System Programming and Compiler Construction

MODULE 5 (3)

COMPILERS : ANALYSIS PHASE

Prof. Sonal Shroff
Computer Engineering Department
TSEC

Specification of tokens

Alphabet

•Any finite set of symbols – Letters, digits and punctuation

•{0,1} – binary alphabet

String

String over an alphabet is a finite sequence of symbols drawn from that alphabet.

•“Compiler” is a string of length eight. ($|s| = 8$)

•The empty string, denoted ε, is the string of length zero

Specification of tokens

Terms for parts of String

• A prefix of string s – any string obtained by removing zero or more symbols from the end of s. ex. ban, banana, ε are prefixes of banana.

•A suffix of string s – any string obtained by removing zero or more symbols from the beginning of s. ex. ana, banana, ε are sufixes of banana

•A substring of s – any string obtained by deleting any prefix and any suffix from s. ex. nan, banana, ε are substrings of banana

➢ Specification of tokens

Terms for parts of String

•The proper prefixes, suffixes and substrings of s are those, prefixes, suffixes and substrings, respectively, of s that are not ε or not equal to s itself.

•A subsequence of s – any string formed by deleting zero or more not necessarily consecutive positions of s. ex. baan is a subsequence of banana.

➢ Specification of tokens

Language

It is any countable set of strings over some fixed alphabet.

Abstract languages like ø, the empty set, or {ε}, the set containing only the empty string.

The meaning to the string is not the requirement here.

Specification of tokens

Operations on Languages

In lexical analysis, the most important operations on languages are

Operation	Definition
Union	L U M = {s | s is in L or s is in M }
Concatenation	L . M = {st | s is in L and t is in M }
Kleene closure of L	$L^*=L^0 \cup L^1 \cup L^2 \ldots$
Positive Closure of L	$L^+ = L^1 \cup L^2 \cup L^3 \ldots$

Specification of tokens

Operations on Languages

Let L be the set of letters {A, B,.....Z,a,b,.....z}and D be the set of digits{0,1,....9}
L and D are the alphabets upper and lower case letters and of digits.
OR L and D are languages, all of whose strings are of length one.

Possible Operations:

1. L U D is the set of letters and digits
2. LD is the set of string consisting of one letter followed by one digit
3. L^4 is the set of all four letter string
4. L* is the set of all strings of letters including empty string ε
5. L(L U D)* is the set of all strings of letters and digits beginning with letter
6. D+ is the set of all strings of one or more digits

Regular Expressions

Let ∑ = { a, b }

- The regular expression a|b denotes the set {a, b}
- The regular expression (a|b)(a|b) denotes {aa, ab, ba, bb}. The set of all strings of a's and b's of length two
- The regular expression a* denotes set of all strings of zero or more a's.

  r = {ε, a, aa, aaa, aaaa, ... }
- If two regular expressions represents same language then we can say that they are equivalent

