Compiler Design

Fatemeh Deldar

Isfahan University of Technology

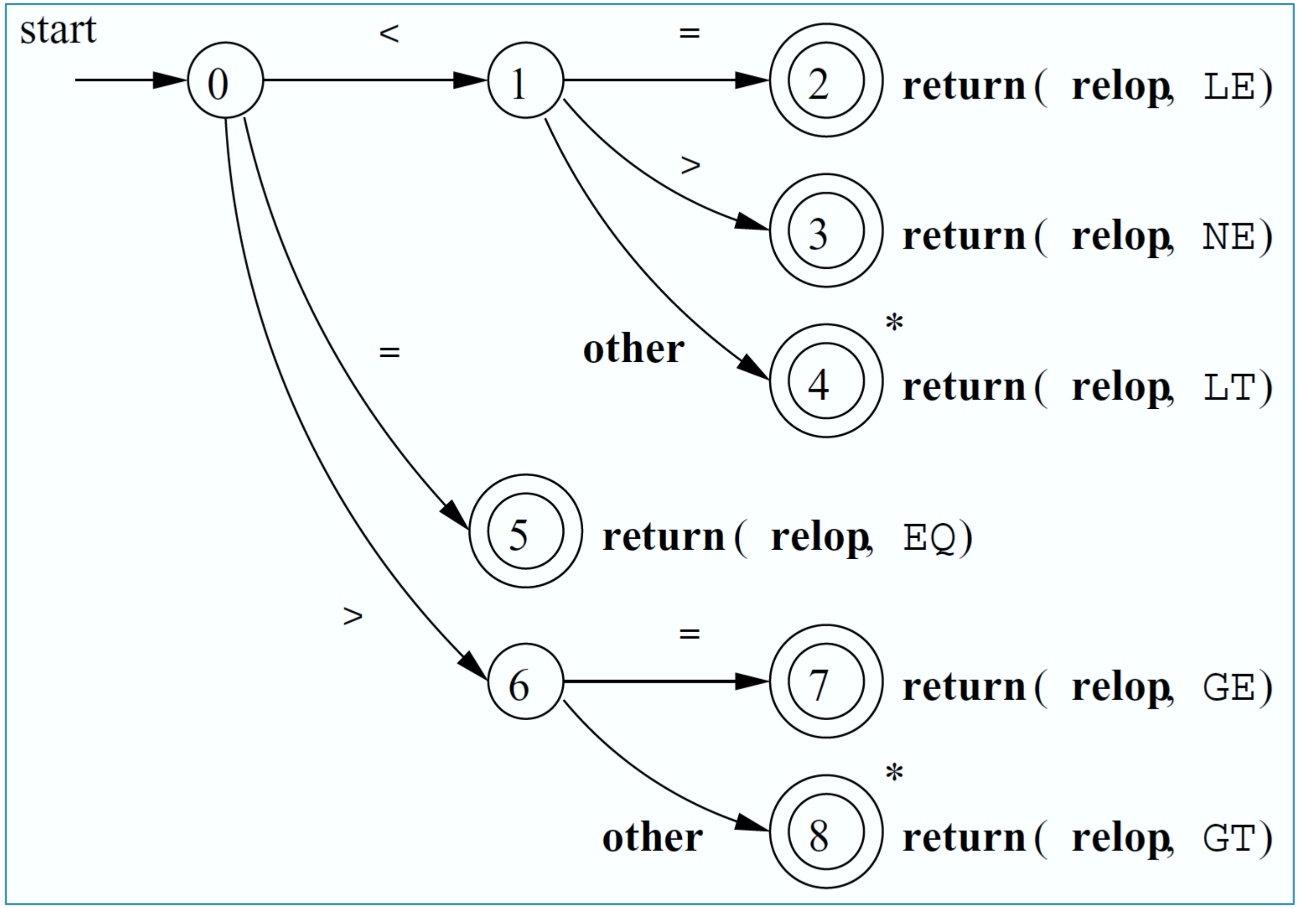
1402-1403

Transition Diagrams

- Transition diagrams have a collection of **nodes** or circles, called **states**
- **Edges** are directed from one state of the transition diagram to another
  - **Accepting or final states** indicate that a lexeme has been found
  - If it is necessary to retract the forward pointer one position, a ***** is placed near that accepting state
  - One state is designated the start state, or **initial state**

