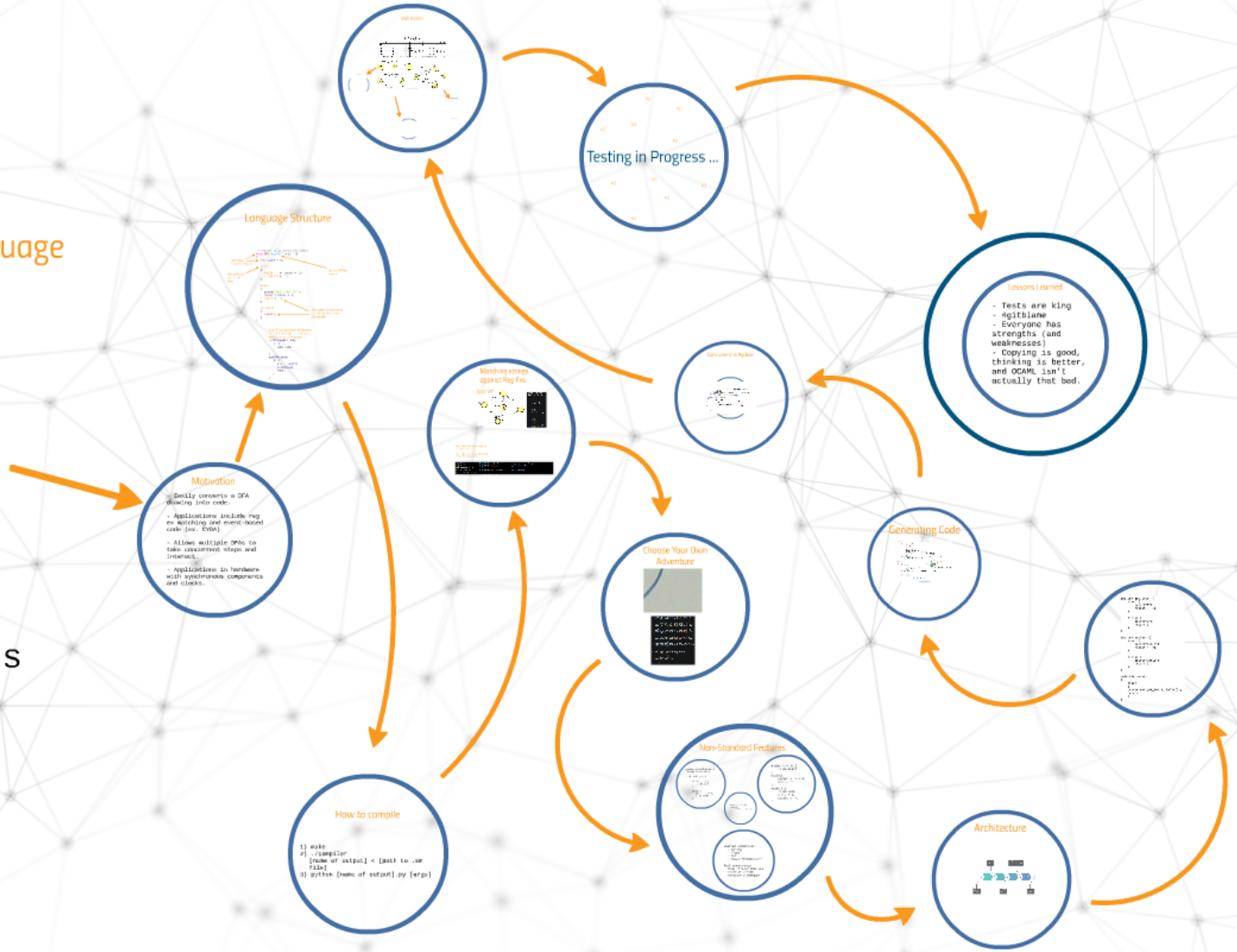
StateMap

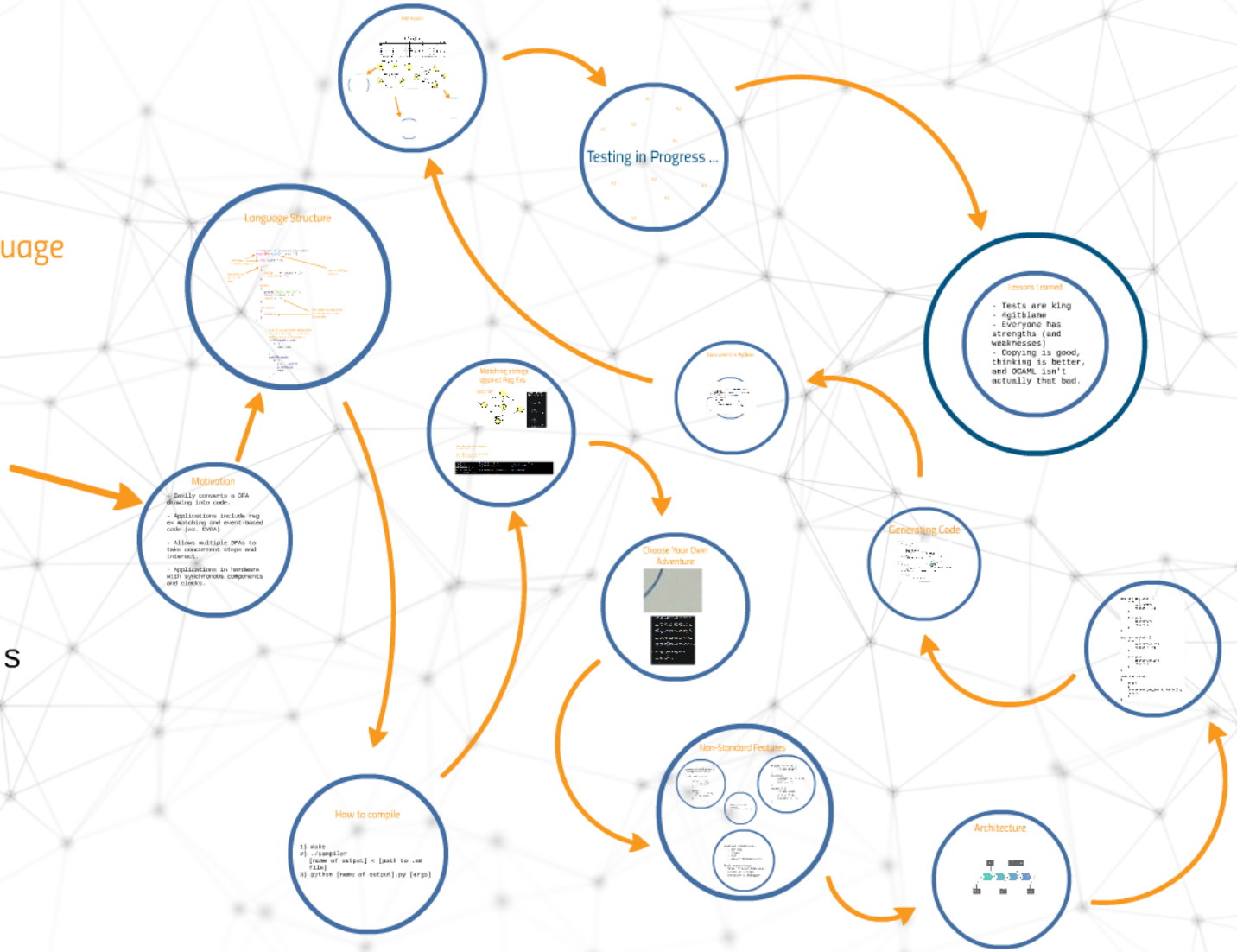
A DFA Simulation Language



Oren Finard
Jackson Foley
Alexander Peters
Brian Yamamoto
Zuokun Yu

StateMap

A DFA Simulation Language



Oren Finard
Jackson Foley
Alexander Peters
Brian Yamamoto
Zuokun Yu

Motivation

- Easily converts a DFA drawing into code.

