Lab1: A net-list simulator

Georges-Axel Jaloyan <georges-axel.jaloyan@ens.fr>
Balthazar Patiachvili <balthazar.patiachvili@ens.psl.eu>

The required files for this lab session can be downloaded on the following page: `https://github.com/hbens/sysnum-2025`.

You can clone the repository using the following commands:

```
git clone --recursive https://github.com/hbens/sysnum-2025
```

More precisely, the following files can be found in it.

- `tp1.pdf`: the document you are reading
- `tp1/*`: sources of the scheduler (that you will have to complete) as well as some tests files and a net-list simulator.
- `project25.pdf`: the project description.
- `carotte.py/*`: sources, and some examples for the CAROTTE.PY compiler generating netlists. In the second part of the project, you are encouraged to look at those files, and modify them as you wish.
- `doc/*`: various documentation on the languages you will use in this course (including the syntax used for *net-lists* as well as specification of CAROTTE.PY domain specific language).

Scheduling

The CAROTTE.PY compiler generates its net-lists (a representation of digital synchronous circuits) in an arbitrary order. As a result, they are not adequate to sequential simulation, as the first line may depend for its inputs on the outputs of a subsequent line. In order to build such a simulator, we are required to do the reordering before sequentially simulating the instructions. We will use a topological sorting algorithm on the data dependency graph of the circuit.

In the first part of this lab session, a net-list simulator is provided to help you testing your code. In the second part (which is the first part of the project), you will be required to write the net-list simulator.

Question 1 *The file `tp1/graph.ml` contains a basic implementation of directed graphs. Complete the following two functions:*

```
(* Returns true if the graph has a cycle, false if not *)
val has_cycle : 'a graph -> bool
(* Returns the list of nodes sorted in topological order *)
val topological : 'a graph -> 'a list
```

You can test and verify your code by typing the following commands in a shell:

```
> ocamlbuild graph_test.byte
> ./graph_test.byte
```

In what follows, we will build the graph processed in the previous question, from a net-list. Each node of the graph represents a variable of the circuit while each edge describes a dependency relationship: `x` $\rightarrow$ `y` *if and only if* there is an equation with input `x` and output `y`.

Question 2 *Implement a function* `read_exp` *(in the file* `tp1/scheduler.ml`*) which returns the input variables of an equation, with type signature:*

```
val read_exp : Netlist_ast.equation -> Netlist_ast.ident list
```

(We ignore for now memory variables, which will be handled in question 4, details on the grammar of net-lists can be found in file `tp1/netlist_ast.ml`*).*

Question 3 *Implement a function* `schedule` *(in the file* `tp1/scheduler.ml`*) which takes as input a net-list and returns the topologically sorted net-list in order to sequentially execute it:*

```
val schedule : Netlist_ast.program -> Netlist_ast.program
```

Your function must raise the `Combinational_cycle` *exception if the net-list contains a cycle (We ignore for now memory variables, which will be handled in question 4).*

To test your scheduler, you can compile your code with:

```
> ocamlbuild scheduler_test.byte
```

and then test your program on a test file such as:

```
> ./scheduler_test.byte test/fulladder.net
```

The program creates a new version of your net-list (like `fulladder_sch.net`), and then runs the simulator on this net-list. You can chose the number of simulation steps with the option `-n <steps>`. The option `-print` allows you to just print the sorted net-list on standard output without simulating it.

Registers allow to break dependency loops by introducing a time delay between its input and output. In this way, the equations whose input depend on the output of a register declaration are allowed to write into an input dependency of this register declaration. This you are able to read the output value of a register before writing into its input value.

Question 4 *Add to your scheduler the handling of registers.*

A net-list simulator (project part 1)

A simulator is a program that evaluates step-by-step a net-list using an *environment* to store the values computed at the previous step. This environment maps each identifier (`Netlist_ast.ident`) to a value (`Netlist_ast.value`).

Question 5 *What is the general architecture of a simulator that manages only loop-less circuits? Write the corresponding pseudo-code.*

Question 6 *How to handle registers and memory?*

The `netlist_simulator.ml` file contains the skeleton for a basic simulator.

Question 7 *Complete* `netlist_simulator.ml` *to implement netlist simulation.*

To test your scheduler, you can compile your code with:

```
> ocamlbuild netlist_simulator.byte
```

and then test your program on a test file such as:

```
> ./netlist_simulator.byte test/fulladder.net
```

Make sure you simulator correctly simulates all given tests.