CS312 Midterm: An Alarm Clock as a DFA

Phillip Goldfarb

October 27 2014

Abstract

A finite automata representing an alarm clock's states and behaviors may be constructed, which may then be converted into a regular expression.

1 Introduction

In conventional finite automata, a state transition only occurs on something considered to be input, like pressing a button or consuming a character. However, a clock is expected to change state every second; a clock would not be very useful if its owner were required to press a button every second to advance its record of time. For the purposes of this construction, an input t will be considered to be an input accepted by the clock every second. This would be most analagous to a real-world real time clock module, capable of accurately sending a pulse every second.

The presence of an input that is accepted every second in addition to input from the user (such as pressing a button on the device) adds potential complexity. The device must be capable of keeping track of time while still providing user interactions. As a simple example, it would be considered unacceptable for a clock to become thirty seconds slow if its 'set alarm' button were held for thirty seconds. A convenient way to think about an implementation that solves this issue is to consider a form of automata that is capable of having multiple 'threads' of state transitions; one thread would be responsible for recording (and displaying) the current time, and the other would be responsible for accepting user input. If we assume that the first thread only executes read operations on memory, and the second only executes write operations, the system will be free of race conditions. Therefore, two finite automata can be constructed to represent the two subsystems of the alarm clock. Following ths construction, it will be shown that the cartesian product of the two automata can be combined to form a single DFA that represents the alarm clock as a whole.

In order for this hypothetical clock to be useful, it will have a number of additional inputs beyond the per-second tick. buttons or switches will be used for the following operations: 'Set Alarm', 'Activate Alarm', 'Increment Hour', 'Increment Minute'. To make the distinction between a button that is meant to be held momentarily, a switch that stays activated until it is switched off, and a button that is held while others are pressed arbitrary, the following notation will be used to denote that an input is 'pushed and held' versus 'released':

X	X pressed momentarily
X^p	X pressed and held
X^r	X released

Table 1: Configurations of input X

Thus, it is required that any X^p input must eventually be followed by a X^r input. Following is a table of every input accepted by the hypothetical device:

t	Automatic per-second tick
A	Alarm on/off toggle
S	Set alarm time
M	Increment minute
H	Increment hour
Z	Snooze

Table 2: Inputs present on the clock

In order for the clock to be able to show the current time and to have an alarm set to some particular time, it must have some kind of memory. For the purposes of this construction, the following variables will be considered as the areas of memory that can be read from and written to:

$\mathbf{T_h}$	Current time hour 0-23
$\mathbf{T_m}$	Current time minute 0-59
$\mathbf{T_s}$	Current time second 0-59
$\mathbf{A_h}$	Set alarm hour 0-23
$\mathbf{A_m}$	Set alarm minute 0-59
$\mathbf{S_h}$	Snooze hour 0-23
$\mathbf{S_m}$	Snooze minute 0-59

Table 3: Internal memory of the clock

Each area of memory is initialized to 0.

Some functionality will need to be represented as subroutines that are executed without implied state changes, such as displaying a particular time, and changing the values of the areas of memory. It's important to note that each of these operations can be represented as its own finite state machine, but due to the number of states, and the simplicity of each operation.

SET(**var**, *val*)	Update the value of **var** to *val*
TICK()	Increase $\mathbf{T_h}$, $\mathbf{T_m}$, and $\mathbf{T_s}$ by one second, including carry operations
DISP(**H**,**M**)	Show the time $mod(\mathbf{H}, 12)$:**M** on the clock display
BLINK()	Causes the display to blink, indicating that the time is not set
BEEP()	Makes the audible alarm noise

