Design and Analysis of Real-Time Systems SS 2013

Jan Reineke
Andreas Abel

Deadline: Thursday, June 27, 2013, 14:15

Assignment 8

Problem 1: At-Most-One-Miss Analysis (5·6 Points)

Note that this assignment is fairly challenging and open. It will be graded accordingly.

We have seen in the lecture that there are limits as to what *may* and *must* cache analyses can prove. Consider for example the following program:

```
R1 := 0;
while (R1 < 100) {
    access a;
    if (R1 > INPUT) {
        access b;
    } else {
        access c;
    }
}
```

If the analysis cannot determine whether the "then" or the "else" branch is taken in a particular loop iteration, then it can never guarantee cache hits for cache accesses in those parts of the program.

On the other hand, for an LRU-cache with a large enough associativity, it is quite obvious that only the first executions of the "then" and the "else" branch may result in cache misses.

The goal of this assignment is to develop an *at-most-one-miss analysis* that determines whether or not accesses to a given memory block may cause more than one cache miss during the execution of a program. If accesses to a memory block may cause at most one cache miss, we call this memory block *at-most-one-miss*.

Specifically, we want to develop an *at-most-one-miss analysis* for fully-associative caches with LRU replacement, and formalize the analysis as an abstract interpretation.

1. Perform a *may* and a *must* analysis on the example program to verify that they indeed cannot determine whether memory accesses inside the conditional lead to cache hits or cache misses. Assume that the cache has associativity 3.

2. Discuss what kind of concrete semantics is required to capture the property of interest. *Hint:* whether a memory block is *at-most-one-miss* cannot be determined by only regarding reachable cache states, but it is a property of "executions".

3. Formalize such a concrete semantics. First, define a domain C for your semantics. Second, specify the effect of a memory access by defining a concrete transformer $\mathit{access} : C \times M \to C$. Also, define a classification function $\mathit{classify} : C \times M \to \{\mathit{at\text{-}most\text{-}one\text{-}miss}, \mathit{not\text{-}at\text{-}most\text{-}one\text{-}miss}\}$.
   You do not have to lift your analysis to the level of a control-flow graph. Assume, this is done in the standard way as in the lecture.

4. The concrete semantics is likely not efficiently computable. Introduce an abstract semantics that is. In particular, define an abstract domain A with a suitable partial order $\sqsubseteq$, such that $(A, \sqsubseteq)$ forms a complete lattice. Also, define a concretization function that relates elements of the abstract domain A with elements of the concrete domain C, and a locally-consistent abstract transformer $\mathit{access}_{abs} : A \times M \to A$. In addition, define an abstract classification function $\mathit{classify}_{abs} : A \times M \to \{\mathit{at\text{-}most\text{-}one\text{-}miss}, \mathit{don't\ know}\}$, that determines whether a given block is guaranteed to be *at-most-one-miss* or not. *Hint:* similar ideas as in the may- and the must-abstractions for LRU may be applicable. However, the analysis needs to implicitly or explicitly distinguish between the case in which a memory block has "already" caused a cache miss and the case in which the memory block has not.

5. Apply your analysis to the example program given above assuming associativity 3.