

제5장 의미론

Semantics

숙명여대 창병모

4차원 공간의 semantics

지금까지 한 것/앞으로 할 것!

● 주제	논리	구현
● 구문법	문법	파서
● 의미론	의미 함수	인터프리터
● 타입	타입 규칙	타입 검사기

의미론의 개요

- 프로그램 의미의 정확한 정의 및 이해
- 소프트웨어의 정확한 명세
- 소프트웨어 시스템에 대한 검증 혹은 추론
- 컴파일러 혹은 해석기 작성의 기초

의미론의 종류

- 작동 의미론(Operational Semantics)
 - 프로그램의 의미를 프로그램 실행(작동) 과정으로 설명한다.
- ✓ ● 표기 의미론(Denotational Semantics)
 - 프로그램의 의미를 함수 형태로 정의하여 설명한다.
- 공리 의미론(Axiomatic Semantics)
 - 프로그램의 시작 상태와 종료 상태를
 - 논리적 선언(assertion) 형태로 정의하여 설명한다.

5.1 수식의 의미

수식의 의미

- 수식 E의 의미
 - 상태에서 수식의 값
 - V : (State, Expr) \rightarrow Value
- 예
 - $s = \{x \mapsto 1, y \mapsto 2\}$
 - $V(s, x+y) = 3$
- 상태 s에서 간단한 수식의 의미
 - $E \rightarrow \text{true} \mid \text{false} \mid n \mid \text{str} \mid \text{id}$
 - $V(s, \text{true}) = T$
 - $V(s, \text{false}) = F$
 - $V(s, n) = n$
 - $V(s, \text{str}) = \text{str}$
 - $V(s, \text{id}) = s(\text{id})$

수식의 의미

- 산술 수식

- $E \rightarrow E + E \mid E - E \mid E * E \mid E / E$

- $V(s, E1 + E2) = V(s, E1) + V(s, E2)$

모든 V로 계산 가능

- ...

- 비교 수식

- $E \rightarrow E > E \mid E < E \mid E == E \mid E != E$

- $V(s, E1 == E2) = T \text{ if } V(s, E1) == V(s, E2)$
F otherwise

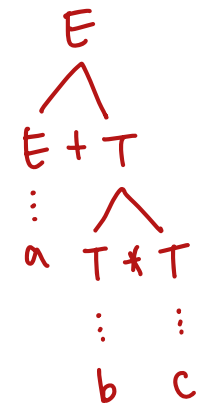
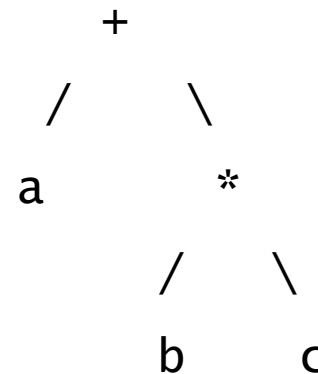
- ...

연산자 우선순위

- 수식의 값
 - 연산자 우선순위와 결합성에 따라 다르다!
- 연산자 우선순위
 - 연산자를 실행하는 순서
 - 우선순위가 높은 연산자가 먼저 실행됨

- 예

- $a + (b * c)$



- 연산자 우선순위 표현

- $E \rightarrow E + T \mid T$
 - $T \rightarrow T * F \mid F$
 - $F \rightarrow n \mid id$

우선순위

- C 연산자 우선순위

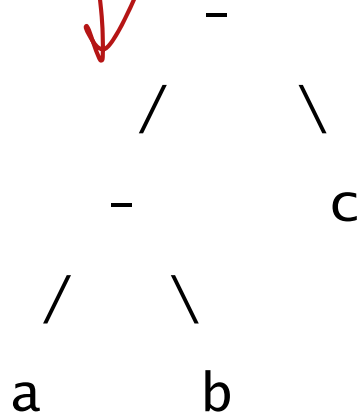
연산자 그룹	연산자	기억 요령
산술 연산자	* / %	계산하여
산술 연산자	+ -	
비교 연산자	< <= > >= == !=	비교한 후
논리 연산자	&&	판단하여
조건 연산자	?:	
대입 연산자	= += -= *= /= %= &= ^= = <<= >>=	저장한다.

