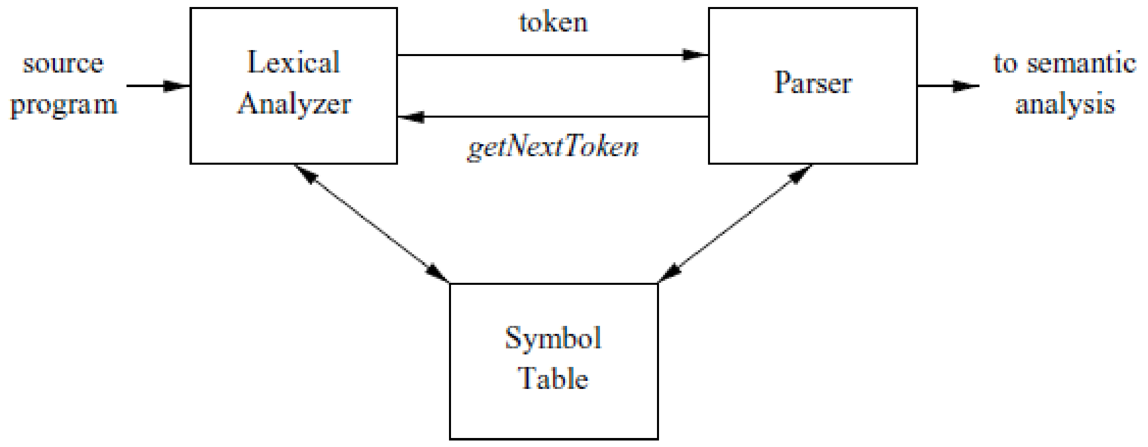
CS 420 - Compilers

Dr. Chen-Yeou (Charles) Yu

- Recognition of Tokens (Ch 3.4)
    - Transition Diagrams (Ch 3.4.1)
    - Recognition of Reserved Words and Identifiers (Ch 3.4.2)
    - Completion of the Running Example (3.4.3)
    - Architecture of a Transition-Diagram-Based Lexical Analyzer (3.4.4) --- TBD, in Part5

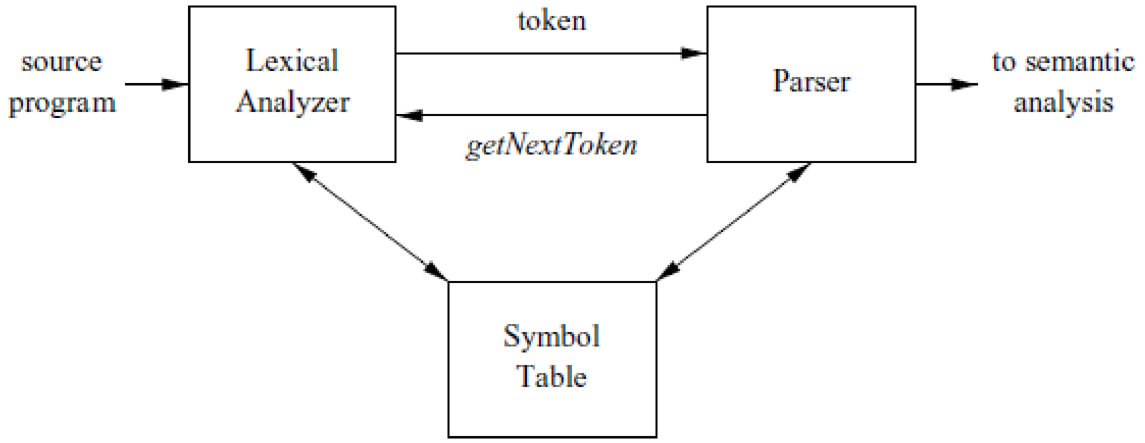
Transition Diagrams

- Remember we first convert patterns into stylized flowcharts, called transition diagrams, in the construction of a lexical analyzer.