- Applications include reg ex matching and event-based code (ex. CYOA)

- Allows multiple DFAs to take concurrent steps and interact.

- Applications in hardware with synchronous components and clocks.

Language Structure

```
// Prints hello world ten times.
void DFA main(/* args */)
{
   int count = 0;

   start
   {
      hello    <- count < 10;
      finished <- *;
   }

   hello
   {
      print("Hello World!");
      count = count + 1;
      start <- *;
   }

   finished
   {
      return;
   }
}
```

Data types (void, int, string, float)

Every DFA must have a start state

One main DFA per program

Every state must have either a return statement or an Else transition

(Sub-DFAs must be declared above the main DFA - if a DFA calls a sub-DFAs, waits for the sub-DFA to return.)

```
int DFA sum(int a, int b) {
    start {
        return (a+b);
    }
}
void DFA main() {
    start {
        int res = sum(1,2);
        print(itos(res));
        return;
    }
}
```

```
// Prints hello world ten times.
void DFA main(/* args */)
{
   int count = 0;

   start
   {
      hello    <- count < 10;
      finished <- *;
   }

   hello
   {
      print("Hello World!");
      count = count + 1;
      start <- *;
   }

   finished
   {
      return;
   }
}
```

ata types
oid, int, string, float)

DFA must
a start

One main DFA per program

Every state must have either a return statement or an Else transition

(Sub-DFAs must be declared above the main DFA - if a DFA calls a sub-DFAs, waits for the sub-DFA to return.)

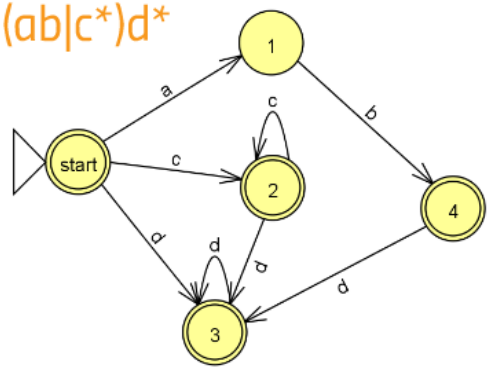
```
int DFA sum(int a, int b) {
	start {
		return (a+b);
	}
}
void DFA main() {
	start {
		int res = sum(1,2);
		print(itos(res));
		return;
	}
}
```

How to compile

```
1) make
2) ./compiler
   [name of output] < [path to .sm
   file]
3) python [name of output].py [args]
```

Matching strings against Reg Exs

(ab|c*)d*



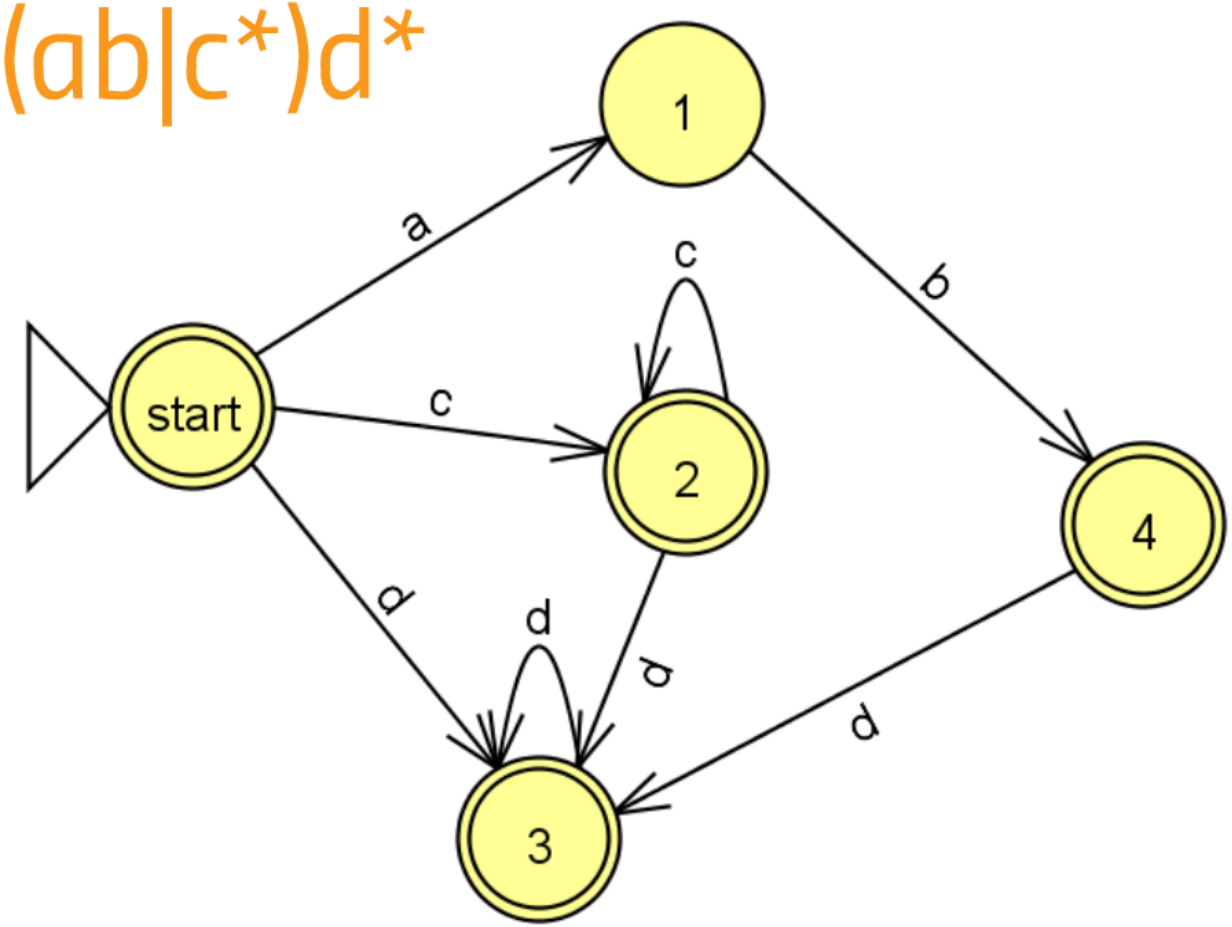
Note: "'string'" will pass the program the argument [s,t,r,i,n,g]

Note to the note: That's a double quote followed by a single quote.

