

TDDD55 Compilers and interpreters

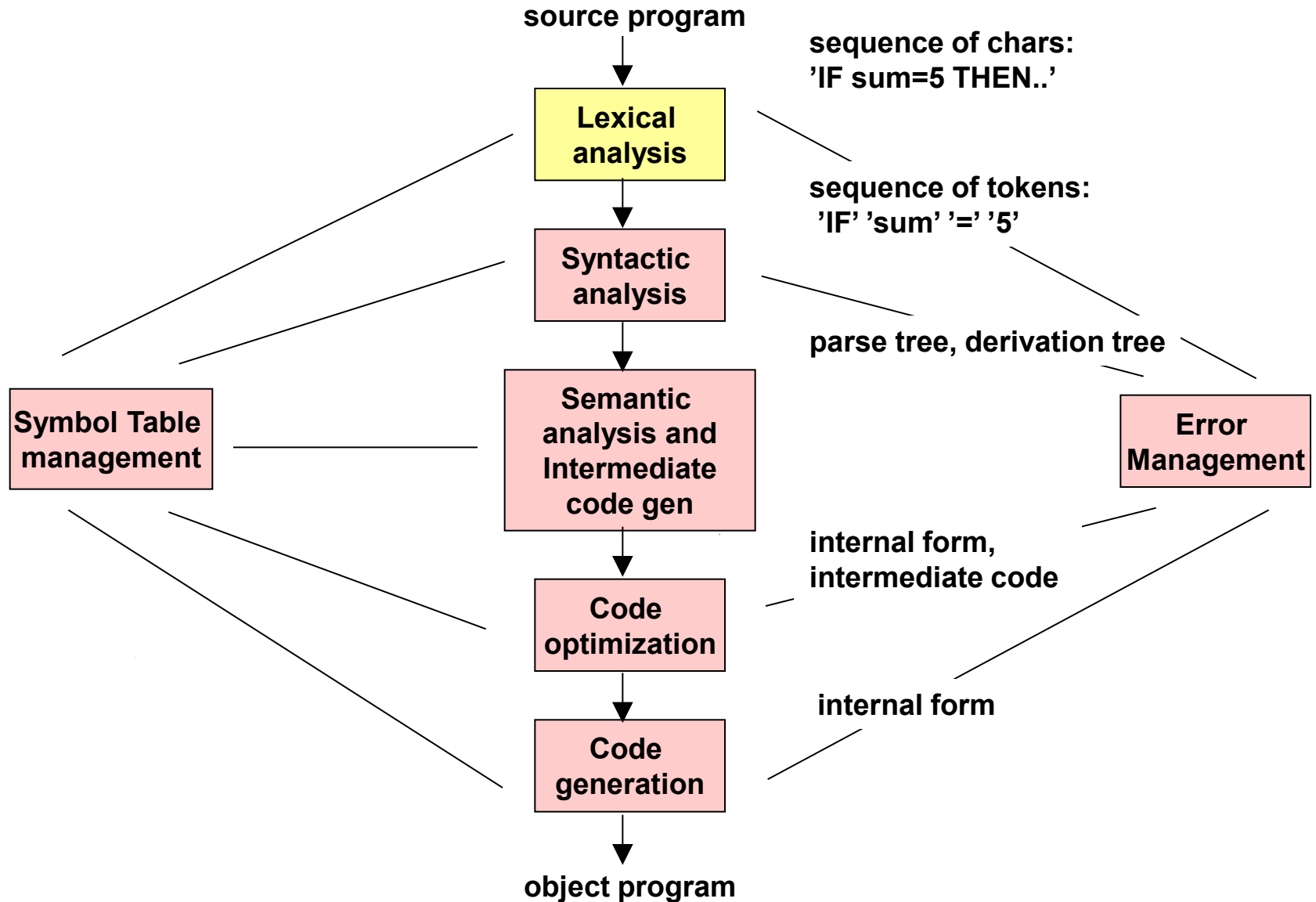
TDDE66 Compiler Construction



Lexical Analysis

Scanners

Lexical Analysis in the Compiler



Lexical Analysis, Scanners

□ Function

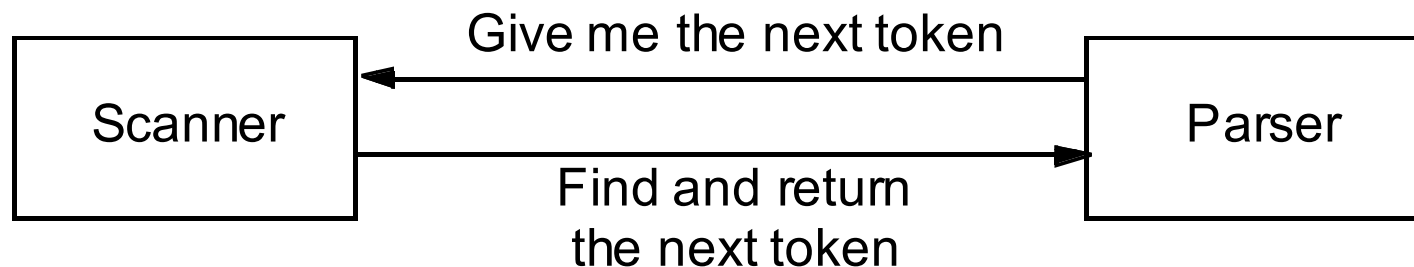
1. Read the input stream (sequence of characters), group the characters into primitives (tokens).
Returns token as *<type, value>*.
2. Throw out certain sequences of characters (blanks, comments, etc.).
3. Build the symbol table, string table, constant table, etc.
4. Generate error messages.
5. Convert, for example, string → integer.

□ Tokens are described using regular expressions

Note: See Lecture 3 on Formal Languages to refresh your knowledge of regular expressions, DFAs, NFAs.

Construction of a Scanner

- ❑ **Tools:** state automata and transition diagrams.
- ❑ Regular expressions enable the automatic construction of scanners.
- ❑ **Scanner generator** (e.g. *Lex*):
In: Regular expressions.
Out: Scanner.
- ❑ **Environment:**



How is a Scanner Programmed?

- ❑ Describe tokens with regular expressions.
- ❑ Draw transition diagrams.
- ❑ Code the diagram as table/program.

Example Scanner

❑ Example. Write a scanner for the following tokens.

Several categories of tokens:

- keyword = **BEGIN | END**
- id = letter (letter | digit)*
- integer = digit+
- op = **+ | - | * | / | // | ↑ | = | :=**