➢ Regular Expressions

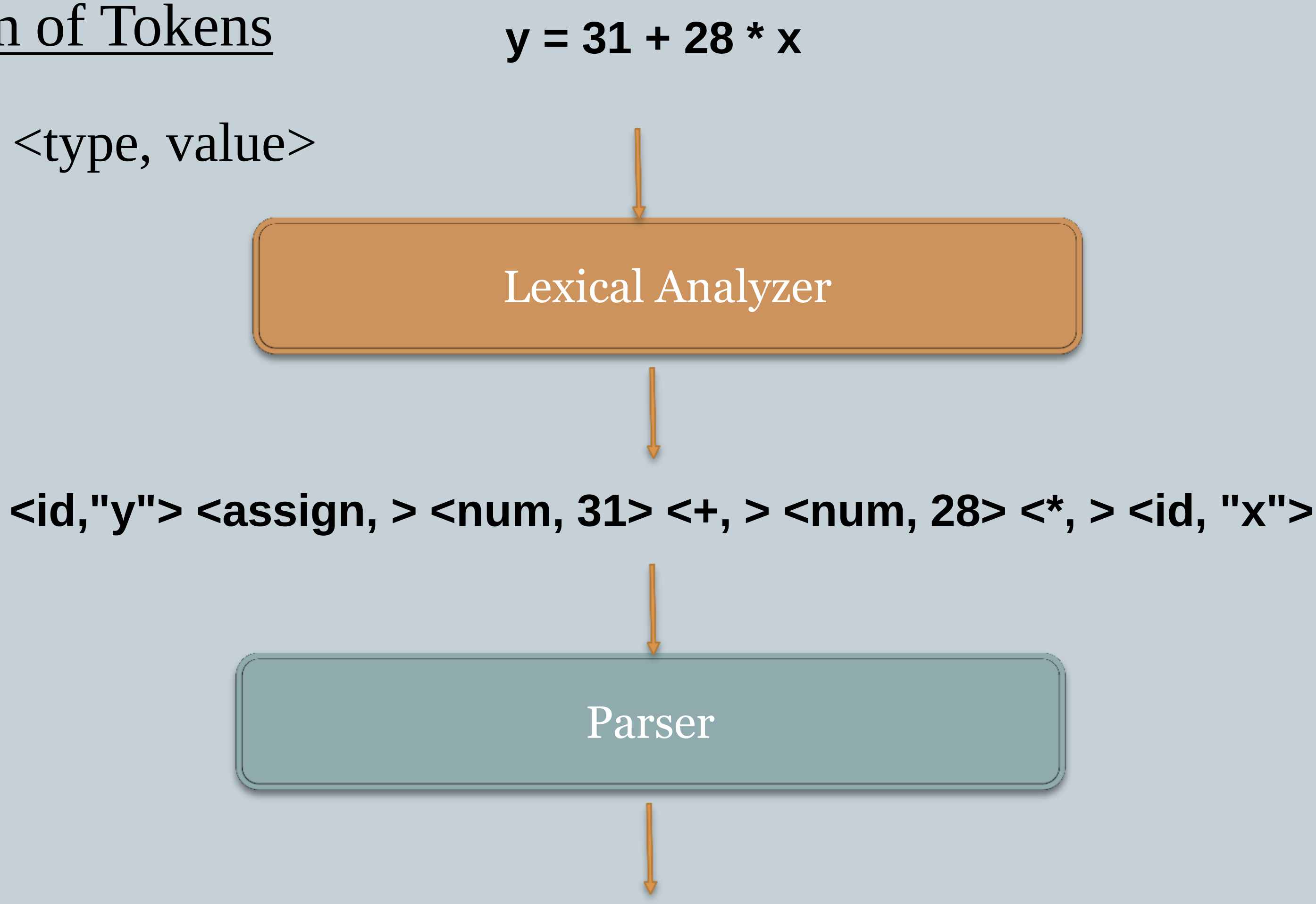
Regular expression are used to specify lexeme patterns.

- letter → A|B|……|Z|a|b|……|z
- letter_ → [A-Za-z_]
- digit → 0|1|2|……|9
- digit → [0-9]
- id → letter_ (letter_ | digit)*
- digits → digit* (.digits)? (E[+-]? digits)?
- op → < | > | <= | >= | = | <>
- ws → (blank|tab|newline)$^+$

Lexical Analysis

- <u>Recognition of Tokens</u>

Token is a pair <type, value>



Recognition of Tokens

A Grammar for branching statement

stmt **->** **if** expr **then** stmt

| if expr **then** stmt **else** stmt

| ε

expr **->** term **relop** term

| term

term **->** **id**

| **number**

Lexical Analysis

- Pattern of Tokens

Token	Pattern
digit	[0–9]
digits	digit+
number	digits (.digits) ? (E [+-] ? digits)?
letter	[A-Za-z]
id	letter (letter | digit)*
if	if
then	then
else	else
relop	< | <= | > | >= | = | <>

