uOttawa

L'Université canadienne
Canada's university

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

COURSE: SEG2106
SEMESTER: WINTER 2009

PROFESSOR: Gregor v. Bochmann
DATE: February 27, 2008
TIME: 13:00 to 14:30

MIDTERM EXAMINATION

Solutions

NAME and STUDENT NUMBER:_____/_____

Mid-Term Exam

There are three (3) types of questions in this examination.

Part 1	**Multiple choice and simple answers**	**12 marks**	
Part 2	**Short development questions**	**36 marks**	
Part 3	**Problem solving**	**8 6 marks**	
Total		**56 54 marks**	

The space allocated for each question is limited. In case of necessity you may use the other side of the pages to continue.

Annex A: The Daemon-Game specification in SDL
Annex B: Algorithm 3.3 (Thompson)
Annex C: Telelogic TAU graphical UML notation

Multiple choice and short-answer questions [2 marks each]:

1. Which of the following statements is correct ?
   1. Validation is an activity to determine whether a system conforms to its specification.
   2. Verification is an activity to determine whether a system satisfies the requirements of the users and owners.
   3. **X** Process quality corresponds to the conformance between the system and the defined requirements.
   4. Process quality and system quality are two different terms that have essentially the same meaning.

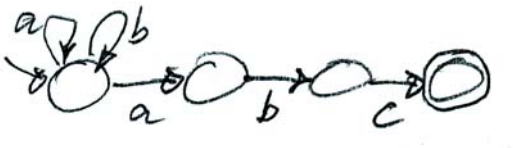
2. What value does the predefined SDL variable PARENT contain ? - Give a short answer !
   Answer: It contains a reference to the process that created the current process (self)

3. What value does the predefined SDL variable SENDER contain ? - Give a short answer !
   Answer: It contains a reference to the process that sent the message that is being processed by the current transition.

4. Explain in a few words why one can say that the Daemon-Game specification given in Annex A uses the factory design pattern. What is the pattern ? – How is it used ?
   Answer: The pattern involves the factory object instance that has a method for creating instances of another type of objects. – In the Daemon-Game, the Monitor plays the role of the factory, and the Game processes are created.

5. Which of the following is not in the set of strings denoted by the regular expression (a* b c*)*
   a) abc
   b) **X** bacc
   c) abcbc
   d) babbc

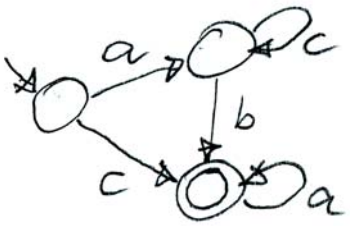
6. Which of the following defines a language different than the others ?
   a. The regular expression (a | b)* a b c
   b. The regular expression (a* b*)* a b c
   c. **X** The regular expression (a | b) (a | b)* c
   d. The accepting automaton below



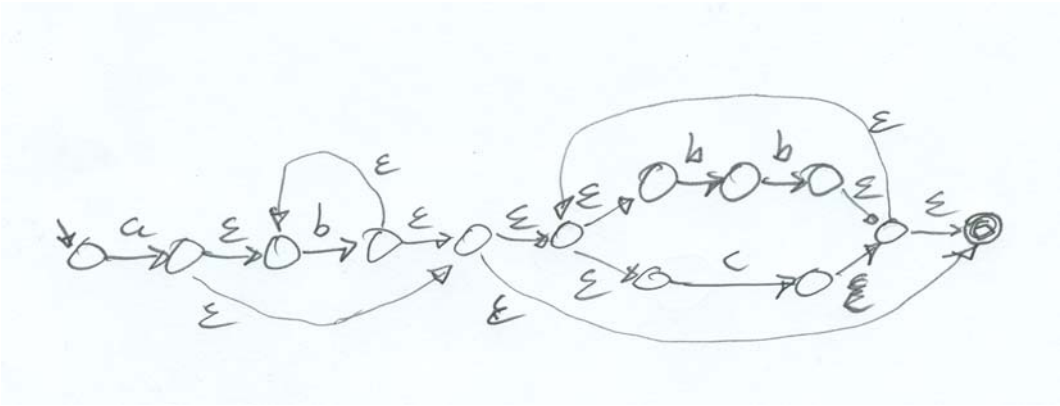
Short development questions

7. **[4 marks]** Write down a regular expression that defines the regular language accepted by the following automaton:

   Answer: **(a c* b | c) a***



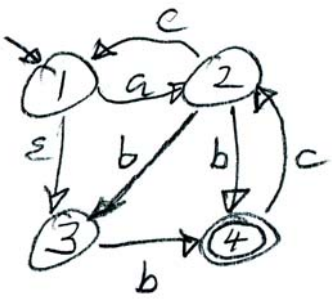
8. **[8 marks]** Closely follow the algorithm of Thomson (see Algorithm 3.3 in Appendix A if needed) to construct a non-deterministic automaton that accepts the language defined by the regular expression " a b* (c | b b)* "

