Prof. Bernd Finkbeiner
Dr. Calogero Zarba
Moritz Hahn

Sample Solution for Problem Set 10
Due: July 17, 2007, 9:14 am

Embedded Systems

Problem 1 (Scheduling)

The solution is given in figure 1.

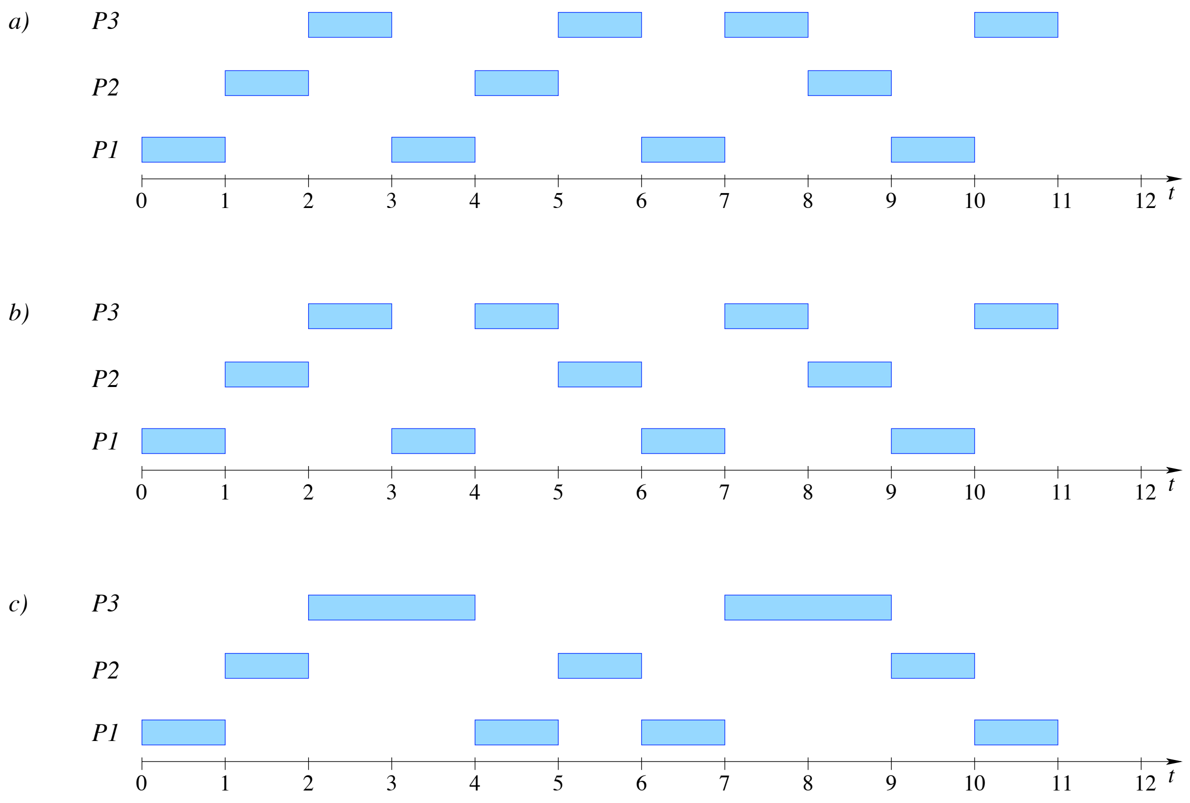


Figure 1: Exercise 1: Scheduling by different strategies

Problem 2 (Scheduling)

(a) The task set stays schedulable, because

$$\sum_{i=1}^{n} \frac{C_i}{T_i} = 2 \cdot \frac{4}{200} + \frac{1}{10} + \frac{2}{40} + \frac{6}{50} = 0.31 \leq 0.74349 \approx n(2^{\frac{1}{n}} - 1)$$

holds. This test is a suffient criterion for RMS schedulability and was introduced in the lecture.
(b) Again we apply the sufficient criterion:

$$\frac{1}{10} + \frac{18}{100} + \frac{2}{20} + \frac{5}{50} + \frac{C_5}{25} \leq 5 \cdot (2^{\frac{1}{5}} - 1)$$

resolving this to C_5 we get:

$$C_5 \leq 6.587$$

So, if the computation time of τ_5 is no larger than 6.587 we can guarantee schedulability. Notice that we asked for the maximal value of C_5 here and didn't ask you to restrict this value to an integer, like 6.

