Compiler Construction

Assignment-01

Automata Design Document

Imaad Fazal 23I-0656 CS-C

Muhammad Immad 23I-0026 CS-C

Regular Expressions for Each Category:

Keywords: (start|finish|loop|condition|declare|output|input|function|return|break|continue|else)

Identifiers: [A-Z][a-z0-9]{0,30}

Integer Literals: [+-]?[0-9]+

Floating-Point Literals: [+-]?[0-9]+\.[0-9]{1,6}([eE][+-]?[0-9]+)?

Boolean Literals: (true|false)

String Literals: \"([^\"\\]|\\.)*\"

Character Literals: '([^'\\]|\\.)'

Arithmetic Operators: (\+|\-|*|\/|%|**)

Relational Operators: (==|!=|<=|>=|<|>)

Logical Operators: (&&|\|\||!)

Assignment Operators: (=|\+=|\-=|*=|\/=)

Inc/Dec Operators: (\+\+|--)

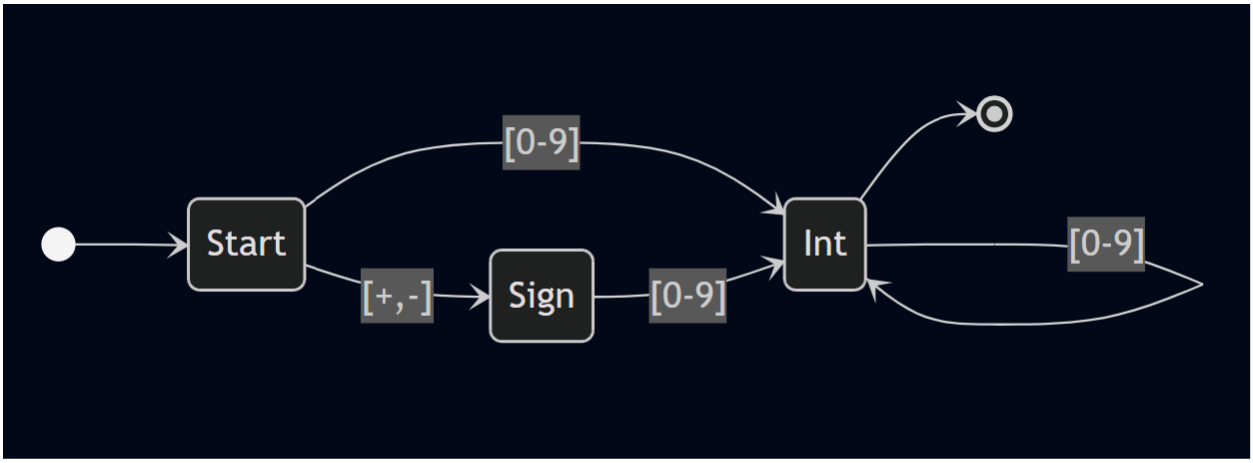
Punctuators: (\(|\)|\{|\}|\[|\],|;|:)

Single-line Comment: ##[^\n]*

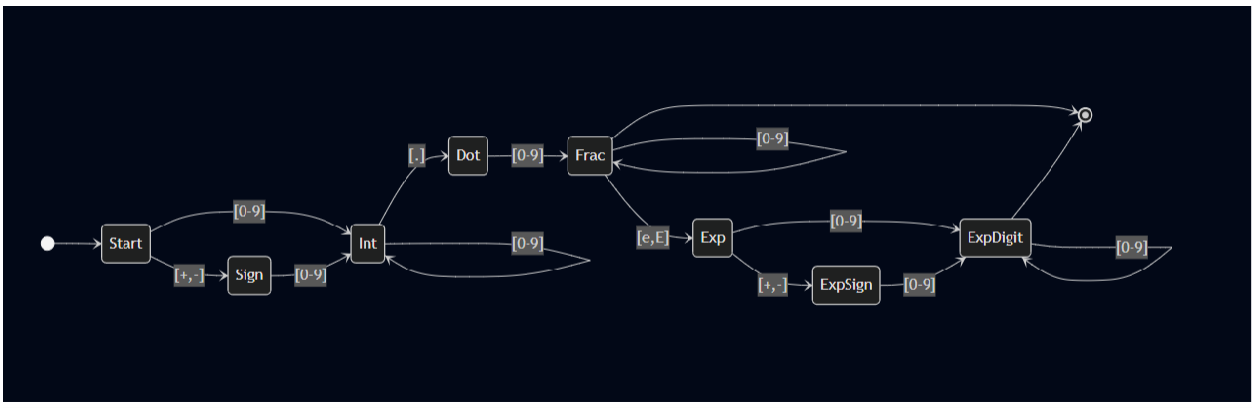
Multi-line Comment: #*([^*]|*+[^#])**+#

NFA Diagrams:

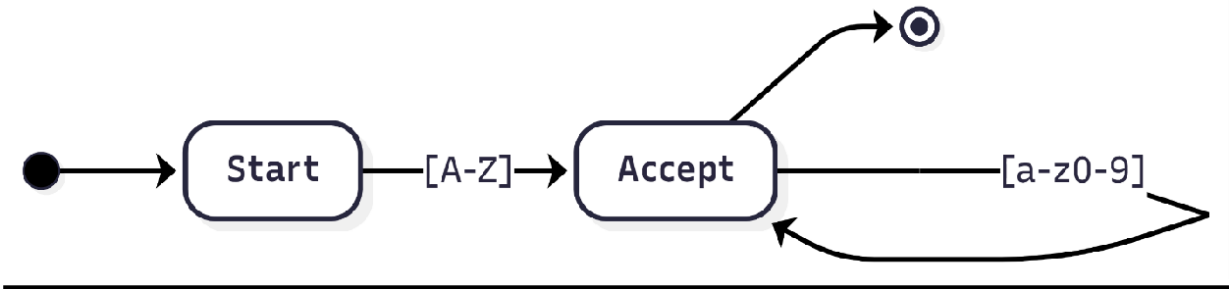
1) Integer Literal (Mandatory):



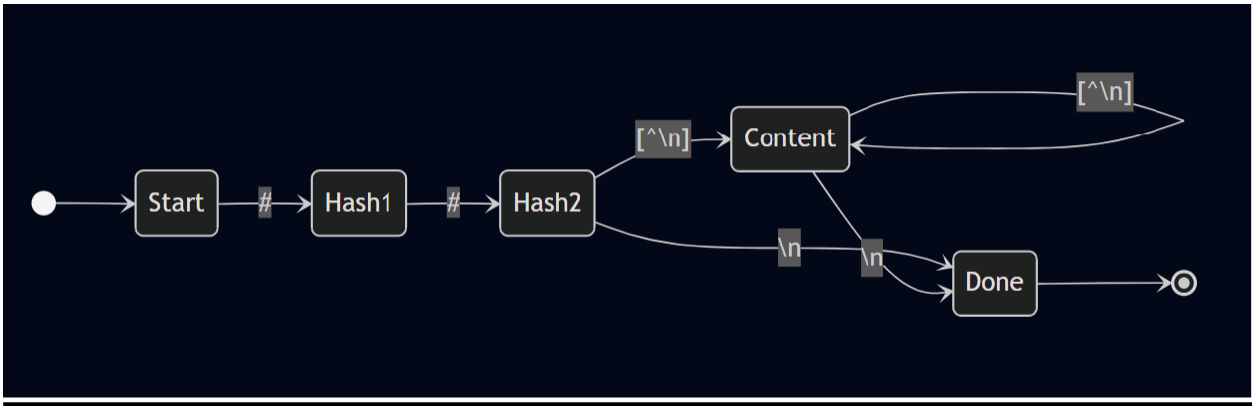
2) Floating Point Literal (Mandatory):



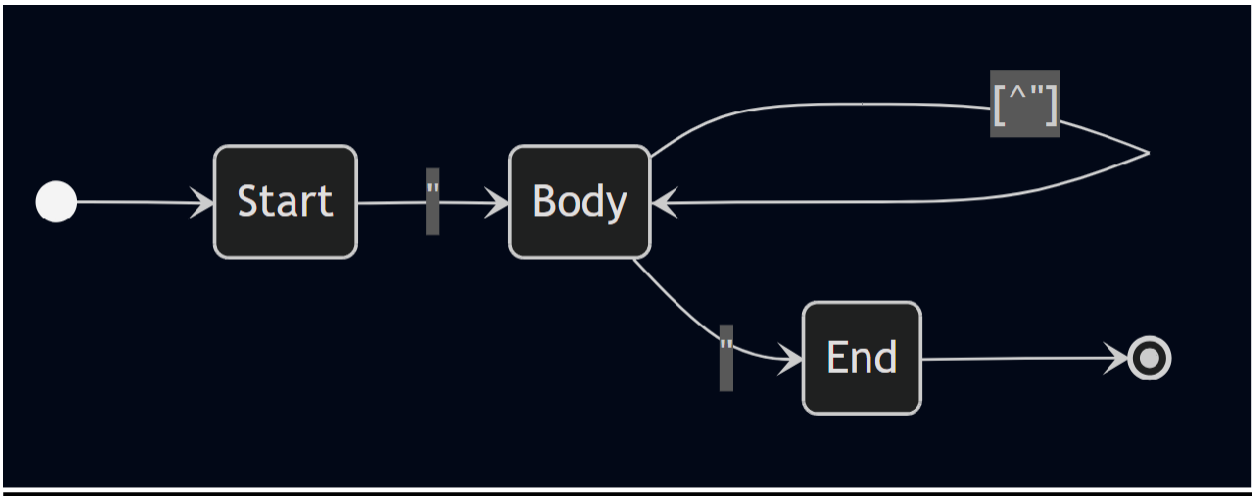
3) Identifier (Mandatory):