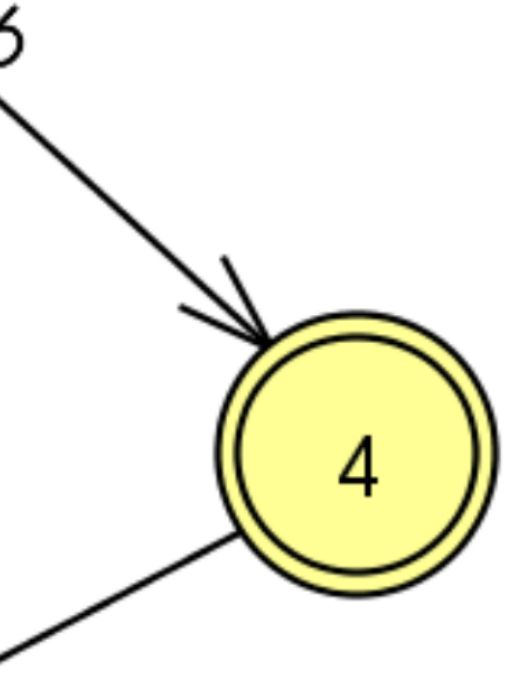
```
bky2102@cairo:~/StateMap/StateMap$ ./compiler < sample_programs/reg_ex_test.sm
bky2102@cairo:~/StateMap/StateMap$ python output.py "'aabbcc'"
Not accepted by the DFA
bky2102@cairo:~/StateMap/StateMap$ python output.py "'abddd'"
Accepted by the DFA.
```

against Reg Exs

(ab|c*)d*




```
/* A StateMap DFA that accepts the
reg ex (ab|c*)d* */
void DFA main(stack<string> args) {
    int accepted = 1; /* acceptance
        state if reach end of stack*/

    start {
        string s = args.peek();

        stateOne    <- s == "a";
        stateTwo    <- s == "c";
        stateThree  <- s == "d";
        accept      <- s == EOS;
        notAccept   <- *;
    }

    stateOne {
        accepted = 0;
        args.pop();
        string s = args.peek();

        stateFour   <- s == "b";
        notAccept   <- *;
    }

    stateTwo {
        accepted = 1;
        args.pop();
        string s = args.peek();

        stateThree  <- s == "d";
        stateTwo    <- s == "c";
        accept      <- s == EOS;
        notAccept   <- *;
```




```
/* A StateMap DFA that accepts the
reg ex (ab|c*)d* */
void DFA main(stack<string> args) {
    int accepted = 1; /* acceptance
        state if reach end of stack*/

    start {
        string s = args.peek();

        stateOne    <- s == "a";
        stateTwo    <- s == "c";
        stateThree  <- s == "d";
        accept      <- s == EOS;
        notAccept   <- *;
    }

    stateOne {
        accepted = 0;
        args.pop();
        string s = args.peek();

        stateFour   <- s == "b";
        notAccept   <- *;
    }

    stateTwo {
        accepted = 1;
        args.pop();
        string s = args.peek();

        stateThree  <- s == "d";
        stateTwo    <- s == "c";
        accept      <- s == EOS;
        notAccept   <- *;
```

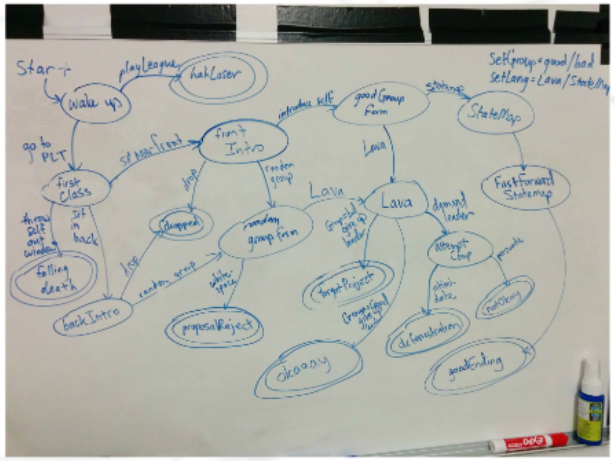
```
stateThree  <- s == "d";
stateTwo    <- s == "c";
accept      <- s == EOS;
notAccept   <- *;
```

Note: "'string'" will pass the program the argument [s,t,r,i,n,g]

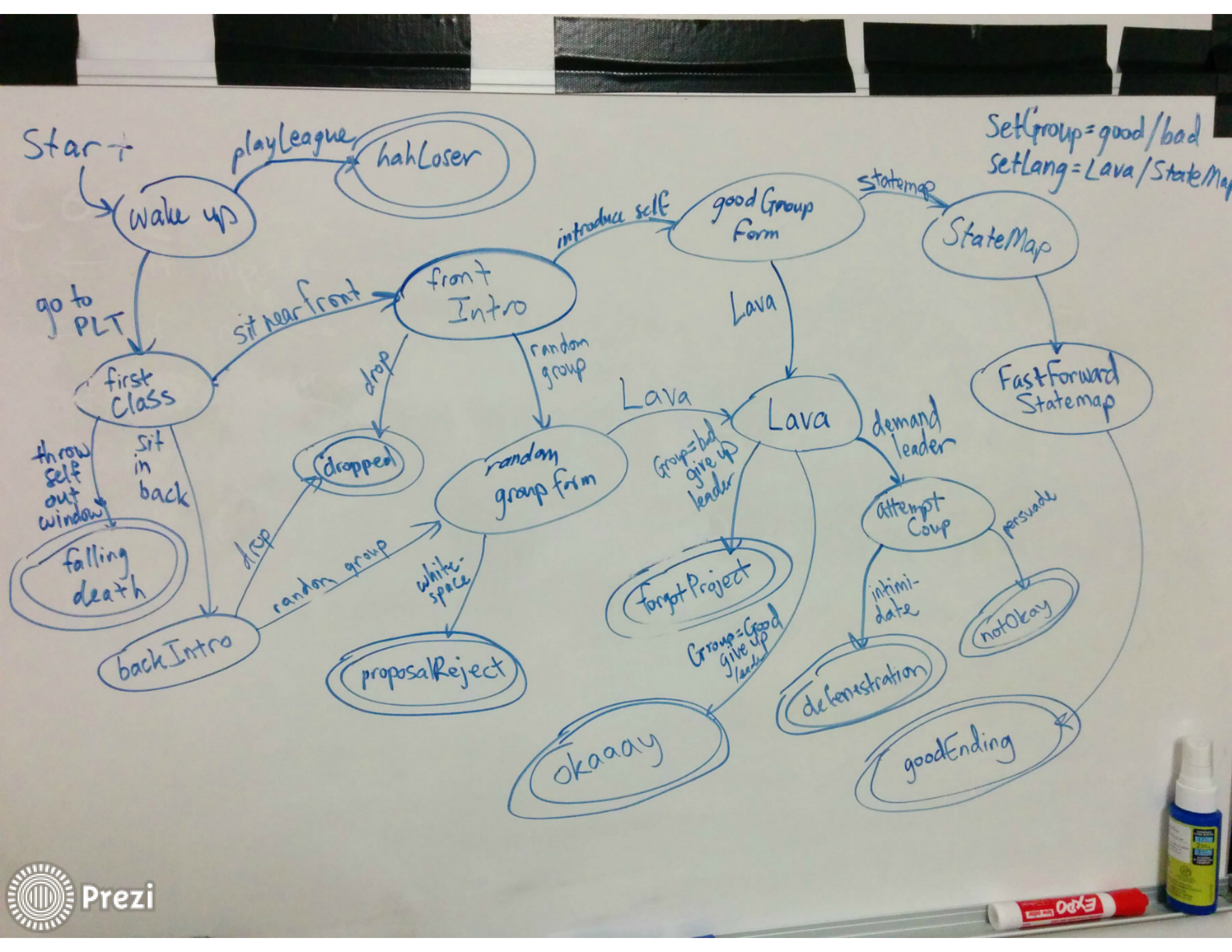
Note to the note: That's a double quote followed by a single quote.

```
bky2102@cairo:~/StateMap/StateMap$ ./compiler < sample_programs/reg_ex_test.sm
bky2102@cairo:~/StateMap/StateMap$ python output.py "'aabbcc'"
Not accepted by the DFA
bky2102@cairo:~/StateMap/StateMap$ python output.py "'abddd'"
Accepted by the DFA.
```

