


프로그래밍언어론

Chapter 3 어휘 및 구문 분석

Syntactic sugar causes cancer of the semicolon.
A. Perlis



목차

3.1 Chomsky Hierarchy

3.2 Lexical Analysis

3.3 Syntactic Analysis



어휘 분석

- 목적: 프로그램 표현 형태를 변환한다.
- 입력: 출력가능한 Ascii 문자열
- 출력: 토큰열
- 무시되는 것: 여백, 주석
- 토큰(token)
하나의 기호를 나타내기 위해 논리적으로 묶인 문자열

토큰 예

- 식별자 - 명칭이라고도 함
- 리터럴 : 123, 5.67, 'x', true
- 키워드 : bool char ...
- 연산자 : + - * / ...
- 구두문자 : ; , () { }

다른 문자열

- 여백: 공백, 탭 문자
- 주석
// any-char* end-of-line
- 줄끝 표시자 End-of-line
- 파일끝 표시자 End-of-file

어휘 분석이 왜 독립된 단계를 이루는가?

- 파서보다 더 간단하고 빠른 모델이 가능
- 컴파일 시간의 75%를 차지 (최적화없을때)
- 서로 다른 문자집합 (ASCII, EBCDIC, Unicode)
- 서로 다른 줄끝 표시자

정규식 Regular Expressions

정규식	의미
x	문자 x
\x	확장문자, 예를 들면 \n
{ name }	명칭 참조
M N	M 또는 N
MN	M 다음에 N이 나타남(접합)
M*	M이 0번 이상 반복됨
M+	M이 1번 이상 반복됨
M?	M이 0번 또는 1번 나타남
[aeiou]	모음 문자 a, e, i, o, u의 집합
[0-9]	숫자 0부터 9까지의 집합
.	임의의 문자 하나

Clite 어휘 구문

부류

- anyChar
- Letter
- Digit
- Whitespace
- Eol
- Eof

정규식 정의

[-~]
[a-zA-Z]
[0-9]
[\t]
\n
\004

Clite 어휘 구문

부류

- Keyword
- Identifier
- integerLit
- floatLit
- charLit

정규식 정의

bool | char | else | false | float |
if | int | main | true | while
{Letter}({Letter} | {Digit})*
{Digit}+
{Digit}+\.{Digit}+
'{anyChar}'

Clite 어휘 구문

부류

정규식 정의

- Operator `= | || | && | == | != | < | <= | > |`
`>= | + | - | * | / | ! | [|]`
- Separator `: | . | { | } | (|)`
- Comment `// ({anyChar} | {Whitespace})* {eol}`

어휘분석기 생성기

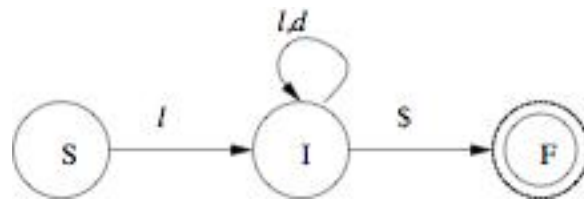
- 입력: 정규식
- 출력: 테이블(배열), 코드
- C/C++: Lex, Flex
- Java: JLex

유한상태 오토마타 Finite State Automata

- 상태 집합: 그래프 노드로 표현
- 입력 알파벳 + 특수 기호(입력의 끝을 나타냄)
- 상태 전이 함수
그래프에서 알파벳 기호가 붙은 아크
- 시작 상태(특별히 지정됨)
- 최종 상태 집합

결정 유한상태 오토마타

- 정의: 유한상태 오토마타에서 각 상태와 입력 기호에 대하여 그 입력 기호가 붙고 그 상태에서 나가는 아크가 많아야 하나 밖에 없을 때 이 유한상태 오토마타는 결정적(deterministic)이라고 한다.
- 식별자를 위한 유한상태 오토마타