연산자 결합성

- 수식 $a - b - c$ 의미

- $(a - b) - c$
- $a - (b - c)$

- 수식의 AST



- 연산자 결합성

- 동일한 우선순위를 갖는 연산자가 두 개 이상 인접한 경우
- 무엇을 먼저 수행하느냐에 대한 규칙

결합성

- 좌결합(left associative)

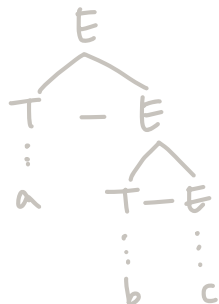
- 왼쪽부터 우선 결합하는 규칙으로 왼쪽에서 오른쪽으로 수행된다.
- 대부분의 이항 연산자들은 좌우선 결합 규칙을 따른다

- 좌순환 규칙(left recursive rule)으로 표현

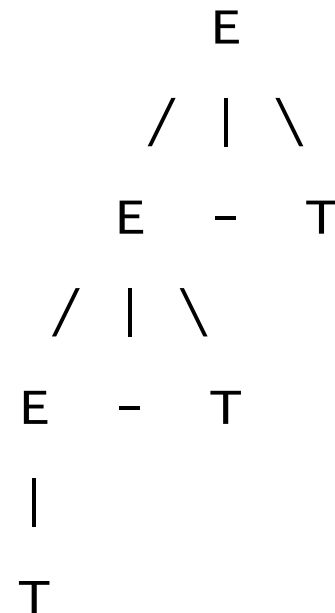
① $E \rightarrow E - T \mid T$

recursive

② $E \rightarrow T - E \mid T$
 $T \rightarrow id$) 라고 할까?



우결합 표현



결합성

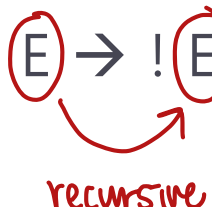
- 우결합(right associative)

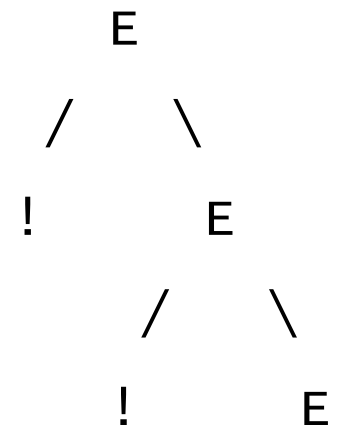
- 오른쪽부터 우선 결합하는 규칙
- 오른쪽의 연산자부터 먼저 수행하는 것을 말한다.
- 대부분의 단항 연산자들과 대입 연산자는 우결합 규칙을 따른다.

- 예

- !!a
- $a = (b = c);$ (in C)

- 우순환 규칙(right recursive rule)으로 표현

- $E \rightarrow !E \mid id \mid \dots$




5.2 구조적 프로그래밍

Fortran 제어 구조

```
10 IF (X .GT. 0.000001) GO TO 20
11 X = -X
IF (X .LT. 0.000001) GO TO 50
20 IF (X*Y .LT. 0.00001) GO TO 30
X = X-Y-Y
30 X = X+Y
...
50 CONTINUE
X = A
Y = B-A
GO TO 11
...
```