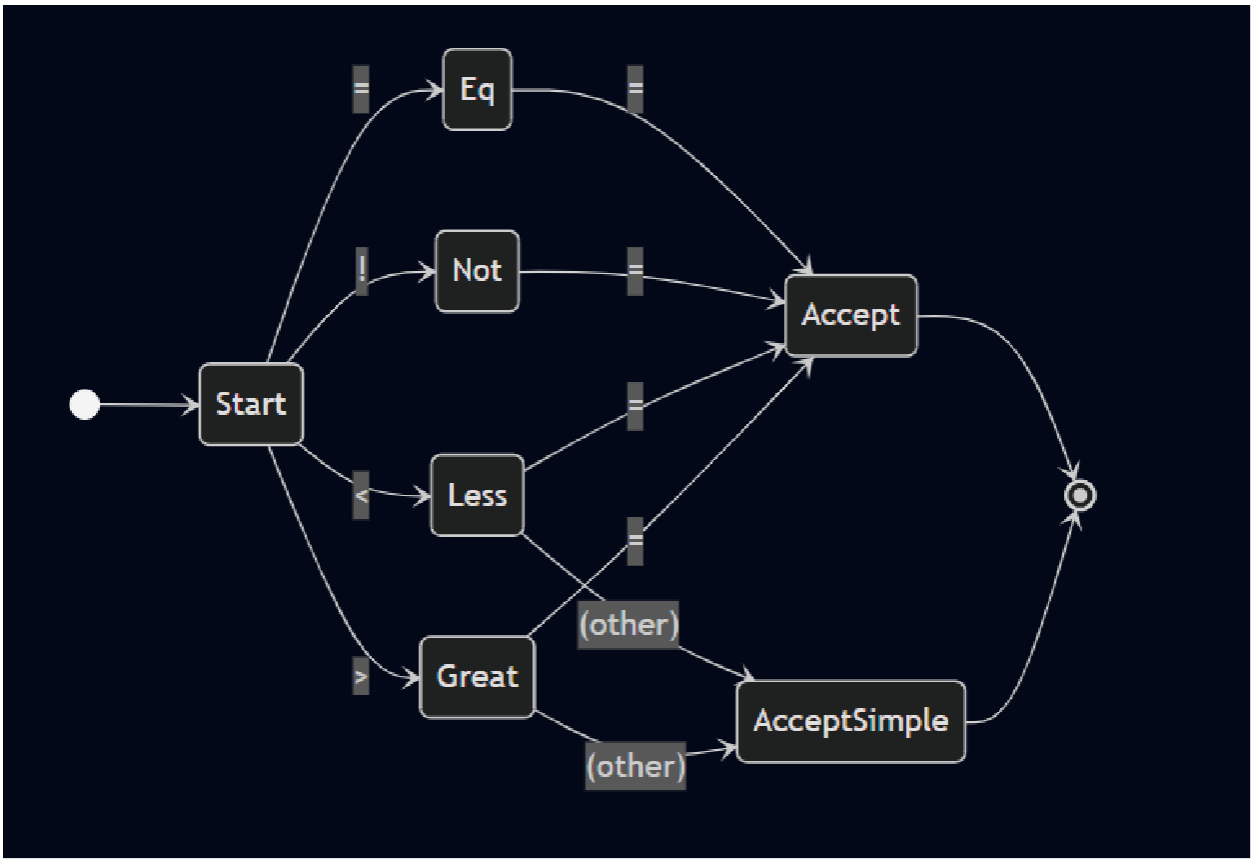
4) Single Line Comment (Mandatory):



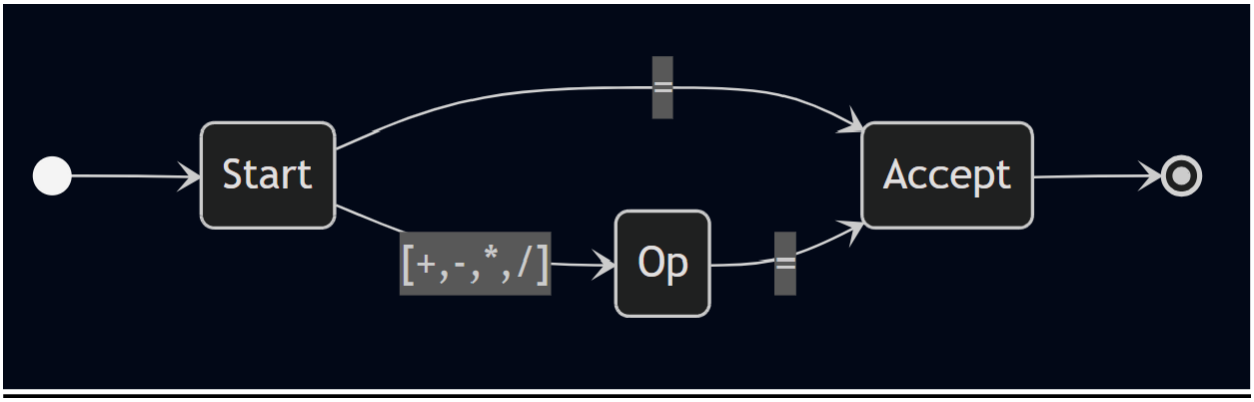
5) String Literal (Choice 1):



