Exercise 2.1: Consider a modified version of the `Delay` component, called `OddDelay`, that has a Boolean input variable *in*, a Boolean output variable *out*, and two Boolean state variables *x* and *y*. Both the state variables are initialized to 0, and the reaction description is given by:

```
if $y$ then $out$ := $x$ else $out$ := 0;
$x$ := $in$;
$y$ := $\neg y$.
```

Describe in words the behavior of the component `OddDelay`. List a possible execution of the component if it is supplied with the sequence of inputs 0, 1, 1, 0, 1, 1 for the first six rounds. ■

Exercise 2.2: Describe the component `OddDelay` from Exercise 2.1 as an extended-state machine with two modes. The mode of the state machine should capture the value of the state variable *y*, while the state variable *x* should be updated using assignments in the mode-switches. ■

Exercise 2.6: Design an event-triggered component `SecondToMinute` with the input event variable *second* and the output event variable *minute* such that *minute* is present every 60^{th} time the event *second* is present. ■

Exercise 2.7: Design a component `ClockedDelay` with a Boolean input variable *x*, an input event variable *clock*, and an output variable *y* of type event(bool) with the following behavior: if *clock* is present during rounds, say, $n_1 < n_2 < n_3 < \cdots$ then in round n_1, the output should be some default value, say 0; in round n_{j+1}, for each *j*, the output should equal the value of *x* in round n_j; and in the remaining rounds (that is, rounds during which the input event *clock* is absent), output should be absent. ■

Exercise 2.9: For the nondeterministic component `Arbiter` of figure 2.7, the reactions are expressed using the extended-state machine notation. Write an equivalent description using straight-line update code. You can use a local variable whose value is assigned nondeterministically using the choose construct. ■

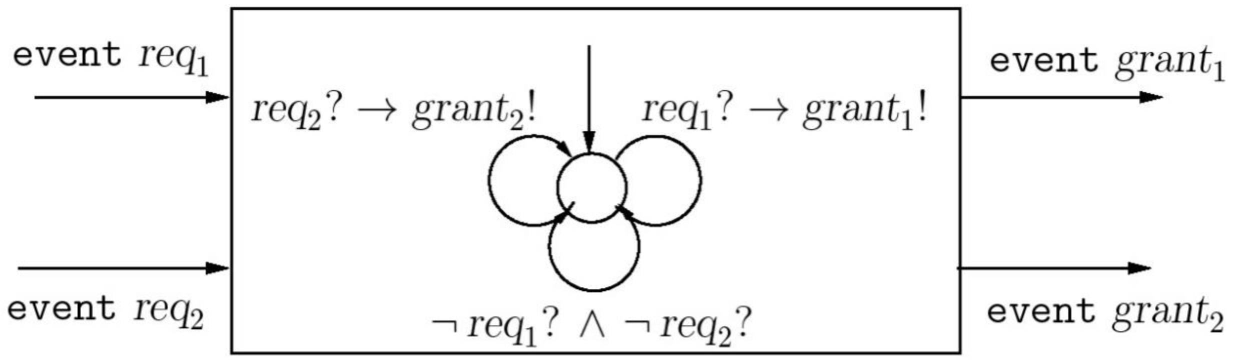

Figure 2.7: Nondeterministic Component `Arbiter`

Exercise 2.10 : Design a nondeterministic component `CounterEnv` that supplies inputs to the counter of figure 2.9. The component `CounterEnv` has no inputs, and its outputs are the Boolean variables *inc* and *dec*. It should produce all possible combinations of outputs as long as the component Counter is willing to accept these as inputs: it should never set both *inc* and *dec* to 1 simultaneously, and it should ensure that the number of rounds with *dec* set to 1 never exceeds the number of rounds with *inc* set to 1. ■

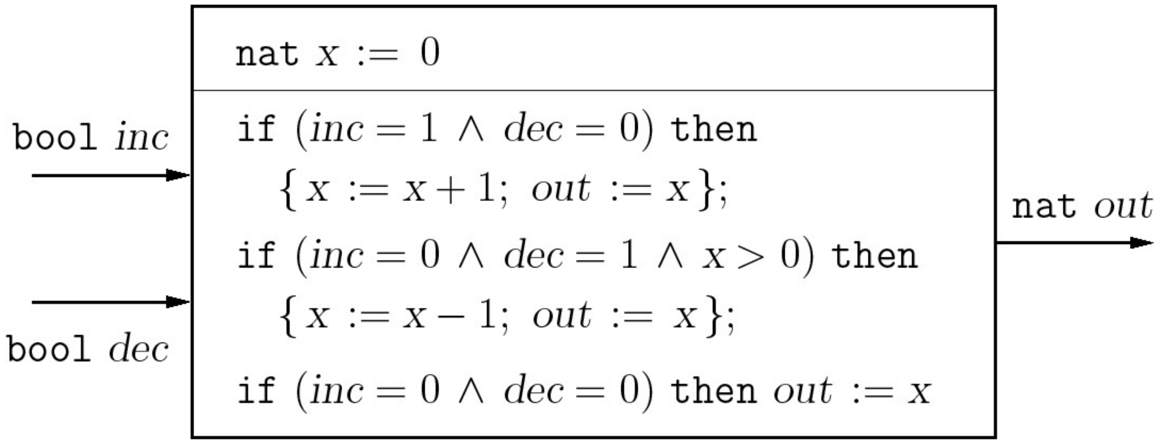
bool inc
bool dec
nat $x := 0$
if $(inc = 1 \wedge dec = 0)$ then
$\{x := x + 1;\ out := x\}$;
if $(inc = 0 \wedge dec = 1 \wedge x > 0)$ then
$\{x := x - 1;\ out := x\}$;
if $(inc = 0 \wedge dec = 0)$ then $out := x$
nat out

