Map Elimination: An implementation on ICFG's

Lisa Kleinlein, Luca Bruder

December 3, 2018

1 Introduction

In 2015 David Monniaux and Francesco Alberti presented an approach for the static analysis of programs handling arrays, with a Galois connection between the semantics of the array program and semantics of purely scalar operations in their paper "A Simple Abstraction of Arrays and Maps by Program Translation".
This document describes an adaption of their algorithm to the programs and data structures considered in the Ultimate program analysis tool.

The aim of this algorithm is to replace any arrays in a given program with a certain amount of variables. These variables each represent one cell of the old array and the amount of cells can be adjusted with a parameter n.
In this process the program that was put in is transformed into a new program which is an overestimation of the old one. All non-array variables stay untouched, as well as the structure of the ICFG. Furthermore the program stays satisfiable, given that it was satisfiable to begin with.

2 Prerequisites

In the following algorithm we will work with programs from the viewpoint of the Ultimate tool. Ultimate represents programs in a graph-like data structure called interprocedural control flow graph (ICFG). These ICFG's are represented with transformulas. We use these transformulas as data structures to represent the semantics of statements of a programming language.
A transformula is a tuple T = (In, Out, Aux, Φ) with
* In is a mapping between program variables and term variables (i.e., variables occurring in Φ)
* Out is a mapping between program variables and term variables
* Aux is a list of auxiliary variables occurring in Φ
* Φ is a formula in SMT

Each edge of this ICFG G is represented with a transformula consisting of InVariables, OutVariables, AuxVariables and the well-formed formula of the edge.
These transformulas are expected to have no nested select-terms and store-terms. And furthermore no terms of the form (= (store a b c) (store x y z)).
The ICFG given as input has to be of the form G = (Loc, l, sl, E, Var)

Loc: A set of locations
E: A subset of edges connecting locations Loc x Loc
sl: A starting location within Loc
l: E $\rightarrow$ TF is a labeling function that labels each transition with a transition formula from TF
Var: Is a set of all program variables used in G

Satisfiability Modulo Theories or SMT are a way to write down and work with well-formed formulas. In SMT each variable has a sort (e.g., Int, Real, Bool, ...). Of these sorts arrays[1] are of particular interest to us. In this theory arrays are functions with possibly multiple indices. The output of such a function has a certain sort. This sort will hence forward be called valuetype of the array.

[1] http://smtlib.cs.uiowa.edu/theories-ArraysEx.shtml

3 Algorithm

Algorithm 1 Map elimination algorithm

1: **procedure** ELIMINATE_MAPS(**ICFG** G, **int** n)

Input: An integer n for the amount of observed indices
An ICFG G of form G = (Loc, l, sl, E, Var)

Output: An overaproximation G' of the old icfg without arrays

2: **ICFG** $G' \leftarrow G$
3: $Var' \leftarrow Var \backslash \{x \in Var | x \text{ has array sort}\} \cup$
$\{\texttt{x\_idx\_i} | x \in Var \wedge\ x \text{ has array sort} \wedge n \geq i \geq 0 \wedge \texttt{x\_idx\_i} \text{ is an integer} \wedge\ \texttt{x\_idx\_i}\ \text{is fresh}\} \cup$
$\{\texttt{x\_val\_i} | x \in Var \wedge x \text{ has array sort} \wedge n \geq i \geq 0 \wedge \texttt{x\_val\_i} \text{ has valuetype of } x\ \wedge\ \texttt{x\_val\_i} \text{ is fresh}\}$
4: **dict** *idxD:* Var → Seq(Var')
5: **dict** *valD:* Var → Seq(Var') ▷ *idxD* and *valD* are filled with n program variables for each array x in G.
6: **for** $e \in E$ **do** ▷ Iterates over every edge of the icfg G
7: **dict** *lowD* ← null
8: **dict** *highD* ← null ▷ Two dictionaries that contain the lowest and highest respective index used for each array
9: ϕ ←l(e)
10: ϕ' ←l(e')
11: **for** *sel* $\in \phi$ where sel ≡ (`select` $\texttt{x}_\texttt{j}$ y) **do** ▷ Iterates over every Select-Term in ϕ
12: Eliminate_Selects(ϕ, ϕ', *e*, *valD*, *idxD*, *auxD*, *sel*)
13: **for** *sto* $\in \phi$ where sto ≡ (= `w(store` $\texttt{x}_\texttt{j}$ y z)) **do** ▷ Iterates over every Store-Term in l(e)
14: Eliminate_Stores(ϕ, ϕ', *e*, *valD*, *idxD*, *sto*)
15: **for** *equ* $\in \phi$ where equ ≡ (= x y) **do** ▷ Iterates over every Equality-Term in l(e) with x and y being arrays
16: Eliminate_Equalities(ϕ, ϕ', e, valD, idxD, equ)
17: **for each** array a in *e* **do**
18: **if** $\exists In(\phi)(\texttt{a})$ **then**
19: $In(\phi') \leftarrow (In(\phi') \setminus \{(\texttt{a}, b_\texttt{j}) \mid (\texttt{a}, b_\texttt{j}) \in In(\phi)\})$
20: **if** j == *lowD*[a] **then**
21: $In(\phi') \leftarrow (In(\phi')\ \cup\ \{(val, val_{lowD[a]})\}$
22: **if** $\exists Out(\phi)(\texttt{a})$ **then**
23: $Out(\phi') \leftarrow (Out(\phi') \setminus \{(\texttt{a}, c_\texttt{k}) \mid (\texttt{a}, c_\texttt{k}) \in In(\phi)\})$
24: **if** k == *highD*[a] **then**
25: $In(\phi') \leftarrow (In(\phi')\ \cup\ \{(val, val_{highD[a]})\}$
26: **for each** idx in idxD[a] and val in valD[a] **do**
27: $In(\phi') \leftarrow (In(\phi')\ \cup\ \{(idx, idx)\}$
28: $Out(\phi') \leftarrow (Out(\phi')\ \cup\ \{(idx, idx)\}$
29: **return** G'

