Project 4

CPSC 439-01 Spring 2023

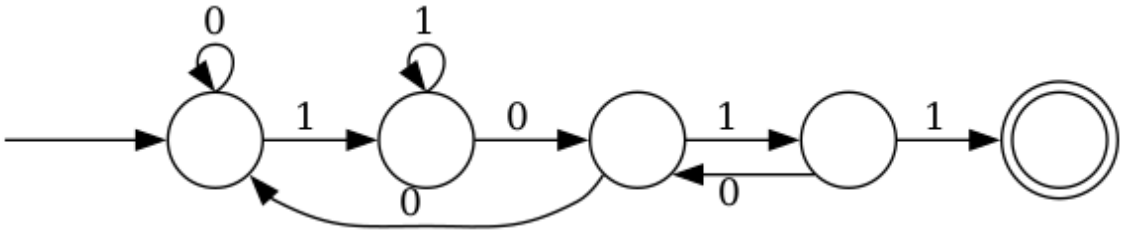
Michael Clausen
Diamond Dinh
Luke Eltiste
Evan Fricker
Anthony Lam
Aaron Lieberman

1. The language we chose was

$$L = \{w : w \text{ contains the substring } 1011\}$$

from exercise 2.1.4 in *Introduction to Theory of Computation.*

DFA:



2. The SAT input:

```
# At least one state in each step
Q0_0 Q1_0 Q2_0 Q3_0 Q4_0
Q0_1 Q1_1 Q2_1 Q3_1 Q4_1
Q0_2 Q1_2 Q2_2 Q3_2 Q4_2
Q0_3 Q1_3 Q2_3 Q3_3 Q4_3
Q0_4 Q1_4 Q2_4 Q3_4 Q4_4
Q0_5 Q1_5 Q2_5 Q3_5 Q4_5
Q0_6 Q1_6 Q2_6 Q3_6 Q4_6

# No more than one state at step 0
~Q0_0 ~Q1_0
~Q0_0 ~Q2_0
~Q0_0 ~Q3_0
~Q0_0 ~Q4_0
~Q1_0 ~Q2_0
~Q1_0 ~Q3_0
~Q1_0 ~Q4_0
~Q2_0 ~Q3_0
~Q2_0 ~Q4_0
~Q3_0 ~Q4_0

# No more than one state at step 1
~Q0_1 ~Q1_1
~Q0_1 ~Q2_1
~Q0_1 ~Q3_1
~Q0_1 ~Q4_1
~Q1_1 ~Q2_1
~Q1_1 ~Q3_1
~Q1_1 ~Q4_1
~Q2_1 ~Q3_1
~Q2_1 ~Q4_1
~Q3_1 ~Q4_1

# No more than one state at step 2
~Q0_2 ~Q1_2
~Q0_2 ~Q2_2
~Q0_2 ~Q3_2
~Q0_2 ~Q4_2
~Q1_2 ~Q2_2
~Q1_2 ~Q3_2
~Q1_2 ~Q4_2
~Q2_2 ~Q3_2
~Q2_2 ~Q4_2
~Q3_2 ~Q4_2

# No more than one state at step 3
~Q0_3 ~Q1_3
~Q0_3 ~Q2_3
~Q0_3 ~Q3_3
~Q0_3 ~Q4_3
~Q1_3 ~Q2_3
~Q1_3 ~Q3_3
~Q1_3 ~Q4_3
~Q2_3 ~Q3_3
~Q2_3 ~Q4_3
~Q3_3 ~Q4_3
```

```
# No more than one state at step 4
~Q0_4 ~Q1_4
~Q0_4 ~Q2_4
~Q0_4 ~Q3_4
~Q0_4 ~Q4_4
~Q1_4 ~Q2_4
~Q1_4 ~Q3_4
~Q1_4 ~Q4_4
~Q2_4 ~Q3_4
~Q2_4 ~Q4_4
~Q3_4 ~Q4_4

# No more than one state at step 5
~Q0_5 ~Q1_5
~Q0_5 ~Q2_5
~Q0_5 ~Q3_5
~Q0_5 ~Q4_5
~Q1_5 ~Q2_5
~Q1_5 ~Q3_5
~Q1_5 ~Q4_5
~Q2_5 ~Q3_5
~Q2_5 ~Q4_5
~Q3_5 ~Q4_5

# No more than one state at step 6
~Q0_6 ~Q1_6
~Q0_6 ~Q2_6
~Q0_6 ~Q3_6
~Q0_6 ~Q4_6
~Q1_6 ~Q2_6
~Q1_6 ~Q3_6
~Q1_6 ~Q4_6
~Q2_6 ~Q3_6
~Q2_6 ~Q4_6
~Q3_6 ~Q4_6

# Transition ("", 1 -> "1")
~Q0_0 ~P0 Q1_1
~Q0_1 ~P1 Q1_2
~Q0_2 ~P2 Q1_3
~Q0_3 ~P3 Q1_4
~Q0_4 ~P4 Q1_5
~Q0_5 ~P5 Q1_6

# Transition ("", 0 -> "")
~Q0_0 P0 Q0_1
~Q0_1 P1 Q0_2
~Q0_2 P2 Q0_3
~Q0_3 P3 Q0_4
~Q0_4 P4 Q0_5
~Q0_5 P5 Q0_6

# Transition ("1", 1 -> "")
~Q1_0 ~P0 Q0_1
~Q1_1 ~P1 Q0_2
~Q1_2 ~P2 Q0_3
```

```
~Q1_3 ~P3 Q0_4
~Q1_4 ~P4 Q0_5
~Q1_5 ~P5 Q0_6

# Transition ("1", 0 -> "10")
~Q1_0 P0 Q2_1
~Q1_1 P1 Q2_2
~Q1_2 P2 Q2_3
~Q1_3 P3 Q2_4
~Q1_4 P4 Q2_5
~Q1_5 P5 Q2_6

# Transition ("10", 1 -> "101")
~Q2_0 ~P0 Q3_1
~Q2_1 ~P1 Q3_2
~Q2_2 ~P2 Q3_3
~Q2_3 ~P3 Q3_4
~Q2_4 ~P4 Q3_5
~Q2_5 ~P5 Q3_6

# Transition ("10", 0 -> "")
~Q2_0 P0 Q0_1
~Q2_1 P1 Q0_2
~Q2_2 P2 Q0_3
~Q2_3 P3 Q0_4
~Q2_4 P4 Q0_5
~Q2_5 P5 Q0_6

# Transition ("101", 1 -> "1011")
~Q3_0 ~P0 Q4_1
~Q3_1 ~P1 Q4_2
~Q3_2 ~P2 Q4_3
~Q3_3 ~P3 Q4_4
~Q3_4 ~P4 Q4_5
~Q3_5 ~P5 Q4_6

# Transition ("101", 0 -> "")
~Q3_0 P0 Q0_1
~Q3_1 P1 Q0_2
~Q3_2 P2 Q0_3
~Q3_3 P3 Q0_4
~Q3_4 P4 Q0_5
~Q3_5 P5 Q0_6

# Transition ("1011", 1 -> "")
~Q4_0 ~P0 Q0_1
~Q4_1 ~P1 Q0_2
~Q4_2 ~P2 Q0_3
~Q4_3 ~P3 Q0_4
~Q4_4 ~P4 Q0_5
~Q4_5 ~P5 Q0_6

# Transition ("1011", 0 -> "")
~Q4_0 P0 Q0_1
~Q4_1 P1 Q0_2
~Q4_2 P2 Q0_3
~Q4_3 P3 Q0_4
~Q4_4 P4 Q0_5
~Q4_5 P5 Q0_6

# Initial State
Q0_0

# Accepted States
Q4_4 Q4_5 Q4_6
```