Choose Your Own Adventure




```
    beginning to feel uncomfortable about this class. You
    dread public speaking. You dread massive, final
    reports.\n");

    print("He continued, \"So, the main thing you're going to
    do in this class, assuming that you don't drop it like I
    want you to, is a semester long TEAM project. This is a
    team programming project. Now, the team part of it is
    easily the worst, most difficult aspect of all of it, but
    just to make it more difficult, I'm going to make you
    design and implement your own language and a compiler for
    it ... you're going to have to work with other human
    beings. This really sucks.\"\n");

    print("At this point, you're strongly considering packing
    up and leaving. You don't know anyone in this class and
    stepping on people's desk did not make a great first
    impression on your prospective classmates. As you sit
    through the rest of the lecture and attempt to focus on
    Professor Edwards' words through your rising panic, you've
    come to a decision.\n");

    print("You've decided to:");

    print("1) Stay in the course and tough it out.");
    print("2) Drop out of the course.");
    choice = input("\n");

    dropkicked  <- choice == "2";
    randomGroup <- choice == "1";
    print("Type 1 or 2 to indicate your choice.\n");
    sittingInTheBackSeat <- *;
}
```


Start
playLeague
hahLoser
wake up
go to PLT
first Class
sit near front
front Intro
introduce self
good Group Form
statemap
StateMap
SetGroup=good/bad
setLang=Lava/StateMap
Lava
FastForward Statemap
throw self out window
sit in back
drop
random group
dropped
falling death
backIntro
drop
random group
random group form
white-space
proposalReject
Lava
Lava
demand leader
Group=bad give up leader
forgotProject
Group=Good give up leader
okaaay
Attempt Coup
persuade
intimi-date
notOkay
demonstration
goodEnding

```
    beginning to feel uncomfortable about this class. You
    dread public speaking. You dread massive, final
    reports.\n");

    print("He continued, \"So, the main thing you're going to
    do in this class, assuming that you don't drop it like I
    want you to, is a semester long TEAM project. This is a
    team programming project. Now, the team part of it is
    easily the worst, most difficult aspect of all of it, but
    just to make it more difficult, I'm going to make you
    design and implement your own language and a compiler for
    it ... you're going to have to work with other human
    beings. This really sucks.\"\n");

    print("At this point, you're strongly considering packing
    up and leaving. You don't know anyone in this class and
    stepping on people's desk did not make a great first
    impression on your prospective classmates. As you sit
    through the rest of the lecture and attempt to focus on
    Professor Edwards' words through your rising panic, you've
    come to a decision.\n");

    print("You've decided to:");

    print("1) Stay in the course and tough it out.");
    print("2) Drop out of the course.");
    choice = input("\n");

    dropkicked <- choice == "2";
    randomGroup <- choice == "1";
    print("Type 1 or 2 to indicate your choice.\n");
    sittingInTheBackSeat <- *;
}
```

Non-Standard Features

Control Flow Expressed Through Transitions

- If, While, For

```
if (x == 4) {
   // do stuff
}

state {
   end <- x != 4;
   // do stuff
}
```

```
while (x < 4) {
    // do stuff
}
state1{
    state2 <- x < 4;
    end <- *;
}
state 2 {
    // do work
    x = x + 1;
    state1 <- *;
}
```

Does this work?

```
state1{
    state1 <- x < 4;
}
```

Limited primitives:
- String
- Float
- Int
- Stack<"Primitive">

Mock concurrency
Step through DFAs one state at a time.
Envision a debugger.

Control Flow Expressed Through Transitions

- If, While, For

```
if (x == 4) {
   // do stuff
}
```

```
state {
   end <- x != 4;
   // do stuff
}
```

Does
stat

```
while (x < 4) {
    // do stuff
}
state1{
    state2 <- x < 4;
    end <- *;
}
state 2 {
    // do work
    x = x + 1;
    state1 <- *;
}
```

Does this work?

```
state1{
    state1 <- x < 4;
}
```

Limited primitives:

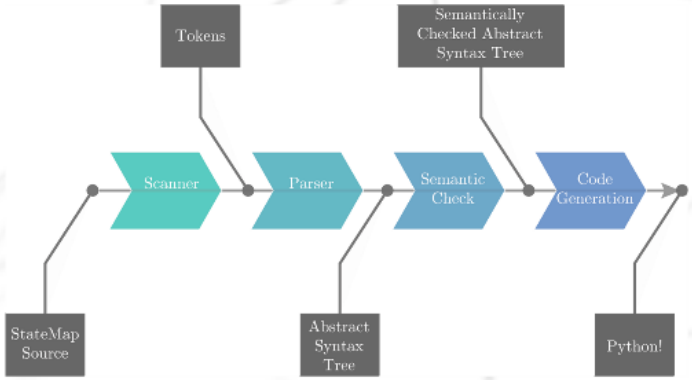
- String
- Float
- Int
- Stack<“Primitive”>