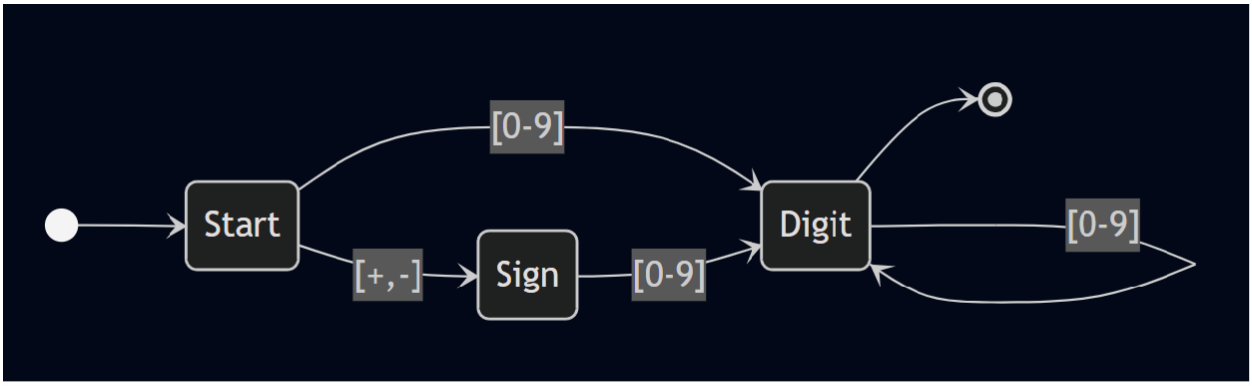
6) Relational Operators (Choice 2):



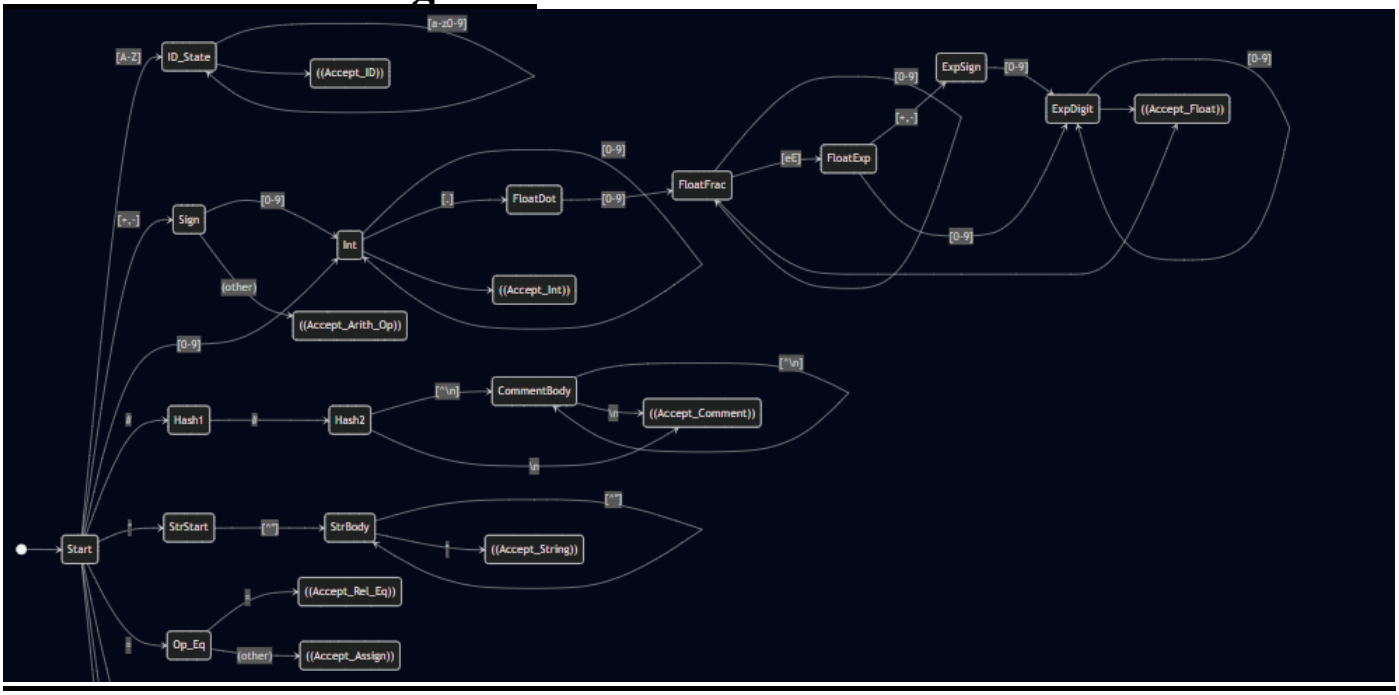
7) Assignment Operators (Choice 3):