Transition Diagrams

- Some important conventions about transition diagrams
  - (Remember we are still in the construction of a lexical analyzer (LA))
  - Certain states are said to be **accepting**, or **final**. We always indicate an accepting state by a double circle, and if there **is an action to be taken** --- typically returning a token and an **attribute value to the parser** --- we shall attach that action to the accepting state.



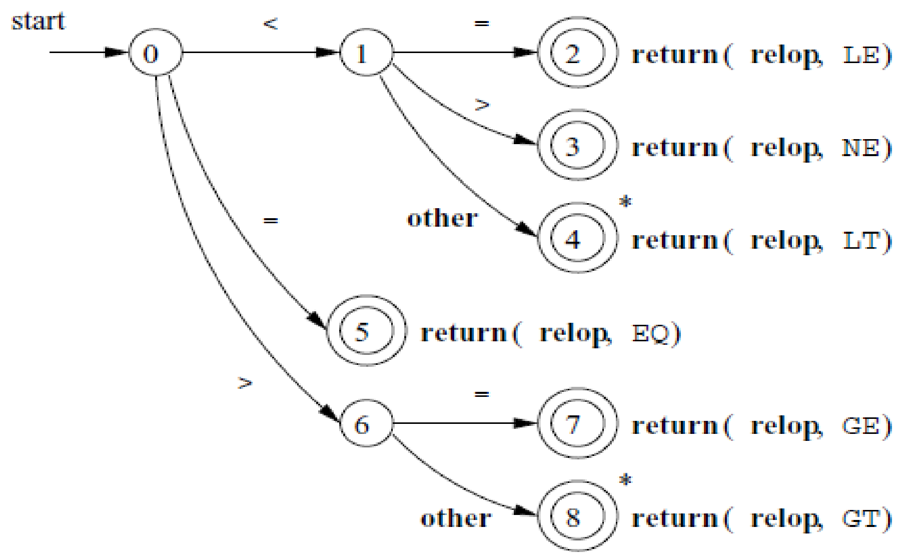


Figure 3.13: Transition diagram for **relop**

Transition Diagrams

- (Conventions Cont.)
- If it is necessary to **retract** the forward pointer one position. In other words, we have moved one or more characters "too far" in finding the token. So one or more stars are drawn.
- For example, the lexeme does not include the symbol (in a transition) that get us to the accepting state, then we shall additionally place a * near that accepting state.
- In this case, we need to adjust the input buffer so that you will read this character again since you have not used it for the current lexeme
- Finally, The transition diagram always begins in the **start** state before any input symbols have been read