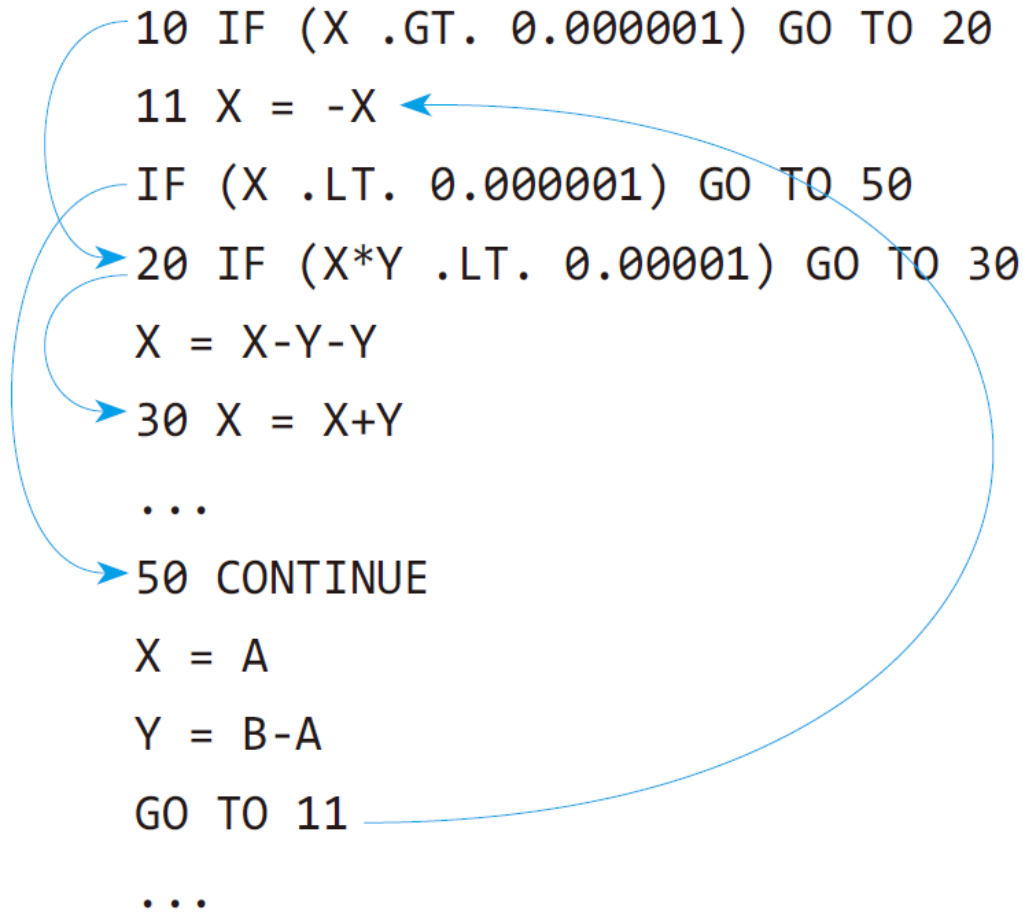


그림 5.5 goto 실행
가동성↓

Goto 문에 대한 역사적 논쟁

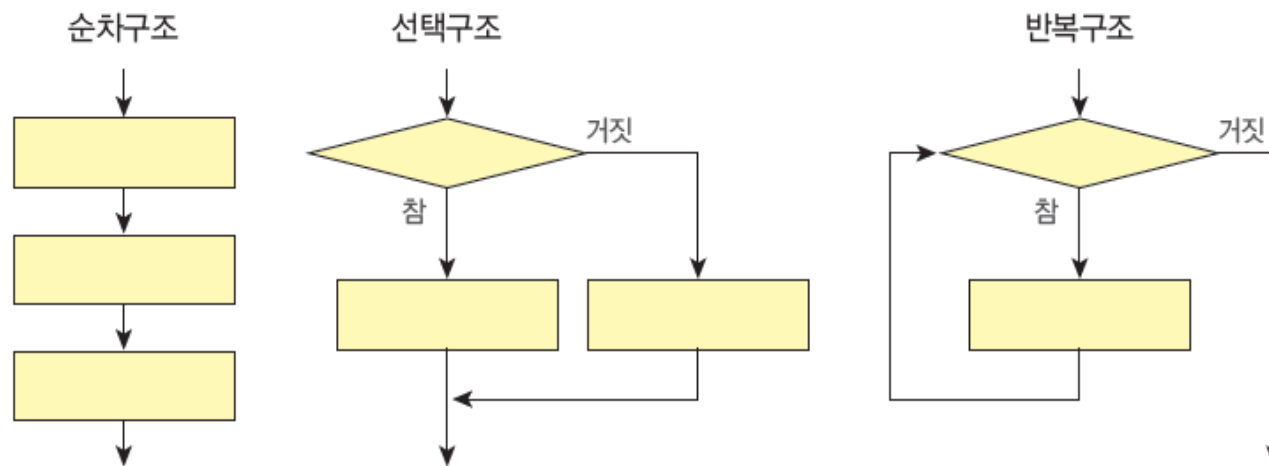
- Dijkstra, Goto Statement Considered Harmful
 - Letter to Editor, *C ACM*, March 1968
- Knuth, Structured Prog. with go to Statements
 - You can use goto, but do so in structured way ...
- Continued discussion
 - Welch, "GOTO (Considered Harmful)ⁿ, n is Odd"
- General questions
 - Do syntactic rules force good programming style?
 - Can they help?

