Thomas Nemeh

k1653003

**Formal Verification Project**

**Stage 1:**

At first, I designed my modulo 8 counter in the most obvious way possible. Using 3 boolean variables(x, y, and z; x being the most significant digit and z being the least significant digit), I defined a rule for each of the 8 state-transitions(the states being 000, 001, 010, 011, 100, 101, 110, and 111, respectively):

NAME Modulo8

VAR

x : boolean;

y : boolean;

z : boolean;

INIT

!x & !y & !z

Rules

!x & !y & !z : !x & !y & z

!x & !y & z : !x & y & !z

!x & y & !z : !x & y & z

!x & y & z : x & !y & z

x & !y & z : x & y & !z

x & y & !z : x & y & z

However, by taking advantage of patterns governing the behavior of the three variables, I developed a simpler model with only 4 rules. This is the final version of my model:

NAME Modulo8V2

VAR

x : boolean;

y : boolean;

z : boolean;

INIT

!x & !y & !z;

RULES

!z :

z := true

z & !y:

z := false; y := true

z & y & x:

z := false; y := false; x := false

z & y & !x:

z := false; y := false; x := true

(See stage 7 for representation of Kripke structure)

**Stage 2:**

To test my model, I wrote 8 CTL properties to test each of the counter’s transitions. My model satisfies all of these properties.

1. AG ((!x & !y & !z) -> AX(!x & !y & z))

2. AG ((!x & !y & z) -> AX(!x & y & !z))

3. AG ((!x & y & !z) -> AX(!x & y & z))

4. AG ((!x & y & z) -> AX(x & !y & !z))

5. AG ((x & !y & !z) -> AX(x & !y & z))

6. AG ((x & !y & z) -> AX(x & y & !z))

7. AG ((x & y & !z) -> AX(x & y & z))

8. AG ((x & y & z) ->AX(!x & !y & !z))

To demonstrate why my properties represent the intuitive specification of what we can expect from the counter, I will analyze one of my properties in detail. This will be sufficient to explain all 8 of my properties, since each one is based on the same CTL framework and differs only in the values plugged in. Thus, take the following property:

AG ((!x & !y & !z) -> AX(!x & !y & z))

This formula can be translated to English thusly: “For every reachable state defined as !x, !y, and !z(corresponding to 0 in decimal), for all paths starting at that state the next state must be defined as !x, !y, and z(corresponding to 1). This property, then, proves that 1 will always follow 0. My next property similarly shows that 2 will always follow 1, next that 3 will always follow 2, etc. Altogether, my properties prove that from each state the next state will always be incremented by 1, except in the case where the counter resets from 7 to 1. This ensures the correctness of my model at each possible transition.

All my properties are safety properties because they define what behavior should always happen, i.e. that nothing “bad” will happen. Moreover, a counterexample to any of my properties would be a sequence finite path showing a violation of the property.

**Stage 3:**

None of my properties pass vacuously in the system. However, for the sake of demonstrating the concept of vacuity, suppose I had written my first property like this:

AG (!x & !y & !z) -> AX(!x & !y & z)

My intention is to check that all occurrences of the state defined as !x, !y, and !z will be followed by a state defined as !x, !y, and z. However, the property written in the above matter instead says the following: “If all reachable states are defined as !x, !y, and !z, then for all paths starting at the initial state, the next state must be defined as !x, !y, and z”. Because the antecedent is false, the result of the implication will be vacuously true, telling us nothing about the behavior of the model. Therefore, the property should be written as I had it originally.

**Stage 4:**

To introduce a bug not caught by any of my properties, I changed the initial state from 0 to 7:

NAME Modulo8Bug

VAR

x : boolean;

y : boolean;

z : boolean;

INIT

x & y & z;

RULES

!z :

z := true

z & !y:

z := false; y := true

z & y & x:

z := false; y := false; x := false

z & y & !x:

z := false; y := false; x := true

(See stage 7 for representation as Kripke structure)

**Stage 5:**

The issue is that my properties simple define, given a possible state, what the next state should be, but there is no property that checks that the model starts in the right position. Therefore, I will define one addition property that ensures the initial state is correct:

9. !x & !y & !z

This is also a safety property because it checks for a behavior that should always happen, that the first state of the counter is defined as !x, !y, and !z. A counterexample, then, would simply be the model producing an initial state that does not assign each variable as false: if we run the model for one state and we get, for example,

x & !y & !z

then the ninth property will fail and we will know that the model is incorrect.

**Stage 6:**

My initial model satisfies the new property because it defines the initial state as !x & !y & !z.