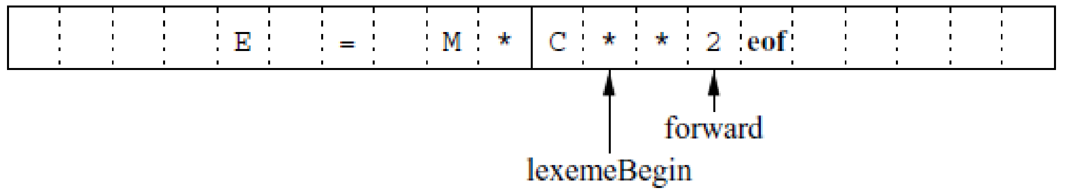


Figure 3.3: Using a pair of input buffers

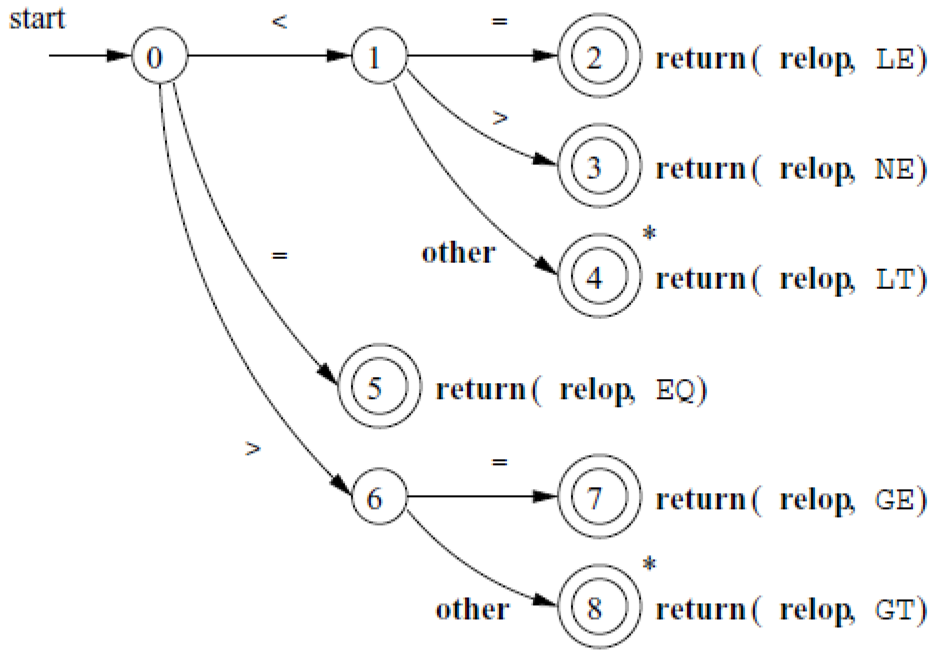


Figure 3.13: Transition diagram for **relop**

Recognition of Reserved Words and Identifiers

- Sometimes, recognizing keywords and identifiers (IDs)is giving us lots of trouble.
- For example, “if”, “else” are “reserved” and they are not IDs
- If we use the transition diagram like Fig. 3.14 to search for ID lexemes, the diagram will also recognize the keywords if, else, then, in our example.
- We need to have special

strategy because our original

plan doesn’t work!

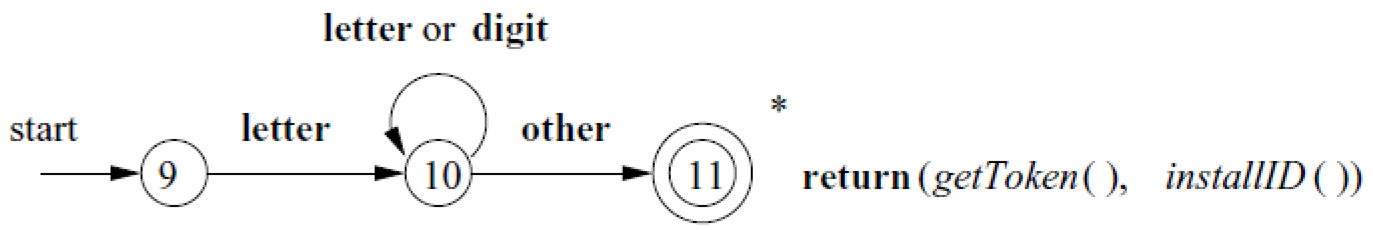


Figure 3.14: A transition diagram for **id**’s and keywords

Recognition of Reserved Words and Identifiers

- Explanation for Fig. 3.14
  - Starting in state 9, it checks that the lexeme begins with a letter and goes to state 10. If so, we stay in state 10 as long as the input contains letters and digits. When we first encounter anything but a letter or digit, we go to state 11 and accept the lexeme found.
  - Since the last character is not part of the identifier, we must retract the input one position

