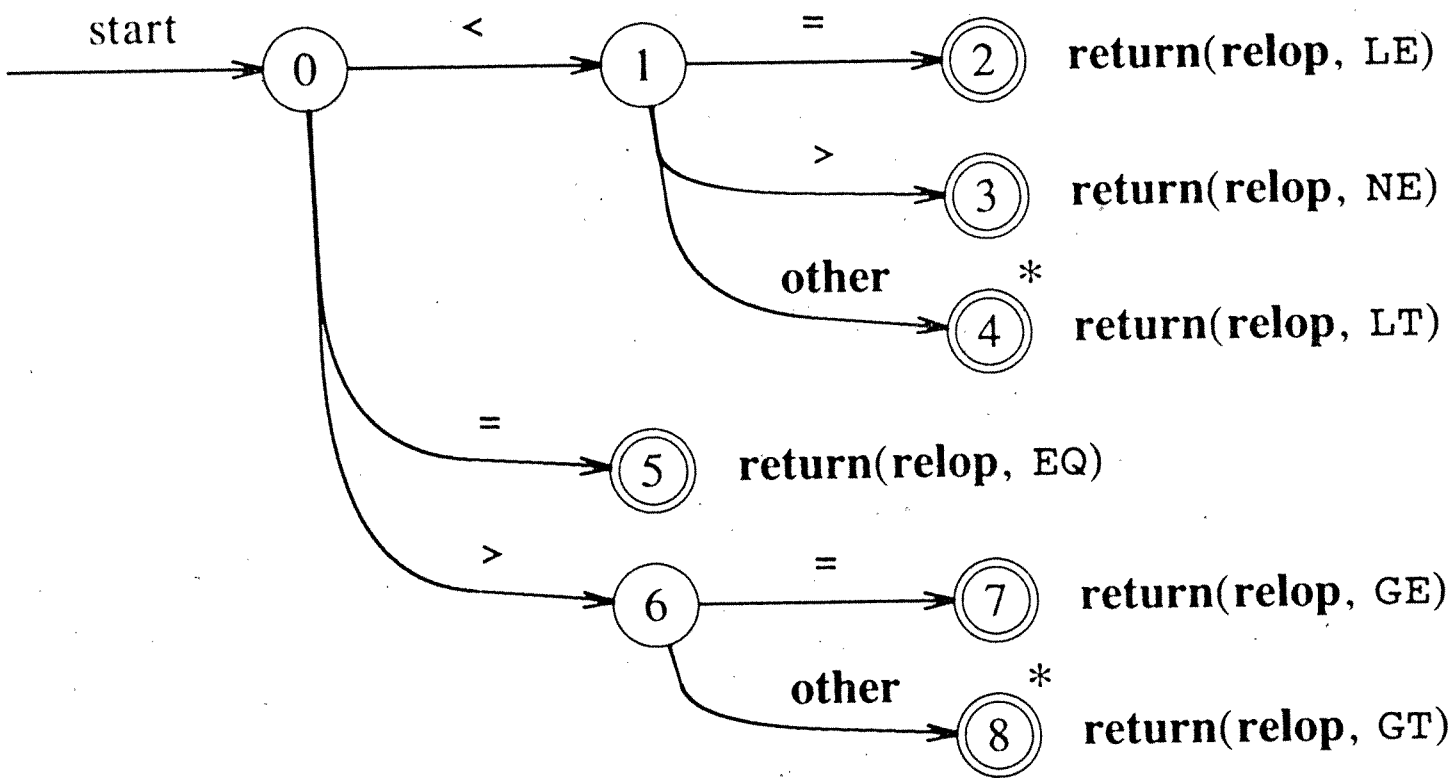

Fig. 3.12. Transition diagram for relational operators.

in a lexical analyzer for a typical programming language is several hundred, while using the trick, fewer than a hundred states will probably suffice.