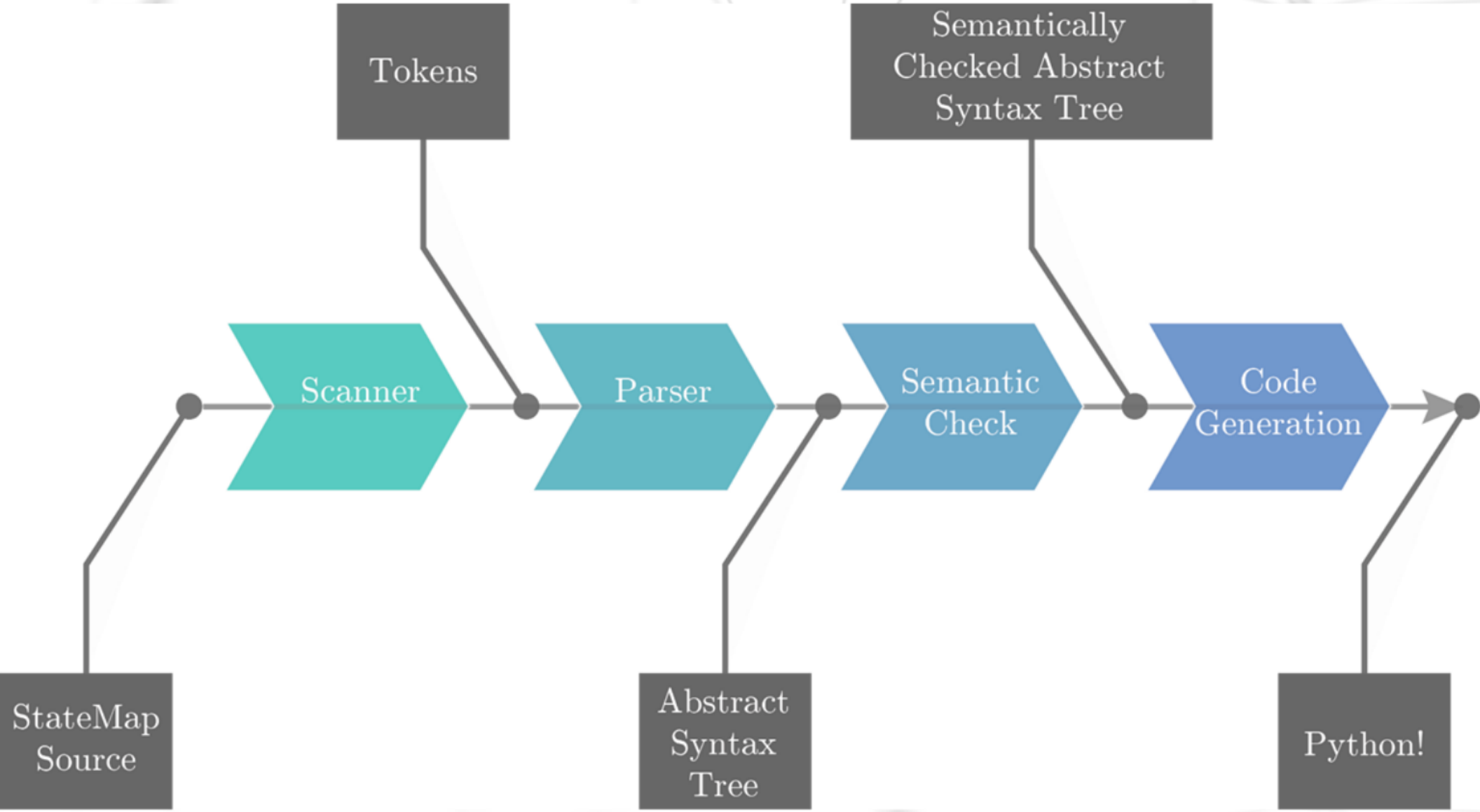
Mock concurrency

Step through DFAs one state at a time.

Envision a debugger.

Architecture




Tokens
Semantically Checked Abstract Syntax Tree
Scanner
Parser
Semantic Check
Code Generation
StateMap Source
Abstract Syntax Tree
Python!

```
int DFA get_int() {
    start {
        print("1");
        finish <- *;
    }

    finish {
        print("2");
        return 1;
    }
}

int DFA hello() {
    start {
        print("hello");
        finish <- *;
    }

    finish {
        print("world");
        return 1;
    }
}

void DFA main()
{
    start
    {
    concurrent(get_int(), hello());
    return;
    }
}
```

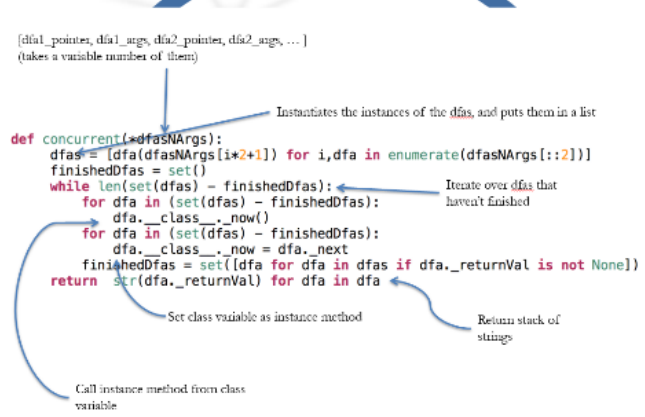
Generating Code

```
class _main:
        _now = _node_start
        def __init__(self,*args):
                try:
                        pass
                except IndexError:
                        print('RuntimeError:Too few arguments provided to dfa "main"')
                        sys.exit(1)
                self._returnVal = None
                _main._now = self._node_start
                self._next = None
                while self._returnVal is None:
                        _main._now()
                        _main._now = self._next
                return
        def _node_start(self):
                concurrent(_get_int, [], _hello, [])
                self._returnVal =  1
                self._next = None

_dfa_Dict["main"] = _main

class _get_int:
        _now = _node_start
        def __init__(self,*args):
                self._returnVal = None
                _get_int._now = self._node_start
                self._next = None
                return
        def _node_finish(self):
                print "2"
                self._returnVal =  1
                self._next = None
        def _node_start(self):
                print "1"
                if(1):
                        self._next = self._node_finish
                        return

_dfa_Dict["get_int"] = _get_int
        class _hello:
                _now = _node_start
                def __init__(self,*args):
                        self._returnVal = None
                        _hello._now = self._node_start
                        self._next = None
                        return
                def _node_finish(self):
                        print "world"
                        self._returnVal =  1
                        self._next = None
                def _node_start(self):
                        print "hello"
                        if(1):
                                self._next = self._node_finish
                                return

        _dfa_Dict["hello"] = _hello
```

```
class _main:
        _now = _node_start
        def __init__(self,*args):
                try:
                        pass
                except IndexError:
                        print('RuntimeError:Too few arguments provided to dfa "main"')
                        sys.exit(1)
                self._returnVal = None
                _main._now = self._node_start
                self._next = None
                while self._returnVal is None:
                        _main._now()
                        _main._now = self._next
                return
        def _node_start(self):
                concurrent(_get_int, [], _hello, [])
                self._returnVal =  1
                self._next = None

_dfa_Dict["main"] = _main

class _get_int:
        _now = _node_start
        def __init__(self,*args):
                self._returnVal = None
                _get_int._now = self._node_start
                self._next = None
                return
        def _node_finish(self):
                print "2"
                self._returnVal =  1
                self._next = None
        def _node_start(self):
                print "1"
                if(1):
                        self._next = self._node_finish
                        return

_dfa_Dict["get_int"] = _get_int
```

```
class _hello:
        _now = _node_start
        def __init__(self,*args):
                self._returnVal = None
                _hello._now = self._node_start
                self._next = None
                return
        def _node_finish(self):
                print "world"
                self._returnVal =  1
                self._next = None
        def _node_start(self):
                print "hello"
                if(1):
                        self._next = self._node_finish
                        return

_dfa_Dict["hello"] = _hello
```

Concurrent in Python



[dfa1_pointer, dfa1_args, dfa2_pointer, dfa2_args, …]
(takes a variable number of them)

Instantiates the instances of the dfas, and puts them in a list

```
def concurrent(*dfasNArgs):
    dfas = [dfa(dfasNArgs[i*2+1]) for i,dfa in enumerate(dfasNArgs[::2])]
    finishedDfas = set()
    while len(set(dfas) - finishedDfas):
        for dfa in (set(dfas) - finishedDfas):
            dfa.__class__._now()
        for dfa in (set(dfas) - finishedDfas):
            dfa.__class__._now = dfa._next
        finishedDfas = set([dfa for dfa in dfas if dfa._returnVal is not None])
    return  str(dfa._returnVal) for dfa in dfa
```

Iterate over dfas that haven't finished

Set class variable as instance method

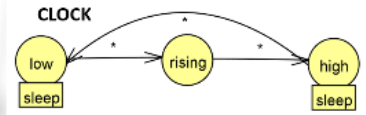
Return stack of strings

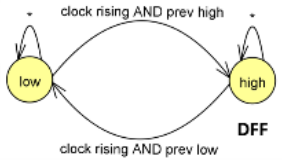
Call instance method from class variable