Problem 3 (Scheduling)

Applying the sufficient criterion, you find out that schedulability is not guaranteed:

$$U = \frac{1}{3} + \frac{2}{4} + \frac{1}{8} \approx 0.96 > 0.78 \approx 3 \cdot (2^{\frac{1}{3}} - 1)$$

However, because this is only a sufficient but not a neccessary criterion and as $U \leq 1$, it may still be possible to find a schedule. It is actually, as one can see in figure 2.
Because of this, the correct answere is "yes".
(b) The task set is not schedulable:

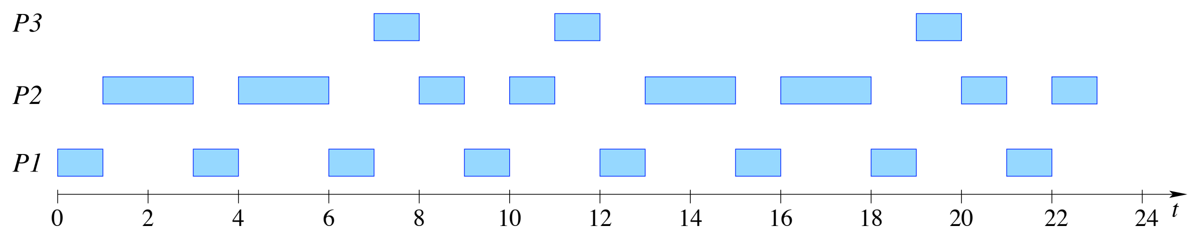


Figure 2: Exercise 3 (a): A schedule is possible

$$U = \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{13}{12} > 1$$

(c) This task set is also not schedulable:

$$U = \frac{1}{2} + \frac{1}{5} + \frac{2}{8} + \frac{1}{10} = \frac{21}{20} > 1$$

Problem 4 (Synthesis)

(a) See figure 3

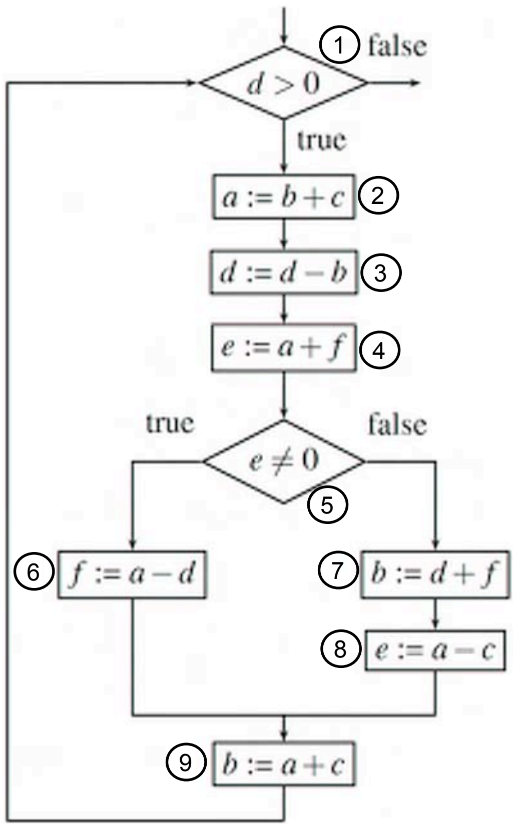


Figure 3: Exercise 4 (a): Control Flow Graph

(b) Before drawing the graph, we need to find out about the dependencies. Let i and j be statements and assume there is a path from i and j. In the lecture three different dependence classes where introduced:

- i defines a resource r, j uses it and the path from i to j does not contain other definitions of r: true dependence, RAW, t
- i uses a resource, j defines it and the path from i to j does not contain other definitions of r: anti dependence, WAR, a
- i and j define the same resource and the path from i to j does not contain other uses or definitions of r: output dependence, WAW, o

If there are several paths from i to j, you only need to find one which fulfills one of the criteria to have a dependence of the respective kind.
With these definitions we can talk now about the dependencies of the nine statements:

- 1: This statement uses d. It is true dependent of 3, because this is the only statement that defines d and we have a path (345691) from 3 to 1 on which d is not defined.
- 2: Here, a is defined and b and c are used. It is dependent on
  - 9 (t), because 9 defines b (used by 2) and there is a path from 9 to 2 in which b is not redefined.
  - 9 (a), because 9 uses a (defined by 2) and there is a path from 9 to 2 (912) on which a is not redefined.
  - 6 (a), because 6 uses a (defined by 2) and there is a path from 6 to 2 (6912) on which a is not redefined.
  - 4 (a), for the same reason
  - 8 (a), for the same reason
- 3: Here, d is defined and d and b are used. Dependencies are
  - 1 (a), because it depends on d (3 defines it) and there is a path (123) from 1 to 3 without any redefinitions of d.
  - 3 (a), because d is both used and defined in this statement (so, take path of length 0)
  - 3 (t), because d is both used and defined in this statement (so, take path of length 0)
  - 6 (a), because d is used in 6 (defined in 3) and there is a path from 6 to 3 (69123) without any redefinitions of d
  - 7 (a), because d is used in 7 (defined in 3) and there is a path from 7 to 3 (789123) without any redefinition of d. Notice that there is no true dependency of 3 from 7: 7 defines b, but it is redefined by 9 on all paths to 3.
  - 9 (t), because 9 defines b (used by 3) and there is a path (9123) from 9 to 3 where is is not redefined.
- 4: Here, e is defined and a and f are used. Dependencies:
  - ...
  - 8 (o), because e is defined by 8 and by 4 and there is a path from 8 to 4 (891234) in which e is e is neither used nor defined. Notice that there is no output dependency of 8 from 4, because e is used in 5 which is executed on all paths from 4 to 8.
  - ...
- ...

The rest of the dependencies can be found likewise. The full dependence graph is given in figure 4. Here, dottet lines represent dependences between different loop iterations.

(c) The set of live variables at the beginning of each statement can be determined by a backward analysis. This means that the edges of the control flow graph are traversed in reverse order. If a variable is used, it is inserted in the set of live variables; at a definition point the variable is removed from it (if it is not inserted at the same time). At joins in the control flow graph the union of the corresponding sets is taken. The analysis proceeds until a stable state has been reached.
In figure 5 this algorithm is illustrated and the final result is given. We simulate the assumption that b, c, d, e and f are live at the end of the loop by placing them at the entry of 1 at the beginning.