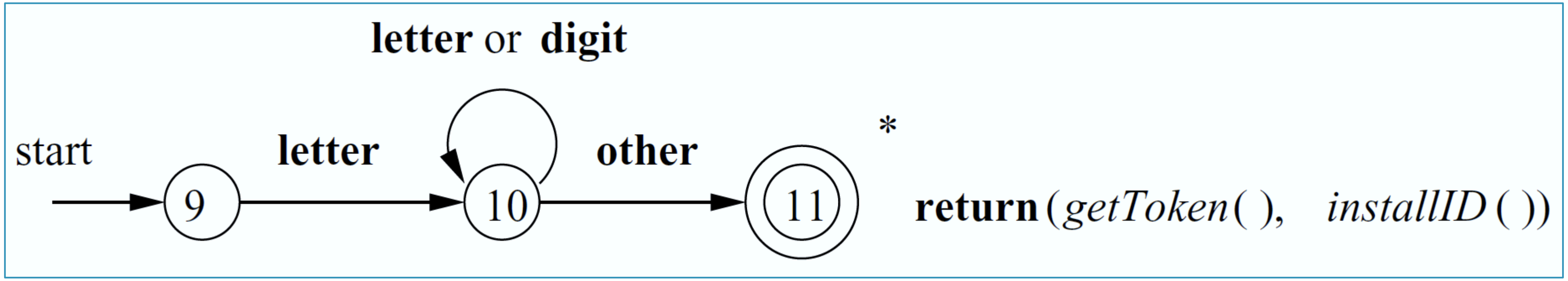
Transition Diagrams

- **Example:** A transition diagram that recognizes the lexemes matching the token relop



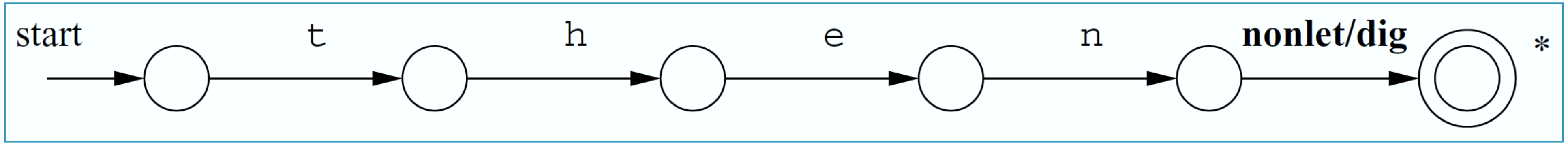
Transition Diagrams

- **Example:** Recognition of Reserved Words and Identifiers



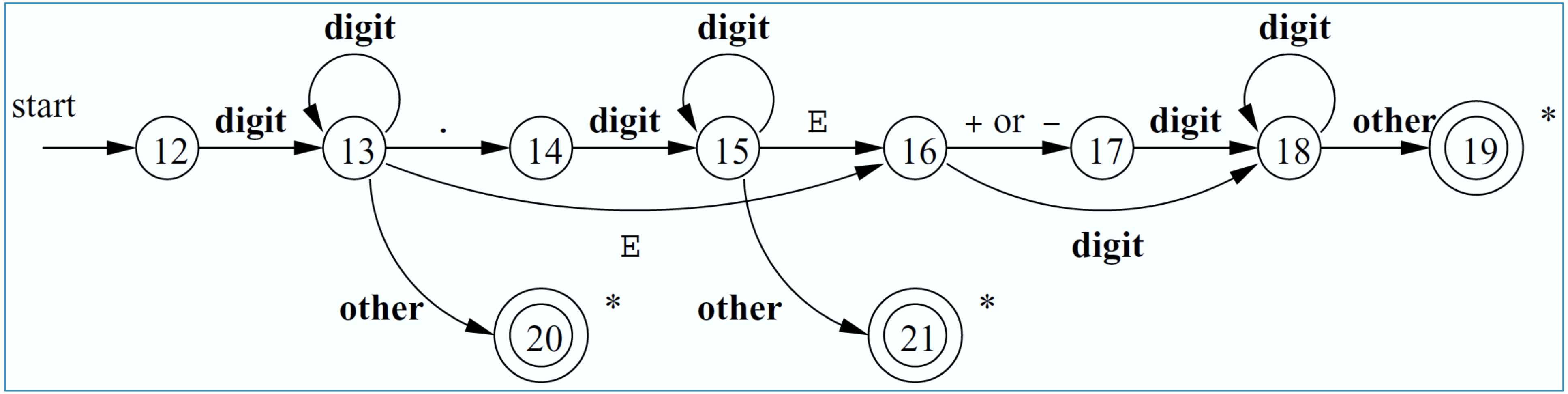
Transition Diagrams

- There are two ways that we can handle reserved words that look like identifiers:
  1. ***Install the reserved words in the symbol table initially***
     - When an identifier is found, a call to **installID** places it in the symbol table if it is not already there
  2. ***Create separate transition diagrams for each keyword***
     - In this method we must prioritize the tokens so that the reserved-word tokens are recognized in preference to id

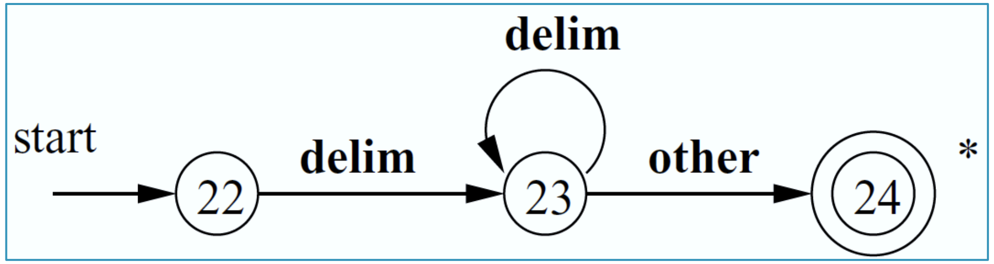


Transition Diagrams

- **Example:** The transition diagram for token number



- **Example:** The transition diagram for whitespace



Transition Diagrams

- **Sketch of implementation of relop transition diagram**
- **Architecture of a Transition-Diagram-Based Lexical Analyzer**
  - Combining all the transition diagrams into one (Combining states 0, 9, 12, and 22 into one start state)

```
TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character processing until a return
                  or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
```