Table 4: Subroutines of the clock

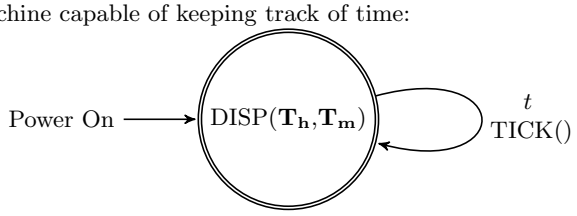
Every memory operation is taken to be atomic, to remove the possibility of a race condition. In addition, for simplicity's sake, DISP() is assumed to show AM/PM:

$$period(\mathbf{H}) = \begin{cases} AM & : \mathbf{H} < 12 \\ PM & : \mathbf{H} \geq 12 \end{cases}$$

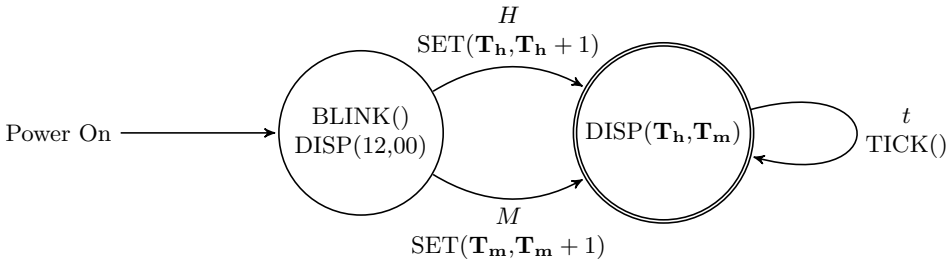
$mod()$ is the normal modulus operation, except that $mod(\mathbf{H}, 12) = 0$ is displayed as 12.

2 Basic Clock

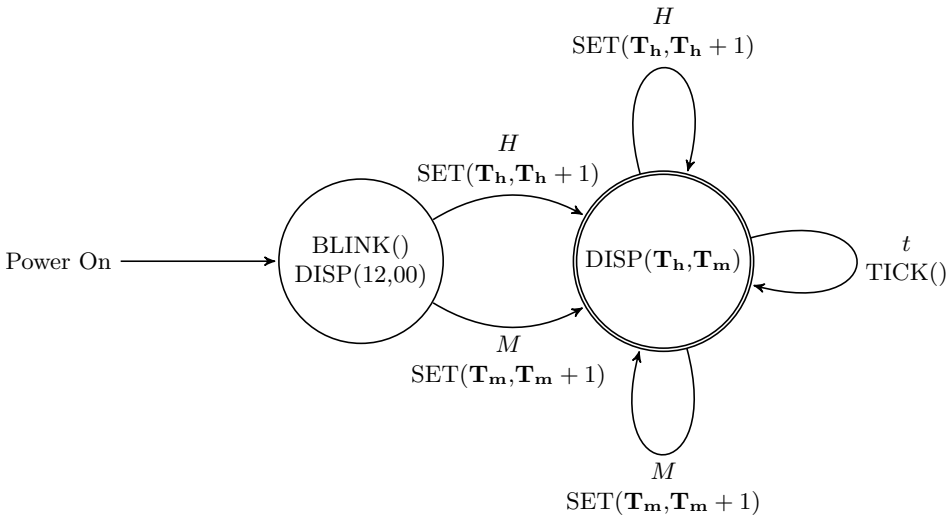
Now that the basic mechanisms are in place for manipulating what information the clock displays, the pieces can be put together to make a machine capable of keeping track of time:



This automata will display the time, but its initialized time is midnight, only making it useful for showing the amount of time that has elapsed. Note that in this automata, a state being accepting indicates that it is considered the machine's idle state, in which it is functioning and waiting for input. This automata can be represented by the regular expression $t^\star$, which makes sense as a machine that does nothing but keep track of how many seconds have elapsed. We can add on to this to make a clock that can requires user input before displaying the time:



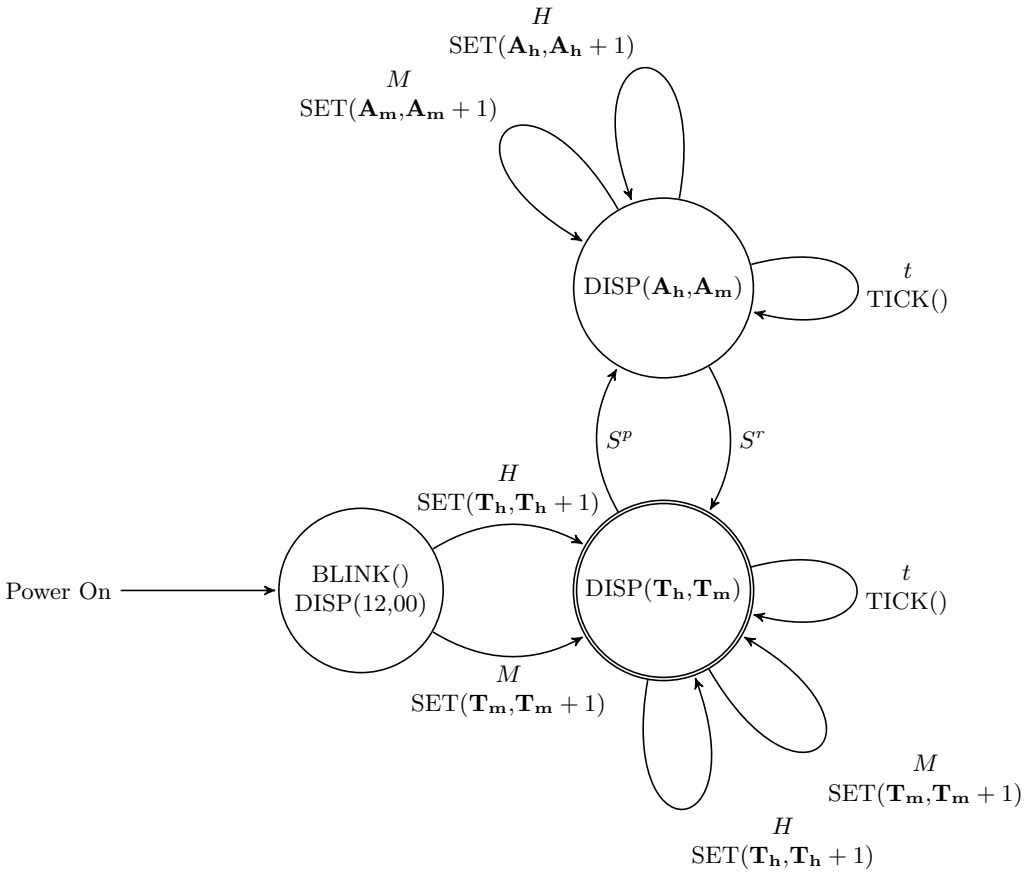
This is better, but it still only allows the time to be initialized to either 1:00 AM or 12:01 AM. To make it more useful, the hour and minute need to be incrementable from the idle state:



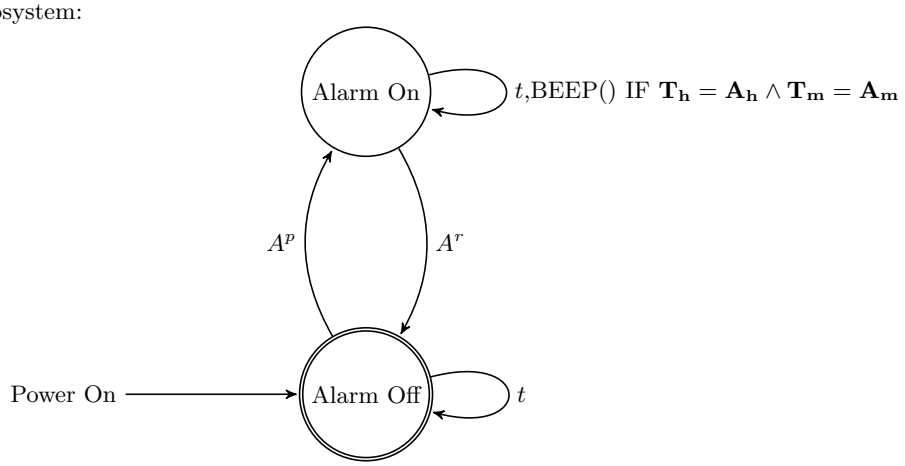
At this point, the finite automata can be represented by the regular expression $(H|M)^\star(H|M|t)^\star$. This makes sense, since the machine waits to be initialized, and then keeps track of the elapsed time, in addition to the time added by the user, providing an accurate representation of the current time of day.

