High Integrity Computing (F23PS2)

High Integrity Software Development (F29HD2)

Flow Analysis - How It Works

Andrew Ireland

Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Overview

- Begin to explore the levels of program analysis supported by the SPARK Toolset.
- Consider aspects of the mechanisms that under-pin the flow analysis capabilities of the SPARK Examiner.

Examiner's Levels of Analysis

Data flow analysis: checks the correct usage of parameters and global variables with respect to their modes; checks that variables are not read before they are written; checks for ineffective code (automatic).

Information flow analysis: checks for consistency between code and derives annotations (automatic).

Formal verification: uses proof annotations in conjunction with the code to generate verification conditions. Proving all but the trivial verification conditions is out with the scope of the Examiner (semi-automatic).

Note that the levels should be seen as progressively more sophisticated. By default the Examiner will conduct data and information flow analysis.

Flow Analysis

- A purely symbolic form of analysis, *i.e.* no specific data values are considered.
- Based upon a number of relationships between variables and expressions.

Statements, Variables & Expressions

- For a statement S:
  - V denotes the set of variables
  - E denotes the set of expressions
- Note that a statement may be an atomic statement or a compound statement, *e.g.* sequences, conditionals, ...
- Example:

  ```
  X := Y * Z; W := X + 2;
  ```

  within the above statement sequence, $V = \{W, X, Y, Z\}$ and $E = \{Y * Z, X + 2\}$.

Classifying Variables

A variable is **defined** when it appears on the LHS of an assignment, otherwise it is **preserved**. This gives rise to 2 sets:

- D the set of variables that S may **define**.
- P the set of variables that S may **preserve**.

Note the use of “may” – if S is conditional then not all paths may traverse all assignments.

Examples

- Given the statement sequence:

```
X := Y * Z; W := X + 2;
```

  we get $D = \{X, W\}$ and $P = \{Y, Z\}$

- Given the conditional statement:

```
if X > Y then X := X + Y; else Y := Y + X;
```

  we get $D = \{X, Y\}$ and $P = \{X, Y\}$

Note that the intersection of D and P may be non-empty when conditional statements are involved.

Some Dependency Relations

$L(u, e)$ is true if the initial value of variable u may be used in computing the value of expression e.

$M(e, v)$ is true if e may be used in computing the final value of variable v.

$R(u, v)$ is true if the initial value of u may be used in computing the final value of variable v.

Example Revisited

- Given:

```
X := Y * Z; W := X + 2;
```

where $V = \{W, X, Y, Z\}$ and $E = \{\underbrace{Y * Z}_{e_1}, \underbrace{X + 2}_{e_2}\}$

- Relations L, M and R are defined:

$$L_{true} = \{(y, e_1), (z, e_1), (y, e_2), (z, e_2)\}$$
$$M_{true} = \{(e_1, x), (e_1, w), (e_2, w)\}$$
$$R_{true} = \{(y, w), (z, w), (y, x), (z, x), (y, y), (z, z)\}$$

Note that $L(x, e_2)$ is false because the initial value of `X` is overwritten before it is used in computing e_2.

Relations As Binary Matrices

$$\begin{aligned} L_{true} &= \{(y, e_1), (z, e_1), (y, e_2), (z, e_2)\} \\ M_{true} &= \{(e_1, x), (e_1, w), (e_2, w)\} \\ R_{true} &= \{(y, w), (z, w), (y, x), (z, x), (y, y), (z, z)\} \end{aligned}$$

$$L = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

$$L_{ij} = 1, \text{ if } L(v_i, e_j) \text{ is true, otherwise false}$$
$$M_{ij} = 1, \text{ if } M(e_i, v_j) \text{ is true, otherwise false}$$
$$R_{ij} = 1, \text{ if } R(v_i, v_j) \text{ is true, otherwise false}$$

Diagonal Matrices: Defined & Preserved

- The defined and preserved relations can also be represented as matrices:

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Note that a 1 along the diagonal of the D matrix denotes a variable that is defined, while a 1 along the diagonal of the P matrix denotes a variable that is preserved.

