Causal Block Diagram assignment (part 1)

Fall Term 2004

General Information

- The due date is **Monday 27 September 2004**, before 23:55.
- Submissions must be done via WebCT. Beware that WebCT's clock may differ slightly from yours. As described on the Assignments page, *all* results must be uploaded to WebCT and accessible from links in the index.html file. There is no need to upload AToM^3.
- The assignment must be made in teams of *two* people. It is understood that all partners will understand the complete assignment (and will be able to answer questions about it).
- Grading will be done based on correctness and completeness of the solution. Do not forget to document your requirements, assumptions, design, implementation *and* modelling and simulation results in detail !
- Extensions, if given, will involve extending not only the alotted time, but also the assignment.

The assignment

In this first part of the Causal Block Diagram assignment, you will describe and explain your algorithms in the form of an HTML document which you will upload to WebCT. The second part of the assignment will then implement the algorithms in meta-modelling environment AToM^3.

The algorithms to describe:

1. The main simulation loop. As a starting point, the following hints. In a discrete-time simulator, the main simulation loop looks like (ignoring things like sorting etc. which are required in `{compute_signals`):

```
current_discrete_time = 0

do

  compute_signals at current_discrete_time
  # Note that compute_signals must behave differently at
  # current_discrete_time = 0 and at current_discrete_time > 0
  # (and possibly take other conditions into consideration).

  current_discrete_time += discrete_time_increment
  # current_discrete_time_increment is ALWAYS 1 !

while (not termination)
# a time-based termination condition could be current_discrete_time > TMAX
# a state-based termination condition is based on the value
# of some signal(s) in the model.
# In our simulator, we allow the user to explicitly model
# ANY termination condition as a signal fed into the
# appropriate port of the simControl block.
```

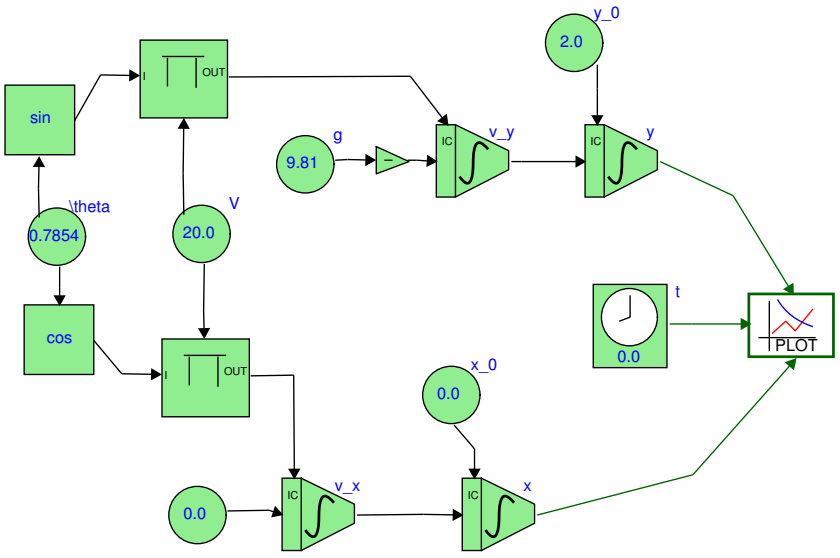

Figure 1: Causal Block Diagram of a thrown ball

Note that the discete time increment is *always* 1. It is part of the discrete-time causal block diagram formalism (its semantics is given by difference equations) and the user cannot set this !

A discretized continuous-time simulator *also* uses the fixed `discrete_time_increment = 1`. The simulator does however compute, in addition to the signals, the special signal `continuous_time` (`current_continuous_time = current_discrete_time*continuous_time_increment` or, if you wish to implement this more efficiently, increment `current_continuous_time` by `continuous_time_increment` in the simulation loop). As we're using a discrete-time simulator, all signals, including the `continuous_time` signal are discretized. The `continuous_time_increment` is user-settable (and will have to be chosen small enough for numerical stability/accuracy if `Integrator` and/or `Derivative` blocks are used). Looking from the continuous point of view, the simulator will give us an approximation of the signals, at distinct points in time which are all `continuous_time_increment` apart. This "slicing of continuous time" has led to the name *time slicing simulator*.