3 Alarm Functionality

To begin adding functionality to the clock to make it useful as an alarm clock, there needs to be some way for the user to set the alarm. To do so, the automata must have a new main state; if the idle state is considered to be the state in which pressing H or M set the current time, then the new state must be the one in which presing H or M set the alarm time. In addition, the new state must display the alarm time, since the user must know what the new alarm time is. However, as previously noted, t pulses must still be accepted, so that the clock continues to have the correct time.



At this point, the automata represents an alarm clock that allows the time to be set, and for a second alarm time to be set. This would be the regular expression $(H|M|^\star(H|M|t|(S^p(H|M|t)^\star S^r)^\star$. However, this still would not be a useful alarm clock, since there is no way for the clock to actually make a sound when the current time matches the alarm time. It makes sense to now make a second automata that represents the alarm subsystem:

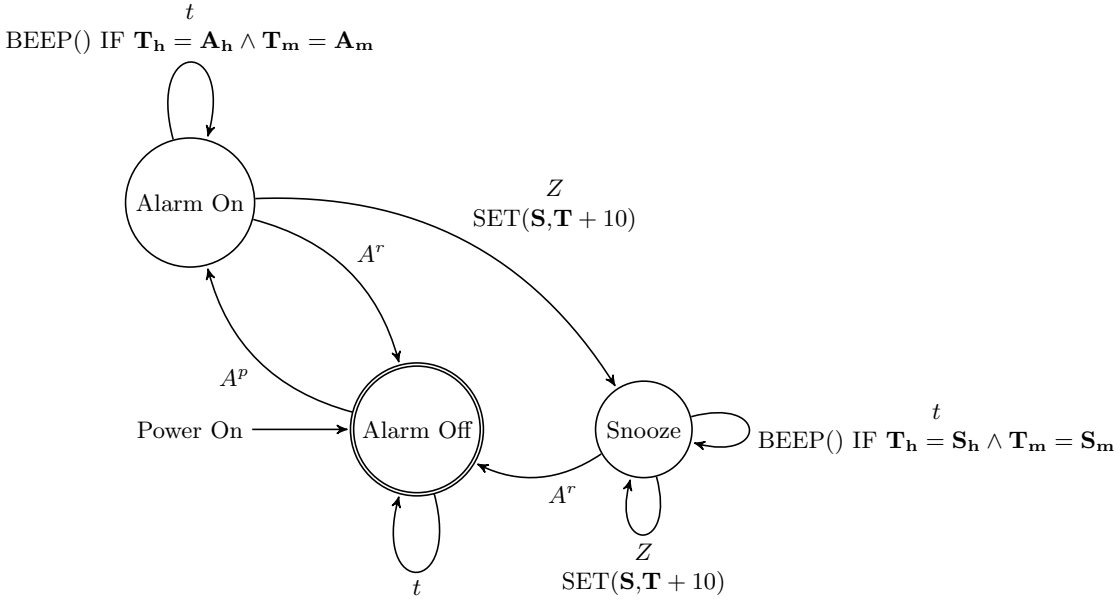


This subsystem is lazier than I expected, but it turns out that it works pretty well; if the alarm is off, then it silently idles. If the alarm is

