Causal Block Diagrams Assignment

1 Discrete Time Causal Block Diagrams

Discrete time Causal Block Diagrams (CBDs), as the name implies, are used to describe processes that evolve intermittently, or that can be abstract as such.

More information about CBD can be found in [1].

The purpose of this assignment is that you build and test a Discrete time CBD simulator by completing the code already provided as a starting point.

1.1 Tasks

1.1.1 Discrete Time CBD simulator

Implement a CBD simulator for Discrete time CBDs by filling in what is missing in the provided source files. Take the time to go through the folder structure and the code to understand the general architecture of the simulator. The python script that you have to fill in is the `CBD.py`, under the `Source` folder.

The main classes `CBD.py` are:

- `BaseBlock` class – An abstract class that represents a Block in a CBD. Specific blocks such as `ConstantBlock` extend this class.
- Specific block classes – A set of classes, each extending `BaseBlock` and representing a specific Block in a CBD.
- `Clock` – A class whose instance is used to keep track of the simulated time in a simulation.
- `CBD` – A class representing an entire CBD diagram.
- `DepGraph` – Represents a dependency graph. This dependency graph is used to determine the order of evaluation of the blocks at each simulation step.
- `DepNode` – Represents a node of the dependency graph.

The file `EvenNumbersCBD.py` shows how to build, draw, simulate and plot the hierarchical CBD shown in Figure 1, using the bokeh plotting library. You are not required to use any specific plotting library.

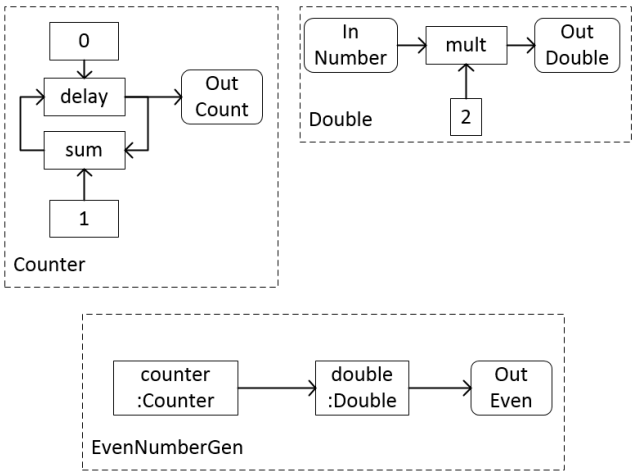

Figure 1: Example of a hierarchical CBD (EvenNumberGen) that computes the sequence of even numbers and makes use of other two CBDs (Counter and Double).

For this task, look for the `TO IMPLEMENT` comments and implement the following functions:

1. `getDependencies` function of the `BaseBlock`. This function should return the list of blocks on which the current block depends on. It will be helpful when building the dependency graph.

2. `compute` function of several blocks. These functions should compute the output value of the block at the iteration `curIteration`. Look at the code of the `BaseBlock` class to understand how the inputs and outputs are represented.

3. `getDependencies` function of the `DelayBlock` class. This function should override the `getDependencies` function of the `BaseBlock` class because delay blocks have different dependencies depending on the current iteration.

4. `__createDepGraph` of the `CBD` class. This function should create a dependency graph for the current CBD, at the iteration `curIteration`.

5. `__isLinear` of the `CBD` class. This function should detect whether a strong component - representing an algebraic loop - is a linear algebraic loop or not.

6. Do not worry about implementing the `DerivatorBlock` and `IntegratorBlock` classes. For now.

1.1.2 CBD Simulator

Use the supplied unit tests to test your implementation and learn how to build your CBD models by instantiating the blocks and connecting them together. Then build your own CBD model that must include the following features:

- At least one inner CBD.
- At least one linear algebraic loop.
- At least one Delay block.

Simulate your model for some time and plot the results.

1.1.3 Discrete Time CBD Denotational Semantics

Code an algorithm that takes as input a CBD Model, and outputs a LaTeX document that contains the equations that the model represents. The CBD model is represented with the classes that are in the code provided.

Use your code to produce the equations that correspond to each of the models that you created in the previous task, and include them in the report.

1.1.4 Document

Write a small report containing the tasks that you have completed and how you have completed them. Include all the plots that you have made from the simulations. The CBD model that you have built in the previous tasks should be displayed graphically in the report. You can use the `draw` function, defined in the `CBDDraw.py` file to export your model as a DOT (Graph Description Language)[1] model. You can then use the generated DOT file to draw the graph with the GraphViz tool or online: `http://graphs.grevian.org/`.

References

[1] Cláudio Gomes, Joachim Denil, and Hans Vangheluwe. Causal-Block Diagrams. Technical report, 2016.

[1] `https://en.wikipedia.org/wiki/DOT_(graph_description_language)`