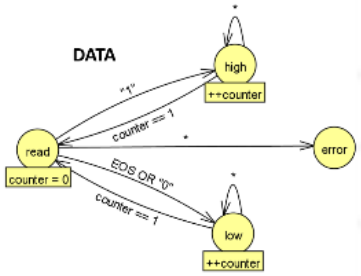
Shift Register

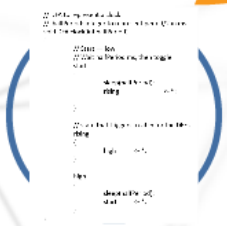
Shift Register

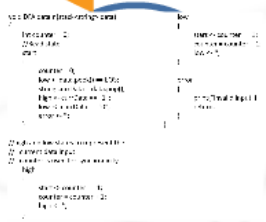
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Clock	L	R	H	L	R	H	L	R	H	L	R	H	L	R	H	L	R
Data	Read	D1	D1	Read	D2	D2	Read	D3	D3	Read	D4	D4	Read	D5	D5	Read	D6
DFF1	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4	D5	D5	D5
DFF2	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4
DFF3	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3
DFF4	L	L	L	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2
Display	start	read	print	start	read	print	start	read	print	start	read	print	start	read	print	start	read





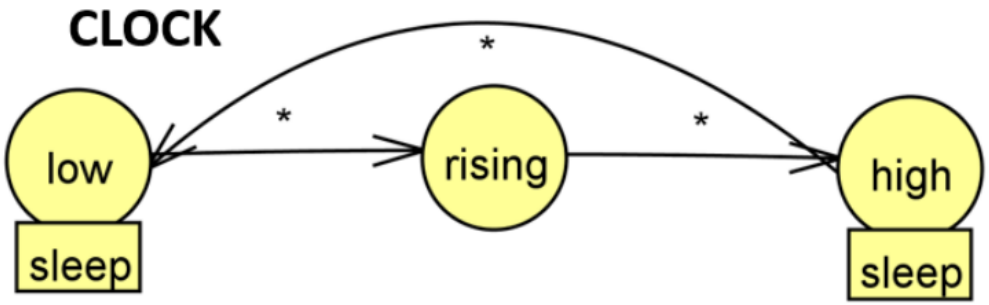


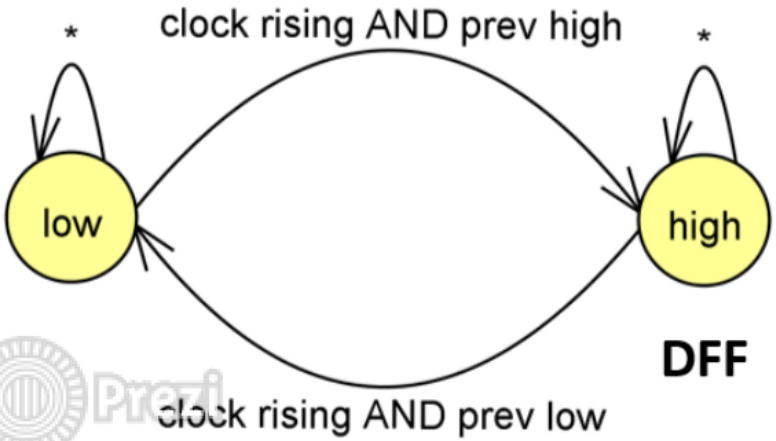


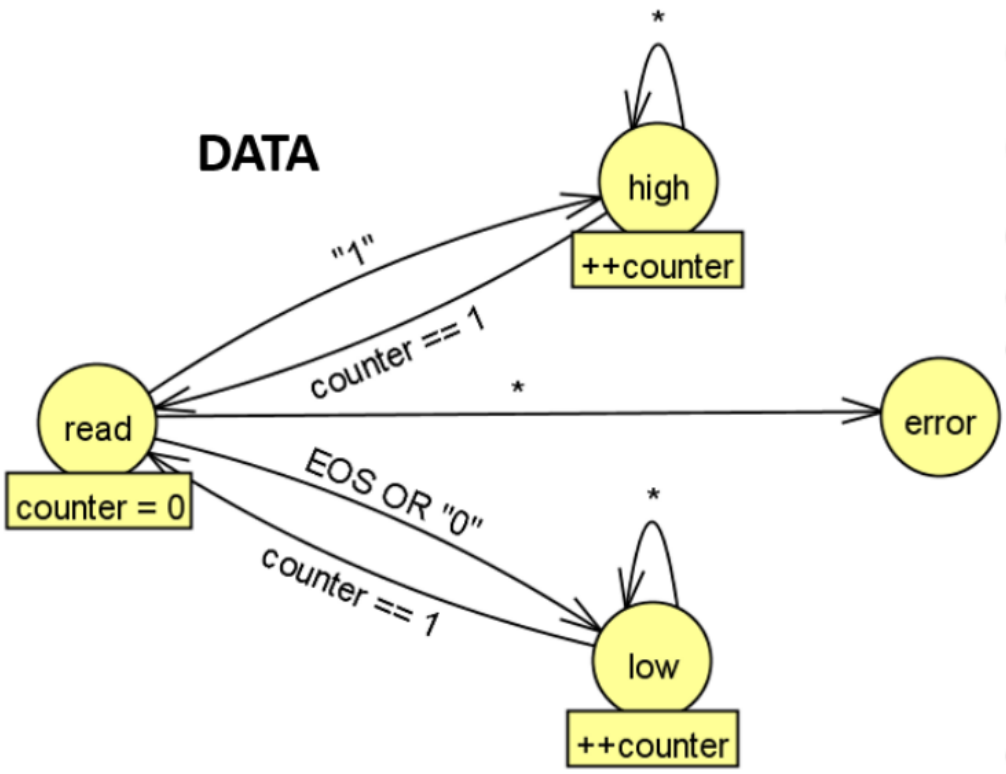


Shift Register

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Clock	L	R	H	L	R	H	L	R	H	L	R	H	L	R	H	L	R
Data	Read	D1	D1	Read	D2	D2	Read	D3	D3	Read	D4	D4	Read	D5	D5	Read	D6
DFF1	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4	D5	D5	D5
DFF2	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4
DFF3	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3
DFF4	L	L	L	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2
Display	start	read	print	start	read	print	start	read	print	start	read	print	start	read	print	start	read