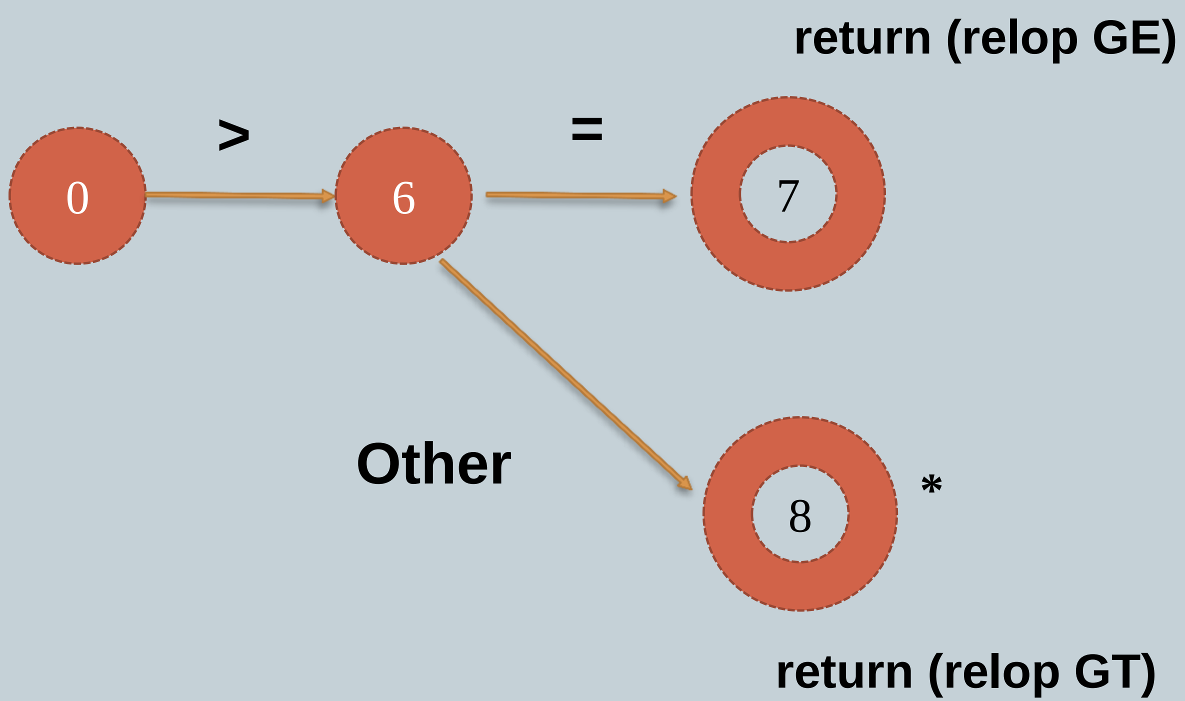
Lexical Analysis

Lexemes	Token Name	Attribute Value
Any ws	--	--
if	if	--
else	else	--
then	then	--
id	id	Pointer to table entry
num	num	Pointer to table entry
<	relop	LT
<=	relop	LE
=	relop	EQ
<>	relop	NE
>	relop	GT
>=	relop	GE