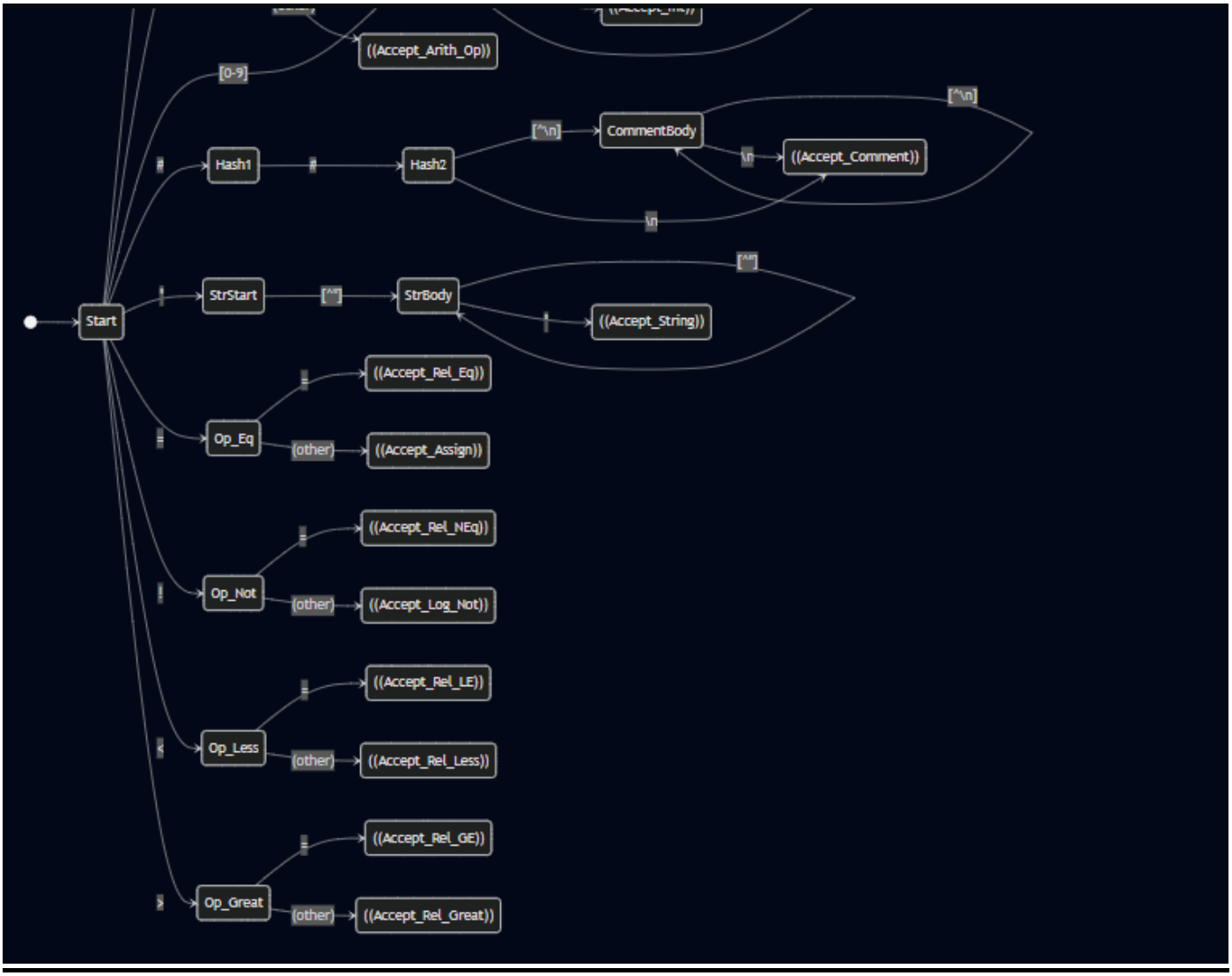


