SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Sebastian Hahn, M.Sc.

Verification of Real-Time Systems SS 2015

Assignment 6

Deadline: June 11, 2015, before the lecture

Exercise 6.1: Cache Analysis: Theory (11=2+1+2+2+4 Points)

(a) Determine the "height" of the LRU must cache lattice $(C_{\text{must}}, \sqsubseteq)$, as defined below, i.e. the length of the longest ordered chain of LRU must caches.

$$\begin{aligned} C_{\text{must}} &:= \{f : \mathcal{B} \to \{0, \ldots, k\} \mid \forall i < k : |\{b \in \mathcal{B} \mid f(b) \leq i\}| \leq i + 1\} \cup \{\bot\} \\ f \sqsubseteq g &:\Leftrightarrow f = \bot \vee (g \neq \bot \wedge \forall b \in \mathcal{B} : f(b) \leq g(b)) \end{aligned}$$

(b) Explain the extra condition $\forall i < k : |\{b \in \mathcal{B} \mid f(b) \leq i\}| \leq i + 1$ in the definition of C_{must} above.

(c) Come up with functions $classify_{must}$ and $classify_{may}$ that classify a given access as either hit or miss for a given must/may cache. First give a suitable target domain. *Hint:* The target domain should be a lattice.

(d) In the lecture, we introduced the concretization function γ_{must} for must caches by an example. Provide a general formal definition of γ_{must}. *Hint:* Do not forget to state the signature of γ_{must}.

(e) Formulate and prove local consistency of the LRU must cache update up_{must} formally.

Exercise 6.2: Cache Analysis: Practice (12=5+3+4 Points)

(a) Perform a cache analysis (must and may) for a 4-way fully associative cache with LRU replacement policy on the access graph depicted in Figure 1. Assume an initially empty cache. Nodes in the graph are program points, an edge denotes an access to the given address. Give the result of your fixed point iteration per program point and classify the accesses using the functions defined in Exercise 3.1.

(b) Accesses within a loop typically behave as follows (given that the cache is big enough): They miss the cache in the first iteration, but hit the cache in all later iterations. What technique could be applied to capture such behaviour (without modifying the program/access graph)? Describe such a technique briefly. How does it affect the lattice the must/may analysis operates on?

(c) Apply your presented technique and give the final results.

Exercise 6.3: Relational Cache Analysis (6=2+2+1+1 Points)

(a) Briefly explain why relational cache analysis can overcome the three main problems (mentioned in lecture) caused by imprecise address information.

(b) For the analysis of set-associative caches, the traditional address-based cache analysis analyses each cache set separately. Why is this possible? Why is this not a good idea for relational cache analysis?

(c) Relational cache analysis typically uses several pre-processing analyses that compute relations between symbolic names. How can the results of these analysis be combined?

(d) Explain why the relation $\overline{\text{ssdb}}$ occurs often in practice and why it is still useful for relational cache analysis.

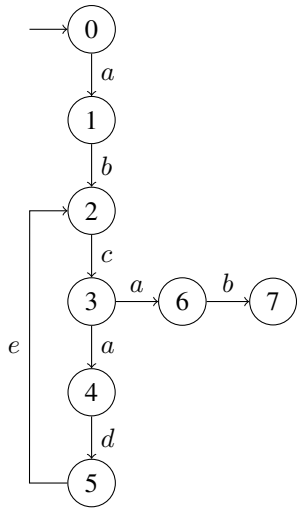

Figure 1: Access graph for exercise 1.