turned on, then it idles until the hour and minute of the current time match the hour and minute of the set alarm time. If those conditions are met (which may occur for a maximum of 60 seconds, conveniently), then the clock beeps once per second. It is also important to notice that only the second automata representing the alarm subsystem calls the BEEP() subroutine, and only the first automata calls SET() and TICK(). The regular expression representing the second automata is fairly simple: $((A^p t^\star A^r)|t)^\star$. In imperative terms, this means that the alarm subsystem will go through cycles in which either (A) its alarm is toggled on, the alarm eventually goes off, and the alarm is toggled off, or (B), it simply sits idly.

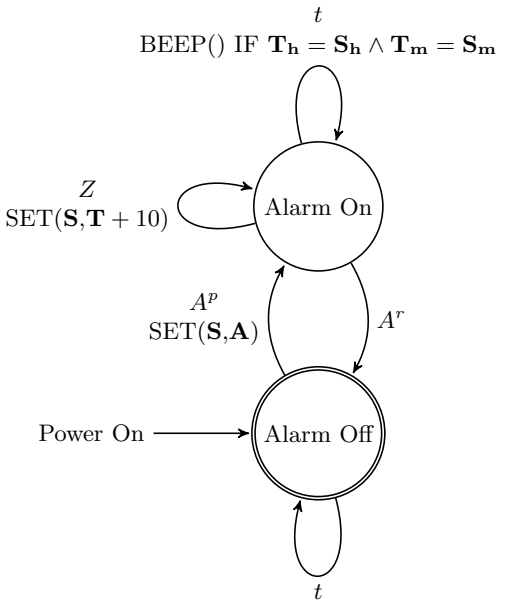
4 Adding a Snooze Function

In order to cater to most clock-purchasing clientele, the alarm clock should have a snooze button. The snooze function is generally expected to work as follows: if the alarm goes off, activating snooze mode should stop the alarm, and resume after a set period of time, such as ten minutes. This can be represented by adding an additional alarm state:



In this automata, SET(**S**,**T** + 10) is a lazy way of saying that pressing the Snooze button adds ten minutes to the alarm time; the SET() notation implies logic for carrying minutes into the hours place. This automata acts the same as the previous alarm subsystem, but in this case, a new alarm is set up when the snooze button is pressed, which then acts the same as the original alarm, without modifying the user-set alarm time. This subsystem is represented by the regular expression $(((A^p(t|(Z|t)^\star)^\star)^\star A^r)|t)^\star$, which can be reduced to the expression $((A^p(t|Z)^\star A^r)|t)^\star$, which essentially shows that when the alarm goes off, either a sequence of seconds or seconds with the user pressing the snooze button will eventually lead to the alarm sounding again, until the alarm is turned off. The structure suggests that the subsystem could be reduced to use fewer states; this could be accomplished by replacing **A** comparison in the Alarm On state