구성상태와 전이

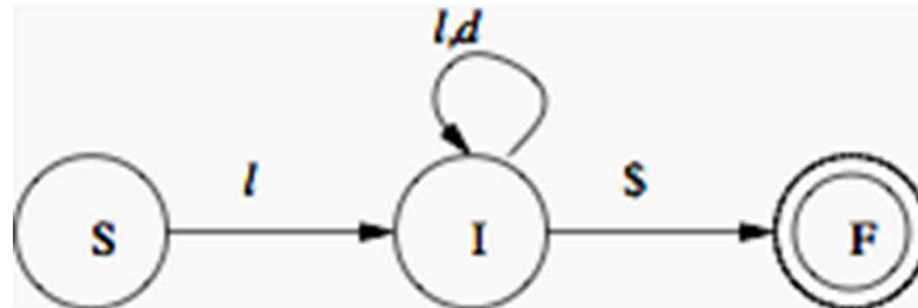
- 오토마타의 구성 상태(configuration)는 하나의 상태와 나머지 입력(입력끝 표시자로 끝난다)으로 이루어진다.
- 하나의 전이(move)는 맨 왼쪽 입력 기호에 대응하면서 현재 상태에서부터 진출하는 아크를 따라가는 것으로 입력 기호 하나를 처리한다.
 - 현재 상태와 입력 기호에 대한 전이가 정의되지 않으면 오토마톤은 오류로서 멈추게 되고 따라서 입력은 거절된다.

입력의 수락