**Stage 7:**

I define the Kripke structure for my model based on the following formula:

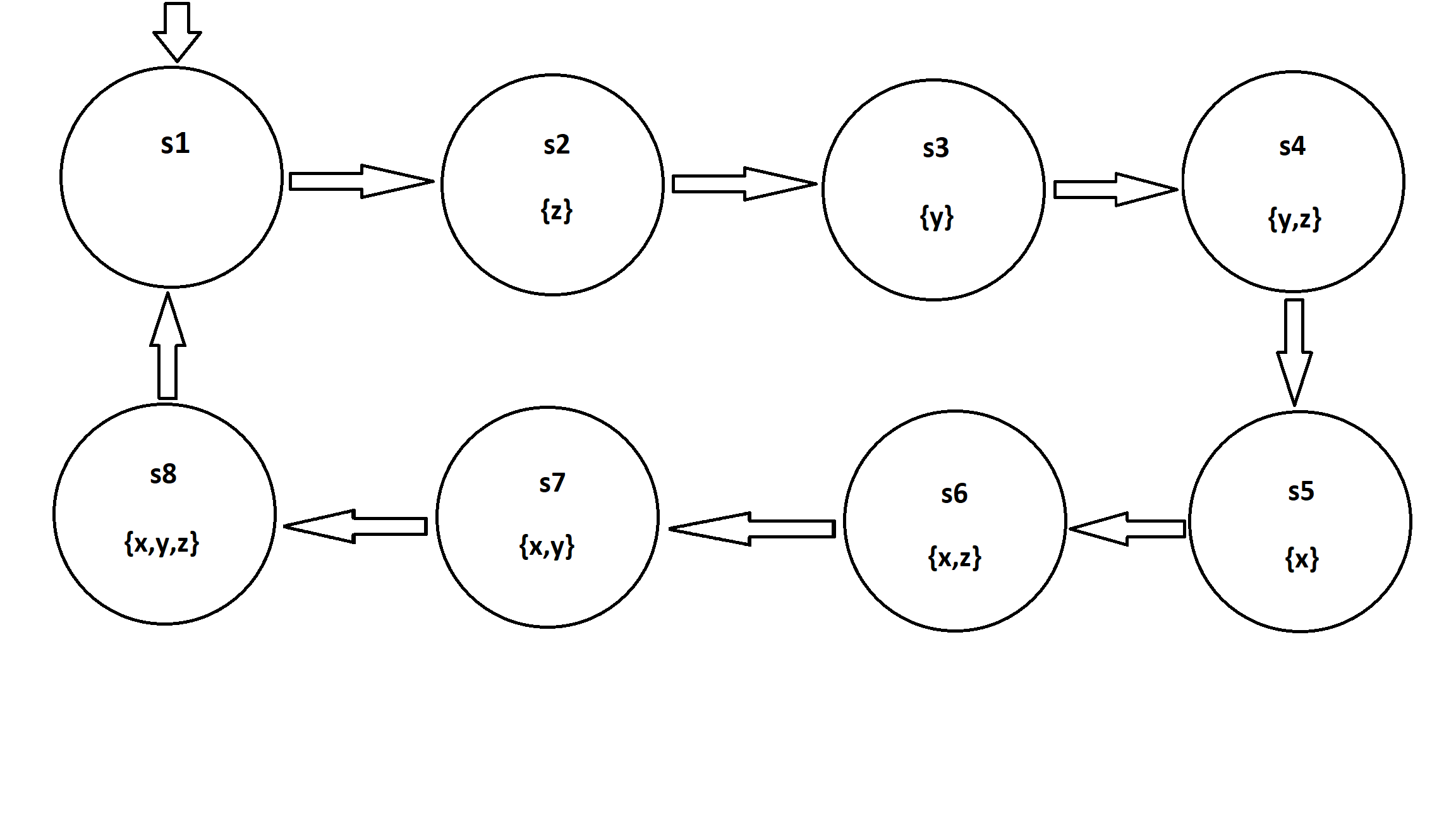
M = (S, S0, R, L)

S – set of all states of the system n

S0 – set of initial states n

R – transition relation between states n

L – a function that associates each state with set of propositions true in that state (labeling)