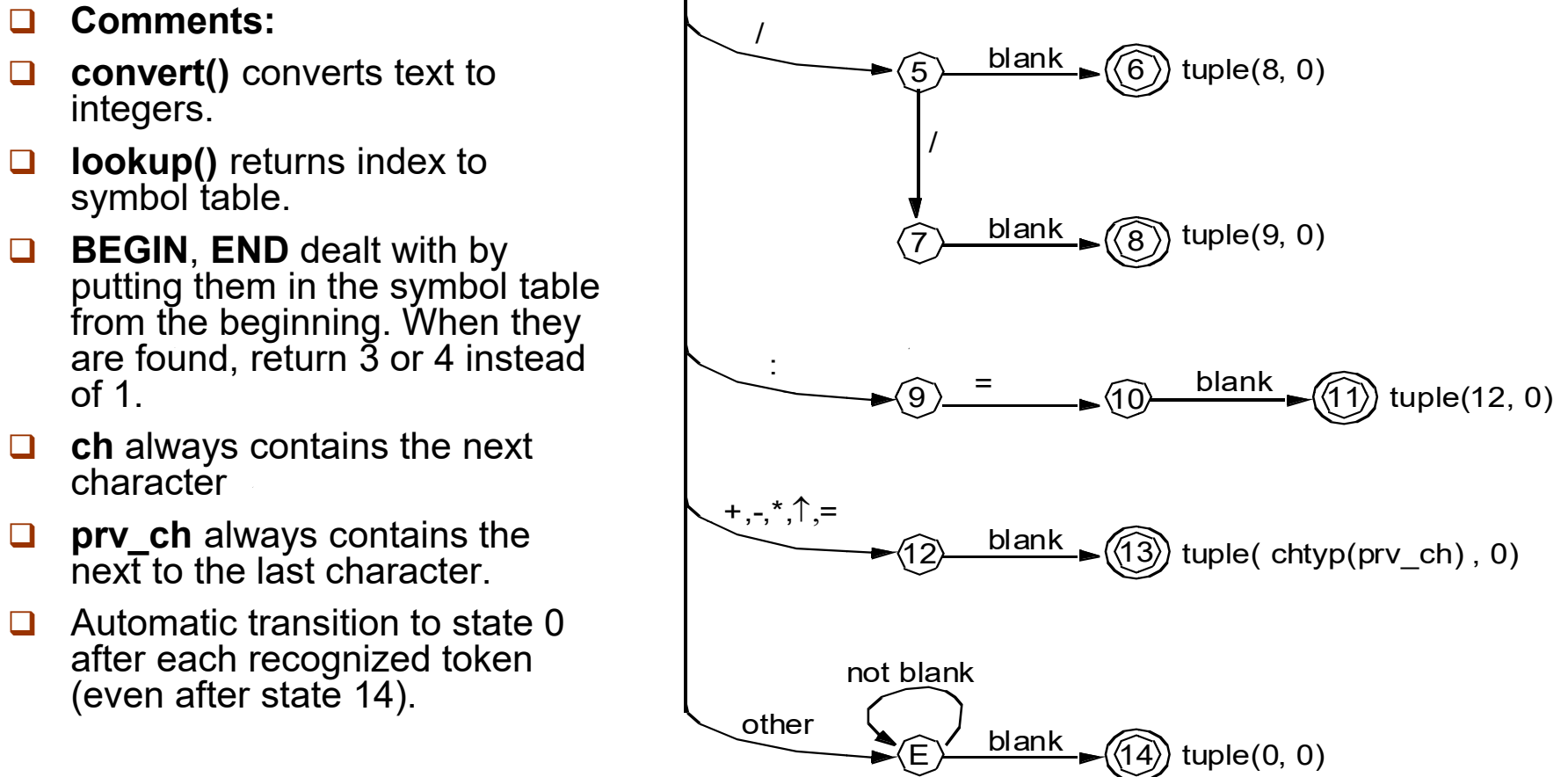
❑ Simplification:

- Assume that there is a blank character after each token.
- This simplification can easily be removed!

The Scanner Represents Tokens as Tuples

Tuple type	< Typecode, value >
undef.	< 0,0 >
id	< 1, table-pointer >
integer	< 2, value >
BEGIN	< 3,0 >
END	< 4,0 >
+	< 5,0 >
-	< 6,0 >
*	< 7,0 >
/	< 8,0 >
//	< 9,0 >
^	< 10,0 >
=	< 11,0 >
:=	< 12,0 >

