Program Normalization

TECHNISCHE UNIVERSITÄT DARMSTADT

Software Development Tools Lab WS 15/16
Carina Oberle

Problem 1 Loop normalization

Problem 1.1 For

- Need to place 'step' in front of every `continue` statement. (Still problematic: labeled continue in cascaded loops.)
- Name collisions possible because we put loop variables ahead of our new loop.

Problem 1.2 ForEach

- `continue` statements are not a problem in this case and need no special handling.
- Translation into `while` loop via Iterator (Java) resp. Enumerator (C#).

Problem 1.3 DoWhile

- Translation into `while` loop not clear. Problem: `continue` and `break` statements.
- A translation into a while(true) loop would be possible, where we exit the loop as soon as the condition evaluates to false at the end of the loop.

Problem 2 Frequency Analysis

Boa (`http://boa.cs.iastate.edu`)

Problem 2.1 Loops

Boa distinguishes the following loop kinds: `while`, `do`, `for`. Figure 1 shows an AST representation of a loop as Statement.

We may distinguish ForEach loops from common for loops via their child nodes. Listing 1 exemplarily shows that a ForEach loop has attributes *kind*, *statements*, *variable_declaration* and *expression*. A common For loop, however, by default has attributes *kind*, *statements*, *initializations*, *updates* and *expression*. Yet, attribute *variable_declaration* is never set, as potential variable declarations are listet in attribut *initializations*.

```
{
"kind": "FOR",
```

Attribute	Type
condition	Expression?
expression	Expression?
initializations	array of Expression
kind	StatementKind
statements	array of Statement
type_declaration	Declaration?
updates	array of Expression
variable_declaration	Variable?

Figure 1: Boa Repräsentation eines Statements

```
"statements": [{"kind": "BLOCK","statements": [{"kind": "EXPRESSION","expression
    ": {"kind": "METHODCALL","expressions": [{"kind": "VARACCESS","variable": "
    canvas"}],"method": "drawBitmap","method_args": [{"kind": "VARACCESS","
    variable": "bitmap"},{"kind": "VARACCESS","variable": "t.src"},{"kind": "
    VARACCESS","variable": "t.dst"},{"kind": "LITERAL","literal": "null"}]}}]}],

"variable_declaration": {"name": "t","variable_type": {"name": "TileRect","kind"
    : "OTHER"}},

"expression": {"kind": "VARACCESS","variable": "tileRects"}}
```

Listing 1: AST representation of a ForEach loop

Problem 2.2 Goto (C#)

Boa has no explicit representation of Goto statements, which is why they are handled as *StatementKind.OTHER*. The dataset "2015 September/GitHub" does not contain any statements of this kind, as there is not yet any AST data available for C# projekts.

Problem 3 SwitchCase

The standard version of a switch block, containing **`break`** statements at the end of each section, may be intuitively translated into an **`if`**/**`else`** construct. Yet, in case of fall-throughs there are quite a few cases that need special handling. Especially **`break`** statements inside **`if`** block obfuscate the control flow. We address important cases in the test classes. [1]

Problem 4 Boolean Expressions

For normalizing boolean expressions, we first collect a mapping from references to assigned expressions for each method. A reference that is assigned more than once or inside a conditional (if or switch) is regarded as unknown.

[1] https://github.com/stg-tud/kave-java/tree/carina-sdt/cc.kave.commons/src/test/java/cc/kave/commons/model/ssts/transformation/switchblock