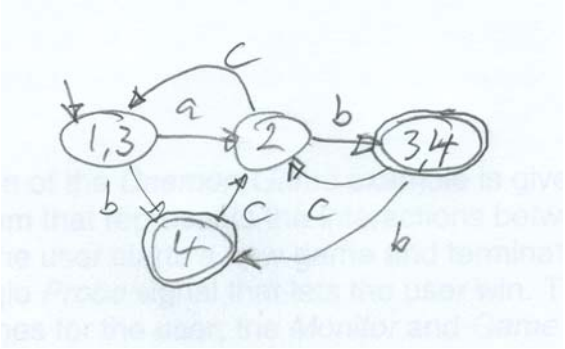


9. **[4 marks]** Write a regular expression that represents the hundred years up to 2005, that is, the numbers between 1906 and 2005 (inclusively).

Answer: 19 (0 (6 | 7 | 8 | 9) | digit1 (0 | digit1) | 200 (0 | 1 | 2 | 3 | 4 | 5)
where digit1 = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

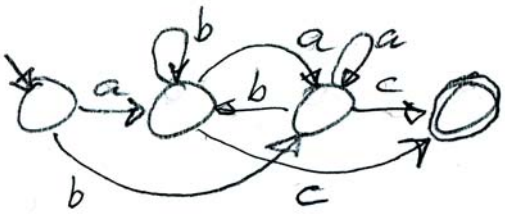
10. **[8 marks]** Closely follow the algorithm discussed in class to convert the following non-deterministic automaton into an equivalent deterministic automaton. **Note**: The minimization of the deterministic automaton is not asked for here.

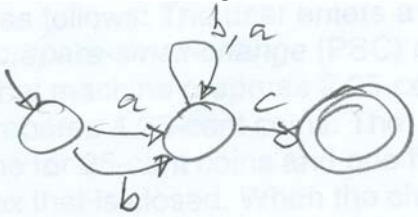




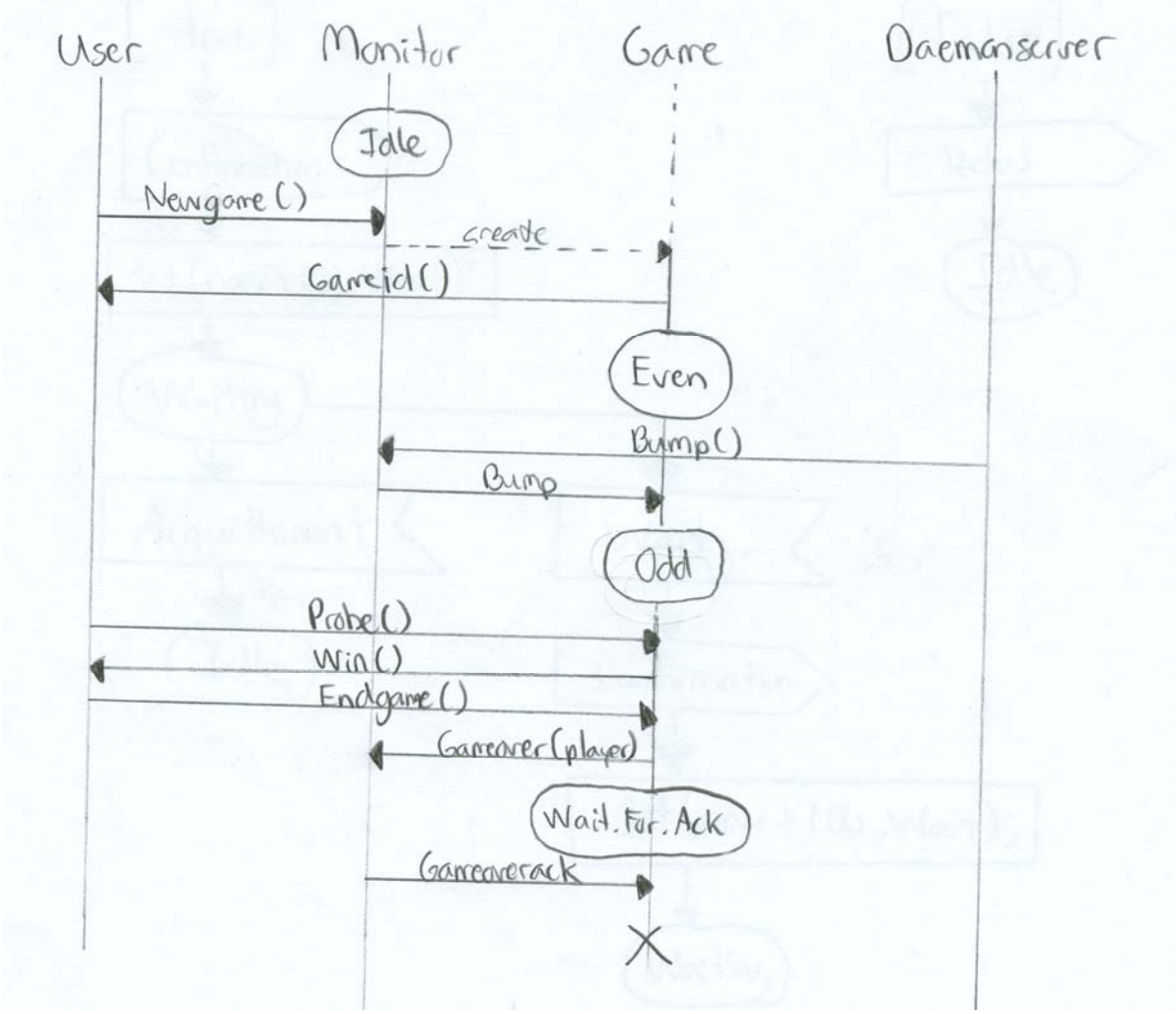
Please note: The subset {3, 4} represents a different state in the deterministic automaton than the subset {4}, even if the latter is included in the former.