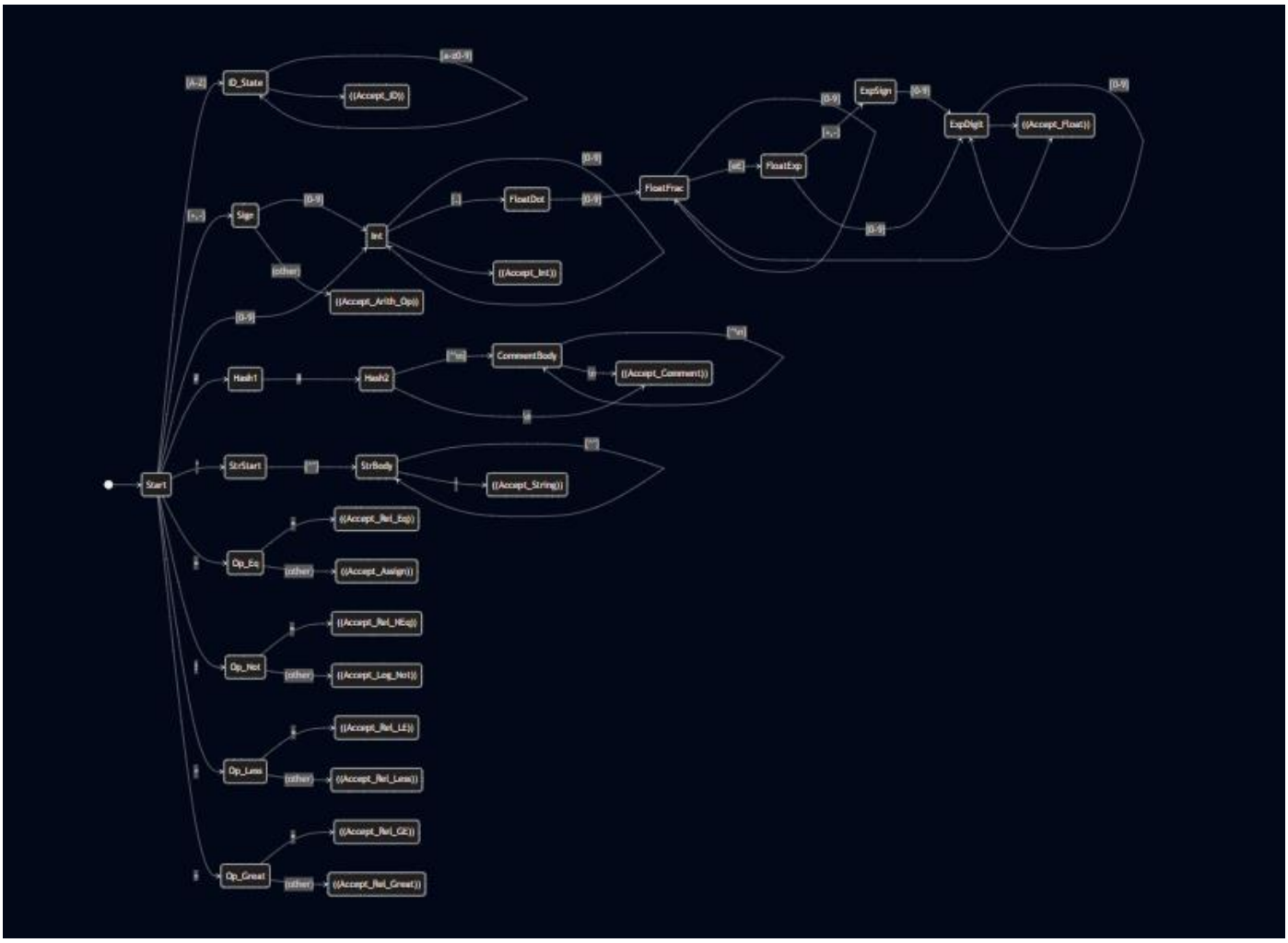
8) Minimized DFA for Integer Literal:



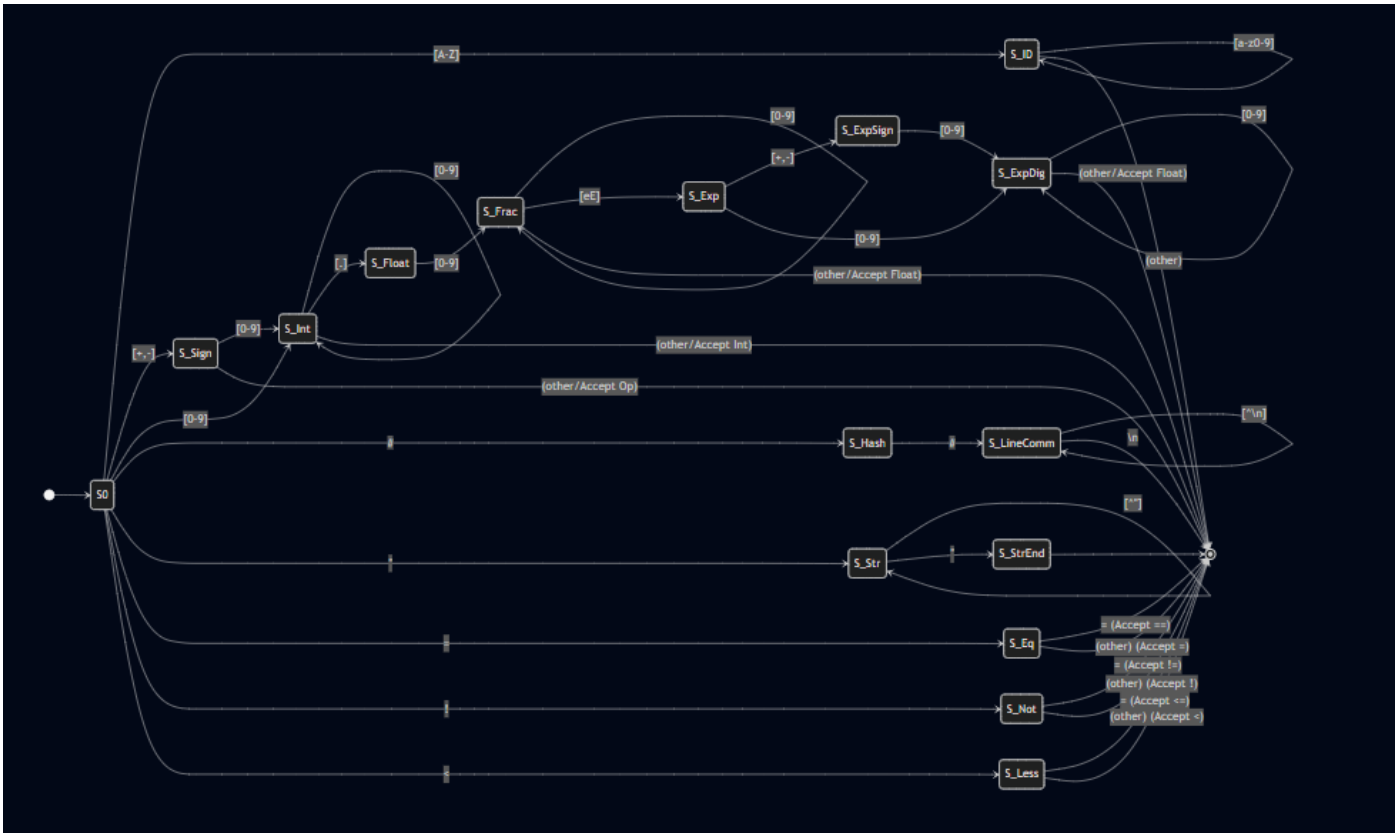
Combined NFA Diagram:




((Accept_Arith_Op))
[0-9]
#
Hash1
#
Hash2
[^\n]
CommentBody
[^\n]
\n
((Accept_Comment))
\n
[^"]
"
StrStart
[^"]
StrBody
"
((Accept_String))
Start
=
((Accept_Rel_Eq))
=
Op_Eq
(other)
((Accept_Assign))
=
((Accept_Rel_NEq))
!
Op_Not
(other)
((Accept_Log_Not))
=
((Accept_Rel_LE))
<
Op_Less
(other)
((Accept_Rel_Less))
=
((Accept_Rel_GE))
>
Op_Great
(other)
((Accept_Rel_Great))



Combined DFA Diagram:



Combined Transition Table:

State	[A-Z]	[a-z0-9]	[0-9]	[+,-]	.	#	"	=	!	<	Other
S0 (Start)	S_ID	Err	S_Int	S_Sign	Err	S_Hash	S_Str	S_Eq	S_Not	S_Less	Err
S_ID	Accept	S_ID	S_ID	Accept	Accept	Accept	Accept	Accept	Accept	Accept	(Retract)
S_Sign	Accept	Accept	S_Int	Accept	Accept	Accept	Accept	Accept	Accept	Accept	Accept (Op)
S_Int	Accept	Accept	S_Int	Accept	S_Float	Accept	Accept	Accept	Accept	Accept	Accept (Int)
S_Float	Err	Err	S_Frac	Err	Err	Err	Err	Err	Err	Err	Err
S_Frac	Accept	Err	S_Frac	Accept	Err	Accept	Accept	Accept	Accept	Accept	Accept