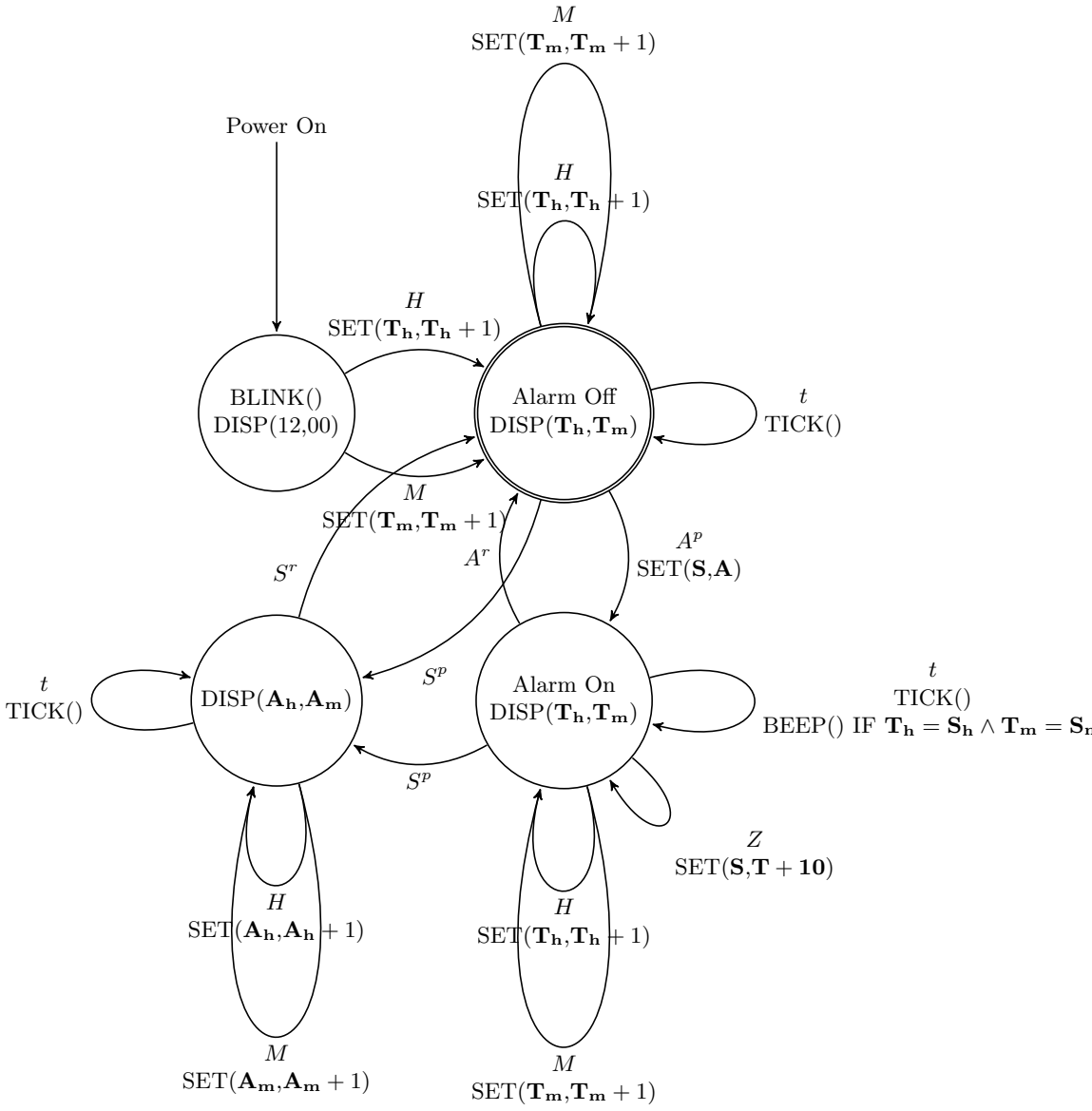
with **S**, and then copying the value of **A** into **S** on the A^p state transition:



This automata is represented by the simpler regular expression $(A^p(t|Z)^\star A^r|t)^\star$.

5 Combined Automata

Now that two subsystems have been constructed that will provide the expected functionality of an alarm clock, it is possible to combine the two automata into one, by effectively taking the cartesian product. That is, for each of the two main states in the displaying/setting automata, there must also be states for the alarm being on, and the alarm being off. This could be simplified by just supposing that the alarm is deactivated when the alarm is being set:



This monstrosity repesents the entire alarm clock, including the alarm subsystem. With all of that work, there is still a flaw that would be considered unacceptable: when the alarm is on, pressing the snooze button will cause the alarm to go off in ten minutes, even if it's hours before the alarm is actually supposed to go off. The regular expression for this automata would be: $(H|M)(H|M|t|S^p(H|M|t)^\star S^r|(A^p((Z|t)^\star A^r)|S^p(H|M|t)^\star S^r))^\star$.