자바에는 없음

구조적 프로그래밍

- 기본 아이디어
 - goto 문을 사용하지 않고 시작과 끝 지점이 일정한 구조적 구문 사용
 - 프로그램에 있는 각각의 구조와 그 사이의 관계를 이해하면 프로그램 전체를 보다 쉽게 이해할 수 있다.
- 구조적 프로그램 정리 : Böhm & Jacopini (1966)
 - 어떤 계산 함수든 순차, 선택, 반복의 3가지 제어 구조로 표현 가능
 - 순차(concatenation):
 - 구문 순서에 따라서 순서대로 수행
 - 선택(selection):
 - 프로그램의 상태에 따라서 여러 구문들 중에서 하나를 선택해서 수행
 - if, switch 문
 - 반복(repetition):
 - 구문을 반복하여 수행.
 - while, repeat, for 문.

구조적 프로그램 정리 : Böhm & Jacopini (1966)



어디서 시작하고 끝나는지 명확하게 보임

언어 S의 문장 : 요약 문법

- 정수
 - $n \in \mathbf{Int}$
- 변수
 - $id \in \mathbf{Var}$
- 식
 - $E \in \mathbf{Exp}$

Stmt S \rightarrow id = E

| S; S

| let T id [= E] in S end

| if E then S [else S]

| while (E) S

| read id

| print E

Expr E \rightarrow n | id | true | false

| E + E | E - E | E * E | E / E | (E)

| E == E | E < E | E > E | !E

예제

```
read x;  
y = 1;  
while (x != 1) {  
    y = y*x;  
    x = x-1;  
}  
print y;
```

1. x 값을 읽는다. $\{ x \mapsto n \}$ if READ n
2. y에 1 배정 $\{ x \mapsto n, y \mapsto 1 \}$
3. x 값이 1이 아닌지 검사
4. 거짓이면 종료
5. 참이면 y 값을 x 값과 현재 y 값의 곱으로 변경 $\{ x \mapsto n, y \mapsto 1*n \}$
6. x 값 1 감소 $\{ x \mapsto n-1, y \mapsto n \}$
7. 3 번부터 반복

설탕 구문 Syntactic sugar

- 언어 S는 필수적인 몇 개의 문장만을 제공하고 있다
- 프로그래밍 편의를 위해 제공하는 부가적인 문장
- do S while (E)

→ (1번 이상 실행 (일반적인 while 문은 S를 0번 이상 실행))

```
S;  
while (E)  
  S;
```

while 문으로 표현 가능

- for (e1; e2; e3) S

```
e1;  
while (e2) {  
  S;  
}  
e3;
```

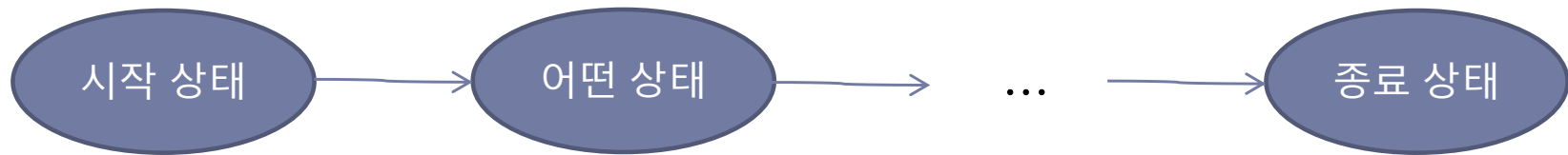
- switch (E) {
 case a : S; ...; break;
 case b : S; ...; break;
 case c : S; ...; break;
}

```
if (E == a) S1;  
else if (E == b) S2;  
  ⋮
```

5.3 문장의 의미

작동 의미론 (Operational Semantics)

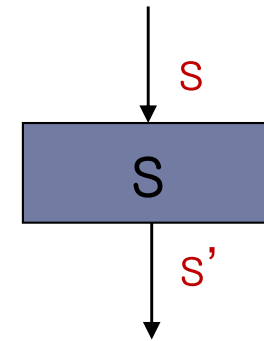
- 기본 아이디어
 - 프로그램의 의미를 프로그램 실행과정으로 설명한다. ↗ command의 나열
 - 실행과정을 상태 전이 (state transition) 과정으로 설명한다.
 - 각 문장 S마다 상태 전이 규칙 정의