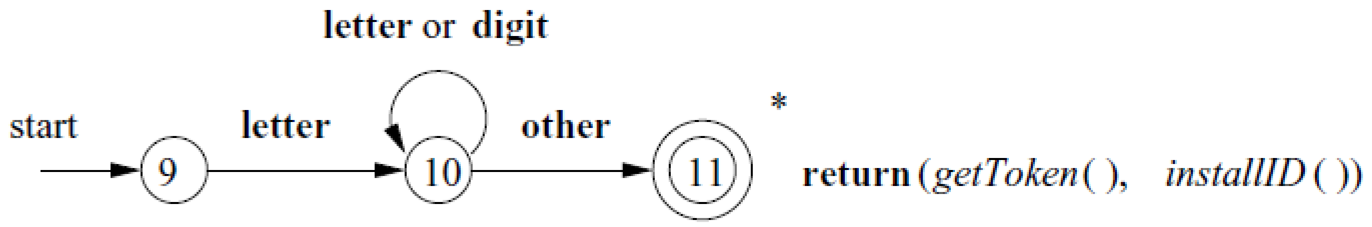


Figure 3.14: A transition diagram for **id**'s and keywords

Recognition of Reserved Words and Identifiers

- Two ways to handle the reserved words
  - Install the **reserved words** in the **symbol table** initially.
    - A field of the symbol-table entry indicates that these strings are never ordinary identifiers, and tells which token they represent
    - When an ID is found, a call to InstallID() place it in the symbol table if it is not already there and returns a pointer to the symbol table entry for the lexeme found
    - By following this logic, any identifier **not** in the symbol table during lexical analysis (LA) **cannot be a reserved word**, so its token is **id**.
    - The function getToken() will examine the symbol table entry for the lexeme found, and returns whatever token name the symbol table says this lexeme represents (either ID or keyword tokens that was initially installed in the table)

Recognition of Reserved Words and Identifiers

- Create **separate** transition diagrams for **each keyword**.
  - An example for the keyword “then”, the transition is shown below
  - Such a transition diagram consists of states representing the situation after each successive letter of the keyword is seen, followed by a test for a nonletter-or-digit. i.e., any character that cannot be the continuation of an identifier
  - It is necessary to check that the identifier has ended.
  - Otherwise, we need to return token “then” to be an ID, because the lexeme would be looks like “thenextvalue” that has “then” as a proper prefix
    - thenextvalue → ID
    - then →reserved
  - If we adopt this approach, then we must prioritize the tokens so that the reserved-word tokens are recognized in preference to ID, especially when the lexeme matches both patterns, ID and reserved.

Recognition of Reserved Words and Identifiers

Non letter / non digit

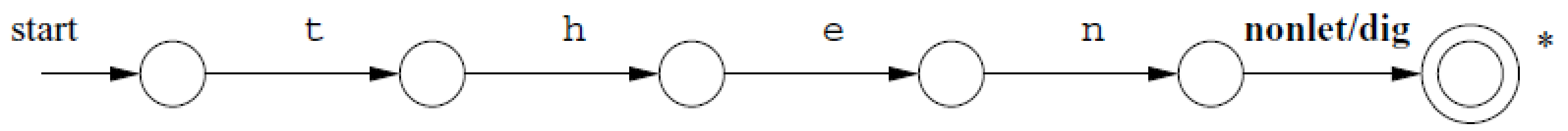
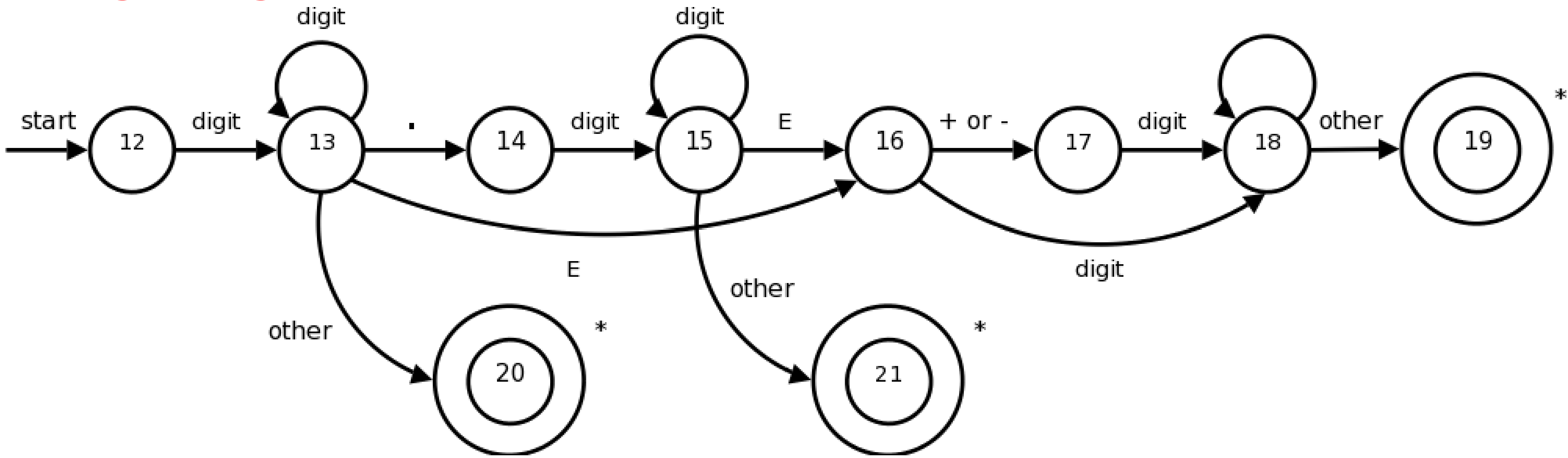


