Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering and Networks Laboratory
Computer Engineering Group (TEC)

HS 2015 Prof. L. Thiele / R. Jacob

Discrete Event Systems

Exercise Sheet 14

This last exercise session tries to put everything together somehow. You will discover a modeling tool for time Petri nets, which is paramount in the evaluation of a system design. You will mainly use the "stepper simulator", which randomly plays the token game. In the last part of the exercise, you will use the embedded model-checker to tune some design parameters.

1 Warming up: Calculating with Petri nets

In this exercise you are supposed to model a function $f_i(x, y)$ using a Petri net. That is, the Petri net must contain two places P_x and P_y that hold x and y tokens respectively in the beginning. Additionally, the net must contain one place P_z which holds $f_i(x, y)$ tokens when the net is dead. The Petri nets are supposed to work for arbitrary numbers of tokens in P_x and P_y.

a) $f_1(x, y) = 5x + y \quad \forall x, y \geq 0$

b) $f_2(x, y) = x - 2y \quad \forall y \geq 0, x \geq 2y$

c) $f_3(x, y) = x \cdot y \quad \forall x, y \geq 0$

Hint: You may have to use inhibitor arcs for **c)**. An inhibitor arc between a place and a transition prevents the transition from firing as long as there is at least one token in the place.

2 Simulate your Petri nets with TINA

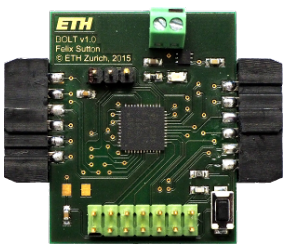
In this exercise, you will learn how to use the modeling tool TINA (**TI**me petri **N**et **A**nalyzer). The objective is to implement your solutions from Exercise 1 and simulate the nets.

a) On my webpage (`http://people.ee.ethz.ch/~jacobr/teaching.html`) you will find a simple tutorial for basic uses of the TINA software. Complete the reading of Section 1 before moving on.

b) To practice, create new net files and implement your solutions from Exercise 1. Using the stepper, try them out for different values of x and y.

3 Queue sizing and overflow management of BOLT

BOLT is an ultra-low power processor interconnect that decouples arbitrary application and communication processors with respect to time, power and clock domains. BOLT supports asynchronous message passing with predictable timing characteristics, and therefore making it possible for the system designer to construct highly-customized platforms that are easier to design, implement, debug, and maintain. You can find more information on the webpage: `http://www.bolt.ethz.ch/`



In this exercise, we look down to the practical problem of queue sizing and overflow management, using BOLT architecture as a case study.

Assume on one end BOLT is connected to a sensor, which delivers data at a given frequency and writes into BOLT queue. At the other end, a communication processor is in charge to read these messages from BOLT and to send them through a wireless sensor network. The problem is that the network is not always available (other processes also need the bandwidth) and the communication processors have limited memory to store messages prior sending them.

The task is to evaluate the design parameters (queue size, resource management scheme, ...) such that overflows in the BOLT queue (which would result in packet losses) are avoided.

3.1 The nice and easy deterministic world

a) Download the initial Petri net (Ex3.1) from my webpage: `http://people.ee.ethz.ch/~jacobr/teaching.html`. Open it in `nd`.

b) This initial model is untimed. Extend it with delays on transitions such that the net follows the earliest firing rule and knowing that:

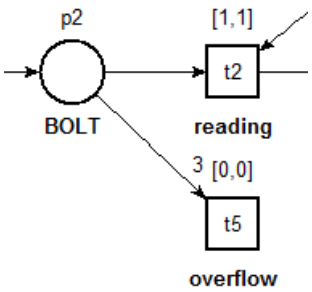
- Task of reading and writing to BOLT as well as sending a message to the network take one time unit each.
- The sensor produces one message every 5 time units.
- At steady state, the communication processor can use the network for 10% of the time.

c) Play the token game using the stepper. Is the net bounded? Why is it not surprising?

d) Assume BOLT has now a maximal capacity of two massages. How would modify the net to implement this?
Hint: The easiest way is to add a capacity to the `BOLT` place...