An Important Relation

- A variable u may be used in computing the value of a variable v in two ways:
  1. u may be used in an expression that in turn is used by v OR
  2. u and v may be the same variable, and the variable is a member of the set P.
- Symbolically this can be expressed by:

$$R = LM \text{ or } P$$

Note that the product of binary matrices is analogous to matrix multiplication where multiplication is replaced by **and** and addition is replaced by **or**, *e.g.* If N is the product of L and M (with components L_{ij} and M_{jk}) then:

$$N_{ik} = (L_{i1} \text{ and } M_{1k}) \text{ or } (L_{i2} \text{ and } M_{2k}) \text{ or } \ldots \text{ or } (L_{in} \text{ and } M_{nk})$$

Sequences Of Statements

- Consider a statement S_1 and associated relations D_1, P_1, L_1, M_1, R_1, and a statement S_2 and associated relations D_2, P_2, L_2, M_2, R_2.
- Now for the composition statement $S_1;\ S_2$, the associated relations are defined as follows:

$$\begin{aligned}
D &= D_1 \text{ or } D_2 \\
P &= P_1 \text{ and } P_2 \\
L &= L_1 \text{ or } R_1 L_2 \\
M &= M_1 R_2 \text{ or } M_2 \\
R &= R_1 R_2
\end{aligned}$$

`Switch` Revisited

```
procedure Switch(X, Y: in out Integer)
--# derives X from Y &
--#         Y from X;
is
   T: Integer;
begin
   T:=X; X:=Y; Y:=T;
end Switch;
```

where $V = \{X, Y, T\}$, $E = \{X, Y, T\}$ and the relations L, M, R for the whole procedure are:

$$L = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad M = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad R = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Some Of The Examiner's Rules

Rule 1: Within matrix M every expression (row) must have a value of 1 for at least one exported variable (column).

Rule 2: Within matrix R every imported variable (row) must have a 1 against at least one exported variable (column).

Rule 3: The sub-matrix of R corresponding to the imported and exported variables must be consistent with the derives clauses.

Note that matrix L provides a basis for detecting the use of undefined variables.

A Buggy Version of `Switch`

```
procedure Switch(X, Y: in out Integer)
--# derives X from Y &
--#         Y from X;
is
   T: Integer;
begin
   T:=X; X:=Y; Y:=X;
end Switch;
```

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad M = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad R = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Ineffective Statements

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

- Within M the row for expression 1 (`X`) has 0 against all the exported variables (`X` and `Y`) – this violates **Rule 1**.
- This means the statement containing the expression associated with the first row of M is `ineffective`, *i.e.* `T := X` has no effect on the computation defined by `Switch`.
- The Examiner generated message:

```
!!!  (  1)  Flow Error  : Ineffective statement.
```


Ineffective Importation

$$R = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- Within R the row for variable `X` has 0 against all the exported variables – this violates **Rule 2**.
- This means that the imported value of `X` does not contribute to the final value of any of the exported variables.
- The Examiner generated message:

```
!!!  (  2)  Flow Error   : Importation of the initial
            value of variable X is ineffective.
```

Code-Annotation Inconsistencies

$$R' = \begin{pmatrix} 0 & 0 & - \\ 1 & 1 & - \\ - & - & - \end{pmatrix}$$

where R' is the sub-matrix of R corresponding to the imported and exported variables.

- Note that the derives clauses state that `Y` is derived from `X`, but R' indicates no such relation holds – violates **Rule 3**.
- The Examiner generated message:

```
!!!  (  3)  Flow Error   : The imported value of X is
                 not used in the derivation of Y.
```


Summary

Learning outcomes:

- Gain insight into how flow analysis can be implemented.
- Construction of dependency relations for simple program statements.
- Expressing and manipulating dependency relations as binary matrices.
- Use of binary matrices in finding flow errors within code.

Recommended reading:

Bergeretti, J.F. & Carré, B.A. Information-Flow and Data-Flow Analysis of while-Programs, *ACM Transactions on Programming Languages and Systems*, ACM, New York, Vol. 7, 1985.