Finite Automata

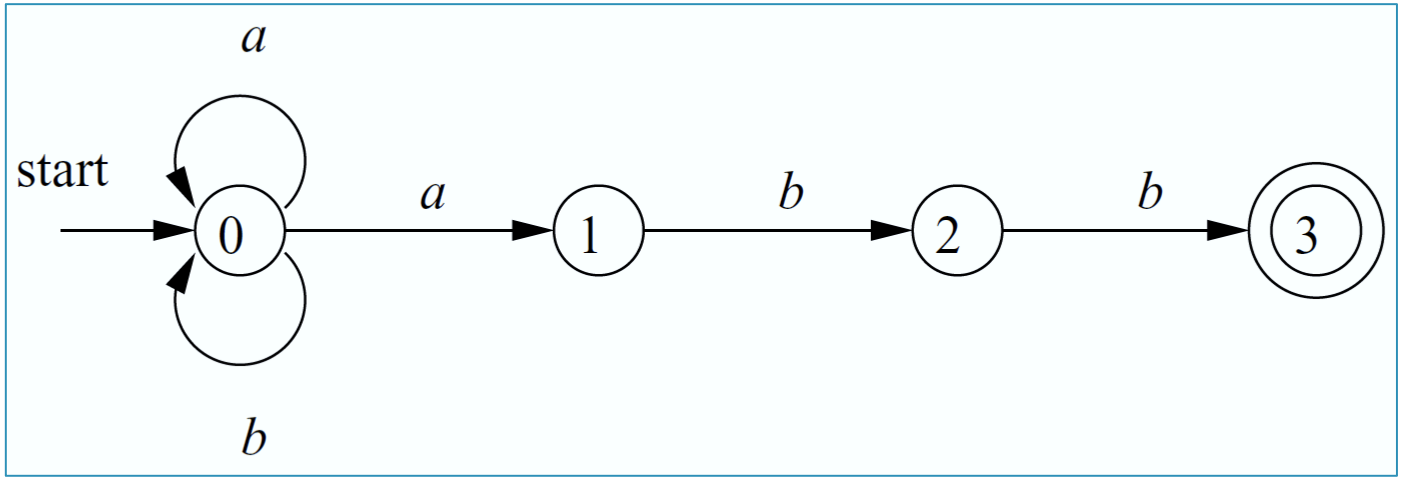
- Finite automata are recognizers
  - They simply say "yes" or "no" about each possible input string
- Finite automata come in two flavors
  - **Nondeterministic Finite Automata (NFA)**
    - It has no restrictions on the labels of their edges
    - A symbol can label several edges out of the same state, and ϵ is a possible label
  - **Deterministic Finite Automata (DFA)**
    - It has, for each state, and for each symbol of its input alphabet exactly one edge with that symbol leaving that state
- Both deterministic and nondeterministic finite automata are capable of recognizing the same languages, **called the regular languages**

Nondeterministic Finite Automata

- **A nondeterministic finite automaton (NFA) consists of:**
  1. A finite set of states S
  2. A set of input symbols $\sum$, the input alphabet
  3. A transition function that gives, for each state, and for each symbol in $\sum \cup \{\epsilon\}$ a set of next states
  4. A state s_0 from S that is distinguished as the start state (or initial state)
  5. A set of states F, a subset of S, that is distinguished as the accepting states (or final states)

Nondeterministic Finite Automata

- **Example**
  - The transition graph for an NFA recognizing the language of regular expression $(a|b)^*abb$



  - The string aabb is accepted by the above NFA

$$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$$

Nondeterministic Finite Automata

- **Transition Tables**
  - An NFA can also represented by a transition table, whose rows correspond to states and columns correspond to the input symbols and ϵ

STATE	a	b	ϵ
0	$\{0, 1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

Deterministic Finite Automata

- A deterministic finite automaton (DFA) is a special case of an NFA where:
    1. There are no moves on input ϵ
    2. For each state s and input symbol a, there is exactly one edge out of s labeled a
- Every regular expression and every NFA can be converted to a DFA accepting the same language

Simulating a DFA

$s = s_0;$
$c = nextChar();$
while (c != **eof**) {
  $s = move(s, c);$
  $c = nextChar();$
}
if (s is in F) **return** "yes";
else return "no";

Conversion of an NFA to a DFA

- The general idea
  - Each state of the constructed DFA corresponds to a set of NFA states
- Operations on NFA states

OPERATION	DESCRIPTION
$\epsilon\text{-}closure(s)$	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
$\epsilon\text{-}closure(T)$	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}\ \epsilon\text{-}closure(s)$.
$move(T, a)$	Set of NFA states to which there is a transition on input symbol a from some state s in T.