Section 2 3 Level 2023170793 تاج الدين سر الختم عبدالواحد عثمان

Section 1 3 Level 2023170789 احمد سفيان احمد حسن

Section 1 3 Level 2023170811 احمد ياسر محمد الفكي

Section 1 3 Level 2023170121 ايمن محجوب عبدالله محمد

Section 1 3 Level 2023170092 إسماعيل إبراهيم بخيت أحمد مسعود

Section 4 3 Level 2023170780 علا عبدالرحيم عبدالعزيز أحمد

Table 1: TinyProgLang Token List & Regular Expressions

Token Category	Token Type or "Lexeme"	Regular Expression or "RE"	Additional Notes
Literals	T_NUMBER	`[0-9]+(\.[0-9]+)?`	
	T_STRING	`"[^"]*"`	
Identifier	T_IDENTIFIER	`([a-z]|[A-Z])([a-z]|[A-Z]|[0-9])*`	Check against keyword list
Keywords	T_INT	`int`	These are literal strings.
	T_FLOAT	`float`	
	T_STRING	`string`	
	T_READ	`read`	
	T_WRITE	`write`	
	T_REPEAT	`repeat`	
	T_UNTIL	`until`	
	T_IF	`if`	
	T_ELSEIF	`elseif`	
	T_ELSE	`else`	
	T_THEN	`then`	
	T_RETURN	`return`	
	T_ENDL	`endl`	
	T_MAIN	`main`	
Double Char Operators	T_ASSIGN	`:=`	
	T_NOTEQUAL	`<>`	
	T_AND	`&&`	
	T_OR	`||`	
Single Char Operators	T_PLUS	`\+`	
	T_MINUS	`-`	
	T_TIMES	`\*`	
	T_DIVIDE	`/`	
	T_LESSTHAN	`<`	
	T_GREATERTHAN	`>`	
	T_EQUAL	`=`	
Symbols	T_LPAREN	`\(`	Left Parenthesis
	T_RPAREN	`\)`	Right Parenthesis
	T_LCURLY	`{`	Left Curly Brace
	T_RCURLY	`}`	Right Curly Brace
	T_COMMA	`,`	
	T_SEMICOLON	`;`	
Ignored	COMMENT	`/\*.*?\*/`	Discard this token.
	WHITESPACE	`[ \t\n\r]+` *or* `\s+`	Discard this token.
Special	T_EOF	`N/A (Generated by scanner)`	Virtualized; Signals the E

1 Alphabet of the TinyProgLang

The alphabet of the TINY language, denoted by Σ, is the set of all valid input symbols from which strings in the language are formed.

$$\Sigma = \{\texttt{a-z}, \texttt{A-Z}, \texttt{0-9}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{:}, \texttt{=}, \texttt{<}, \texttt{>}, \texttt{\&}, \texttt{|}, \texttt{.}, \texttt{"}, \texttt{(}, \texttt{)}, \texttt{\{}, \texttt{\}}, \texttt{,}, \texttt{;}, \texttt{space}, \texttt{\textbackslash t}, \texttt{\textbackslash n}, \texttt{\textbackslash r}\}$$

The words in the language are composed of sequences of strings over Σ . And the RE for the tokens in our language are built from this alphabet. The main character classes are enumerated as follows: a) Letters; defined by the RE `[a-zA-Z]`. and b) Digits, defined by the RE `[0-9]`; and d) Symbols, where each character is a literal (e.g., `\+`, `;`); and e) Whitespace, defined by the RE `[ \t\n\r]`. The Language allows Comments matched with the RE `(/* .*?  */)`. The `Table 1` shows the full description of tokens TingProgLang does allow.

Token Priority

When constructing a scanner, the order in which token rules are defined is critical. The scanner follows two fundamental principles to resolve ambiguity,

1. **The Longest Match Rule:** The scanner will always attempt to match the longest possible string of characters from the input. For example, given the input `<>`, the scanner will match the single token `T_NOTEQUAL` rather than two separate tokens, `T_LESSTHAN` `<` and `T_GREATERTHAN` `>`.

2. **The Priority Rule (First Match Wins):** Whenever two or more rules match the *same* longest string, the scanner will use the rule that was defined *first* in the priority list.

We apply the second rule to TinyProgLang by defining our tokens from most-specific to least-specific. Prioritizing keywords over identifiers and distinguishing both of these entities is one classical example (i.e. with the input `if` which matches both the `T_IF` and the `T_IDENTIFIER` rules (RE `[a-zA-Z][a-zA-Z0-9]*`) with the same length.)

Token Priority Order for TinyProgLang

We list the relevant tokens in order of priority, from highest to lowest. The scanner consults this priority list to decide which regular expression to use when it is taking input symbols.

Token Priority Rationale

1. **Reserved Keywords:** These are the most specific literal strings. They must be listed before the general `T_IDENTIFIER` rule.

2. **Multi-Character Operators:** These must be listed before their single-character sub-parts to satisfy the Longest Match Rule.

3. **General Pattern Tokens:** These compose the match-all patterns in our TinyProgLang, and they must come after all keywords.

4. **Single-Character Operators & Symbols:** The relative order of these unit tokens does not matter. `"Unit"` here refers to the fact that they are composed of complete, separate and unambigious patterns.

5. **Ignored Patterns:** These are matched last and discarded Or by the scanner.

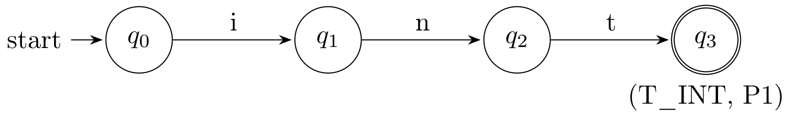
Token Priority List And Their RE Mapping

From above rules we declare the enumerated list, and sign them into four priority classes (from P1 to P5, with P1 the highest) as follows:
{`T_INT`, `T_FLOAT`, `T_STRING_KEY`, `T_READ`, `T_WRITE`, `T_REPEAT`, `T_UNTIL`, `T_IF`, `T_ELSEIF`, `T_ELSE`, `T_THEN`, `T_RETURN`, `T_ENDL`, `T_MAIN` } { `T_ASSIGN`, `T_NOTEQUAL`, `T_AND`, `T_OR`} { `T_NUMBER`, `T_IDENTIFIER`, `T_STRING` } { `T_PLUS`, `T_MINUS`, `T_TIMES`, `T_DIVIDE`, `T_LESSTHAN`, `T_GREATERTHAN`, `T_EQUAL`, `T_LPAREN`, `T_RPAREN`, `T_LCURLY`, `T_RCURLY`, `T_COMMA`, `T_SEMICOLON` } {`COMMENT`, `WHITESPACE`}

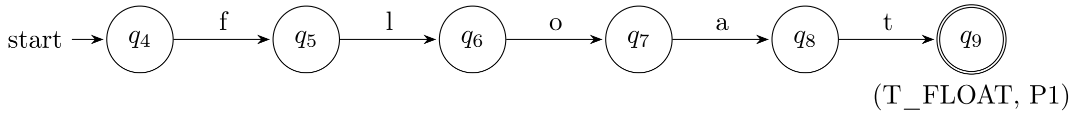
Mapped to RE list:
{ (`int`), (`float`), (`string`), (`read`), (`write`), (`repeat`), (`until`), (`if`), (`elseif`), (`else`), (`then`), (`return`), (`endl`), (`main`) } { (`:=`), (`<>`), (`&&`), (`||`) } { (`[0-9]+(\.[0-9]+)?`), (`([a-z]|[A-Z])([a-z]|[A-Z]|[0-9])*`), (`"[^"]*"`) } { (`\+`), (`-`), (`\*`), (`/`), (`<`), (`>`), (`=`), (`\(`), (`\)`), (`{`), (`}`), (`,`), (`;`) } { (`/\*.*?\*/`), (`[ \t\n]+`) }

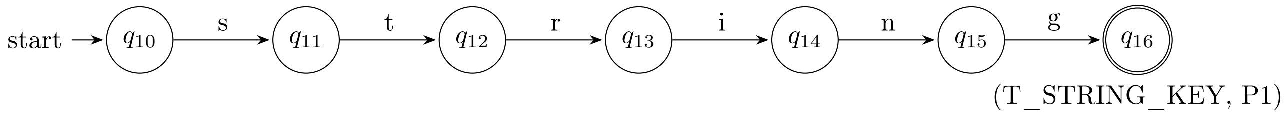
From those two lists we attempt to Build our NFAs using Thompson Construction Method, which is used to build the subsequent DFA via the Subset Construction Method. The prioritized classes would then help us resolve conflicts label the end state for each sub-finite automata.