문장의 의미

- 문장 S 의 의미
 - 문장 S 가 전상태 s 를 후상태 s' 으로 변경시킨다.
 - 상태 변환
- 상태 변환 함수 (state transform function)
 - Eval : (State, Statement) \rightarrow State
 - $\text{Eval}(s, S) = s'$ for each statement S
- 의미론
 - 각 문장 S 마다 상태 변환 함수 정의
 - 프로그램의 실행과정을 상태 변환 과정으로 설명한다.

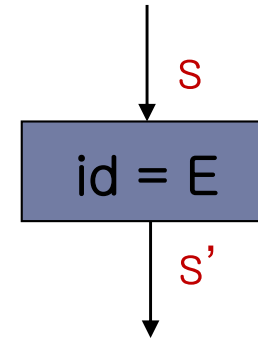
Notational Semantics



f: A \rightarrow B 형태

대입문, 복합문의 의미

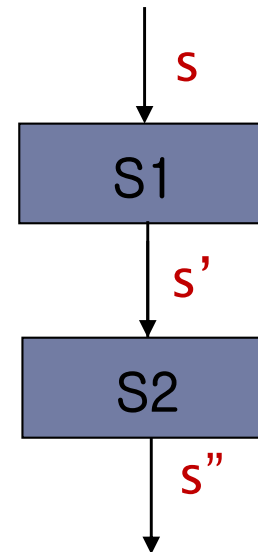
- 대입문 $id = E$ *id의 값이 expression 값으로 바뀜*
 $Eval(s, id = E) = s[id \mapsto V(s, E)]$



- 복합문 $S1; S2$
 $Eval(s, S1; S2) = Eval(Eval(s, S1), S2)$

$$\frac{(s, S1) \rightarrow s', (s', S2) \rightarrow s''}{(s, S1; S2) \rightarrow s''} \text{ p.a.}$$

relation으로 표현함



예

- (1) 실행

- $s = \{x \mapsto 0\}$ *자동초기화*
- $\text{Eval}(s, x = 1) = s[x \mapsto 1] = \{x \mapsto 1\}$

```
let int x; in
```

```
  x = 1;           (1)
```

```
  let int y; in
```

```
    y = 2;         (2)
```

```
    x = x + y;     (3)
```

```
  end;
```

```
end;
```

- (2) 실행

- $s = \{x \mapsto 1, y \mapsto 0\}$
- $\text{Eval}(s, y = 2) = s[y \mapsto 2] = \{x \mapsto 1, y \mapsto 2\}$

- (3) 실행

- $s = \{x \mapsto 1, y \mapsto 2\}$
- $\text{Eval}(s, x = x + y) = s[x \mapsto V(s, x + y)] = s[x \mapsto 3] = \{x \mapsto 3, y \mapsto 2\}$

함수의 유효범위는 stack으로

Let 문

- let T id [=E] in S end

- 실행 과정

- 변수 선언을 만나면 변수가 유효해 지므로
 - 이 변수를 위한 엔트리를 실행 전상태 s에 추가
- 이 상태에서 문장 S를 실행하고
- 실행이 끝나면
 - 선언된 변수는 유효하지 않으므로 실행 후 상태 s'에서 해당 엔트리 제거
 - 이 상태가 let-문을 실행한 후에 상태가 된다.

- 상태 변환 함수

$\text{Eval}(s, \text{let } T \text{ id} = E \text{ in } S \text{ end}) = s'$
if $s'[\text{id} \mapsto *] = \text{Eval}(s[\text{id} \mapsto V(s, E)], S)$
any value

$$\frac{(s[\text{id} \mapsto V(s, E)], S) \rightarrow s'[\text{id} \mapsto *]}{(s, \text{let } T \text{ id} = E \text{ in } S \text{ end}) \rightarrow s'}$$

[예제 1]

```
0 let int x=1; in
1   let int y=1; in
2     y = y + 1;
3     x = x + y;
4   end;
5 end;
6
```