- 시작 상태에서 시작하여 오토마톤이 모든 입력기호를 처리한 후에 최종 상태에서 멈추게 될 때 입력은 수락된다(accepted).

예제

- $(S, a2i\$) \models (I, 2i\$)$
 $\models (I, i\$)$
 $\models (I, \$)$
 $\models (F,)$



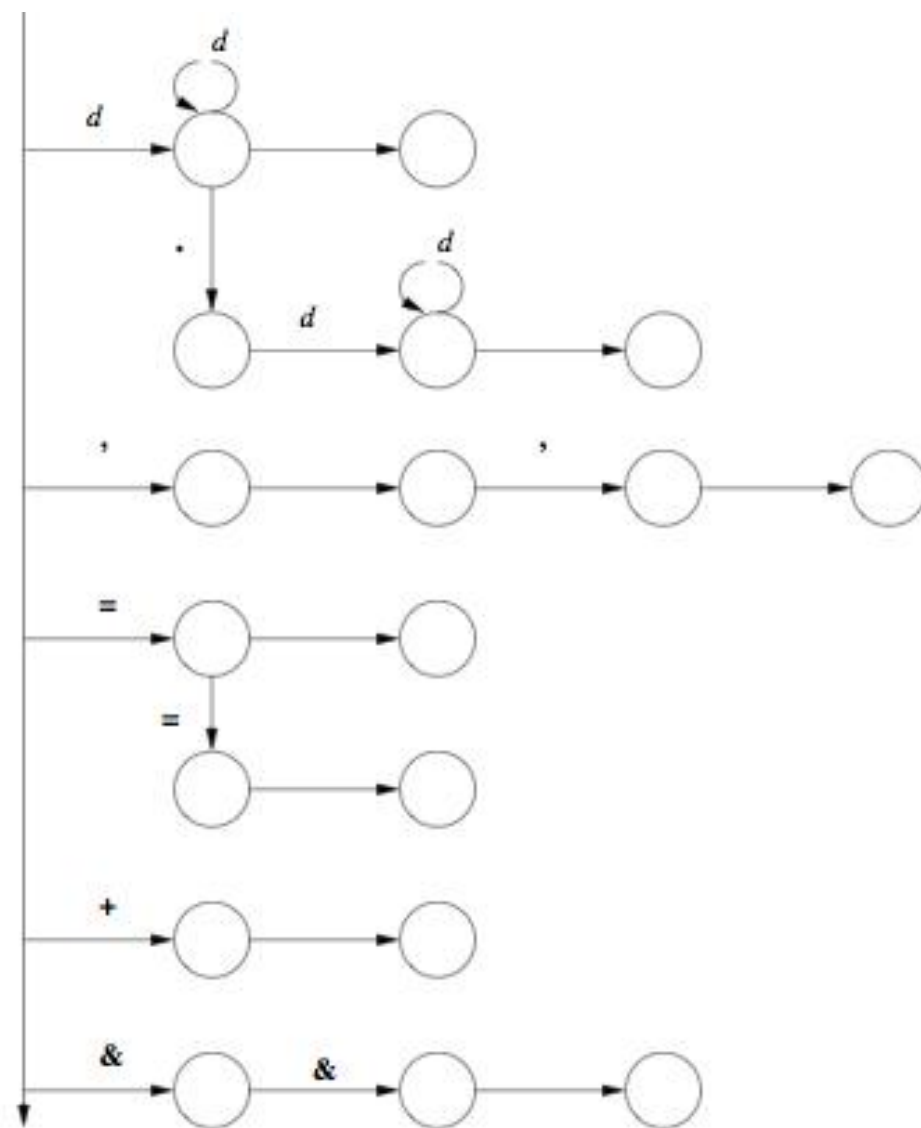
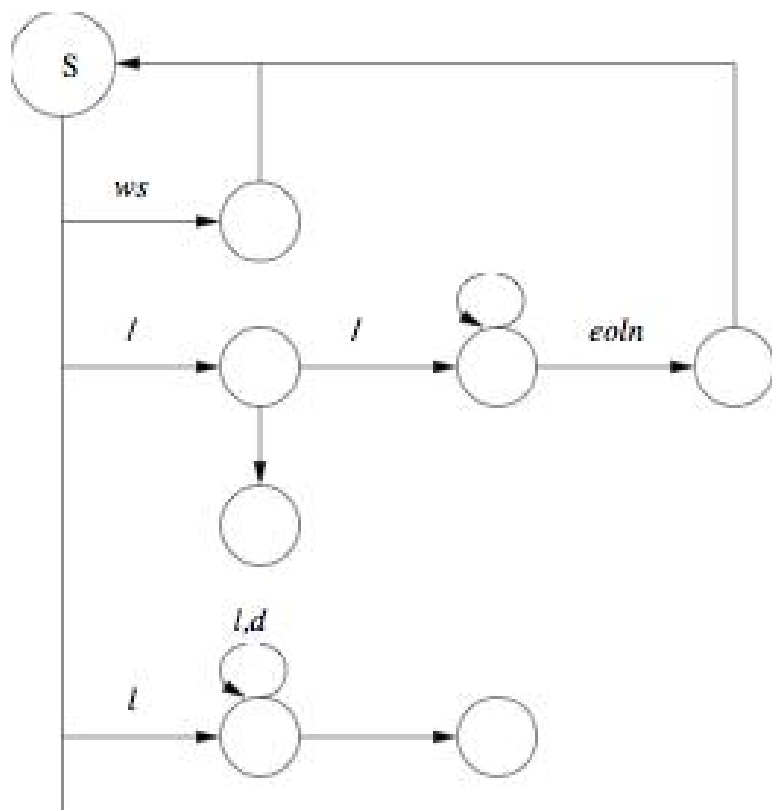
따라서 $(S, a2i\$) \models^* (F,)$

