Data flow

Data Flow

A language's data flow aspect allows you to find unreachable statements, detect unused assignments, check whether a variable might not be initialized before it's read, and so on. It also allows performing some code transformations, for example the 'extract method' refactoring.

Most users of data flow analyses aren't interested in details of its inner working, but are interested in getting the results they need. They want to know which of their statements are unreachable, and what can be read before it's initialized. In order to shield a user from the complexities of these analyses, we provide assembly-like intermediate language into which you translate your program. After translation, this intermediate presentation is analyzed and a user can find which of the statements of original language are unreachable etc.

For example here is the translation of a 'for' loop from baseLanguage:

```
code for node.iterable
label condition
ifjump after node
write node.variable
code for node.body
{jump after condition}
```

First, we translate the expression for node.iterable. Then we emit a label so we can jump after it. Then we perform a conditional jump after the current node. Then we emit code for writing to node.variable. This means that we change the value of node.variable on each iteration. We don't need to know what we write to node.variable, since this information isn't used by our analysis. Finally, we emit code for the loop's body, and jump to the previously emitted label.

Commands of intermediate language

Here are the commands of our intermediate language:

- read x - reads a variable x
- write x - writes to variable x
- jump before node - jumps before node
- jump after node - jumps after node
- jump label - jumps to label
- ifjump ((before|after)node)| label - conditional jump before/after node / to label
- code for node - insert code for node
- ret - returns from current subroutine

Can be unreachable

Some commands shouldn't be highlighted as unreachable. For example we might want to write some code like this:

```
for (Object o : list) {
  return o;
}
```

If you generate data flow intermediate code for this statement, the last command: jump after condition will be unreachable. On the other hand, it's a legal base language statement, so we want to ignore this command during reachable statement analysis. To do so we mark it is as 'may be unreachable,' which is indicated by curly braces around it. You can toggle this settings with the appropriate intention.

Links:

http://www.brics.dk/~mis/static.pdf - good introduction to static analyses including data flow and type systems.

Previous Next