On the physical system, BOLT cannot prevent a write operation if its queue is full. Hence, the previous solution to avoid overflows is not satisfying, as it yields that the write operation is forbidden if there is no more memory available in BOLT, so the systems is actually waiting for the resource to be available. In our case, we would like to be sure instead that we have enough network resource available such that there is no overflow in BOLT, without controlling the write operations.

As a result, we need our Petri net to model the (possible) overflow of BOLT in order to test our design. Among other options, we choose to do that by adding a sink transition downstream the `BOLT` place, as shown thereafter.



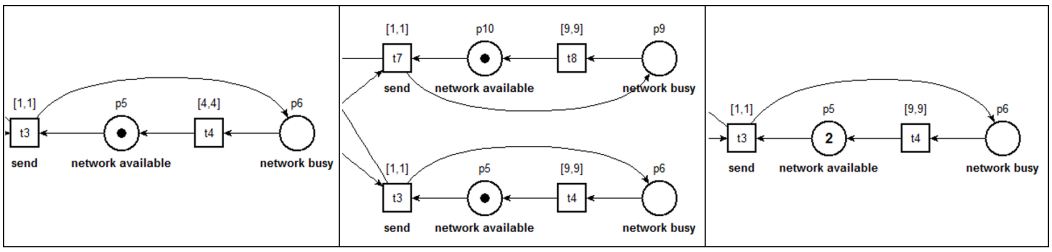
If it happens that 3 tokens are in the `BOLT` place, the overflow transition is enabled and can fire. This is used as a flag to detect an overflow of BOLT.

e) In our case, we want to avoid overflows by increasing the amount of `network` resource available. More precisely, we want to double it. This can be done in several ways:

- Assume you get the network 20% of the time.

- Assume there are two networks, that you get 10% of the time each. Further assume the communication processor can read one message per network (e.g., if there are two networks, you should have 2 tokens in place $p4$). You can either model explicitly two independent networks, or set a second token in the `network available` place.

These three options are illustrated below



a) b) c)

Try out these options. Why solution **c)** does not solve our problem? Which of solution **a)** or **b)** seems the more flexible to you?

f) Assume now the incoming packets arrive in burst of 5 messages every 10 time units. Consider again solutions of type **a)** and **b)** with increased bandwidth:

- One network that you get 50% of the time.
- Five networks that you get 10% of the time each.

Model the bursts and try out these options. Which solution seems the more flexible to you now?

As you can see, even in this very basic scenario, a correct analysis of the system design is not so easy. Coming up with correct design parameters is hard. In practice, it is almost impossible for real-life systems. This is why test and verification methods have been developed and increasingly used over the past decades.

3.2 Realism yields non-determinism

Let us now consider a slightly more realistic model.

a) Download the new Petri net structure (Ex3.2) from my webpage: `http://people.ee.ethz.ch/~jacobr/teaching.html`. Open it in `nd`.

For each task, there is no precise execution time anymore but a time interval instead. This represents the Best- and Worst-Case Execution Time (BCET/WCET). The resource management mechanism and the communication processor are also different.

- The processor can read and store several messages from BOLT before sending them.
- When the processor accesses the network to send messages, it will keep on sending (i.e., no releasing the resource) until it has no more message left in memory.
- It is uncertain when the network will be available again.

b) Complete the reading of Section 2 of the TINA tutorial.

The question is to determine how much memory we need in the communication processor to ensure that BOLT will never overflow, no matter what happens. To do that, we will use the LTL model-checker embedded within TINA. LTL is a different logic than CTL. For our simple purposes here, we can say the difference is that the E quantifier doesn't exist and the A quantifier is always implied. Furthermore, G writes $\Box$, F writes $\Diamond$, and X writes $\circ$. This is summarized in the table below:

CTL	AF a	AG a	EF a	AX a	...
LTL	$\Diamond$ a	$\Box$ a	$\neg$ AG $\neg$ a	$\circ$ a	...
in TINA	`<> a ;`	`[] a ;`	`- [] - a ;`	`() a ;`	...

c) What does the LTL property $\Diamond t5$ mean in natural language? Does our model verify this property? Are we happy about this?

d) What is the LTL property our system must verify in order to guarantee that BOLT will never overflow?

e) What is the smallest capacity for the place $p4$ that makes this true.