$s0 = \{ \}$

$s1 = s0[x \mapsto 1] = \{ x \mapsto 1 \}$

$\text{push}((x \mapsto 1), s0)$

$s2 = s1[y \mapsto 1] = \{ x \mapsto 1, y \mapsto 1 \}$

$\text{push}((y \mapsto 1), s1)$

$s3 = \{x \mapsto 1, y \mapsto 2\}$

$s4 = \{x \mapsto 3, y \mapsto 2\}$

$s5 = s4 - \{y \mapsto *\} = \{x \mapsto 3\}$

$\text{pop}((y \mapsto *), s4)$

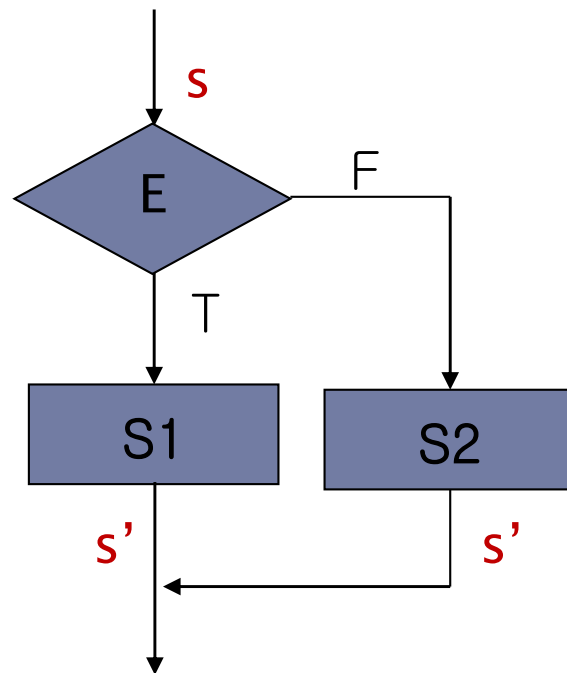
$s6 = s5 - \{x \mapsto *\} = \{ \}$

$\text{pop}((x \mapsto *), s5)$

↓
제거
(교재에는 \가 있었음)

조건문의 의미

- if E then S1 else S2



$$\frac{(s, S1) \rightarrow s'}{(s, \text{if } E \text{ then } S1 \text{ else } S2) \rightarrow s'} \quad \text{if } V(s, E) = T$$

$$\frac{(s, S2) \rightarrow s'}{(s, \text{if } E \text{ then } S1 \text{ else } S2) \rightarrow s'} \quad \text{if } V(s, E) = F$$

- $C = \text{if } E \text{ then } S1 \text{ else } S2$
- $\text{Eval}(s, C) = \text{Eval}(s, S1)$ if $V(s, E) = T$
- $\text{Eval}(s, C) = \text{Eval}(s, S2)$ if $V(s, E) = F$

[예제 2]

```
0 let int x; int y; int max; in
1   x = 1;
2   y = 2;
3   if (x > y) then max =x; // C
4   else max =y;
5 end;
```

- 실행 과정

$s0 = \{ \}$

$s1 = \text{Eval}(s0, \text{int } x; \text{int } y; \text{int } \text{max}) = \{ x \mapsto 0, y \mapsto 0, \text{max} \mapsto 0 \}$

$s2 = \text{Eval}(s1, x = 1) = \{ x \mapsto 1, y \mapsto 0, \text{max} \mapsto 0 \}$

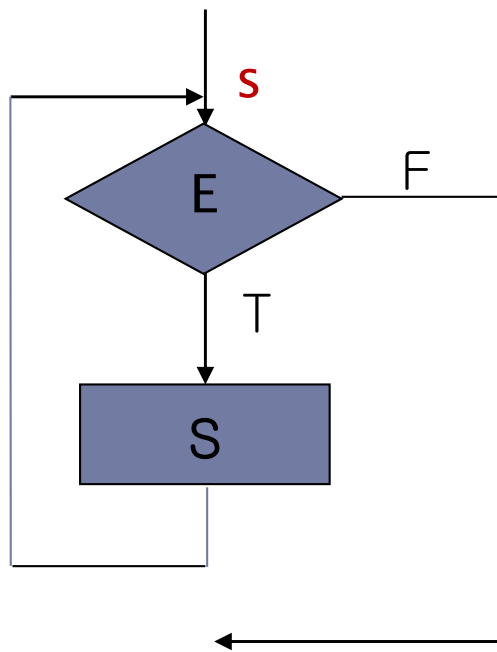
$s3 = \text{Eval}(s2, y = 2) = \{ x \mapsto 1, y \mapsto 2, \text{max} \mapsto 0 \}$