Figure 3.15: Hypothetical transition diagram for the keyword `then`

Completion of the Running Example

- So far we have transition diagrams for identifiers (the diagram also handles keywords) and the relational operators (relop)
- What remains are **whitespace**, and **numbers**
- This is the recognizing for numbers



Completion of the Running Example

- State # are just for illustration, not carrying super meaningful information.
- It only has the info. about the “relative order” of states
- In the state 20, for example, the token number is returned and a pointer to a table of constants where the found lexeme is entered
- If we instead see a dot in state 13, then we have an “optional fraction“, state 14 is entered, and we look for one or more additional digits in state 15.
- If we see an E, then we have an optional exponent,“ whose recognition is the job of states 16 through 19.
- If we are in state 15, see anything not an E or a digit, then we can come to an end of a fraction and return the lexeme just found, in state 21

Completion of the Running Example

- This is the recognizing for whitespaces!

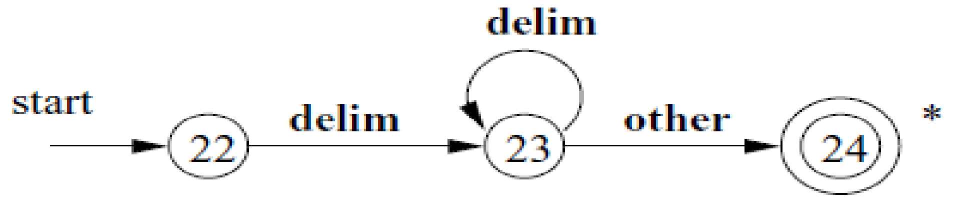

$$\begin{tikzcd} \arrow[r, "\text{start}"] & 22 \arrow[r, "\text{delim}"] & 23 \arrow[loop above, "\text{delim}"] \arrow[r, "\text{other}"] & 24^{*} \end{tikzcd}$$

Figure 3.17: A transition diagram for whitespace

  - The "delim" in the diagram represents any of the whitespace characters, say space, tab, and newline.
  - The final star is there because we needed to find a non-whitespace character in order to know when the whitespace ends and this character begins the next token.
  - There is no action performed at the accepting state.
  - Indeed the lexer does not return to the parser, but starts again from its beginning as it still must find the next token.
  - Parser do not deal with white spaces. It has to be filtered out in this stage!

Architecture of a Transition-Diagram-Based Lexical Analyzer

- TBD, in Part5