All the computations happening inside the do/while loop form one simulation "step". A single step computes values of signals at the `current_continuous_time` (based on other signals at that same time and some at past time points) and then increments the time. The `continuous_time_increment` is sometimes called the "step-size".

2. The construction of the dependency graph (in different phases of the simulation). You may assume that a routine is given which will subsequently find algebraic loops (if any) and will topologically sort the graph.

3. Given an algebraic loop, detection of whether it is linear.

4. Given a linear algebraic loop, construct input for a linear solver.

5. For all models (not only for ODE models), an algorithm to print out the LaTeXform of a model (the denotational semantics). Both the "naieve" and the "sophisticated" format are possible. In the "naieve" format, for each block in the model, the relationship between all connected signals is given. Note that this will only give a meaningful result if *all* signals have a non-empty-string name. In the "sophisticated" format (as a human modeller would write it), some signals may have an empty name as they will not be written explicitly. In this format, all constant equations are written separately and there is one equation for each `Integrator`, `Derivative`, or `Delay` block. For example, the Causal Block Diagram of a thrown ball depicted in Figure 1 could be represented in LaTeX as

```
\documentclass[12pt]{article}
\usepackage{pslatex}
\begin{document}
```

```
\[
  \left\{
    \begin{array}{ll}
      \dot{v_x} = 0.0, & v_x(t_0) = v_{x,0}\\
      \dot{v_y} = -(g) , & v_y(t_0) = v_{y,0}\\
      \dot{x} = v_x, & x(t_0) = x_0\\
      \dot{y} = v_y, & y(t_0) = y_0\\
      \\
      v_{x,0} = cos(\theta)\cdot V\\
      \theta = 0.7854\\
      V = 20.0\\
      g = 9.81\\
      v_{y,0} = V\cdot sin(\theta)\\
      x_0 = 0.0\\
      y_0 = 2.0\\
    \end{array}
  \right.
\]
\end{document}
```

The rendered version of this is

$$\begin{cases} \dot{v_x} = 0.0, & v_x(t_0) = v_{x,0} \\ \dot{v_y} = -(g), & v_y(t_0) = v_{y,0} \\ \dot{x} = v_x, & x(t_0) = x_0 \\ \dot{y} = v_y, & y(t_0) = y_0 \\ \\ v_{x,0} = cos(\theta) \cdot V \\ \theta = 0.7854 \\ V = 20.0 \\ g = 9.81 \\ v_{y,0} = V \cdot sin(\theta) \\ x_0 = 0.0 \\ y_0 = 2.0 \end{cases}$$

6. For each of the blocks, write the computation they need to perform (in terms of their attributes and connected signals) when called upon by the main simulation algorithm. This computation may depend on the phase in the simulation.

Useful information

1. The block attributes are `block_name`, `block_out_value`, `block_out_port`, `block_IC_port`, `block_type`, `block_operator`, `block_in_port`, `block_TRUE_in_port`, `block_FALSE_in_port`, and `block_filename`.

2. The block types are: `Generic`, `Negator`, `Inverter`, `Adder`, `Product`, `Delay`, `Integrator`, `Derivative`, `Constant`, `Test`, and `FileIO`. If a `FileIO` block is present in your model, you should output a warning and ignore it. A block's type is found in the `block_type` attribute.

3. The `SimControl` block has the following attributes: `simControl_discreteTime`, `simControl_terminateMonitor`, `simControl_discreteTime_port`, `simControl_name`, `simControl_continuousTime_port`, `simControl_timeIncrer` and `simControl_timeIncrement_port`.