$s5 = \text{Eval}(s3, C) = \text{Eval}(s3, \text{max}=y) = \{ x \mapsto 1, y \mapsto 2, \text{max} \mapsto 2 \}$

왜냐하면 $V(s3, x > y) = F$

상태 전이 규칙

- while (E) S



$$(s, \text{while } E \text{ do } S) \rightarrow s \quad \text{if } V(s, E) = F$$

$$\frac{(s, S) \rightarrow s', (s', \text{while } E \text{ do } S) \rightarrow s''}{(s, \text{while } E \text{ do } S) \rightarrow s''} \quad \text{if } V(s, E) = T$$

- $L = \text{while } E \text{ do } S$
- $\text{Eval}(s, L) = \text{Eval}(\text{Eval}(s, S), L) \quad \text{if } V(s, E) = T$
- $\text{Eval}(s, L) = s \quad \text{if } V(s, E) = F$

[예제 3] *루프 종료*

```
0 let int x; int y; in
1   read x;
2   y = 1;
3   while (x != 1) // Loop (L)
    { // S
      y = y*x;
      x = x-1;
    }
end;
```

$s0 = \{ \}$

$\text{Eval}(s0, \text{int } x; \text{int } y) = s1 = \{x \mapsto 0, y \mapsto 0\}$

$\text{Eval}(s1, \text{read } x) = s2 = \{x \mapsto 3, y \mapsto 0\}$

$\text{Eval}(s2, y = 1) = s3 = \{x \mapsto 3, y \mapsto 1\}$

$\text{Eval}(s3, L) = \text{Eval}(\text{Eval}(s3, S), L)$

$V(s3, x \neq 1) = \text{T}$ *한번 실행한 상태*

$\text{Eval}(s3, S) = \text{s4} = \{x \mapsto 2, y \mapsto 3\}$

$\text{Eval}(s4, L) = \text{Eval}(\text{Eval}(s4, S), L)$

$V(s4, x \neq 1) = \text{T}$

$\text{Eval}(s4, S) = s5 = \{x \mapsto 1, y \mapsto 6\}$

$\text{Eval}(s5, L) = s5 = \{x \mapsto 1, y \mapsto 6\}$

$V(s5, x \neq 1) = \text{F}$

상태 전이 규칙

- **read id**

$\text{Eval}(s, \text{read id}) = s[\text{id} \mapsto n]$ if n is read.

- **print E**

$\text{Eval}(s, \text{print E}) = s$ if $V(s, E)$ is printed.

5.4 언어 S의 인터프리터

주요 주제

주제	논리	구현
구문법	문법	파서
의미론	상태 변환 함수	인터프리터

변수의 현재 값
(메모리에 저장됨)

