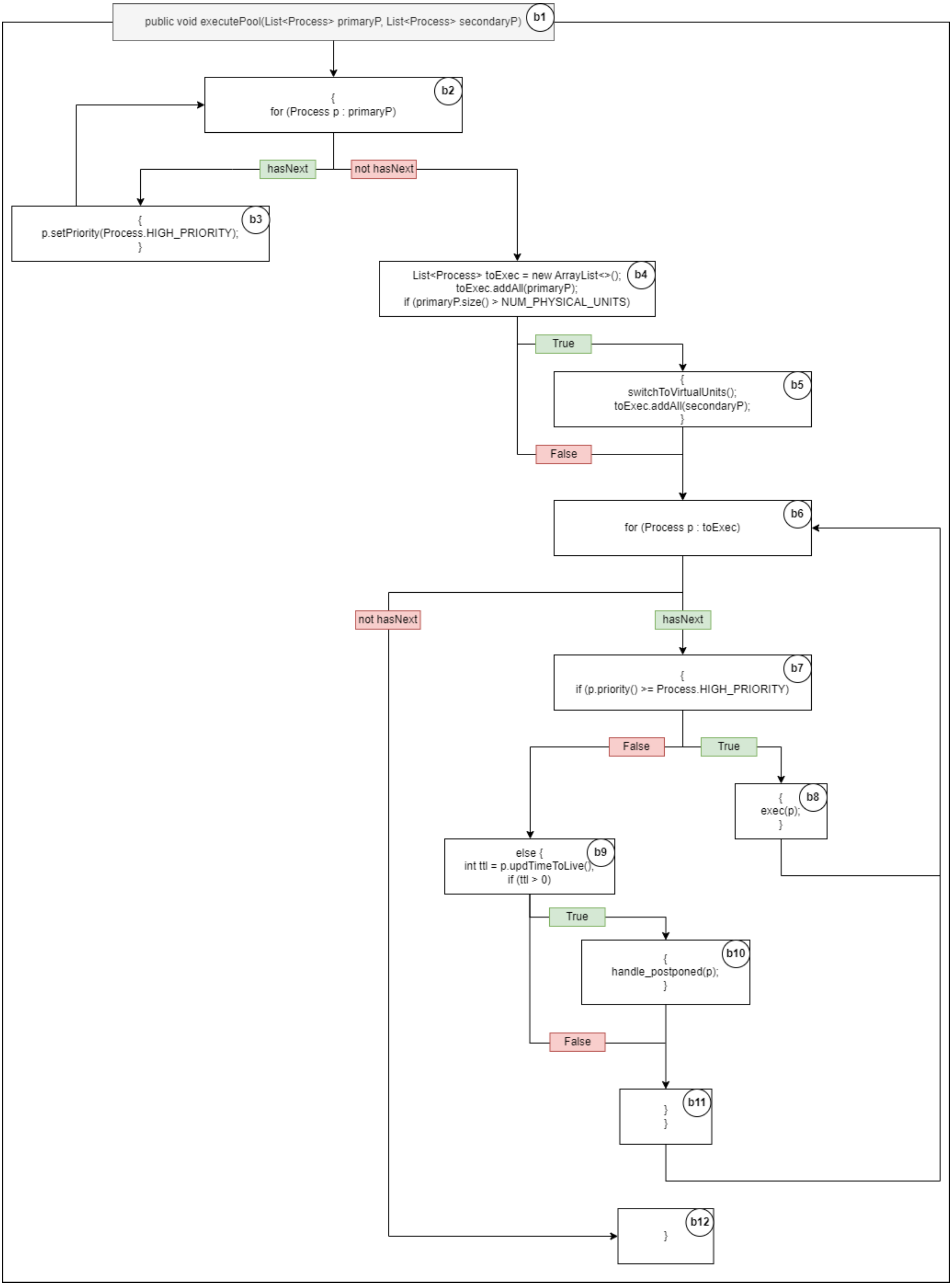
Piacente Cristian 866020

Assignment A2 – Finite models

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Task 1) Draw the control flow graph (CFG) of the method executePool(...)


public void executePool(List<Process> primaryP, List<Process> secondaryP)
b1
{
for (Process p : primaryP)
b2
hasNext
not hasNext
{
p.setPriority(Process.HIGH_PRIORITY);
}
b3
List<Process> toExec = new ArrayList<>();
toExec.addAll(primaryP);
if (primaryP.size() > NUM_PHYSICAL_UNITS)
b4
True
{
switchToVirtualUnits();
toExec.addAll(secondaryP);
}
b5
False
for (Process p : toExec)
b6
not hasNext
hasNext
{
if (p.priority() >= Process.HIGH_PRIORITY)
b7
False
True
{
exec(p);
}
b8
else {
int ttl = p.updTimeToLive();
if (ttl > 0)
b9
True
{
handle_postponed(p);
}
b10
False
}
}
b11
}
b12

Piacente Cristian 866020

Assignment A2 – Finite models

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Question 1) Show a CFG path that leads to executing method "handle_postponed" without never executing method "switchToVirtualUnits".

To execute **handle_postponed,** the basic block b10 must belong to the path and to **avoid** executing **switchToVirtualUnits** we must avoid the basic block b5 in our path.

Here is an example of a **path**: `b1-b2-b3-b2-b4-b6-b7-b9-b10`.

The last basic block in the path is b10 because handle_postponed, which is invoked in that basic block, contains an assertion which would fail considering this path, and so the program execution would stop.

If we want to ignore the fact that we know about the assertion, and so if we want to conduct an analysis based only on the CFG, a path would be: `b1-b2-b3-b2-b4-b6-b7-b9-b10-b11-b6-b12`.

Question 2) Assume that a software analyzer reports an alarm for the program path of Question 1: The report warns that the assertion executed in method "handle_postponed" fails with that program path. Is that report a true alarm or a false alarm, and why?

The assertion in handle_postponed fails if isUsingVirtualUnits is false, which is what the default value is and so the assertion **fails if switchToVirtualUnits** (which is the only method that can set the value to true) **never gets invoked**.

If the given path is **feasible then** the assertion would **fail**, since we avoid executing switchToVirtualUnits.

We must analyze the program and the given path.

We can assume there's only one process in the passed list primaryP, since it's enough for the analysis.
It's not important to know what secondaryP contains, since it doesn't get referenced in the given path.

First of all (basic block **b3**), the method always sets the priority of the process in primaryP to Process.HIGH_PRIORITY.
Then (basic block **b4**), the list toExec contains the process in primaryP, which now has a priority that equals to Process.HIGH_PRIORITY.
Since we want to avoid switchToVirtualUnits (from the basic block b4 jump to **b6**), primaryP.size() must be <= 8 (NUM_PHYSICAL_UNITS): considering the fact that we have a size of 1, it is perfect.
Finally, the foreach loop (basic block **b6**) iterates over the list toExec which only contains one process: we reach the basic block **b7**, but the problem is that we **can't jump to** the basic block **b9** because the priority of the process was set to Process.HIGH_PRIORITY and so **b8 gets executed instead**.

In conclusion, the report is a **false alarm**, because the given path is **infeasible** since if we skip the call to switchToVirtualUnits then toExec only contains the processes from primaryP, but every primaryP process has the priority set to Process.HIGH_PRIORITY and so we will always execute the basic block b8 instead of b9.

EXTRA: the conclusion also takes into consideration the case where we pass an empty primaryP list (so the size equals to 0 that is also <= 8), since the executed path would be b1-b2-b4-b6-b12: **handle_postponed can't get executed in any path where switchToVirtualUnits is skipped**.