Program Analysis 02242 - Guarded Command Language

Niels Thykier s072425
Melvin Winstrøm-Møller s072435
Morten Sørensen s072440

October 17, 2010

Contents

List of Figures

Chapter 1

Introduction

Chapter 2

Design

2.1 General definitions

When arguing for the correctness of algorithms, defining the flow and labels of the language is valuable. Below different definitions and functions is given, mirroring those of chapter 2 of the book. These include init(C), final(C), blocks(C), labels(C) and flow(C). Note that C is used instead of S to indicate commands instead of statements.

In the following, the guarded commands are a part of the domain of commands.

2.1.1 Termination

An important point in regards to the flow is whether or not the program is terminated at a certain point. For instance, the program may terminate at an if, but is not certain to do so. On the other hand, an abort is certain to terminate, and no flow occurs in a command that looks like: abort; C_2. To get a precise analysis, termination will be considered.

2.1.2 Labelling

A label is given to every elementary block "el". The elementary blocks consists of the commands assign, skip, abort, read, write, and test. Furthermore, a label is given to if and do, to simplify definitions later. For all intentions and purposes,

the label of if and do can be seen as being the label of a skip elementary block.

2.1.3 Init

init(C) denotes the entry point for the flow of a command. For instance, the entry point of init(C_1 ; C_2) is equal to init(C_1), since C_1 contains the entry point. The domain is:

$$init\colon Command \to Label$$

It should be noted that the domain is different from that of the book.

init(C):
init([x := a]$_l$) = l
init([skip]$_l$) = l
init([abort]$_l$) = l
init([read x]$_l$) = l
init([write x]$_l$) = l
init(C_1; C_2) = init(C_1)
init({C}) = init(C)
init([if]$_l$ gC fi) = l
init([do]$_l$ gC od) = l

The reasoning behind not simply defining the entry point of an if or a do as the first test in that if or do is that not only the first test, but all the tests, are entry points. Thus, simply choosing the first label would be wrong. One way to include all tests would be to extend the definition of init to map to the powerset of all labels, but this can complicate the analysis later. Instead, all the tests are included by labelling the if and do.

2.1.4 Final

final(C) denotes the exit points for the flow of a command. For instance, the exit point of final([read x]$_l$)$isl.The domain is : init\colon Command \to \mathcal{P}(Label)$

final(C):
final([x := a]$_l$) = {l}
final([skip]$_l$) = {l}
final([abort]$_l$) = {l}
final([read x]$_l$) = {l}
final([write x]$_l$) = {l}
final(C_1; C_2) = $\begin{cases} (\text{el in } C_1 \wedge \text{el} = \text{abort}) \cup \text{final}(C_2) \text{ if } (\text{el in final}(C_1 \wedge \text{el} \not= \text{abort}) \neq \emptyset \\ (\text{el in } C_1 \wedge \text{el} = \text{abort}) \text{ else} \end{cases}$
final({C}) = final(C)

final(if gC fi) = final(gC)
final($[\text{do}]_l$ gC od) = l $\cup$ (el in final(gC) $\wedge$ el = abort)
final($[\text{e}]_l \rightarrow C$) = final(gC1)
final(gC1 [] gC2) = final(gC1) $\cup$ final(gC2)

- The reasoning behing the definition of the semicolon command is that the final commands of C_2 will only be reached if there is any flow at all between C_1 and C_2: this is not the case if C_1 aborts the program before reaching C_2.
- The reasoning behind the definition of final for if is that the exit points of an if consists of the final commands in all the commands of the if.
- The reasoning behind the definition of final for do is that the exit points of a do consists of all the tests in the do (represented by $[\text{do}]_l$) as well as the final commands that aborts, since the normal final commands simply flow back to the tests.

2.1.5 Blocks

blocks(C):
blocks($[\text{x} := \text{a}]_l$) = $[x := a]_l$
$blocks([skip]_l) = [skip]_l$
$blocks([abort]_l) = [abort]_l$
$blocks([readx]_l) = [readx]_l$
$blocks([writex]_l) = [writex]_l$
$blocks(C_1; C_2) = blocks(C_1)joinblocks(C_2)$
$blocks(C) = blocks(C)$
$blocks(ifgCfi) = blocks(gC)blocks(dogCod) = joinblocks(gC)blocks([e]_l- >$
$C) = [e]_l joinblocks(C)blocks(gC_1[]gC_2) = blocks(gC_1)joinblocks(gC_2)flow(C) :$
$flow([x := a]_l) = \emptyset flow([skip]_l) = \emptyset flow([abort]_l) = \emptyset flow([readx]_l) =$
$\emptyset flow([writex]_l) = \emptyset = flow(C_1)unionflow(C_2)union(l, init(C_2))|lfinal(C_1)flow(C_1; C_2)flow(C) =$
$flow(C) = flow(gC)union((l, l')|l, l'label(expressions(gC)andl! = l')flow(ifgCfi)(Anexpressionflowstoits$
$flow(gC)union((l, l')|l, l'label(expressions(gC)andl! = l')union((l, l')|lfinal(gC)andl'label(expressions(gC))$
$C) = (l, init(C))unionflow(C)flow(gC1[]gC2) = flow(gC1)unionflow(gC2)expressions([e]_l- >$
$C) = [e]_l 3/17expressions(gC1[]gC2) = expressions(gC1)unionexpressions(gC2)used(C) :$
$used([x := a]) = fv(a)used([skip]) = \emptyset used([abort]) = \emptyset used([readx]) =$
$\emptyset used([writex]) = xused([e]) = fv(e)$

2.2 Program Slicing

Please describe program slicing here.

Chapter 3

Implementation

Chapter 4

Conclusion

Chapter 5

Appendix

5.1 Abbreviations and acronyms

This section contains a list of abbreviations and acronyms used in the report.

- RD - Reaching Definitions