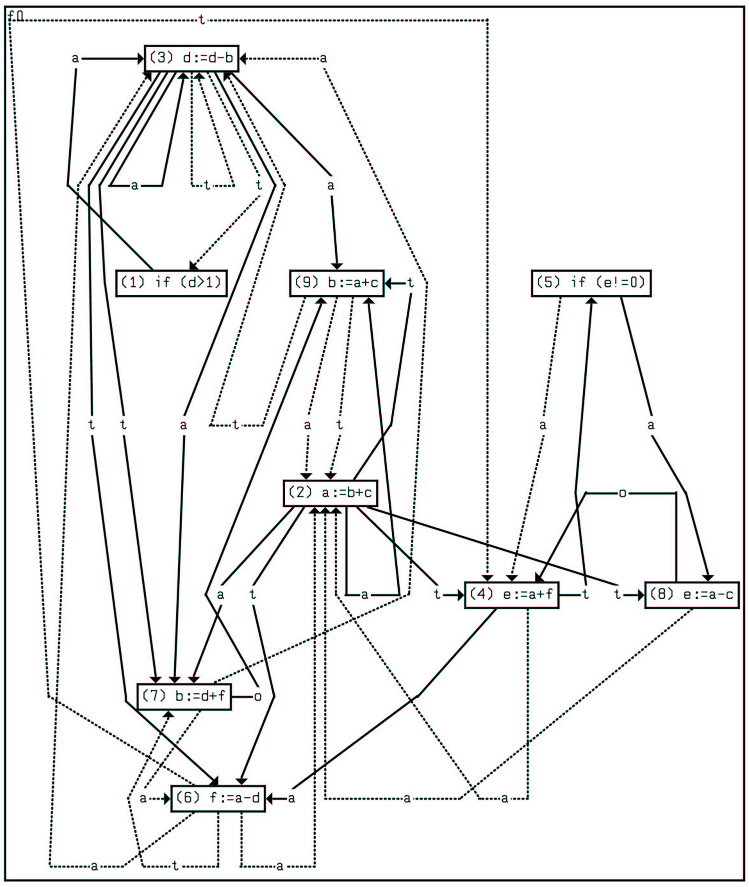
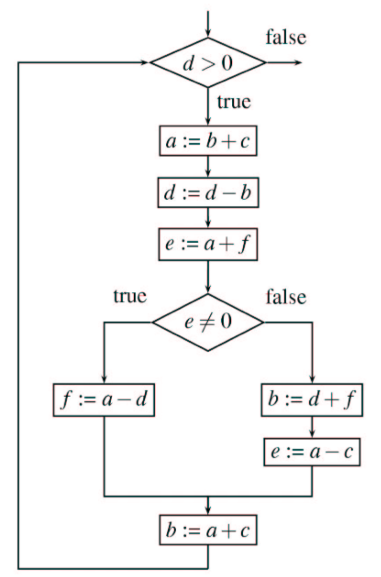
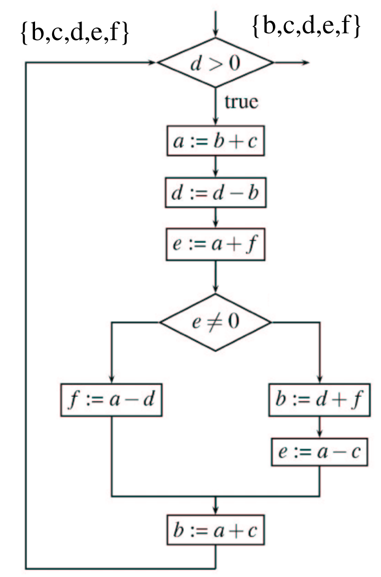


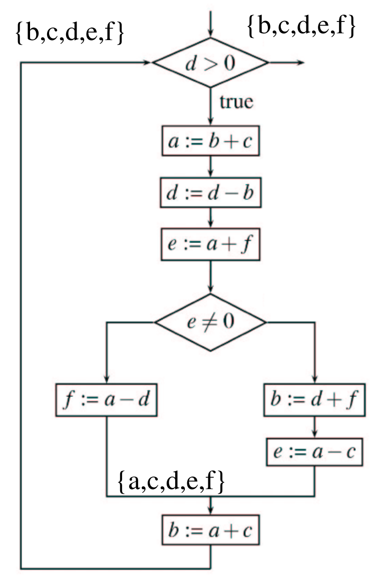
Figure 4: Exercise 4 (b): Data Dependence Graph