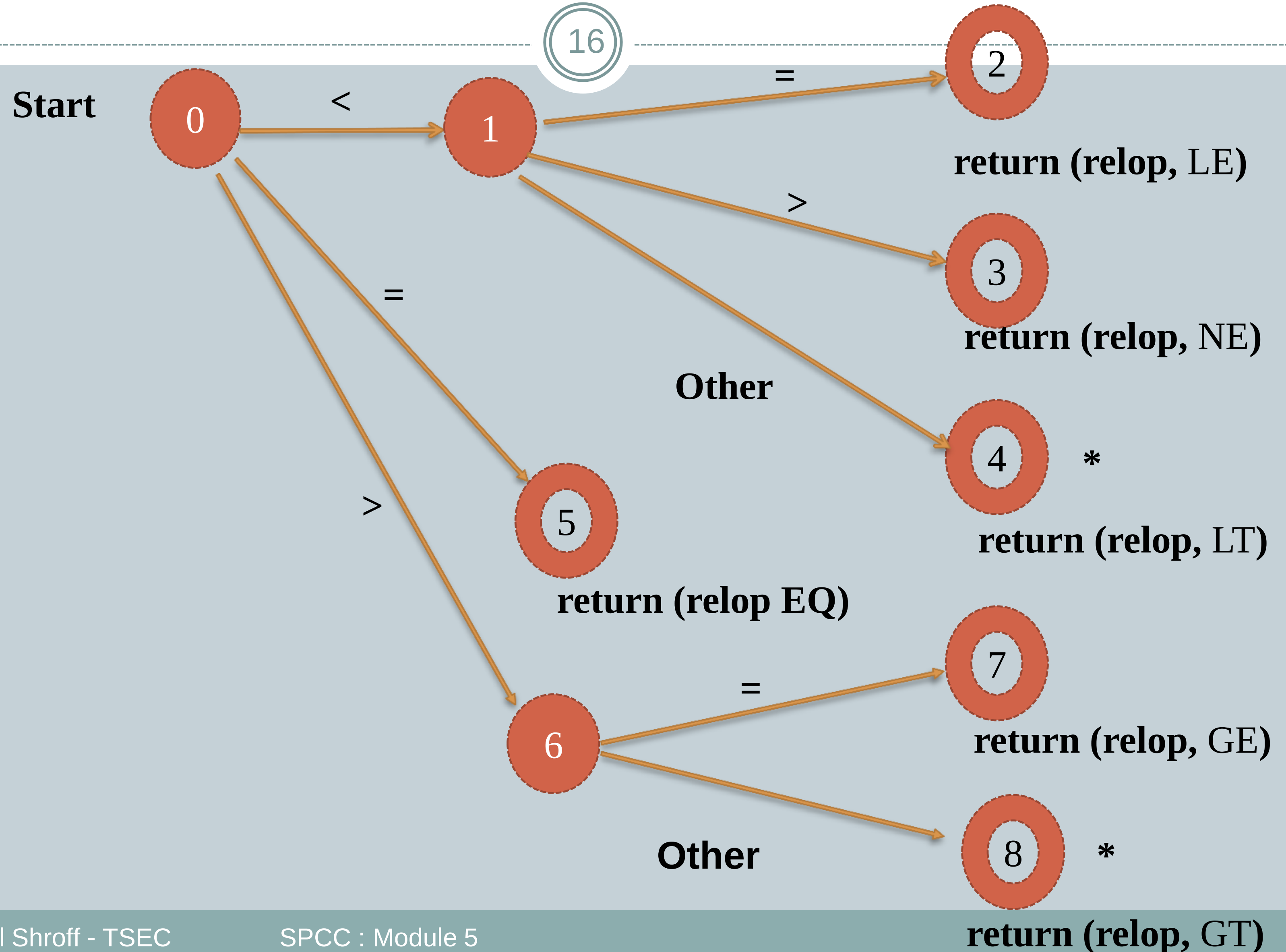
Transition Diagrams

- We convert patterns into stylized flowcharts, called "transition diagrams"
- Transition diagrams have a collection of nodes or circles, called states
- Each state represents a condition that could occur during the process of scanning the input looking for a lexeme that matches one of several patterns
- Edges are directed from one state of the transition diagram to another
- Each edge is labeled by a symbol or set of symbols