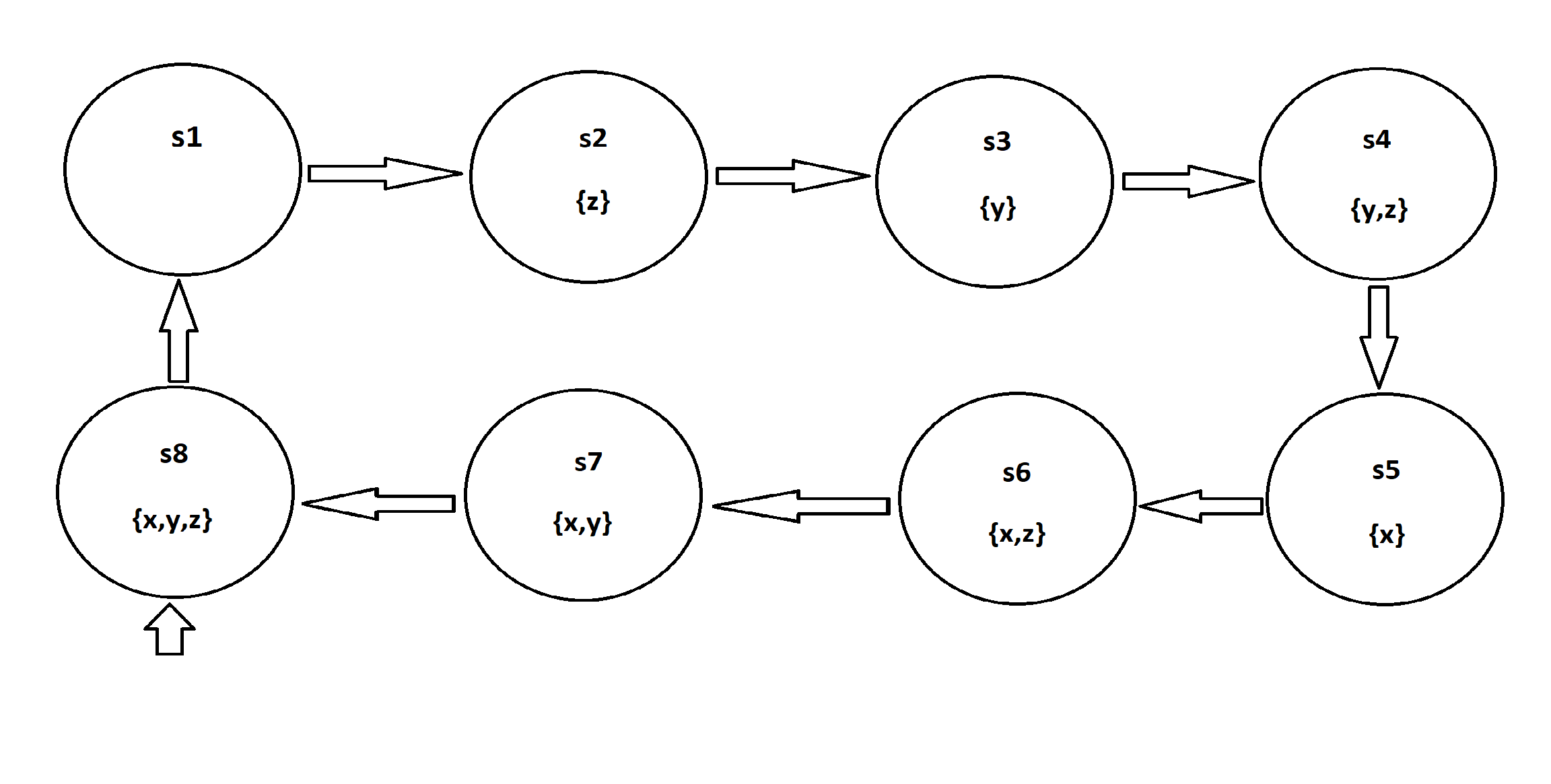
*S* = {s1,s2,s3,s4,s5,s6,s7}

*I* = {s1}.

*R* = {(s1,s2),(s2,s3)(s3,s4),(s4,s5),(s5,s6)(s6,s7),(s7,s8), (s8,s1)}.

*L* = {(s1, {}), (s2, {z}), (s3, {y}),(s4,{y,z}),(s5,{x})(s6,{x,z}),(s7,{x,y}),(s8,{x,y,z})}

My buggy model:

****

*S* = {s1,s2,s3,s4,s5,s6,s7}

*I* = {s8}.

*R* = {(s1,s2),(s2,s3)(s3,s4),(s4,s5),(s5,s6)(s6,s7),(s7,s8),(s8,s1)}.

*L* = {(s1, {}), (s2, {z}), (s3, {y}),(s4,{y,z}),(s5,{x})(s6,{x,z}),(s7,{x,y}),(s8,{x,y,z})}