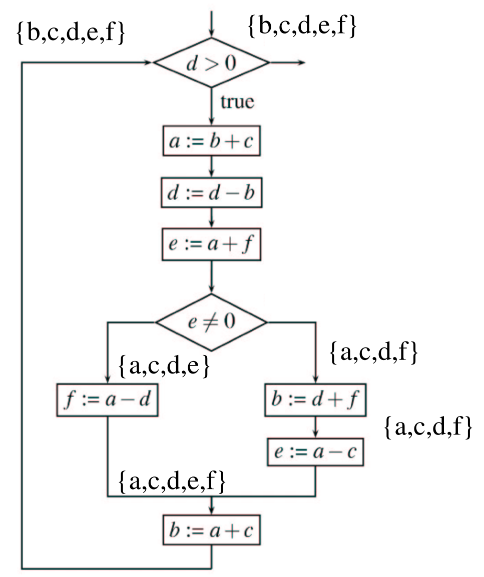
1: Before executing the algorithm



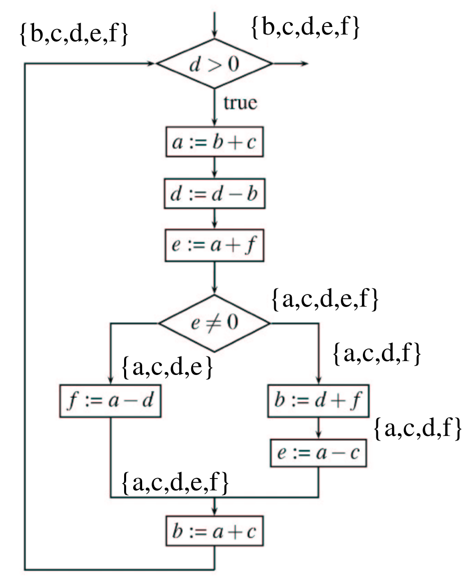
2: handling d > 0: d is added to set, but it’s already there



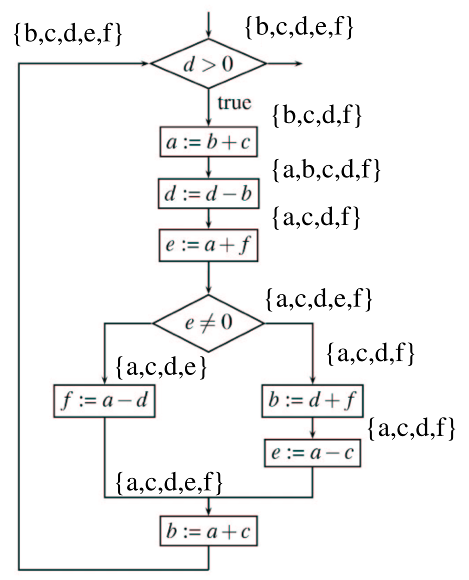
3: handling b:=a + c: b is removed, a and c are added



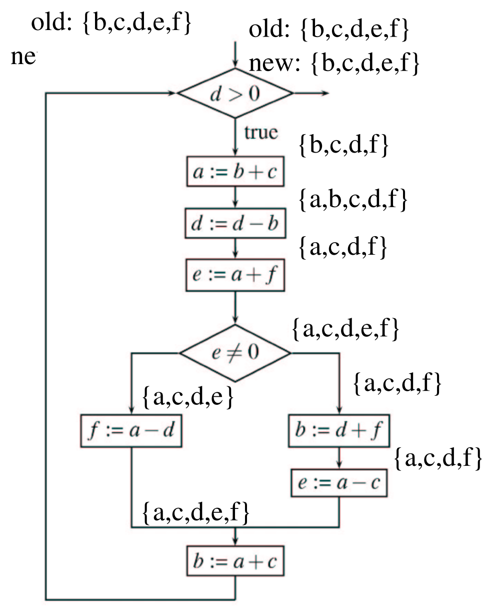
4: handling several instructions as before on the two seperate branches



5: merging



6: handling instructions as before



7: merging, replacing old variable set

Continuing in the loop again, you would notice that nothing changes, because we start with the same values as be– fore. Because of this, we get the following set of live va– riables from 7:
(statement #) : set
1: {b,c,d,e,f}
2: {b,c,d,f}
3: {a,b,c,d,f}
4: {a,c,d,f}
5: {a,c,d,e,f}
6: {a,c,d,e}
7: {a,c,d,f}
8: {a,c,d,f}
9: {a,c,d,e,f}

Figure 5: Exercise 4 (c): Illustration of the Algorithm