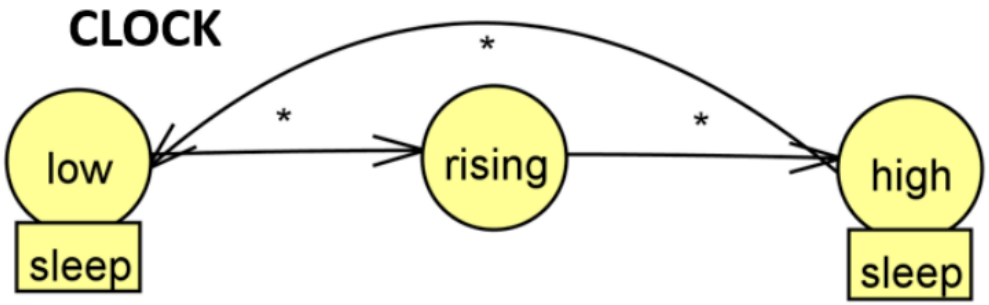


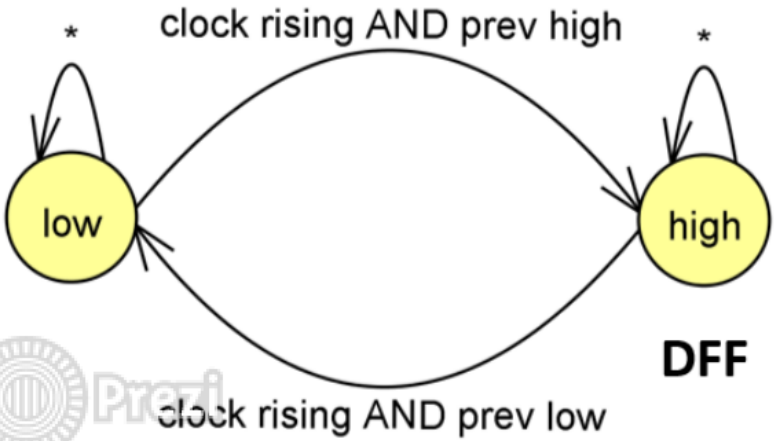


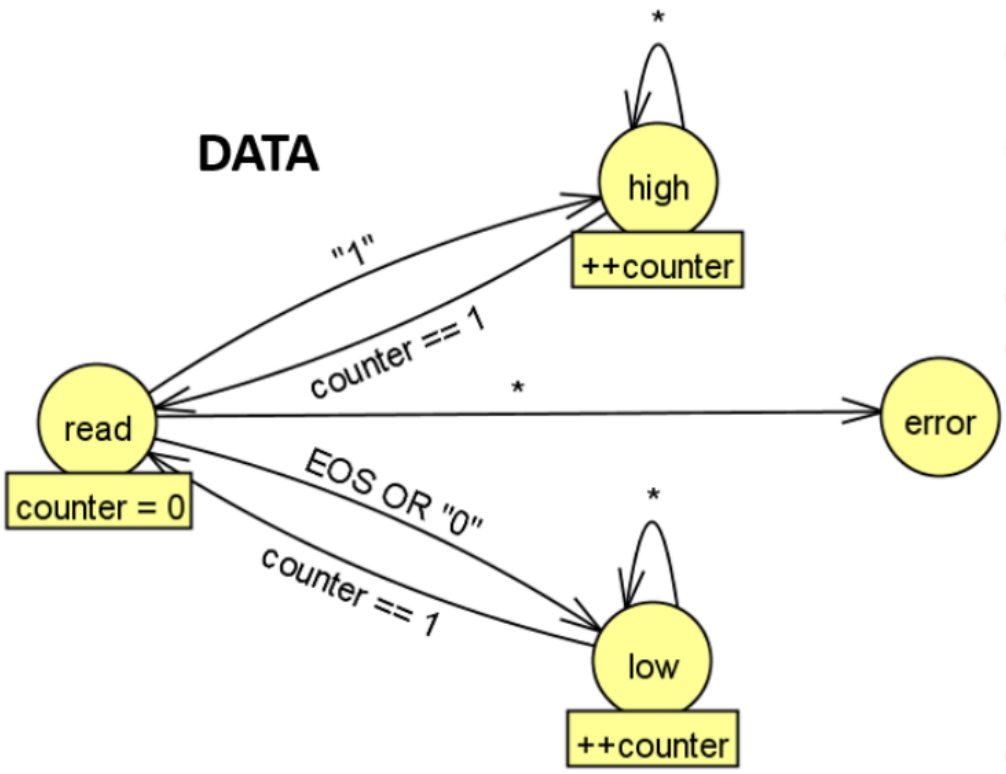
```
// DFA to represent a clock
//  halfPeriod: integer to represent period/2 in ms
void DFA clock(int halfPeriod)
{
        // Start == low
        // Wait halfPeriod ms, then toggle
        start
        {
                sleep(halfPeriod);
                rising                  <- *;

        }

        // state that triggers a catch for the DFFs
        rising
        {
                high        <- *;
        }

        high
        {
                sleep(halfPeriod);
                start       <- *;
        }
}
```

Shift Register

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Clock	L	R	H	L	R	H	L	R	H	L	R	H	L	R	H	L	R
Data	Read	D1	D1	Read	D2	D2	Read	D3	D3	Read	D4	D4	Read	D5	D5	Read	D6
DFF1	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4	D5	D5	D5
DFF2	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4
DFF3	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3
DFF4	L	L	L	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2
Display	start	read	print	start	read	print	start	read	print	start	read	print	start	read	print	start	read