The SAT input can also be found in this gist.

There's also a generator for converting the DFA to the SAT input at dfa2sat.py.

3. The input can be evaluated using the following command:

```
$ python sat.py -a --starting_with P --input substring.in
```

After removing duplicates:

```
~P0 ~P1  P2 ~P3  P4  P5
~P0  P1 ~P2  P3  P4 ~P5
~P0  P1 ~P2  P3  P4  P5
 P0 ~P1  P2  P3 ~P4 ~P5
 P0 ~P1  P2  P3 ~P4  P5
 P0 ~P1  P2  P3  P4 ~P5
 P0 ~P1  P2  P3  P4  P5
 P0  P1  P2 ~P3  P4  P5
```

This tells us that there are 8 unique inputs that are accepted by the regular language. In string form, they are:

```
001011
010110
010111
101100
101101
101110
101111
111011
```

4. We decided to use Z3 with python bindings as the SAT solver of choice. Similar to simple-sat, we created a CLI.

The full code can be found here

The CLI options

```
~/Code/GitHub/z3-sat-solver  main ?1
 python main.py --help
Usage: main.py [OPTIONS]

Options:
  -i, --input TEXT         The input file of the sat problem  [required]
  -p, --input_prefix TEXT  The prefix of all input states
  -a, --all                Whether all satisfiable inputs should be listed
  -s, --string             Convert all input states to a binary string
  --help                   Show this message and exit.
```

We can use

`python main.py -a -s -i inputs/substring.in`

to view all the inputs (in string form) that satisfy the regular language provided in the input

```
…\dev\school\CPSC_439_Theory_Of_Computation\z
→ python main.py -a -s -i inputs\substring.in
101100
010110
101110
010111
101111
101101
001011
111011
```

Learning Z3 took a bit of time since the documentation on the python bindings are not the best, and it has its own abstractions of python data types that are not easily converted to native python data types. Once we learned these quirks, it was relatively easy and fun to use Z3 to accomplish the goals set out in the project.

We used a python CLI library that made converting the program to a CLI really easy.