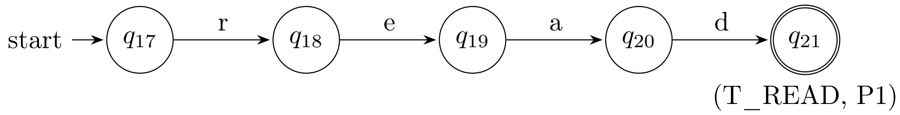
Sub-NFAs for Keywords; Priority 1:

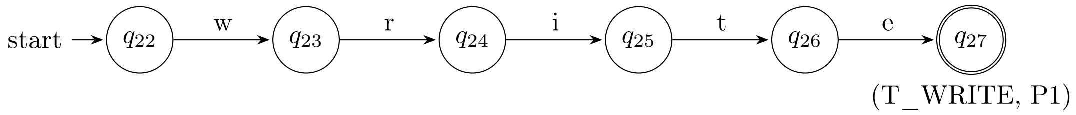


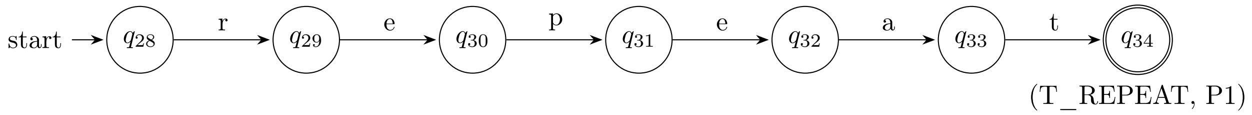
NFA$_1$: RE = int

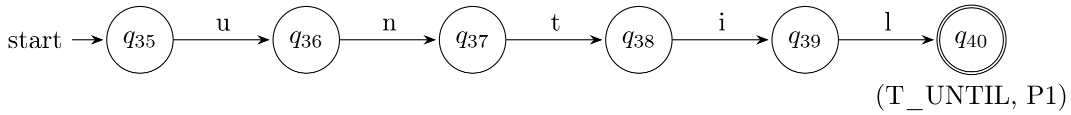


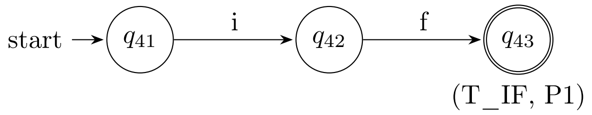

start
q_{10} s q_{11} t q_{12} r q_{13} i q_{14} n q_{15} g q_{16}
(T_STRING_KEY, P1)

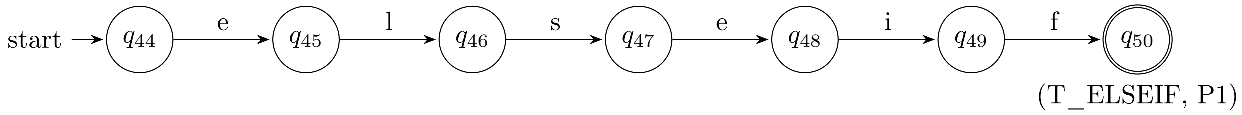


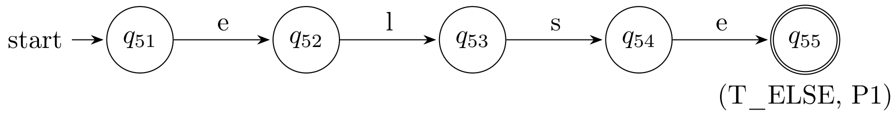

start
q_{22} w q_{23} r q_{24} i q_{25} t q_{26} e q_{27}
(T_WRITE, P1)

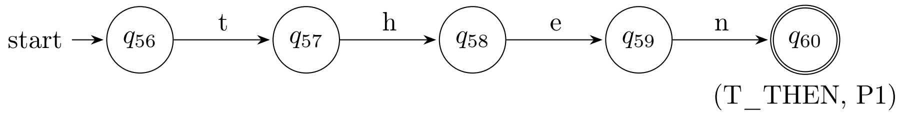

start
q_{28} r q_{29} e q_{30} p q_{31} e q_{32} a q_{33} t q_{34}
(T_REPEAT, P1)

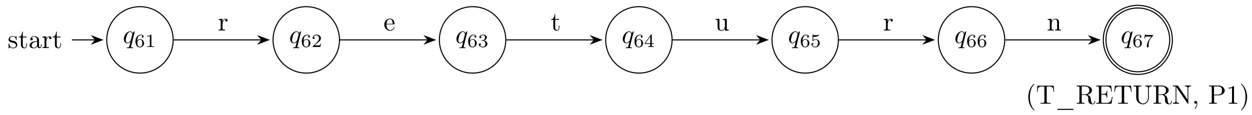

start
q_{35} u q_{36} n q_{37} t q_{38} i q_{39} l q_{40}
(T_UNTIL, P1)

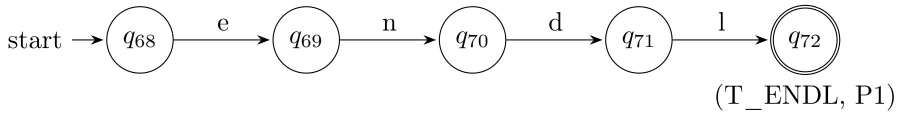


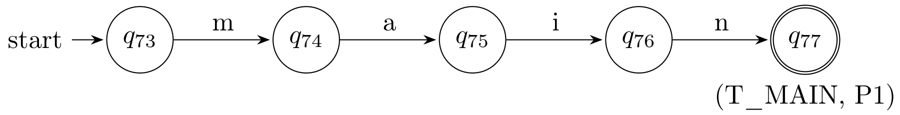

start
q_{44} e q_{45} l q_{46} s q_{47} e q_{48} i q_{49} f q_{50}
(T_ELSEIF, P1)



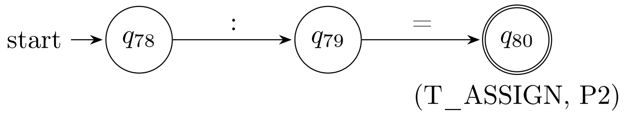


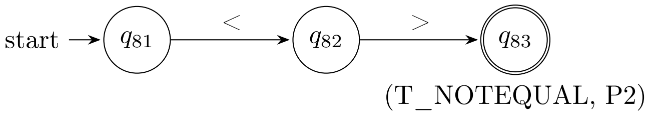

start
q_{61} r q_{62} e q_{63} t q_{64} u q_{65} r q_{66} n q_{67}
(T_RETURN, P1)