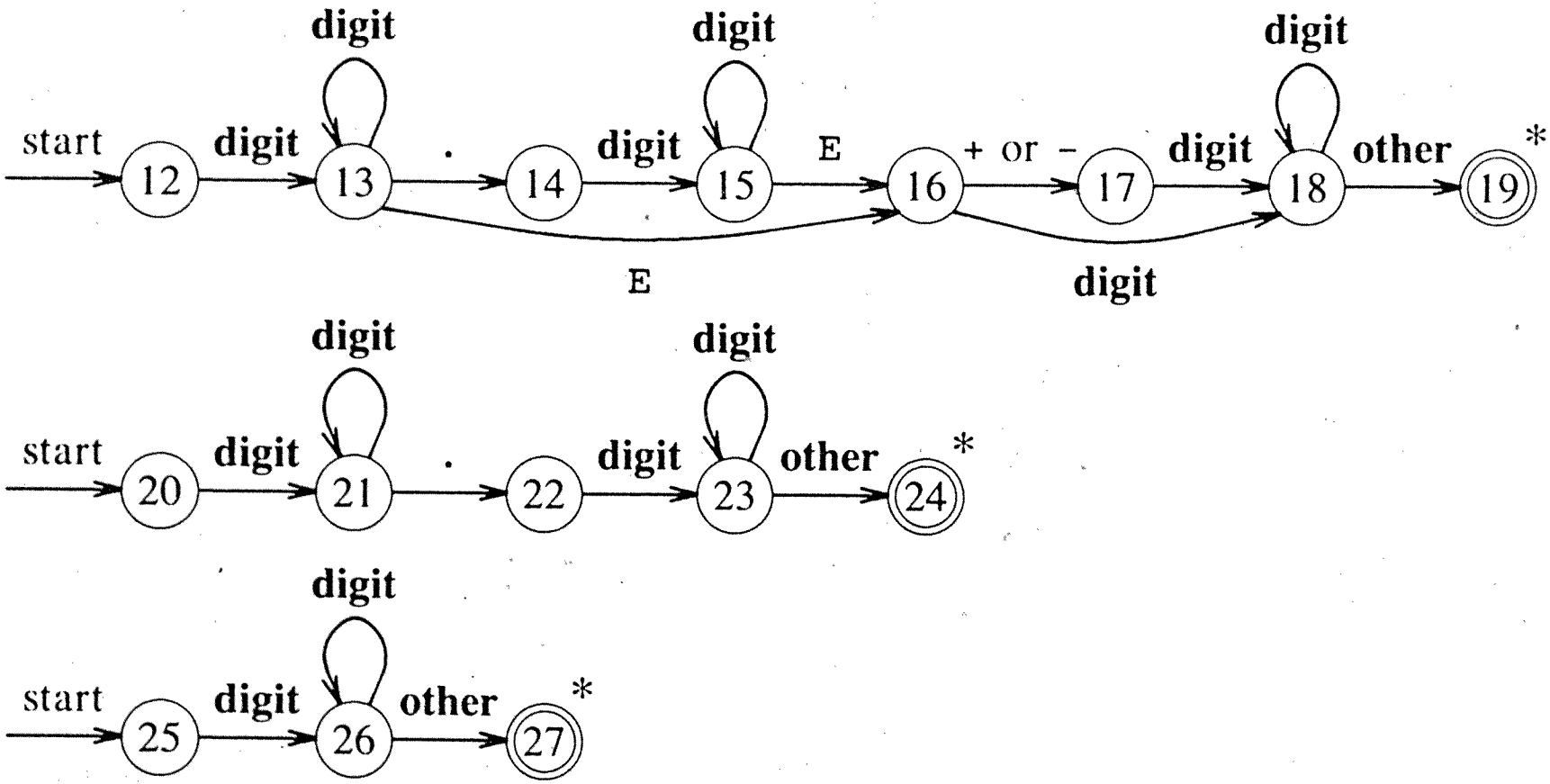

Fig. 3.14. Transition diagrams for unsigned numbers in Pascal.

Example 3.9. A number of issues arise when we construct a recognizer for

```
token nexttoken()
{   while(1) {
        switch (state) {
        case 0:   c = nextchar();
            /* c is lookahead character */
            if (c==blank || c==tab || c==newline) {
                state = 0;
                lexeme_beginning++;
                    /* advance beginning of lexeme */
            }
            else if (c == '<') state = 1;
            else if (c == '=') state = 5;
            else if (c == '>') state = 6;
            else state = fail();
            break;

            ... /* cases 1-8 here */

        case 9:   c = nextchar();
            if (isletter(c)) state = 10;
            else state = fail();
            break;
        case 10:  c = nextchar();
            if (isletter(c)) state = 10;
            else if (isdigit(c)) state = 10;
            else state = 11;
            break;
        case 11:  retract(1); install_id();
            return ( gettoken() );

            ... /* cases 12-24 here */

        case 25:  c = nextchar();
            if  (isdigit(c)) state = 26;
            else state = fail();
            break;
        case 26:  c = nextchar();
            if (isdigit(c)) state = 26;
            else state = 27;
            break;
        case 27:  retract(1); install_num();
            return ( NUM );
        }
    }
}
```

Fig. 3.16. C code for lexical analyzer.

```
int state = 0, start = 0;
int lexical_value;
    /* to "return" second component of token */

int fail()
{
    forward = token_beginning;
    switch (start) {
        case 0:   start = 9; break;
        case 9:   start = 12; break;
        case 12:  start = 20; break;
        case 20:  start = 25; break;
        case 25:  recover(); break;
        default:   /* compiler error */
    }
    return start;
}
```

Fig. 3.15. C code to find next start state.