Figure 2.9: Component Counter with Input Assumptions

Exercise 2.12: Consider a synchronous reactive component *C* with an input variable *x* and output variables *y* and *z*. The component has two tasks, A_1 and A_2, such that the output *y* belongs to the write-set of the task A_1, and the output *z* belongs to the write-set of the task A_2. If we know that the output *y* awaits the input *x*, but the output *z* does not await *x*, then what can we conclude regarding the precedence constraints between the tasks A_1 and A_2? ■

Exercise 2.13: Consider the synchronous reactive component shown in figure 2.14. List all the possible reactions of the component. Does the output y await x? Does the output z await x? ■

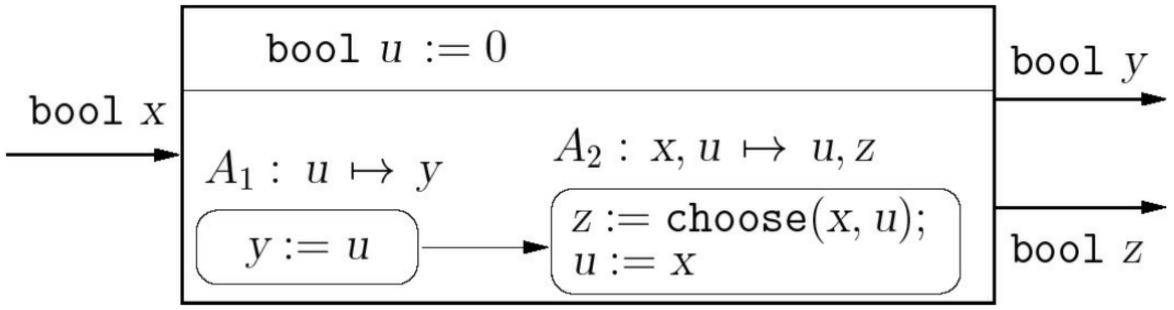

Figure 2.14: Component Specified Using Task Graph

Exercise 2.14: Design a synchronous reactive component ComputeAverage with an integer input variable x, an input event variable *clock*, and a real-valued output variable y with the following behavior: in the first round, the output y is 0; in a subsequent round i, let $j < i$ be the most recent round before round i in which the input event *clock* is present (if *clock* is absent in all rounds before i, then let $j = 0$), the output should be the *average* of input values for x in rounds $j, j+1$, upto $i-1$. The following is a sample behavior of the desired component:

Clock	$\perp$	$\perp$	$\top$	$\perp$	$\perp$	$\perp$	$\top$	$\perp$
x	5	2	−3	1	6	5	−2	11
y	0	5	3.5	−3	−1	1.33	2.25	−2

The component should be designed so that the output y does not await any of the input variables. ■

Exercise 2.16: Consider the component `DoubleSplitDelay` defined as

$$(\texttt{SplitDelay}[\,out \mapsto temp\,] \parallel \texttt{SplitDelay}[\,in \mapsto temp\,]) \setminus temp$$

This component is similar to the component `DoubleDelay` except we use instances of the component `SplitDelay` instead of `Delay`. Show the "compiled" version of `DoubleSplitDelay`, that is, list its state, input, output, and local variables, tasks, and precedence constraints. What are the await dependencies among output and input variables for `DoubleSplitDelay`? ■

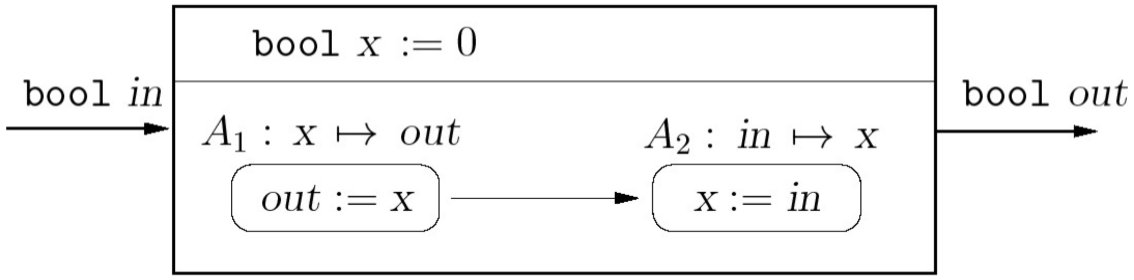

Figure 2.11: Component `SplitDelay` with `Split` Reaction

General rule for adding cross-component precedence constraint:

If a task A belonging to one component reads a var y, which is an output var of the other component, then add a precedence edge from the unique task that writes y to the task A.