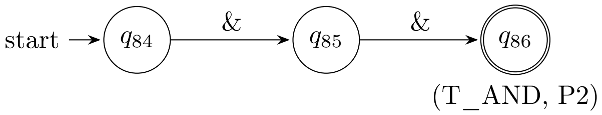


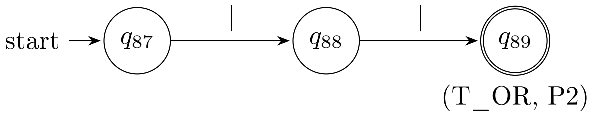


Sub-NFAs for Double Char Operators; Priority 2:

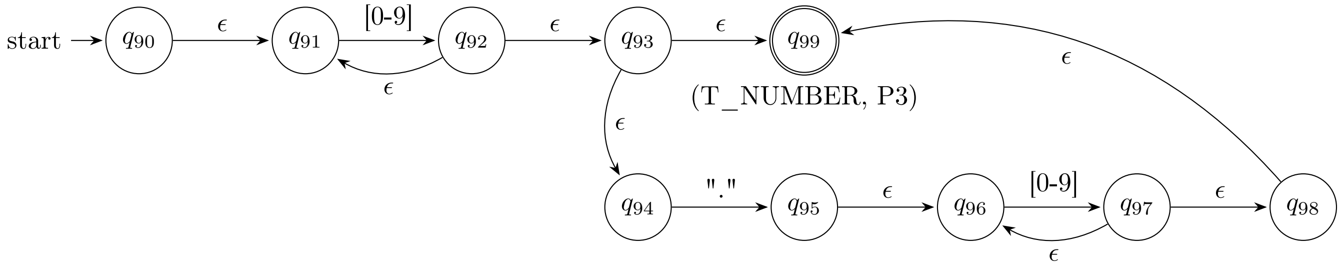


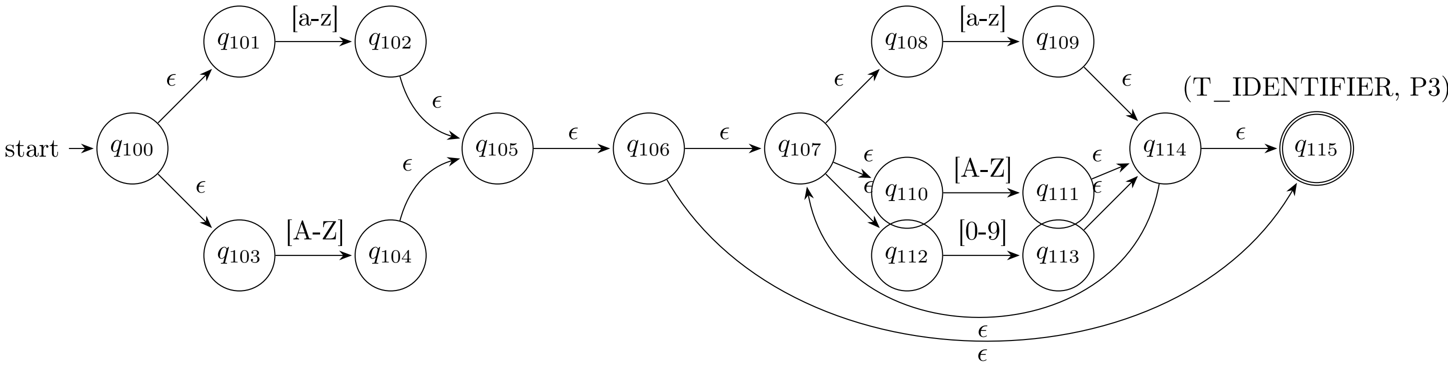


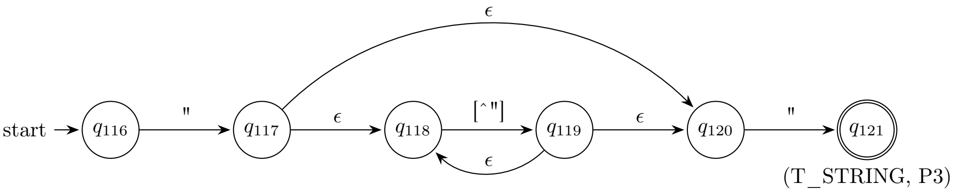




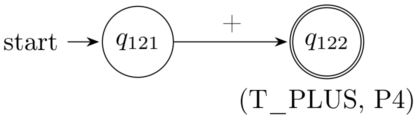
Sub-NFAs for Literals and Identifiers; Priority 3:

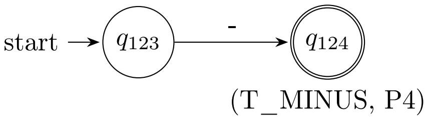


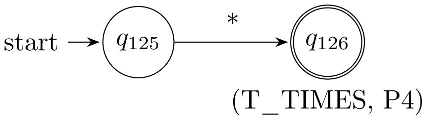


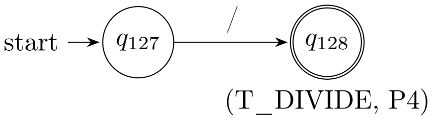


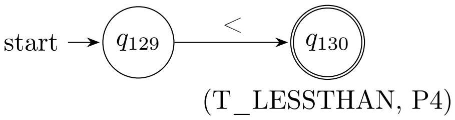
Sub-NFAs for Single Char Operators; Priority 4:

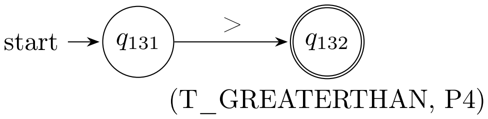


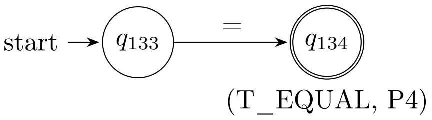


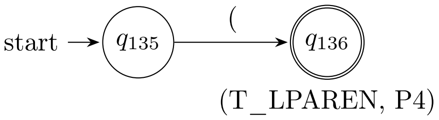


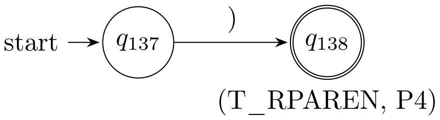


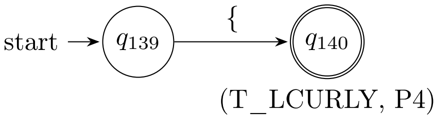


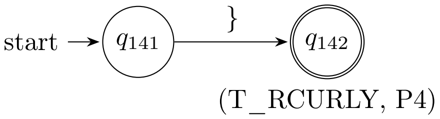


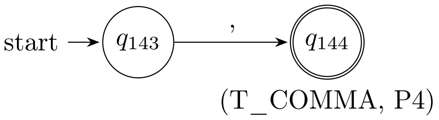


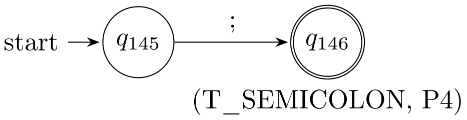




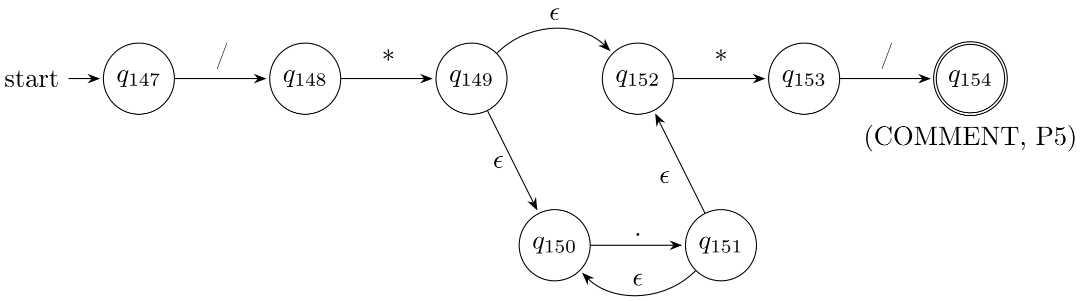


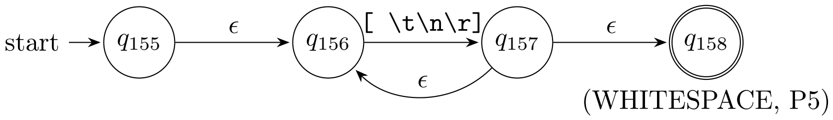


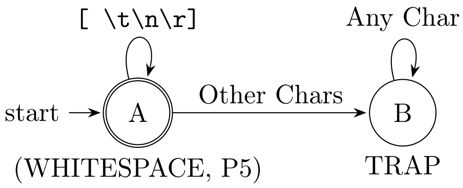




Ignored Tokens; Priority 5:

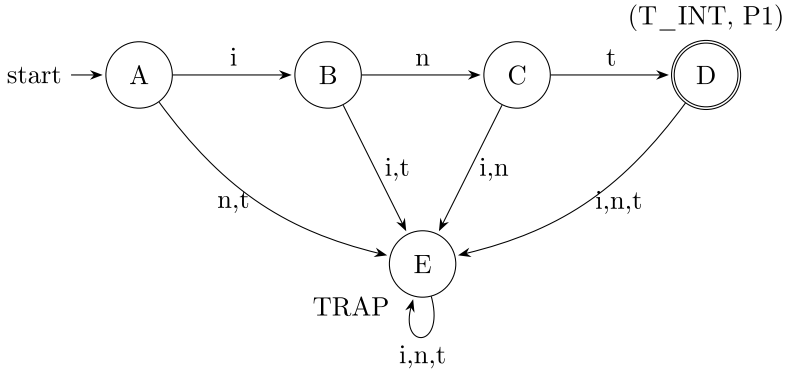


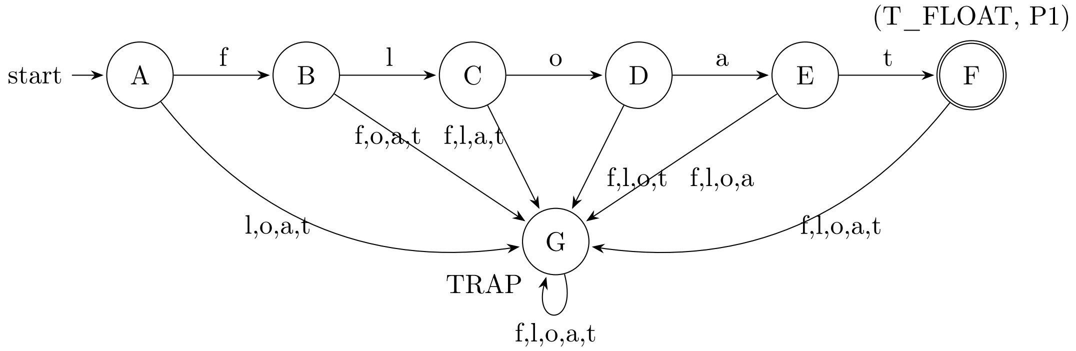


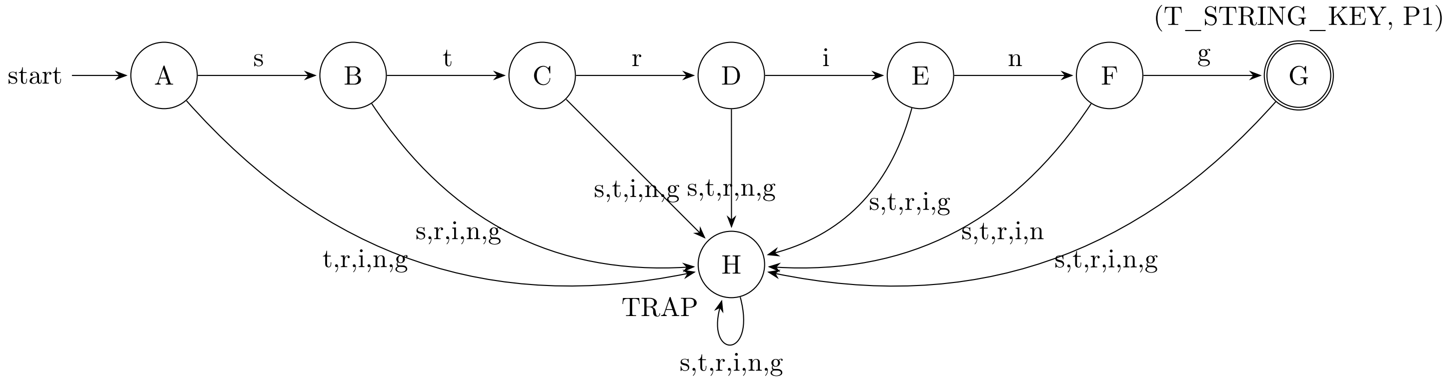


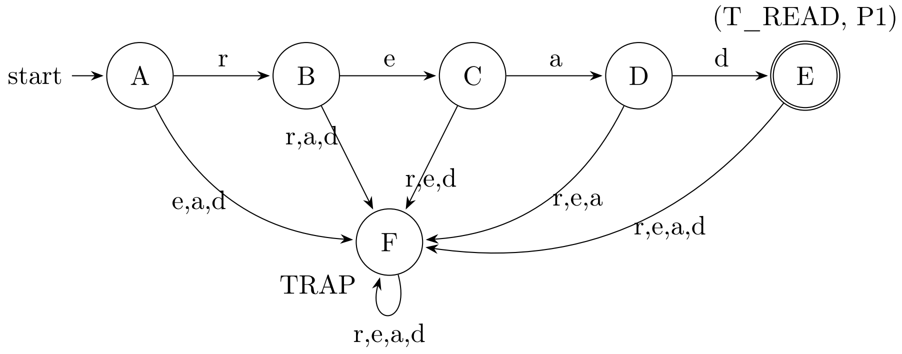
Constructed and Minimized DFAs

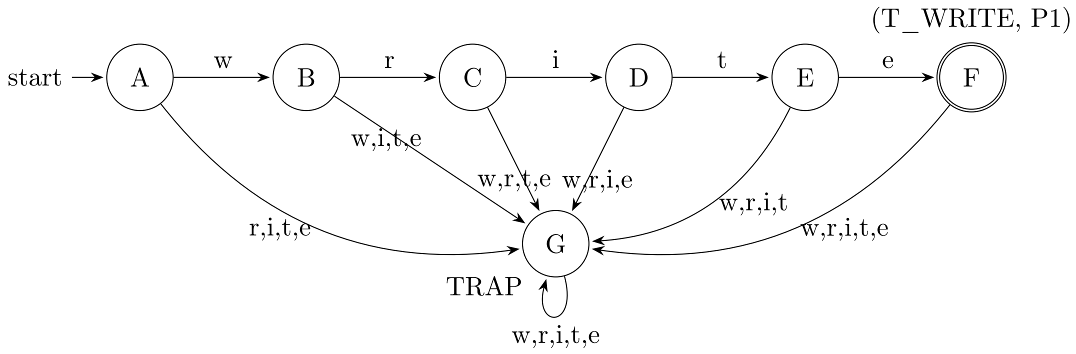
Keywords; Priority 1

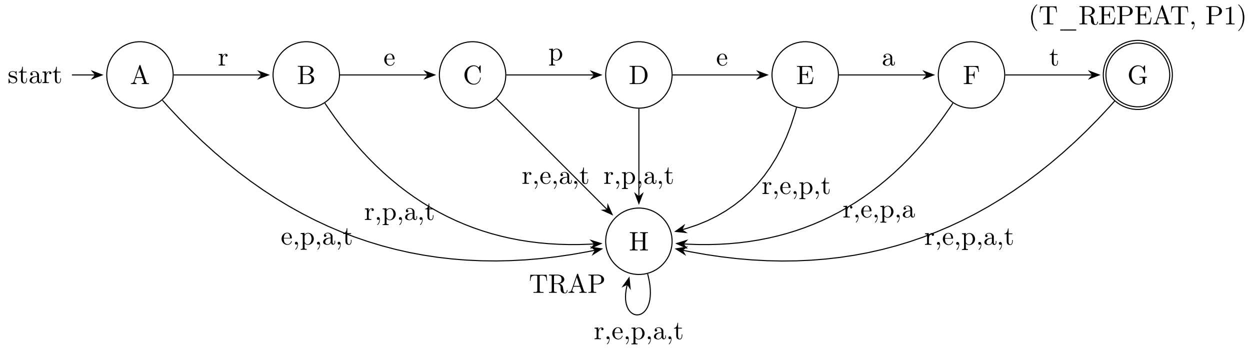


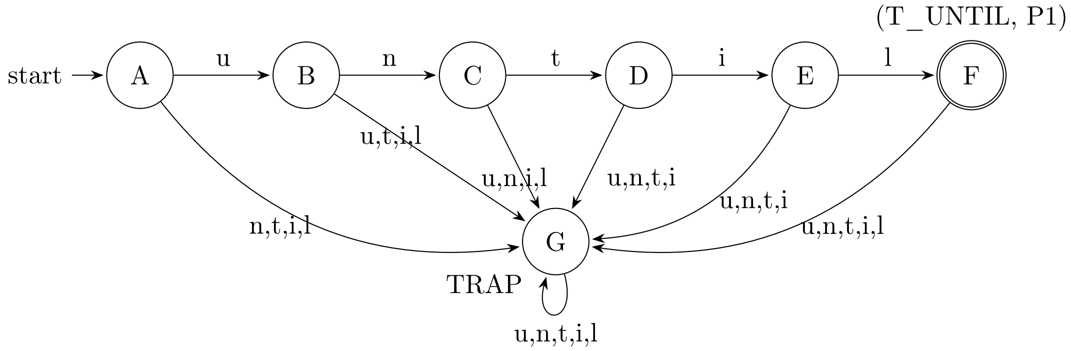


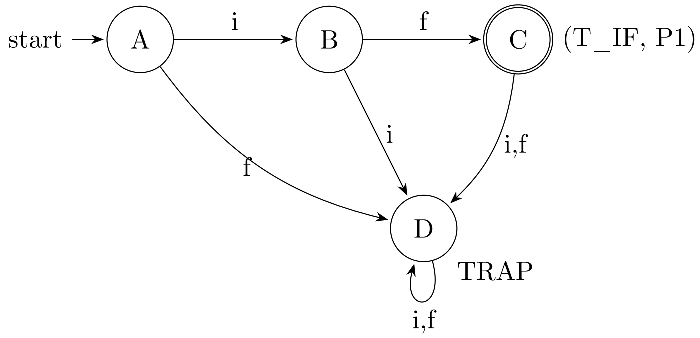


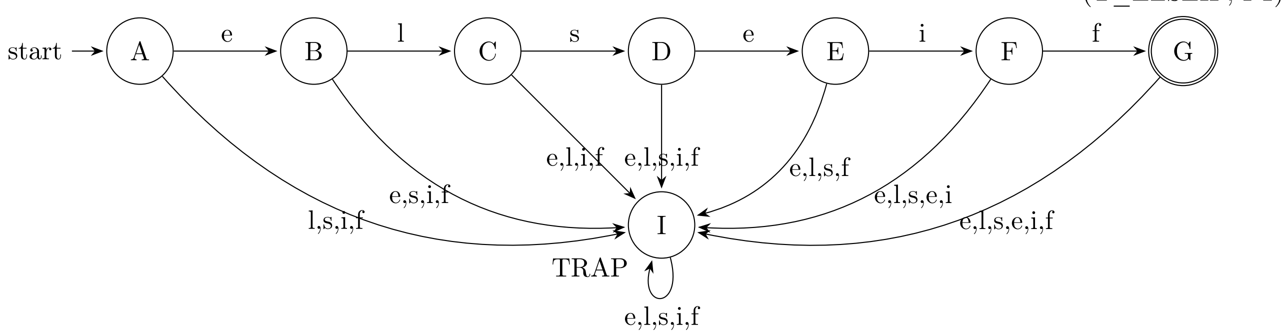


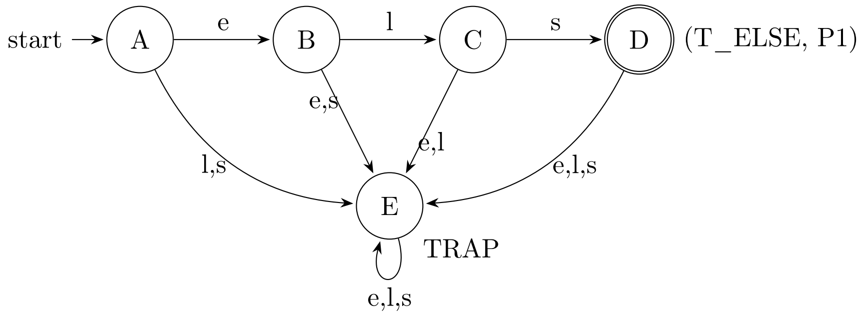

(T_WRITE, P1)
start
A
B
C
D
E
F
G
w
r
i
t
e
w,i,t,e
w,r,t,e
w,r,i,e
w,r,i,t
r,i,t,e
w,r,i,t,e
TRAP
w,r,i,t,e

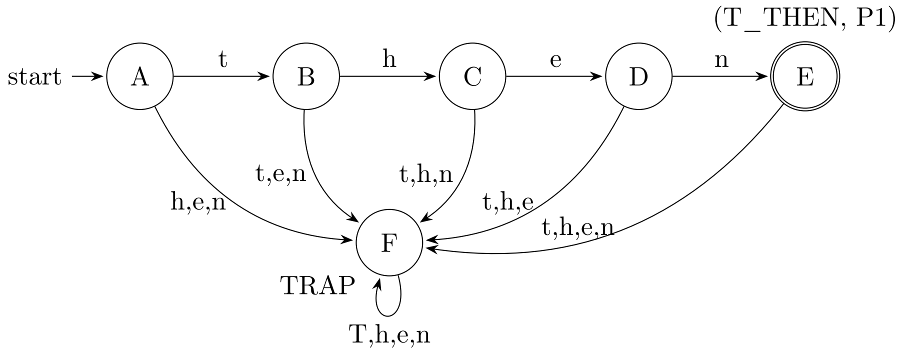

(T_REPEAT, P1)
start
A
B
C
D
E
F
G
H
r
e
p
e
a
t
r,e,a,t
r,p,a,t
r,e,p,t
r,p,a,t
r,e,p,a
e,p,a,t
r,e,p,a,t
TRAP
r,e,p,a,t