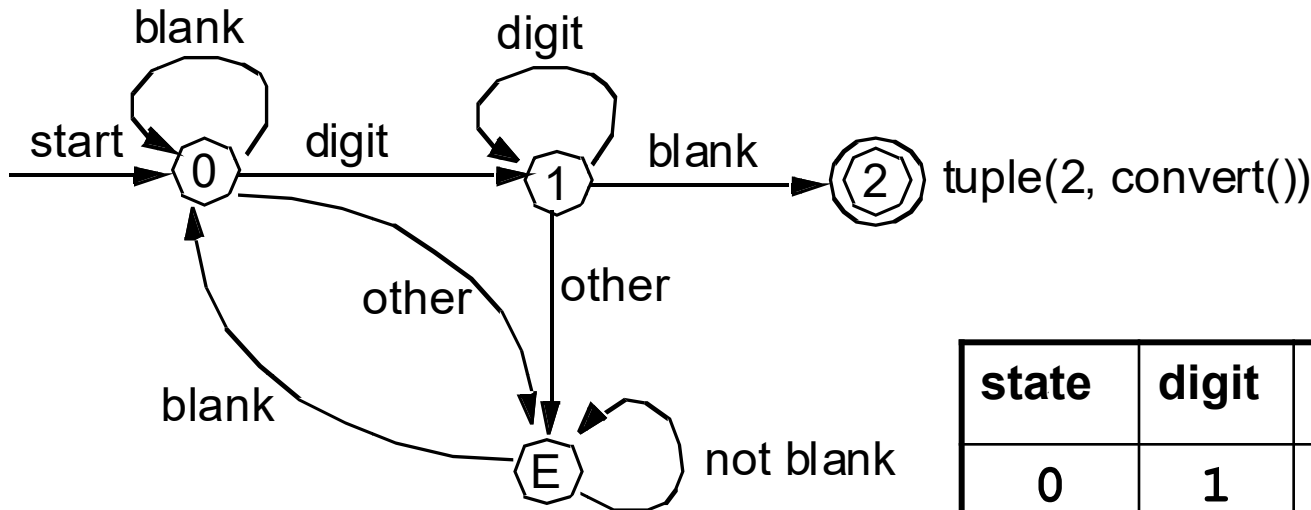
1. Draw the Transition Diagram



2. Translating the Transition Diagram

- Translate the diagram to a transition table, perform simple interpretation of the table.

Example: Transition-diagram and transition-table for integers:



state	digit	blank	Other	Accept
0	1	0	E	false
1	1	2	E	false
2	–	–	–	true
E	E	0	E	false

3. Interpreting the Table

```

Token t = new_Token();
int state = 0;
while ( 1 ) {
    char ch =getc ( inputfile );
    int oldstate = state;
    state = table [ state ][ ch ]; // transition
    // update t->tokenval with ch as appropriate:
    accumulate ( ch, state, t );
    if (is_error_state( state ))
        error_handler( oldstate, ch, ... );
    else if (is_accepting_state( state )) {
        t->tokentype = tokentype( state );
        return t;
    }
  }

```

state	digit	blank	other	accept
0	1	0	E	0
1	1	2	E	0
2	-	-	-	1
E	E	0	E	0

table

Generic Scanner, Interpreting the Table Using Global Data Structures

state	digit	blank	other	accept
0	1	0	E	0
1	1	2	E	0
2	-	-	-	1
E	E	0	E	0

// global data structures:

```
int table [ Nstates ][ Nchars ]  
        = ... (read in or initialize)  
typedef struct {  
    int tokentype;  
    union {  
        int ival; float fval; double dval; ...  
        symboltable *stptr;  
    } tokenval;  
} *Token;
```

// scanner routine, called from parser:

Token getNextToken(void)

```
{  
    Token t = new_Token();  
    int state = 0;  
    while ( 1 ) {  
        char ch = getc ( inputfile );  
        int oldstate = state;  
        state = table [ state ][ ch ]; // transition  
        // update t->tokenval with ch as appropriate:  
        accumulate ( ch, state, t );  
        if ( is_error_state( state ) )  
            error_handler( oldstate, ch, ... );  
        else if ( is_accepting_state( state ) ) {  
            t->tokentype = tokentype( state );  
            return t;  
        }  
    }  
}
```

4. Goto-Representation of the Table

b) Direct Jumps

```
state0:
    ch = getc;
    if ch >= '0' && ch <= '9' goto state1;
    if ch == " " goto state0;
    goto stateE;    /* in other cases */

state1:
...
```

c) using a Switch statement

```
switch (state) {
    case 0:
        switch (ch) {
            case '0': state = 1; break;
            ...
            case '9': state = 1; break;
            case ' ': state = 0; break;
            default: state = E; }
        break;
    case 1:
        ...
```

state	digit	blank	other	accept
0	1	0	E	0
1	1	2	E	0
2	-	-	-	1
E	E	0	E	0

5. Direct Coding of Diagrams (not via a table)

Data Structures and Functions

Variables:

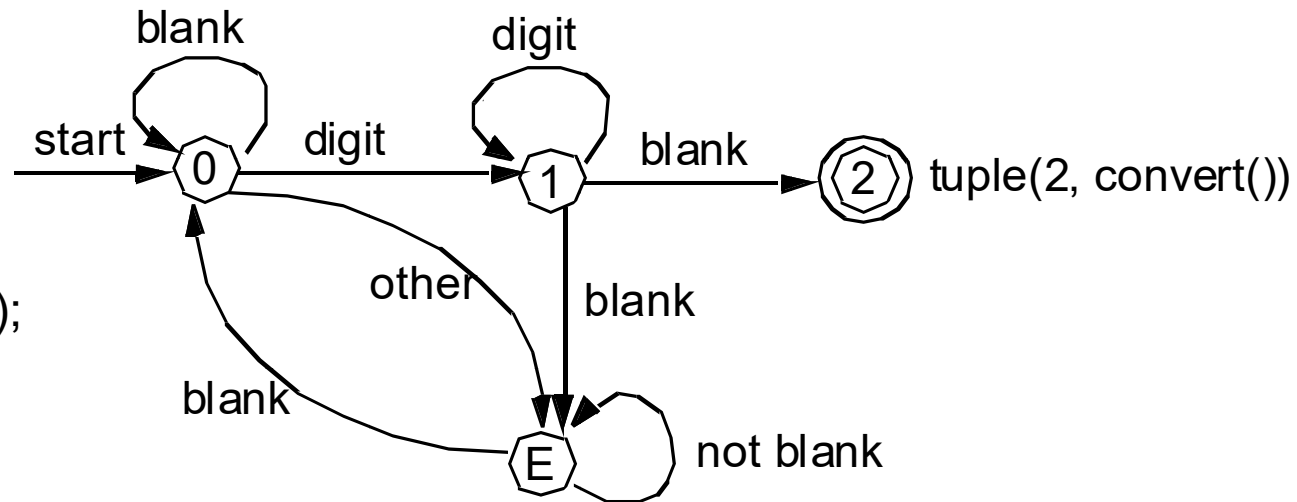
- `t->tokentype` = current symbol class
- `value` = value
- `ch` = current character
- `chtyp` = vector for 1-character tokens
- `symtab` = symbol table

Initialization:

- initialize **chtyp** according to the previous description;
- initialize the symbol table with reserved words;

Functions:

- `getc;`
- `skip_blanks;`
- `symtab_lookup(id);`
- `is_letter(ch);`
- `is_digit(ch);`