몇가지 유의점

- 입력끝 표시자(\$)는 각 토큰마다 나타나지 않고 전체 프로그램에 대해서만 나타남.
- 레이블이 없는 아크는 다른 모든 입력 기호를 나타냄.
- 토큰의 인식은 최종상태에서 종결됨.
- 토큰이 아닌 입력에 대한 인식(여백, 주석)은 시작상태로 돌아가는 아크를 가짐.

몇가지 유의점

- 파일끝의 인식은 소스 프로그램의 끝을 나타내고 종료상태에서 종결됨.
- 오토마타는 결정적이어야 함.
 - 키워드 인식을 따로 포함하지 않음.
 - 앞부분이 같은 토큰들은 모두 함께 고려해야 함.



렉서 코드

- 파서는 새로운 토큰 하나를 얻기 위해서 렉서를 호출한다.
- 렉서는 입력 문자열을 어디까지 읽었는지를 기억해야 한다.
- 무조건 한 문자씩 읽어가는 방식은 토큰 인식 과정에서 한 문자를 더 읽어서 잃어버리게 된다.
 - 엿보기 함수(peek function)
 - 되돌림 함수(pushback function)
 - 시작상태에서는 입력문자 처리를 하지않음.

설계에서 구현으로

```
private char ch = ' ';  
public Token next ( ) {  
    do {  
        switch (ch) {  
            ...  
        }  
    } while (true);  
}
```

유의점

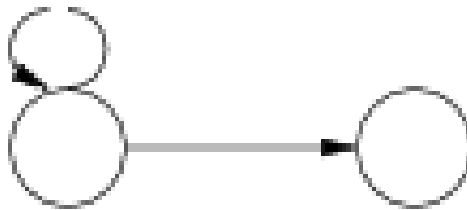
- 루프는 토큰이 인식될 때만 종료된다.
- 루프는 `return`문을 통해 종료된다.
- 변수 `ch` 는 전역변수이어야 하고
공백으로 초기화되어야 한다.
- 토큰의 실제 표현(문자열이거나 정수이거나)
은 설계와 무관하다.

전이 규칙

- A에서 B로 아크를 따라서 이동:
 - 아크가 x로 레이블되면 `ch == x` 이지 검사한다.
 - 아크가 레이블이 없으면 if/switch문의 else/default 선택절에 해당된다.
 - 아크가 유일한 아크라면 검사는 불필요하다.
 - A가 시작상태가 아니면 다음 문자를 읽어온다.

전이 규칙

- 자기자신으로 가는 아크를 가진 노드는 do-while 루프에 해당한다.
 - 반복조건은 아크의 레이블에 대응한다.

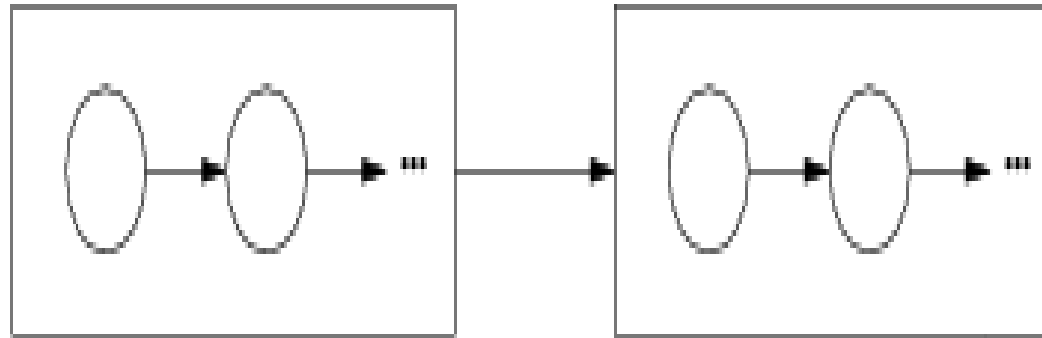


전이 규칙

- 그 외의 전이는 if/switch문으로 변환된다.
 - 각 아크는 각 선택절에 대응함.
 - 레이블이 없는 아크는 기본값 선택절(default).
- 연속된 전이는 변환된 문장의 연속으로 표현된다.

전이 규칙

- 복잡한 하위 다이어그램을 박스로 묶어서 변환한다.
 - 각 박스를 바깥에서 안쪽으로 들어가면서 변환함.



보조 함수 isLetter

```
private boolean isLetter(char c) {  
    return ch >= 'a' && ch <= 'z' ||  
        ch >= 'A' && ch <= 'Z';  
}
```

보조 함수 concat

```
private String concat(String set) {  
    StringBuffer r = new StringBuffer("");  
    do {  
        r.append(ch);  
        ch = nextChar( );  
    } while (set.indexOf(ch) >= 0);  
    return r.toString( );  
}
```

렉서의 토큰 생성

```
public Token next( ) {  
    do { if (isLetter(ch) { // ident or keyword  
        String spelling = concat(letters+digits);  
        return Token.keyword(spelling);  
    } else if (isDigit(ch)) { // int or float literal  
        String number = concat(digits);  
        if (ch != '.')  
            return Token.mkIntLiteral(number);  
        number += concat(digits);  
        return Token.mkFloatLiteral(number);  
    }  
}
```

렉서의 토큰 생성

```
} else switch (ch) {  
    case ' ': case '\t': case '\r': case eofCh:  
        ch = nextCh( ); break;  
    case eofCh: return Token.eofTok;  
    case '+': ch = nextChar( );  
        return Token.plusTok;  
    ...  
    case '&': check('&'); return Token.andTok;  
    case '=': return chkOpt('=', Token.assignTok,  
        Token.eqqTok);
```

소스 프로그램

토큰

```
// a first program
// with 2 comments
int main ( ) {
    char c;
    int i;
    c = 'h';
    i = c + 3;
} // main
```

```
int
main
(
)
{
char
Identifier      c
;

```