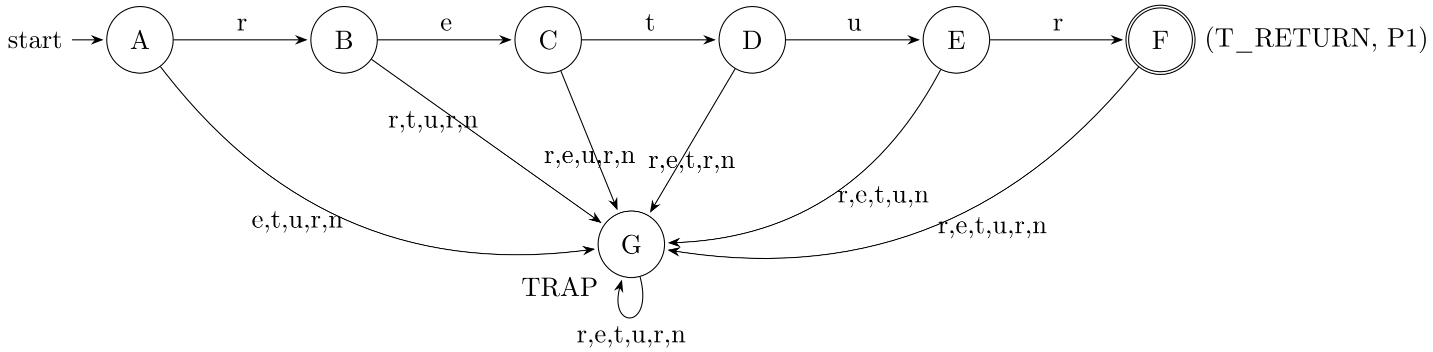

(T_UNTIL, P1)
start
A
B
C
D
E
F
G
u
n
t
i
l
u,t,i,l
u,n,i,l
u,n,t,i
u,n,t,i
n,t,i,l
u,n,t,i,l
TRAP
u,n,t,i,l

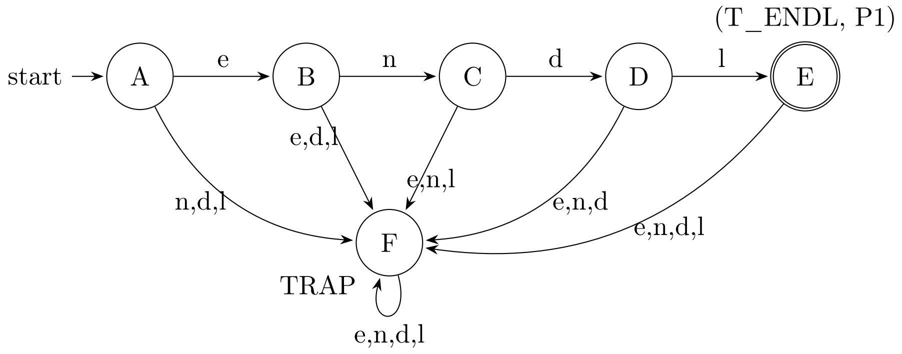

start
A
B
C
(T_IF, P1)
i
f
i
i,f
f
D
TRAP
i,f

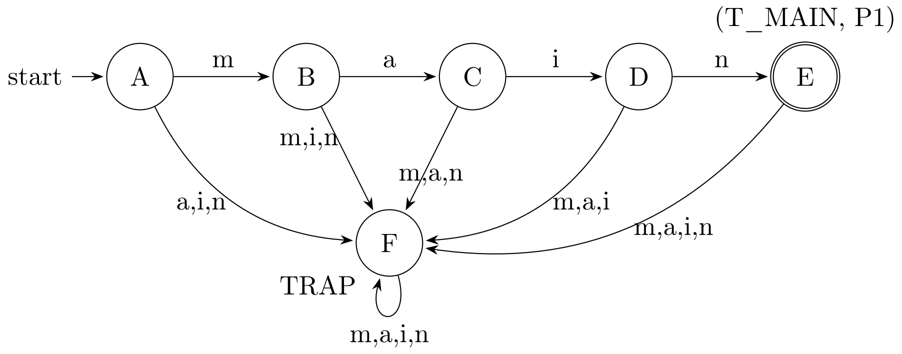

(T_ELSEIF, P1)
start
A
B
C
D
E
F
G
e
l
s
e
i
f
e,l,i,f
e,l,s,i,f
e,l,s,f
e,s,i,f
e,l,s,e,i
l,s,i,f
e,l,s,e,i,f
I
TRAP
e,l,s,i,f


start
A
B
C
D
E
e
l
s
(T_ELSE, P1)
e,s
e,l
l,s
e,l,s
TRAP
e,l,s

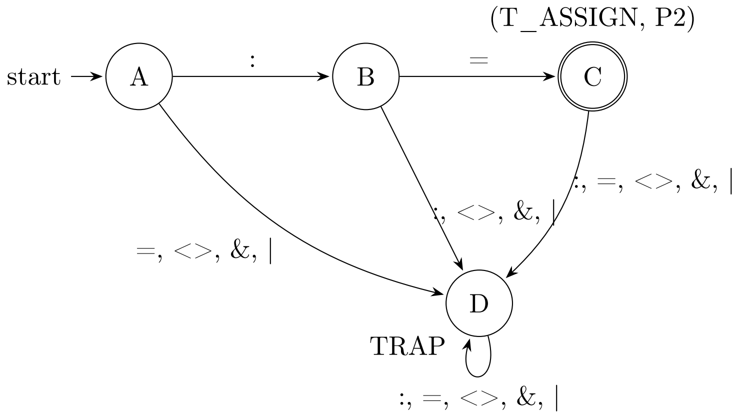

(T_THEN, P1)
start
A
B
C
D
E
F
t
h
e
n
t,e,n
t,h,n
h,e,n
t,h,e
t,h,e,n
TRAP
T,h,e,n

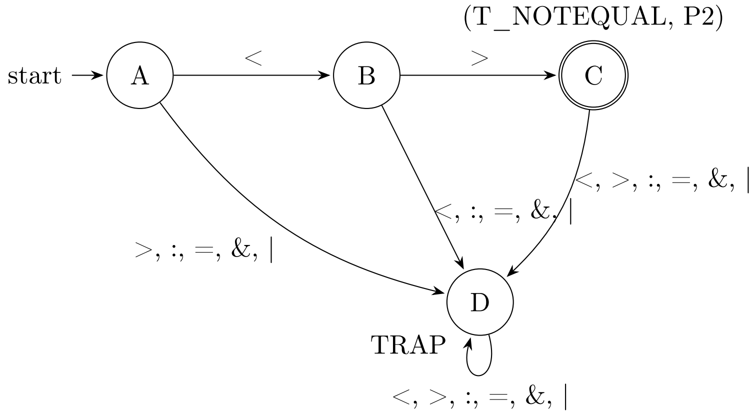

start
A
B
C
D
E
F
G
r
e
t
u
r
(T_RETURN, P1)
r,t,u,r,n
r,e,u,r,n
r,e,t,r,n
r,e,t,u,n
e,t,u,r,n
r,e,t,u,r,n
TRAP
r,e,t,u,r,n