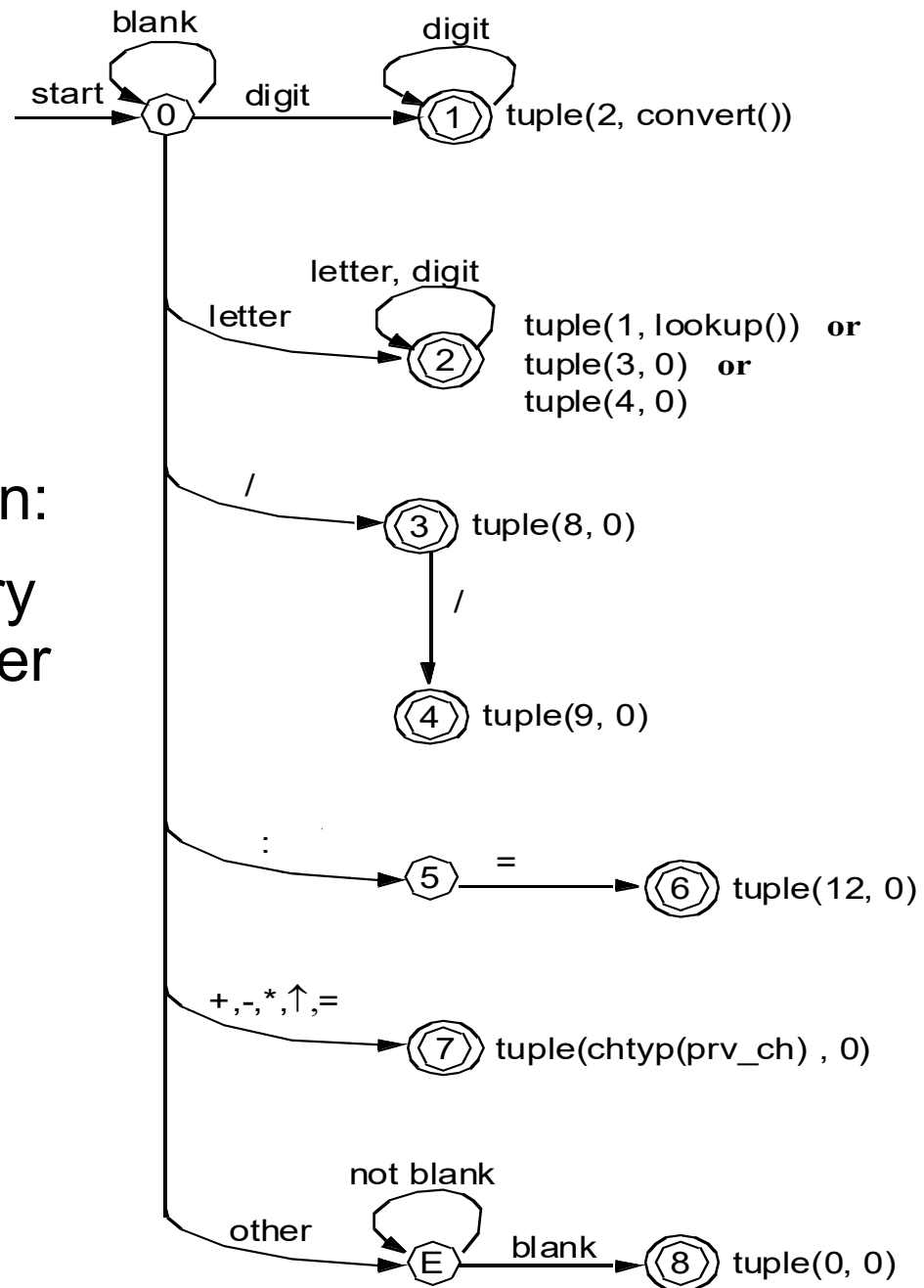
5. Scanner Fragment with Direct Coding Continued

```
Token getNextToken( void )
{
    char ch = getc( inputfile );
    char idstr [ ... ]; // lexeme buffer for identifiers
    t = new_Token();
    while (is_blank(ch))
        ch = getc ( inputfile ); // eat whitespace
    if (is_letter(ch)) { // identifier:
        while (is_letter(ch) || is_digit(ch) ) {
            append( ch, idstr );
            ch = getc( inputfile );
        }
        if (is_blank(ch))
            t->tokenval.stptr =
                symtab_lookup( idstr );
        else error( ... );
    }
    ...
}
```

```
...
else if (is_digit(ch)) { // int-constant:
    int ivalue = ch - '0';
    ch = getc( inputfile );
    while (is_digit(ch)) {
        ivalue *= 10;
        ivalue += ch - '0';
        ch = getc( inputfile );
    }
    if (is_blank(ch))
        t->tokenval.ival = ivalue;
    else error( ... );
}
...
else {
    // others (single-char. symbols):
    t->tokentype = chtyp[ ch ];
    if (t->tokentype == 0)
        error( ... );
}
return t;
}
```

Diagram with simplification removed

- ❑ Removed simplification:
- ❑ Space is not necessary as concluding character



Scanner Lookahead Problems

❑ **Lookahead** is sometimes needed to determine symbol type.

❑ Example: in FORTRAN

- **DO 10 I = 1.25** is an assignment, but
- **DO 10 I = 1,25** is a for-statement.

It is '.' or ',' which determines whether the scanner returns **DO10I** or **DO**

❑ Another Example: in Pascal.

Two character lookahead needed

- **715..816**