State	[A-Z]	[a-z0-9]	[0-9]	[+,-]	.	#	"	=	!	<	Other
											(Float)
S_Hash	Err	Err	Err	Err	Err	S_LineComm	Err	Err	Err	Err	Err
S_LineComm	S_LC	S_LC	S_LC	S_LC	S_LC	S_LC	S_LC	S_LC	S_LC	S_LC	(Loop until \n)
S_Str	S_Str	S_Str	S_Str	S_Str	S_Str	S_Str	S_StrEnd	S_Str	S_Str	S_Str	S_Str
S_StrEnd	Accept	Accept	Accept	Accept	Accept	Accept	Accept	Accept	Accept	Accept	(Retract)
S_Eq	Accept	Accept	Accept	Accept	Accept	Accept	Accept	Accept (==)	Accept	Accept	Accept (=)

Transition Tables:

1) Identifier (Mandatory):

State	[A-Z]	[a-z0-9_]	Other
Start	Accept	Error	Error
Accept	Accept	Accept	(Retract)

2) Integer Literal (Mandatory):

State	[+ , -]	[0-9]	Other
Start	Sign	Digit	Error
Sign	Error	Digit	Error

Digit	Error	Digit	(Retract)

3) Floating Point Literal (Mandatory):

State	**[+ , -]**	**[0-9]**	**. (Dot)**	**[e , E]**	**Other**
Start	Sign	IntPart	Error	Error	Error
Sign	Error	IntPart	Error	Error	Error

State	[+ , -]	[0-9]	. (Dot)	[e , E]	Other
IntPart	Error	IntPart	Dot	Error	(Retract)
Dot	Error	FracPart	Error	Error	Error
FracPart	Error	FracPart	Error	Exp	(Retract)
Exp	ExpSign	ExpDigit	Error	Error	Error
ExpSign	Error	ExpDigit	Error	Error	Error
ExpDigit	Error	ExpDigit	Error	Error	(Retract)

4) Single Line Comment (Mandatory):

State	# (Hash)	\n (Newline)	Other Char
Start	Hash1	Error	Error
Hash1	Hash2	Error	Error
Hash2	Content	Done	Content
Content	Content	Done	Content
Done	(Accept)	(Accept)	(Accept)

5) String Literal (Choice 1):

State	" (Quote)	Other Char
Start	Body	Error
Body	End	Body
End	(Retract)	(Retract)

6) Relational Operators (Choice 2):

State	=	!	<	>	Other
Start	Eq	Not	Less	Great	Error
Eq	Accept (==)	Error	Error	Error	(Retract)
Not	Accept (!=)	Error	Error	Error	Error
Less	Accept (<=)	Error	Error	Error	Accept (<)
Great	Accept (>=)	Error	Error	Error	Accept (>)

7) Assignment Operators (Choice 3):

State	=	+ , - , * , /	Other
Start	Accept (=)	Op	Error
Op	Accept (+=, etc)	Error	(Retract)
Accept	(Retract)	(Retract)	(Retract)

DFA Minimization Algorithm:

1. Algorithm Used:

We implemented **Moore's Algorithm** (Partition Refinement).

This algorithm works by initially partitioning states into two groups: **Final** and **Non-Final**. It then iteratively refines these partitions by checking if states within the same group behave differently (i.e., transition to different groups) for the same input symbol.

2. Steps:

1. **Initialization:** Start with partition P = {F, Q - F} (Final states vs Non-Final states).
2. **Iteration:** For every group G in P, check if all states s in G transition to the same group G' for every input symbol alpha.
3. **Split:** If two states s_1, s_2 in the same group transition to different groups for the same input, split the group.
4. **Termination:** Repeat until no more splits occur.
5. **Merge:** All states remaining in the same group are equivalent and can be merged.

3. Example Output (from DFAMinimizer.java):

Input DFA has redundant states q3 (Final) and q4 (Final).

Initial Partition: `{q0, q1, q2}` (Non-Final), `{q3, q4}` (Final)

Iteration 1:

- `q0` goes to `q1` (Group 0) on 'a', `q2` (Group 0) on 'b'.
- `q1` goes to `q1` (Group 0) on 'a', `q3` (Group 1) on 'b'.
- **Split!** `q0` and `q1` behave differently on input 'b'.

Final Result:

`q3` and `q4` remain in the same group, proving they are equivalent and can be merged.

```
PS C:\Users\Hp\OneDrive\Desktop\23i-0656-23i-0026-C> javac src/DFAMinimizer.java
PS C:\Users\Hp\OneDrive\Desktop\23i-0656-23i-0026-C> java src.DFAMinimizer
  --- Original DFA States ---
  State q0  -> a->q1 b->q2
  State q1  -> a->q1 b->q3
  State q2  -> a->q1 b->q2
  State q3*  -> a->q1 b->q2
  State q4*  -> a->q1 b->q2

  --- Running Minimization Algorithm ---
  Minimized Partitions (Merged States):
  New State M0: { q1 }
  New State M1: { q2 q0 }
  New State M2: { q3 q4 }
PS C:\Users\Hp\OneDrive\Desktop\23i-0656-23i-0026-C>
```