Algorithm 2 Select

procedure ELIMINATE_SELECTS(**term** ϕ, **term** ϕ', **dict** valD, **dict** idxD, **term** sel)

Input: Two transformulas
Two dictionaries containing program variables for values and indices
The term that has to be substituted

Output: The operations are made on global variables and therefore no direct output is needed

if $lowD[\mathtt{x}] > \mathtt{j}$ **then** $lowD[\mathtt{x}] \leftarrow \mathtt{j}$

if $highD[\mathtt{x}] < \mathtt{j}$ **then** $highD[\mathtt{x}] \leftarrow \mathtt{j}$

$Aux(\phi') \leftarrow Aux(\phi) \cup$ fresh auxiliary variable $\mathrm{aux}_{\mathtt{x}}$

Substitute in ϕ' (`Select` $\mathtt{x_j}$ y) **with** $aux_{\mathtt{x}}$

for each idx in $idxD[\mathtt{x}]$ and val in $valD[\mathtt{x}]$

Add to $\phi' \wedge$ $(\rightarrow (= \mathtt{y}\ \mathtt{idx})\ (= \mathtt{val_j}\ aux_{\mathtt{x}}))$

Algorithm 3 Store

procedure ELIMINATE_STORES(**term** ϕ, **term** ϕ', **dict** valD, **dict** idxD, **term** sto)

Input: Two transformulas
Two dictionaries containing program variables for values and indices
The term that has to be substituted

Output: The operations are made on global variables and therefore no direct output is needed

if $lowD[\mathtt{x}] > \mathtt{j}$ **then** $lowD[\mathtt{x}] \leftarrow \mathtt{j}$

if $highD[\mathtt{x}] < \mathtt{j}$ **then** $highD[\mathtt{x}] \leftarrow \mathtt{j}$

Substitute in $\phi' =$ `expr` (`Store` $\mathtt{x_j}$ y z) **with** :

$\wedge$ and **for each** idx in $idxD[\mathtt{x}]$ and val in $valD[\mathtt{x}]$

$(\rightarrow (= \mathtt{y}\ \mathrm{idx})\ (= \mathrm{val}_{\mathtt{j+1}}\ \mathtt{z}))$

$\wedge(\rightarrow (\mathtt{distinct}\ \mathrm{idx}\ \mathtt{y})(= \mathrm{val}_{\mathtt{j+1}}\ \mathrm{val}_{\mathtt{j}})$

Algorithm 4 Equality

procedure ELIMINATE_EQUALITIES(**term** ϕ, **term** ϕ', **dict** valD, **dict** idxD, **term** equ)

Input: Two transformulas
Two dictionaries containing program variables for values and indices
The term that has to be substituted

Output: The operations are made on global variables and therefore no direct output is needed

if $lowD[\mathtt{x}] > \mathtt{j}$ **then** $lowD[\mathtt{x}] \leftarrow \mathtt{j}$

if $highD[\mathtt{x}] < \mathtt{j}$ **then** $highD[\mathtt{x}] \leftarrow \mathtt{j}$

if $lowD[\mathtt{y}] > \mathtt{k}$ **then** $lowD[\mathtt{y}] \leftarrow \mathtt{k}$

if $highD[\mathtt{y}] < \mathtt{k}$ **then** $highD[\mathtt{y}] \leftarrow \mathtt{k}$

Substitute in $\phi' = \mathtt{x_j}\ \mathtt{y_k}$ **with** : ▷ Where $\mathtt{x}_j$ and $\mathtt{y}_k$ are two different Arrays

$\wedge$ and **for each** idx in $idxD[\mathtt{x}]$ and val in $valD[\mathtt{x}]$

$(\rightarrow (= \mathtt{x\_idx_j}\ \mathtt{y\_idx_k})(= \mathtt{x\_val_j}\ \mathtt{y\_val_k})$

4 Examples

The following examples show the transformula of an edge before and after the transformation by the algorithm with parameter n = 1.

Example 1:
(and (= a_0 b_1)(= a_0 c_0)(= a_1 (store a_0 12 24)))
InVars : {(a → a_0)}
OutVars : {(a → a_1)}

(and
(→ (= a_idx_1 b_idx_1)(= a_val_1$_0$ b_val_1$_1$))
(→ (= a_idx_1 c_idx_1)(= a_val_1$_0$ c_val_1$_0$))
(→ (= 12 a_idx_1)(= 24 a_val_1$_1$))
(→ (distinct a_idx_1 12)(= a_val_1$_1$ a_val_1$_0$)))
InVars : {(a → a_val_1$_0$)}
Outvars : {(a → a_val_1$_1$), (b → b_val_1$_0$), (c → c_val_1$_0$)}

Example 2:
(and (= 5 (select a_0 5))(= a_0 c_3))
InVars : {(a → a_0)}
OutVars : {(a → a_1)}

(and
(= 5 a_aux_1)
(→ (= a_idx_1 c_idx_1)(= a_val_1$_0$ c_val_1$_3$))
(→ (= 5 a_idx_1) (= a_aux_1 a_val_1$_0$)))
InVars : {(a → a_val_1$_0$)}
Outvars : {(a → a_val_1$_0$), (c → c_val_1$_3$)}