11. **[4 marks]** Find the minimal automaton equivalent to the following automaton.





12. **[8 marks]** The SDL specification of the *Daemon-Game* example is given in Annex A. Please draw a sequence diagram that represents the interactions between the user and the Daemon-Game when the user starts a new game and terminates the session after having sent a single *Probe* signal that lets the user win. The sequence diagram should show vertical lines for the user, the *Monitor* and *Game* processes and the Daemon that communicates with the *Monitor* through the *Daemonserver* interface.



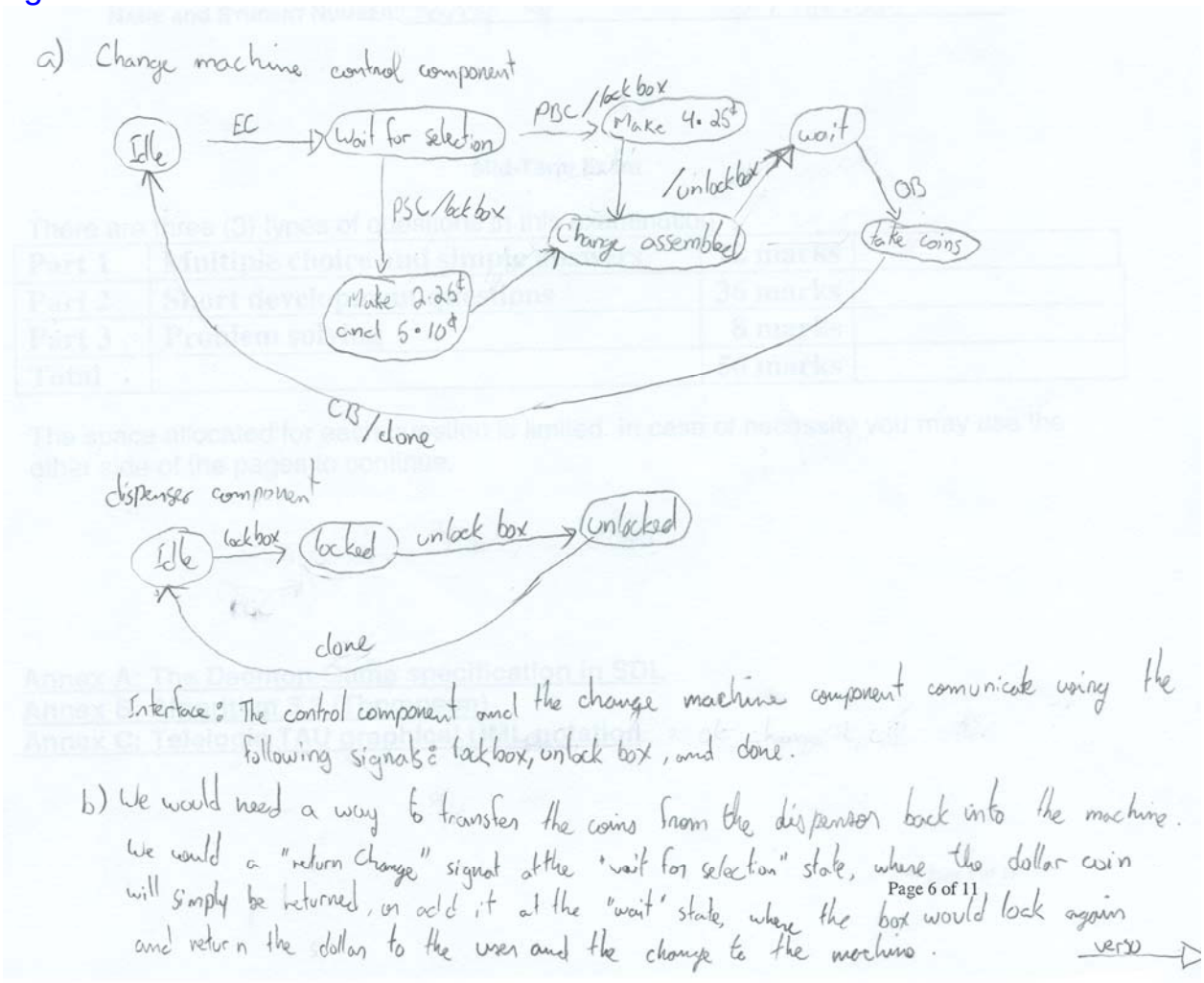
Problem Solving

13. **[8 marks]** [I did not find this question very clear. Therefore I have reduced the number of points from 8 to 6] A simple coin change machine works as follows: The user enters a one-dollar coin (EC) then pushes one of the buttons *prepare-small-change* (PSC) or *prepare-big-change* (PBC). If PSC was pushed that machine prepares 2 25-cent coins and 5 10-cent coins, in the other case, it prepares 4 25-cent coins. The machine contains two dispenser components, one for 25-cent coins and one for 10-cent coins. The change is first assembled in a box that is closed. When the change is assembled, the user may open the box (OB) and take out the change. Then the user has to close the box (CB) and the machine goes back into its initial state.

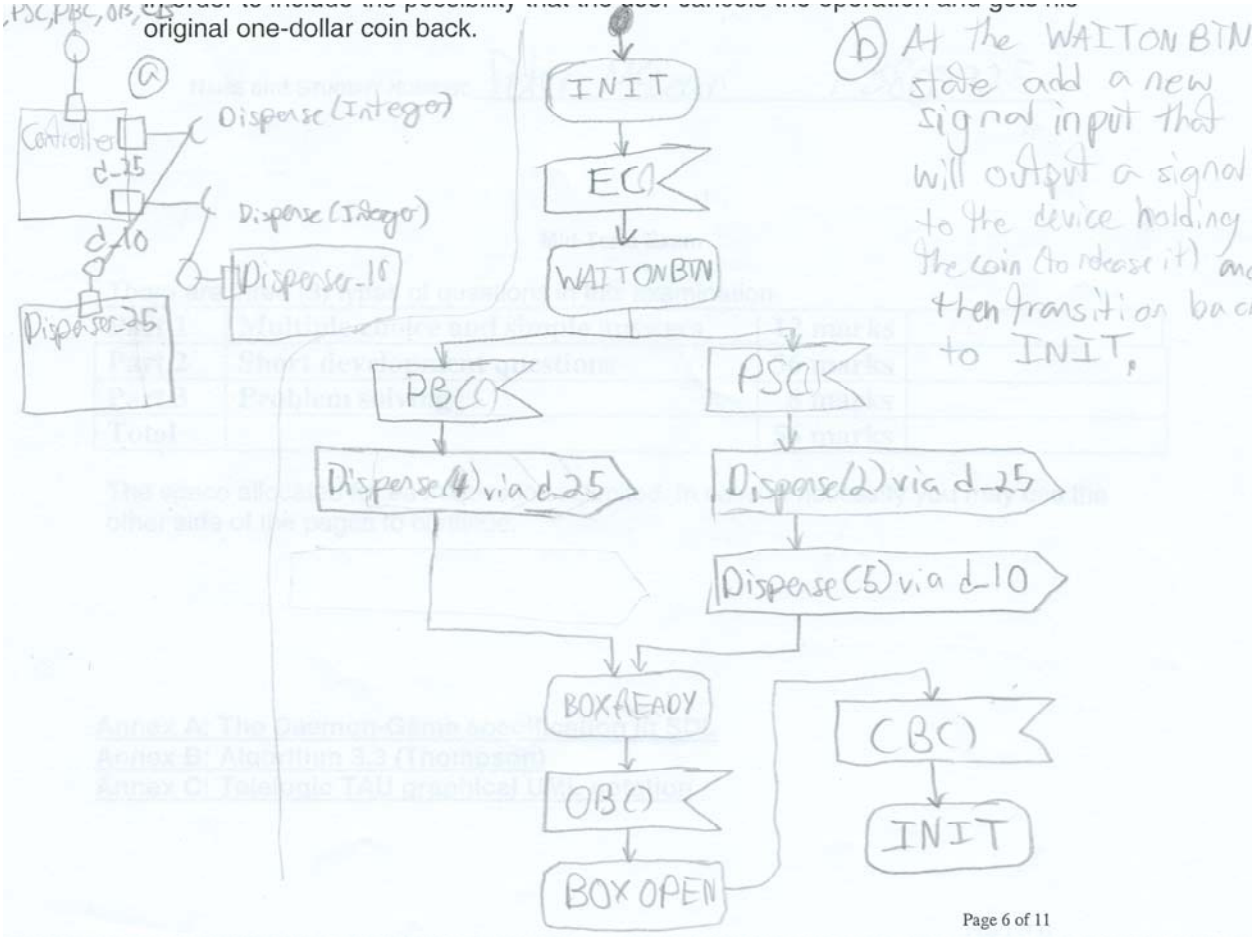
(a) Please make a model, in the form of an LTS drawn as a state diagram, of the behavior of the control component of the change machine. This component communicates with the user through the interactions EC, PSC, PBC, OB, CB, and with the dispenser components through appropriate interactions. You should define the interface with the dispenser components and give an LTS which models the behavior of the controller.

