UFR-IMAG
Université Joseph Fourier

Programming Language and Compiler Design,
2010/2011
Marion Daubignard
Yassine Lakhnech
Laurent Mounier

Some intuitions on optimization

Available expressions

Available expressions are computed for a given control point of a program (typically, the end of a basic block). They are expressions of which you know the value at this point. Hence, to compute them, you use a forward analysis: you need to know what you executed to know what you know the value of.

Intuitively, if in the previous block instruction $y := x + t$ is performed, $x + t$ becomes available: you know its value and do not need to recompute it if you want to use it again. Nevertheless, if x or t are reassigned to be worth new values, $x + t$ becomes unavailable. Indeed, its value has changed, and in case you need it once more, you have to recompute it!

More formally, for a given basic block, to compute the available expressions at the output of the block, $Out(B)$, we need to know $In(B)$. Then, we need to compute:

- $Kill(B)$, the set of expressions B makes unavailable. Those are expressions of $In(B)$ containing a variable which is reassigned in B. It's logical that these expressions are made unavailable since their values have now changed.
- $Gen(B)$, the set of expressions B makes available. Those are all the expressions appearing on the righthandside of an assignment, minus those containing a variable which is reassigned after their computation. (Example: $t = a + b; a = 0$ appears in block B, but $a + b$ is not made available (so not in $Gen(B)$). Indeed, a's value changes right after we compute $a + b$).

After that, you apply the formula $Out(B) = (In(B) - Kill(B)) \cup Gen(B)$ to get $Out(B)$.

When you have the whole set of equations, to solve it, either you can use substitutions and the solution is quite straightforward, or you have to apply the fix-point method. This is a forward analysis so you have to start with all your $In(B) = \emptyset$.

Active variables

At a control point of a program, a variable can be active or inactive. It is said to be active when you need to know its value in the sequel of the program. It is inactive if you do not use it anymore until you reach the end of the execution. The idea is that inactive variables can be thrown away because of their uselessness... For example, variables you reassign at the entry of a block are inactive at the input of this block. Indeed, what need do you have to know the previous value of x if you start your block by $x := 0$? None, which translates into x being an inactive variable at the entry of the block. Of course, the analysis is this time performed backwards. To know what we are going to use in the sequel of a program, we need to go through the program from the end to the beginning.

More formally, to compute the set of active variables at the input of a block ($In(B)$), you need to know $Out(B)$, set of active variables at the output of this block. You also need to compute:

- $Kill(B)$, set of variables rendered inactive in the block. They are all variables that you assign in the block.
- $Gen(B)$, set of variables you render active, that is, set of variables you generate a need for. They are all variables appearing on the righthandside of assignments, plus those appearing in boolean expressions of if statements, minus those which are assigned in this block before they are used (see the example in the previous paragraph). Note that variables which are reassigned *after* they

are used are active: if the block is $t := b + 1; b := 0$, you indeed need to know b at the input of the block!

To conclude, you apply the formula for backward analysis to get $In(B)$: $In(B) = (Out(B) - Kill(B)) \cup Gen(B)$.

When you have the whole set of equations, to solve it, either you can use substitutions and the solution is quite straightforward, or you have to apply the fix-point method. This is a backward analysis so you have to start with all your $Out(B) = \emptyset$.