occurring in it are redefined
- (The aim of the Very Busy Expressions Analysis is to determine) For each program point, which expressions must be very busy at the exit from the point

- We are interested in the largest sets satisfying the equation for VB_{exit}

2.10 2.1.4 Live Variables Analysis

- A variable is live at the exit from a label if there exists a path from the label to a use of the variable that does not re-define the variable
- (The Live Variables Analysis will determine) For each program point, which variables may be live at the exit from the point
- This analysis might be used as the basis for Dead Code Elimination
- We are interested in the smallest sets satisfying the equation for LV_{exit}

2.11 2.1.5 Derived Data Flow Information

- Use-Definition chains or ud-chains: Links that, for each use of a variable, associate all assignments that reach that use
- Definition-Use chains or du-chains: Links that, for each assignment, associate all uses
- One application of ud- and du-chains is for Dead Code Elimination
- Another application is in Code Motion (moving code around)

2.12 2.6 Shape Analysis

- Shape Analysis will allow us to statically detect errors like deferencing a nil-pointer

3 See also

4 References

- Nielson, Flemming, Hanne R. Nielson, and Chris Hankin. Principles of program analysis. Springer, 1999.