(b) Explain in a few words what kind of changes are required in the behavior model in order to include the possibility that the user cancels the operation and gets his original one-dollar coin back.

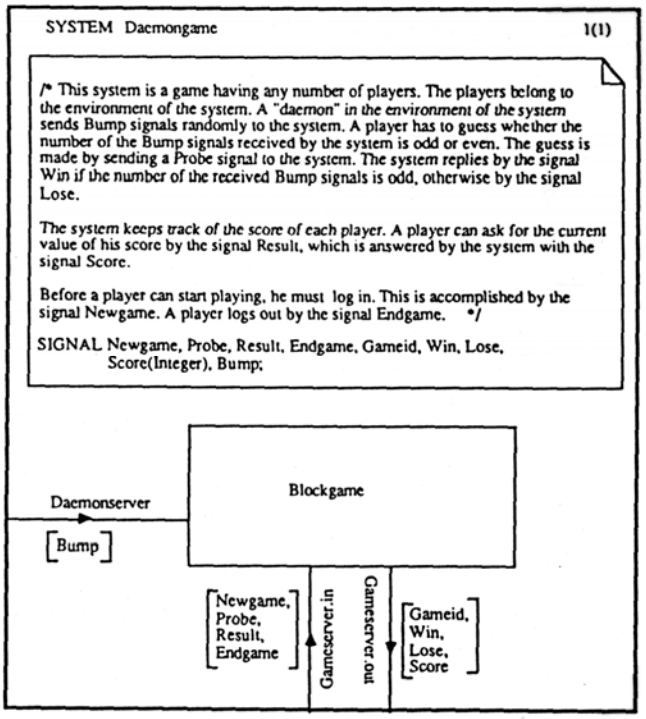
The following are two good solutions, one using the LTS notation, the other using the UML-SDL notation.

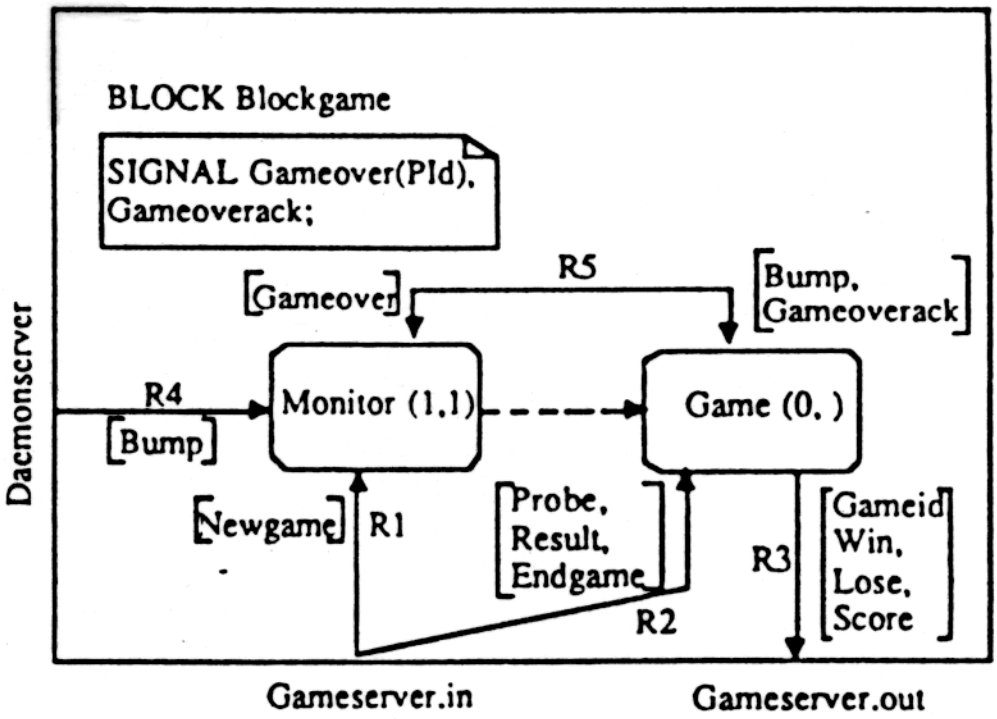


,PSC,PBC,OB, order to include the possibility that the user cancels the operation and gets his original one-dollar coin back.