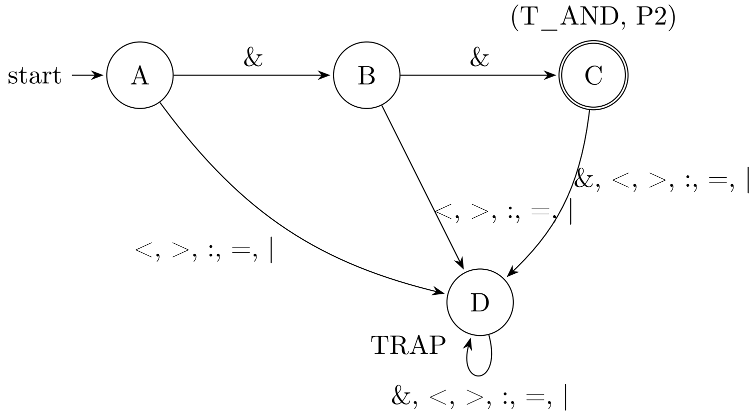

(T_ENDL, P1)
start
A
B
C
D
E
F
e
n
d
l
e,d,l
e,n,l
n,d,l
e,n,d
e,n,d,l
TRAP
e,n,d,l

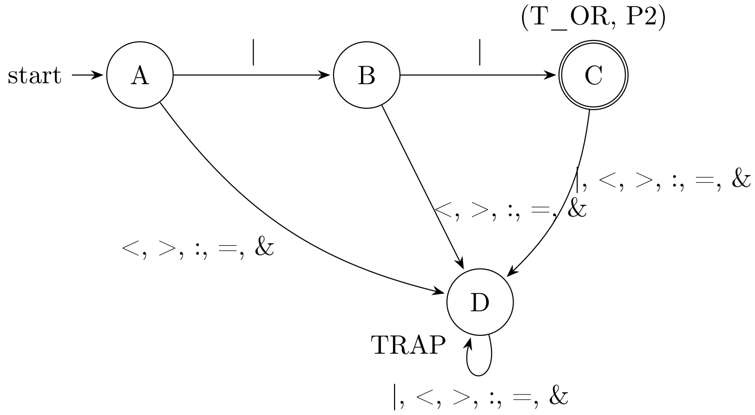

(T_MAIN, P1)
start
A
B
C
D
E
F
m
a
i
n
m,i,n
m,a,n
a,i,n
m,a,i
m,a,i,n
TRAP
m,a,i,n