Transition Diagram for >=

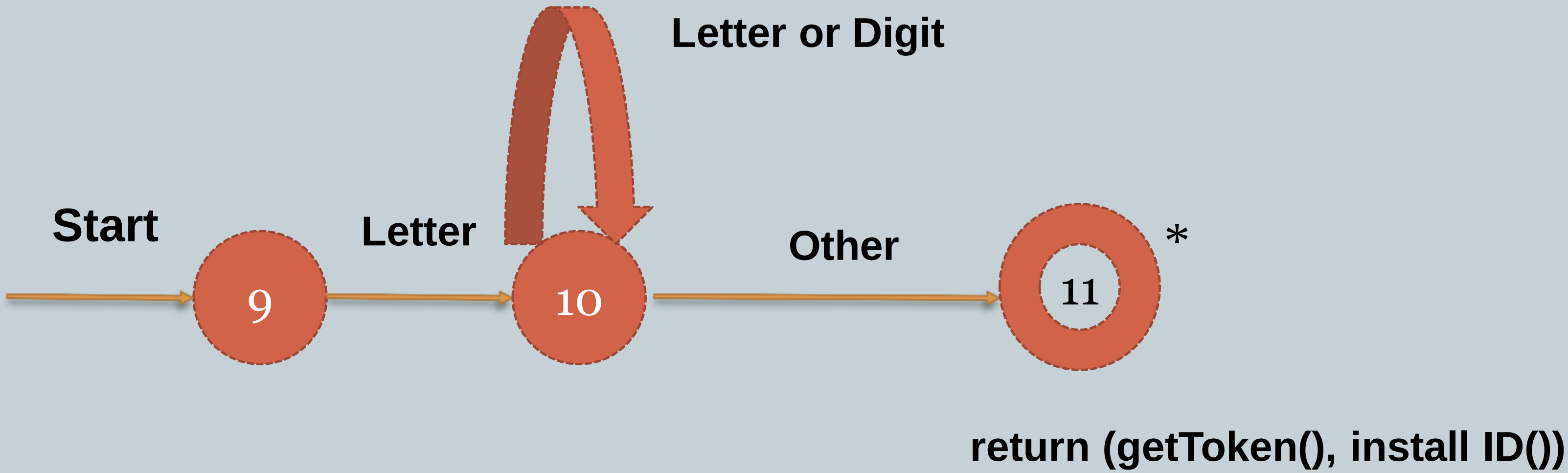


Transition Diagram for **relop**

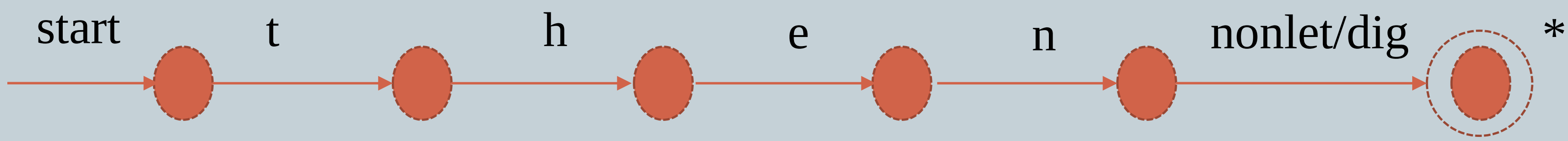


Lexical Analysis

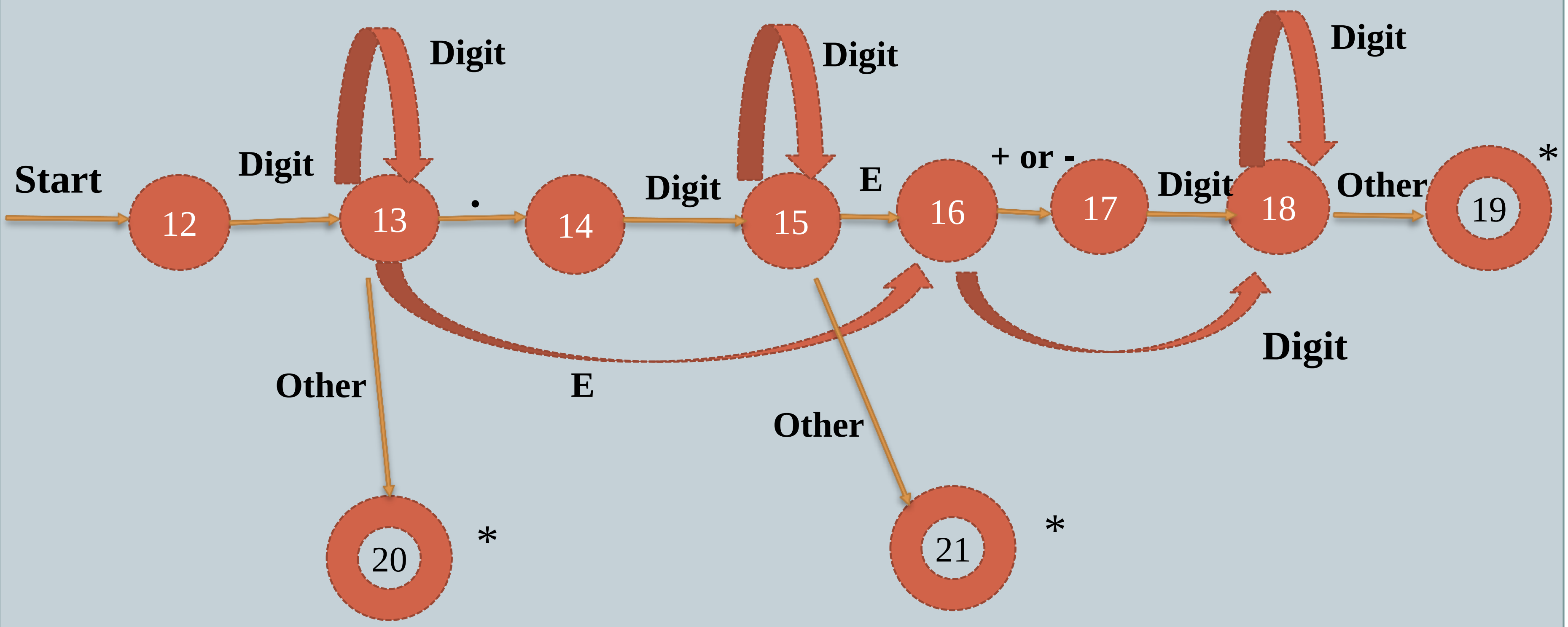
- **Transition Diagram for identifiers and keywords**



- **Hypothetical Transition Diagram for keyword ‘then’**



- **Transition Diagram for unsigned number**



- **Transition Diagram for White Spaces**