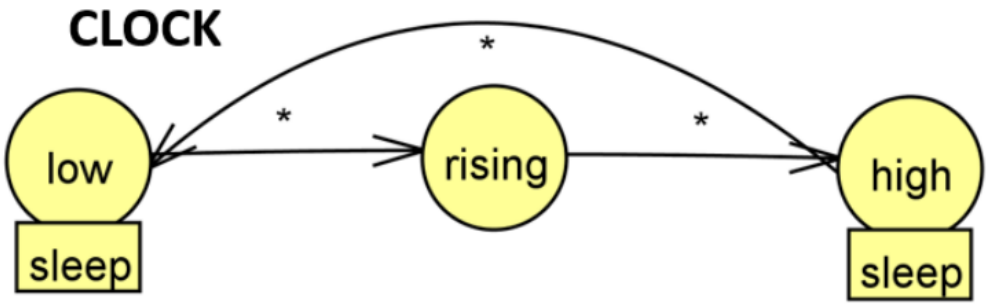


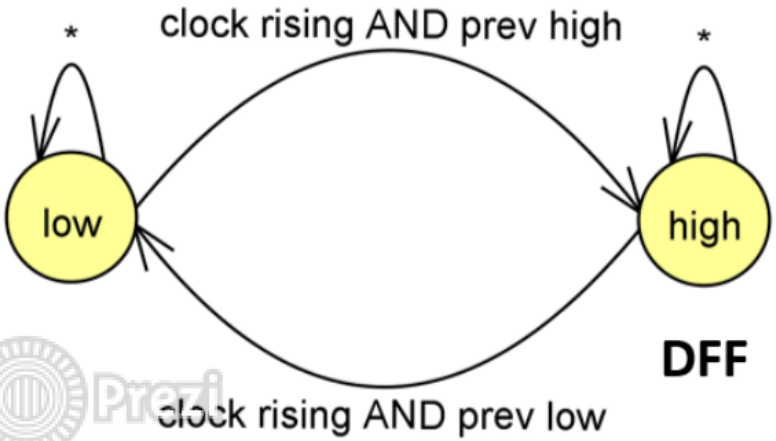


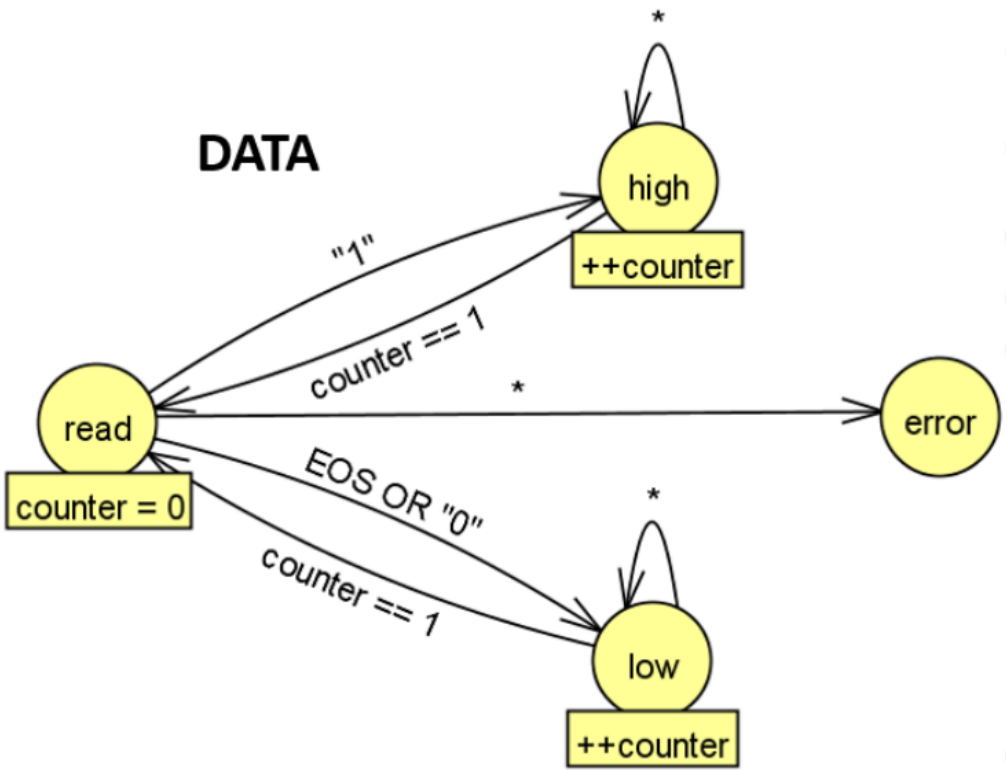
```
// 1st T-FlipFlop in Shift Register
//  Catches data on every rising clock
void DFA DFF1()
{
        // low output
        start
        {
                high    <- (state("clock") == "rising” && state("dataIn") == "high");
                start   <- *;
        }

        // high output
        high
        {
                start   <- (state("clock") == "rising” && state("dataIn") == "low");
                high    <- *;
        }
}
```

Shift Register

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Clock	L	R	H	L	R	H	L	R	H	L	R	H	L	R	H	L	R
Data	Read	D1	D1	Read	D2	D2	Read	D3	D3	Read	D4	D4	Read	D5	D5	Read	D6
DFF1	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4	D5	D5	D5
DFF2	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3	D4	D4	D4
DFF3	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2	D3	D3	D3
DFF4	L	L	L	L	L	L	L	L	L	L	L	D1	D1	D1	D2	D2	D2
Display	start	read	print	start	read	print	start	read	print	start	read	print	start	read	print	start	read







```
void DFA dataIn(stack<string> data)
{
	int counter = 0;
	//Read state
	start
	{
		counter = 0;
		low <- data.peek() == EOS;
		string currData = data.pop();
		high <- currData == "1";
		low  <- currData == "0";
		error <- *;
	}

//high and low states to represent the
//   current data input.
//   counter is used for synchronicity
	 high
	 {
		start <- counter == 1;
		counter = counter + 1;
		high <- *;
	 }
	 low
	 {
		start <- counter == 1;
		counter = counter + 1;
		low <- *;
	 }

	 error
	 {
		print("invalid input");
		return;
	 }
}
```

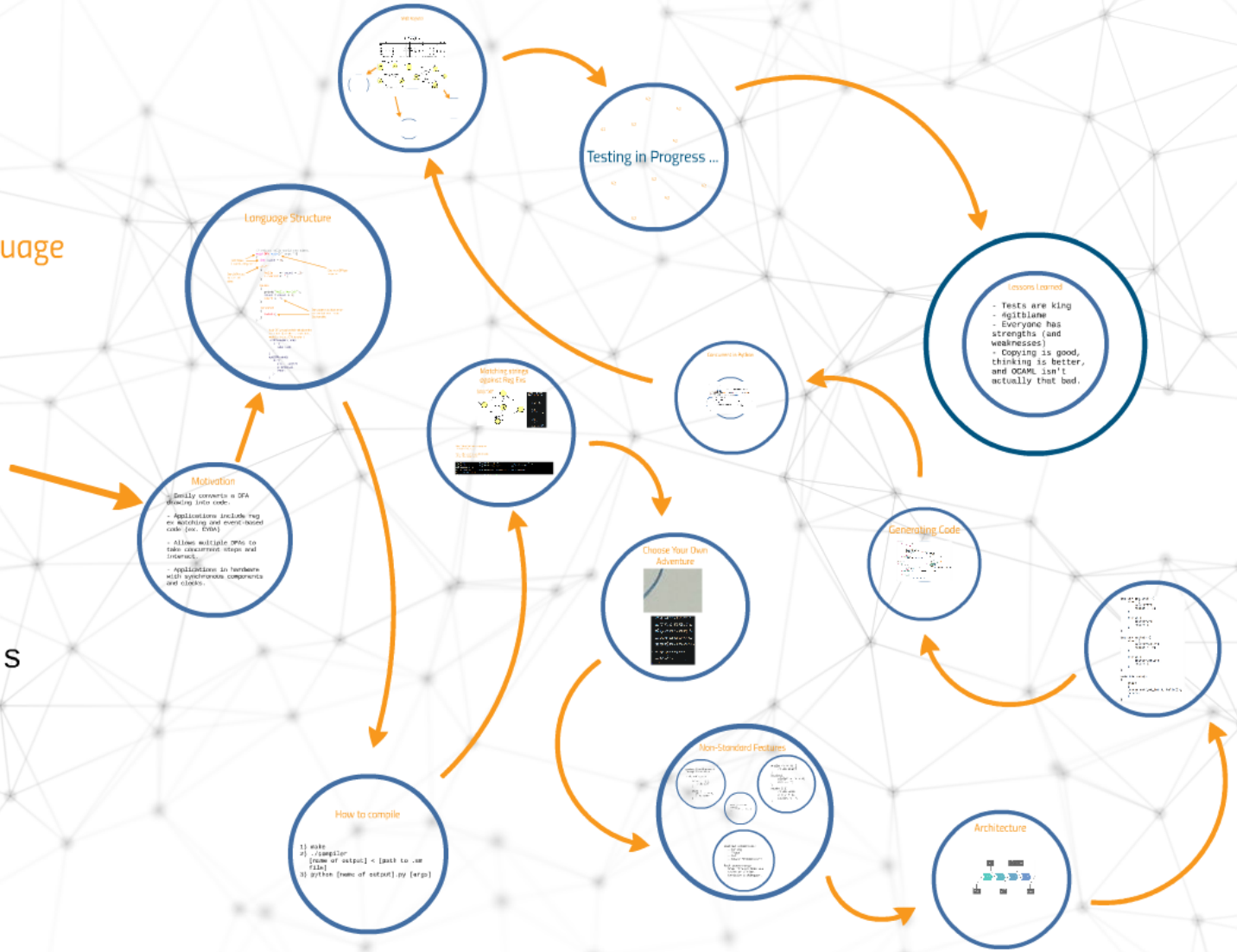

42
42
42
42
42
Testing in Progress ...
42
42
42
42
42
Prezi

Lessons Learned

- Tests are king
- #gitblame
- Everyone has strengths (and weaknesses)
- Copying is good, thinking is better, and OCAML isn't actually that bad.

StateMap

A DFA Simulation Language



Oren Finard
Jackson Foley
Alexander Peters
Brian Yamamoto
Zuokun Yu