Annex A: The Daemon-Game specification in SDL





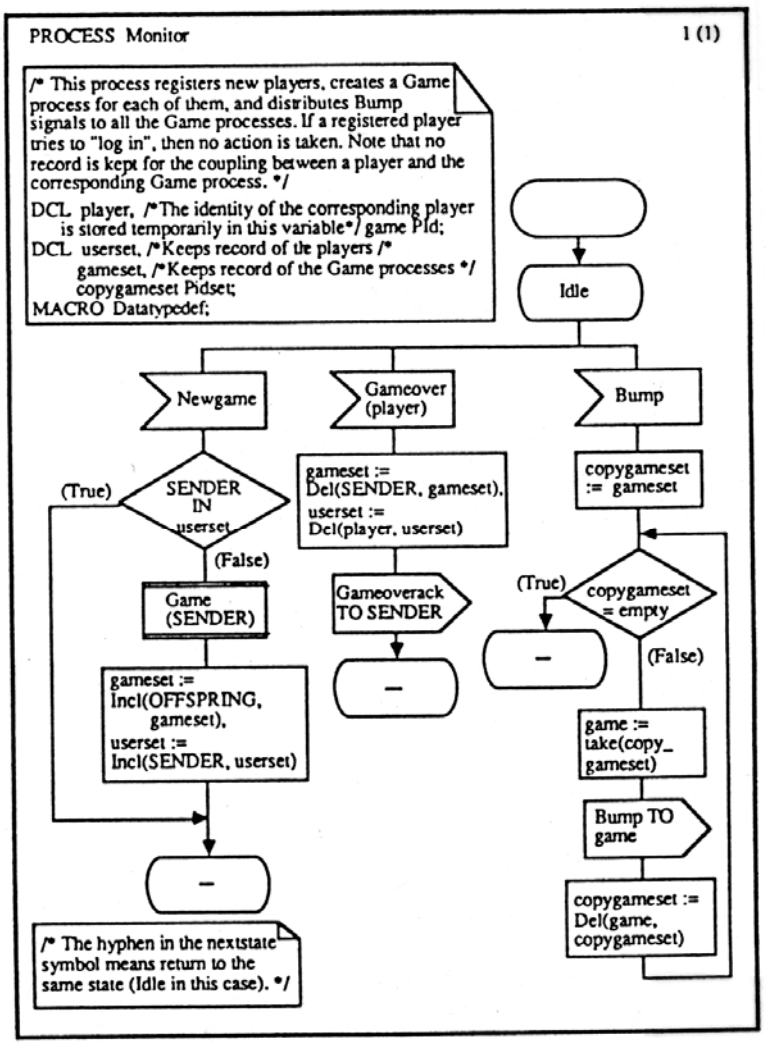
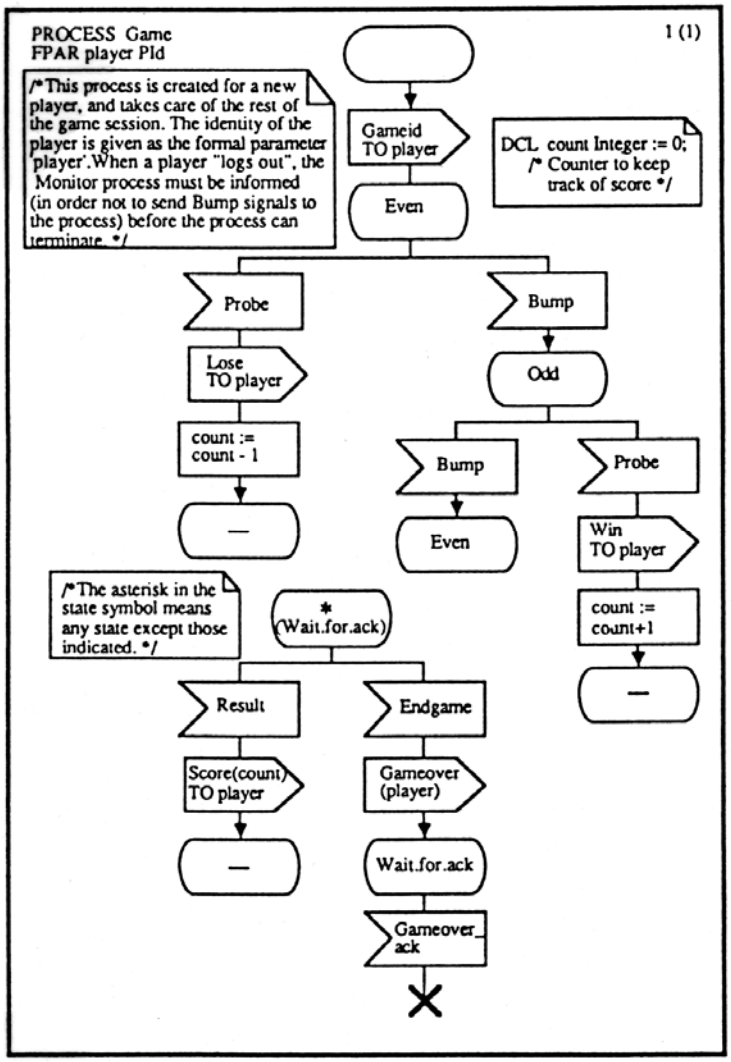
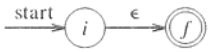
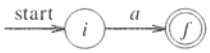