Double Char Operators;Priority 2



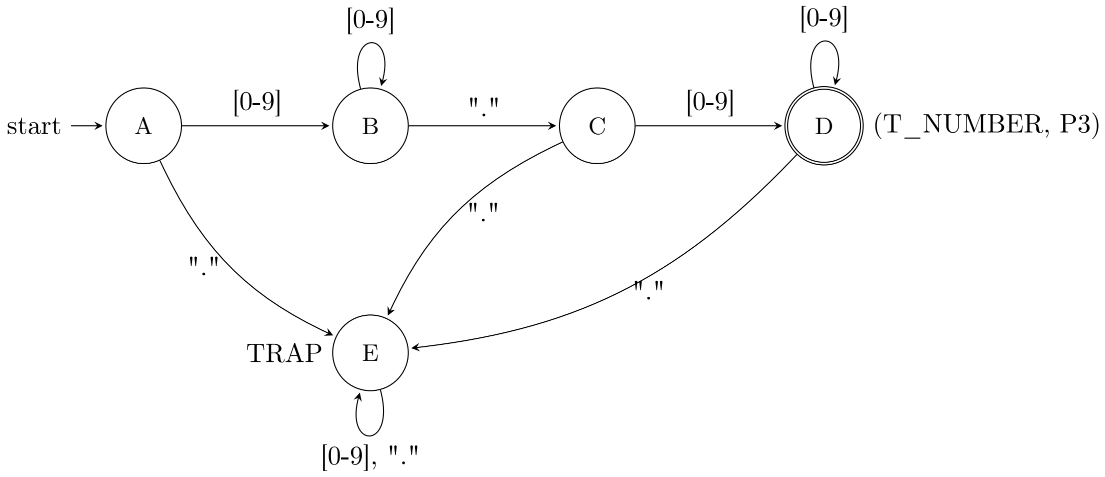


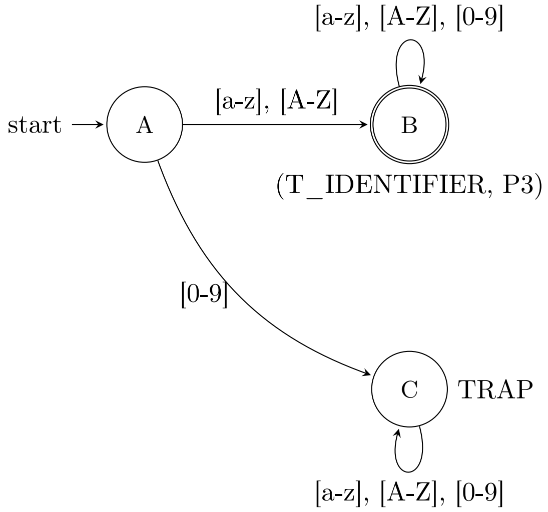




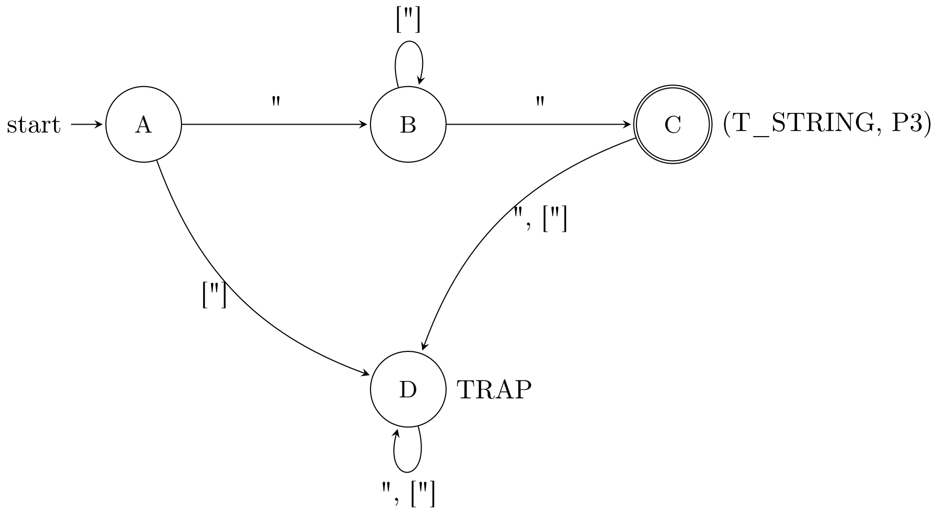
en

Literals And Identifiers ; Priority 3

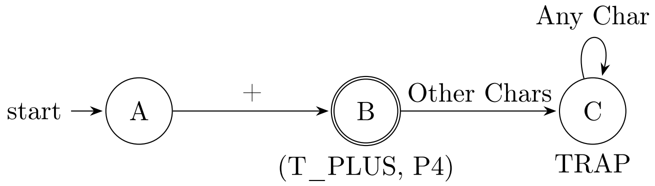


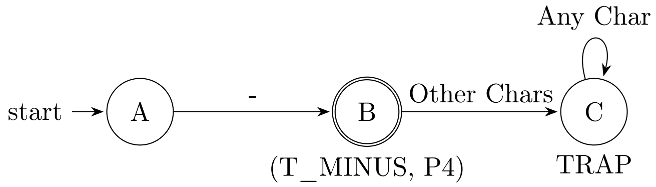


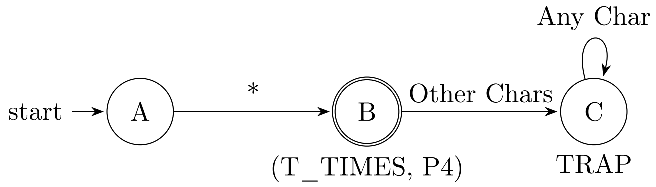
String Recognition; Priority 3

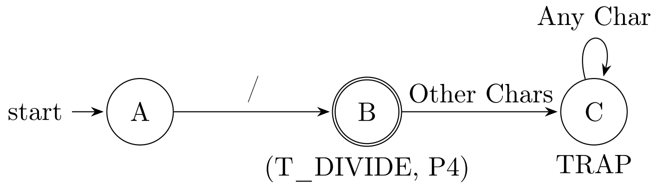


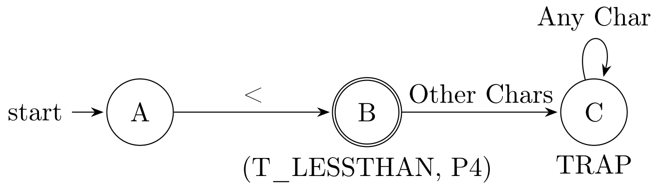
Single Char Operators ;Priority 4

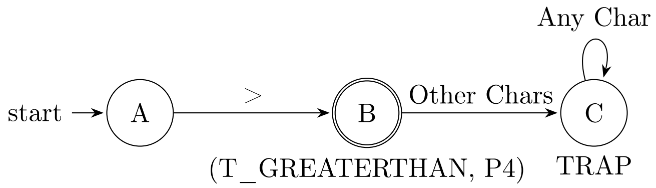


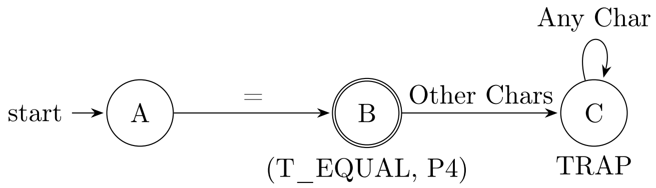


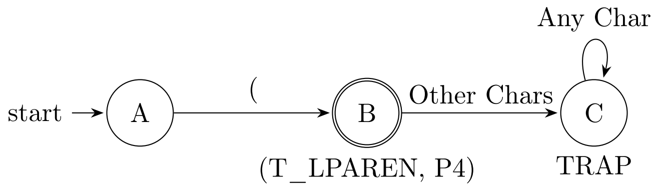


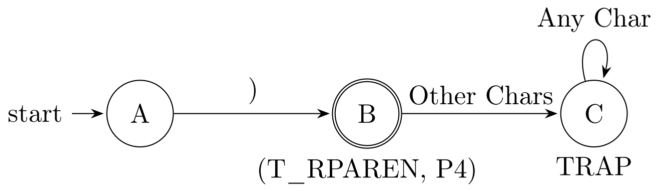


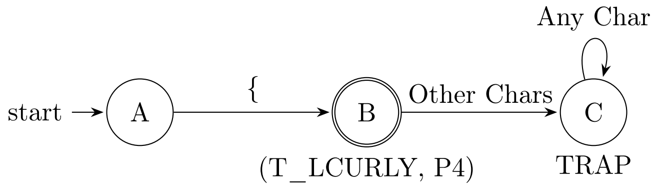


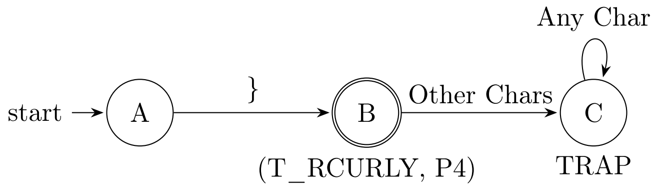


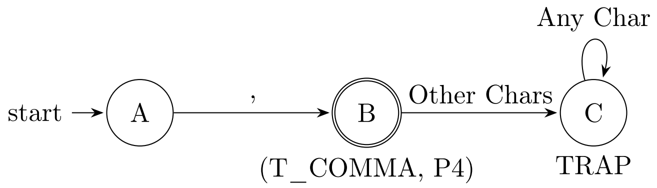


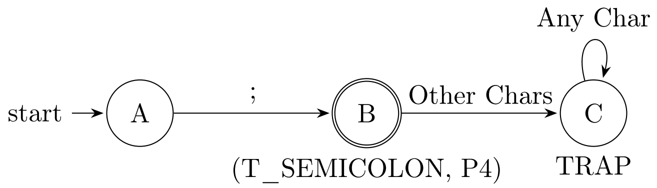












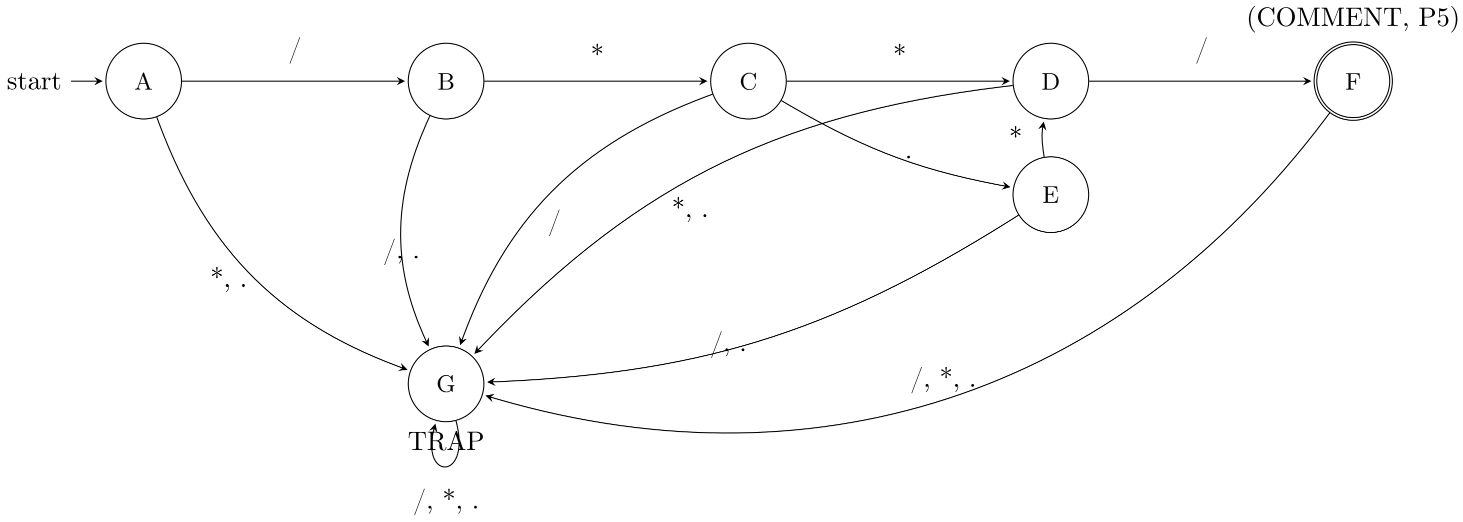
[1]

[1] Two labels on the transition state, namely "Any Char" and "Other char" are not formal REs in TinyProgLang but are shorthand conventions used to satisfy requirement of the DFAs:

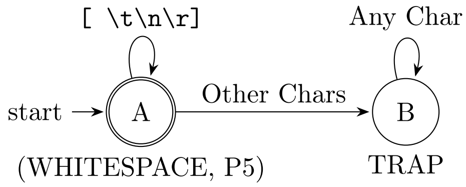
† **Any Char:** Shorthand for Σ.

‡ **Other Chars:** Shorthand for Σ minus the single successful symbol expected from input (e.g., $\Sigma \setminus \{+\}$ at start state of `T_PLUS`).

Ignored Tokens; Priority 5



icture


[\t\n\r]
Any Char
start
A
Other Chars
B
(WHITESPACE, P5)
TRAP