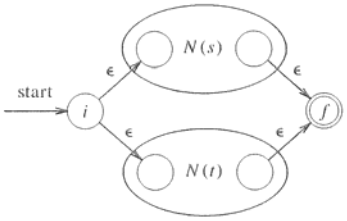
Fig. 20. Process Monitor.

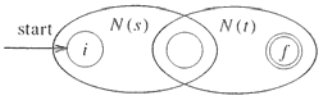


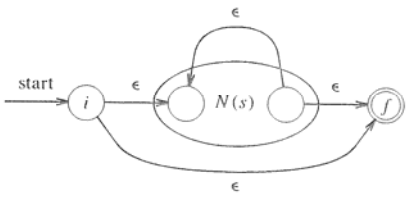
Annex B: Algorithm 3.3 (Thompson)

- First parse *r* into its constituent subexpressions.
- Construct NFA's for each of the basic symbols in *r*.
  - for ε 

  - for *a* in Σ 

  - For the regular expression *s*|*t*, 

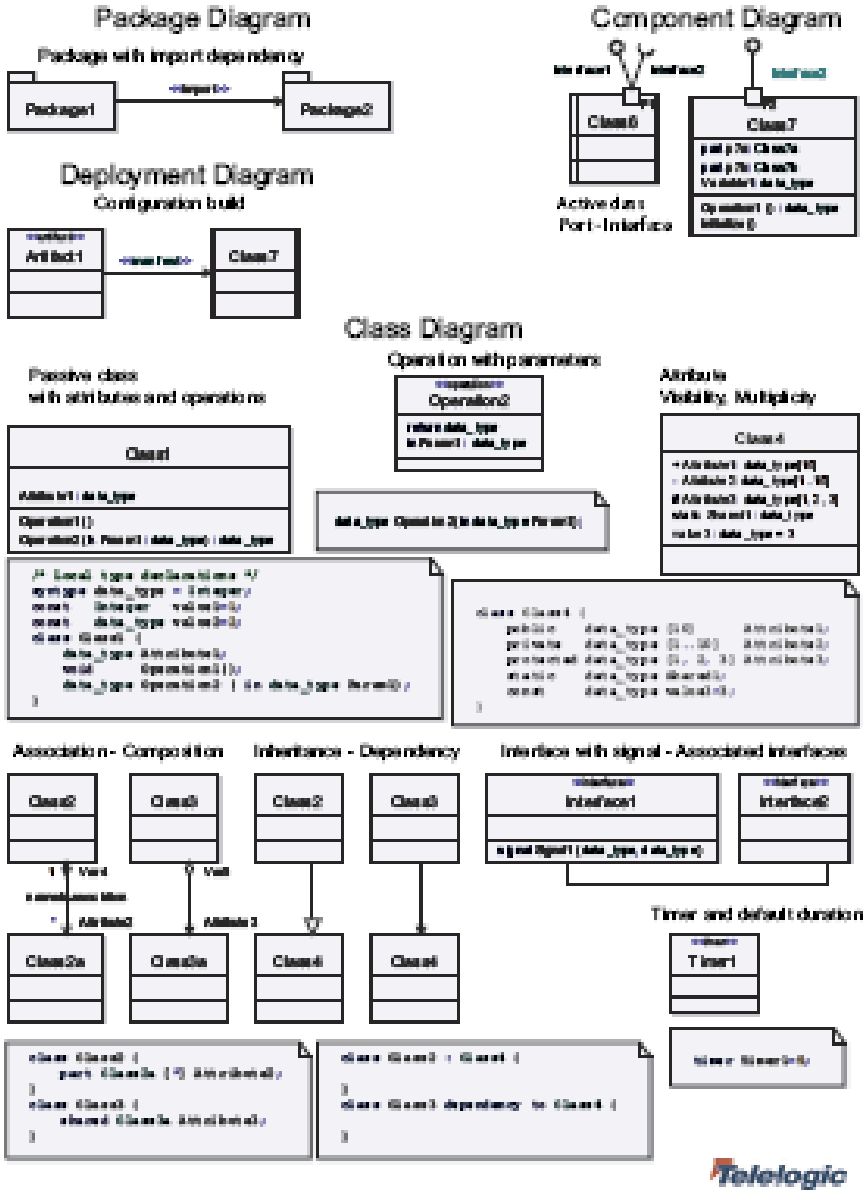
  - For the regular expression *st*, 

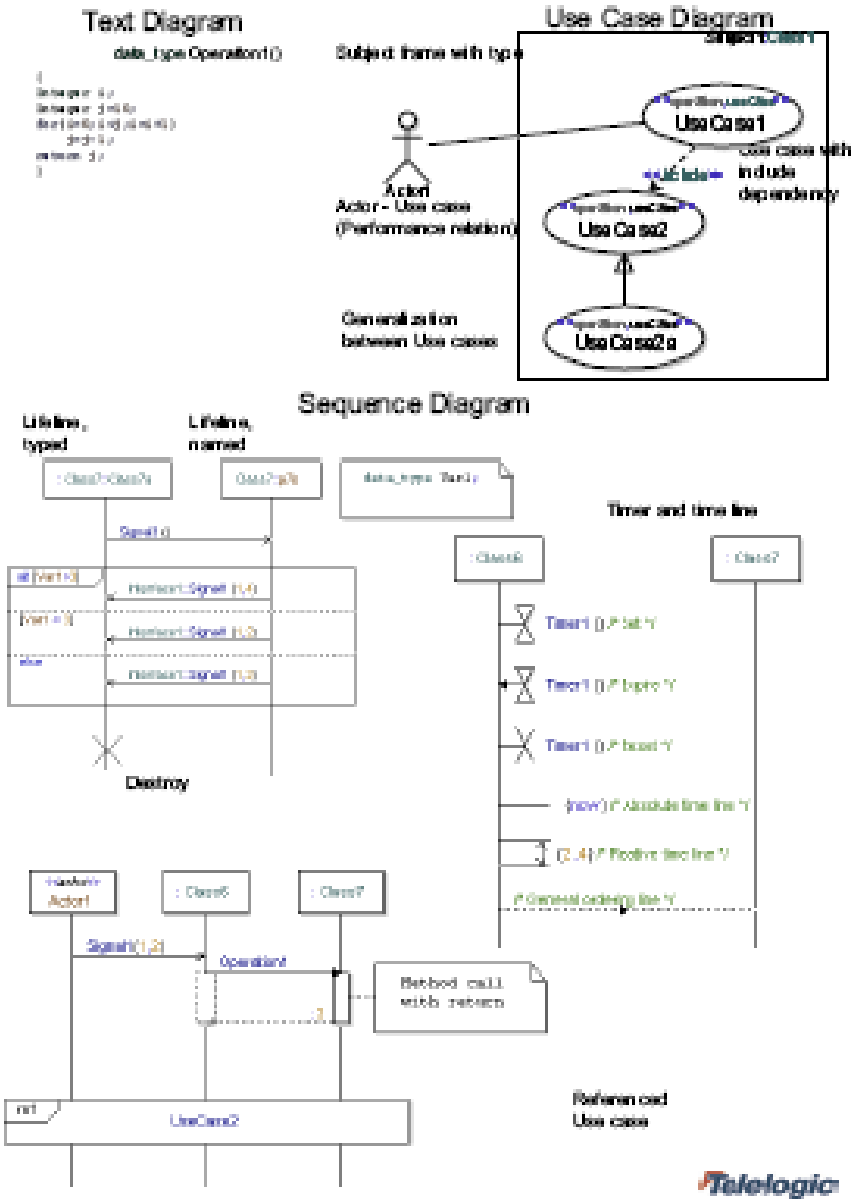
  - For the regular expression *s**, 

  - For the parenthesized regular expression (*s*), use *N(s)* itself as the NFA.

Every time we construct a new state, we give it a distinct name.

Annex C: Telelogic TAU graphical UML notation


Package Diagram
Package with import dependency
Package1
Package2
Component Diagram
Class6
Class7
Active class
Port - Interface
Deployment Diagram
Configuration build
Artifact1
Class7
Class Diagram
Passive class
with attributes and operations
Class1
Operation with parameters
Operation2
Attributes
Visibility, Multiplicity
Class4
Association - Composition
Inheritance - Dependency
Interface with signal - Associated interfaces
Interface1
Interface2
Class2a
Class3a
Class4
Class5
Timer and default duration
Timer1


Text Diagram
Use Case Diagram
Actor
Actor - Use case
(Performance relation)
UseCase1
UseCase2
UseCase2a
Use case with
include
dependency
Generalization
between Use cases
Sequence Diagram
Lifeline,
typed
Lifeline,
named
Destroy
Timer and time line
Actor1
Operation1
Method call
with return
ref
UseCase2
Referenced
Use case