AST를 반환하는 것

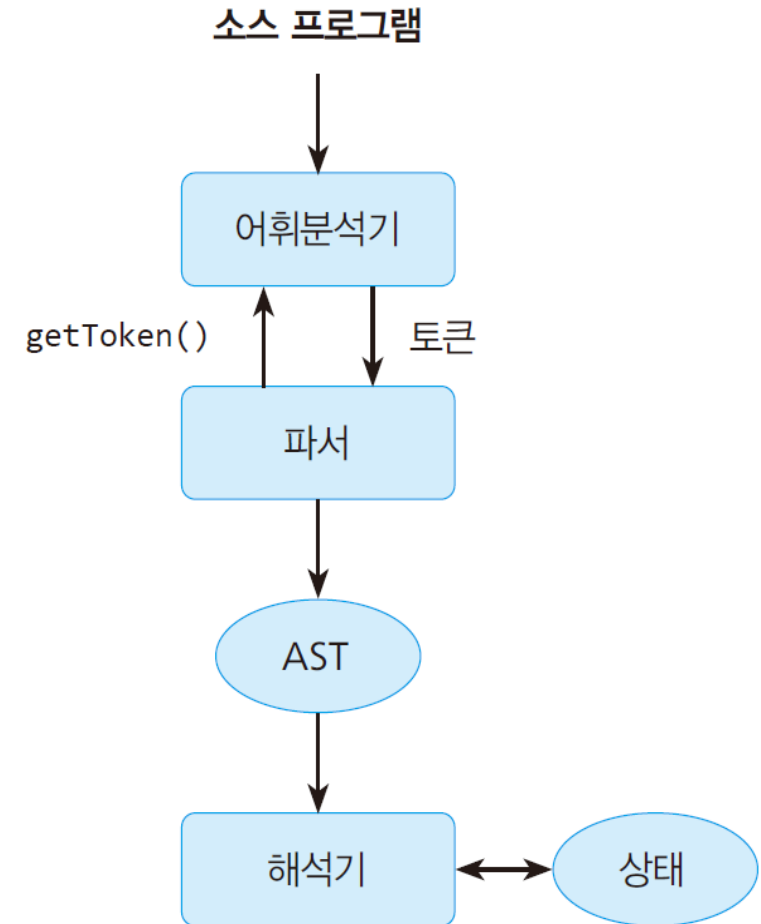


그림 3.1 어휘분석기, 파서, 해석기 구현

[언어 S의 문법 2]

$\langle \text{program} \rangle \rightarrow \{ \langle \text{decl} \rangle \mid \langle \text{stmt} \rangle \mid \langle \text{function} \rangle \}$

$\langle \text{decl} \rangle \rightarrow \langle \text{type} \rangle \text{ id } [= \langle \text{expr} \rangle];$

$\langle \text{stmt} \rangle \rightarrow \text{id} = \langle \text{expr} \rangle;$

$\mid \{ \langle \text{stmts} \rangle \}$

$\mid \text{if } (\langle \text{expr} \rangle) \text{ then } \langle \text{stmt} \rangle [\text{else } \langle \text{stmt} \rangle]$

$\mid \text{while } (\langle \text{expr} \rangle) \langle \text{stmt} \rangle$

$\mid \text{read id};$

$\mid \text{print } \langle \text{expr} \rangle;$

$\mid \text{let } \langle \text{decls} \rangle \text{ in } \langle \text{stmts} \rangle \text{ end};$

$\langle \text{stmts} \rangle \rightarrow \{ \langle \text{stmt} \rangle \}$

$\langle \text{decls} \rangle \rightarrow \{ \langle \text{decl} \rangle \}$

$\langle \text{type} \rangle \rightarrow \text{int} \mid \text{bool} \mid \text{string}$

인터프리터

- 인터프리터
 - 프로그램의 AST를 순회(traverse) 하면서 수식의 값을 계산하고 문장의 의미에 따라 각 문장에 대한 해석(interpret)을 수행한다.
- 인터프리터 함수 Eval
 - $\text{Eval: (Statement, State)} \rightarrow \text{State}$

print 문 구현

- print <expr>;

```
State Eval (Print p, State state) {  
    System.out.println(V(p.expr, state));  
    return state;  
}
```

Print

|

Expr

read 문 구현

- read id;

Id의 type에 따라
read할 값이 다름

```
State Eval (Read r, State state) {  
    if (r.id.type == Type.INT) {  
        int i = sc.nextInt();  
        state.set(r.id, new Value(i));  
    }  
    if (r.id.type == Type.BOOL) {  
        boolean b = sc.nextBoolean();  
        state.set(r.id, new Value(b));  
    }  
    return state;  
}
```

Read

|

Id

복합문 구현

- {<stmt>}

```
State Eval (Stmts ss, State state) {  
    for (Stmt stmt : ss.stmts) {  
        state = Eval(stmt, state);  
    }  
    return state;  
}
```

Stmts
/
\
Stmt ... Stmt

array list

하나씩 꺼내서
for ~ each 은

if 문 구현

- if (<expr>) then <stmt> else <stmt>

```
State Eval (If c, State state) {  
    if (V(c.expr, state).boolValue( ))  
        return Eval (c.stmt1, state);  
    else  
        return Eval (c.stmt2, state);  
}
```

```
      If  
    /  |  \  
Expr Stmt Stmt
```


while 문 구현

- while (<expr>) <stmt> *recursive call*

```
State Eval (While l, State state) {  
    if (V(l.expr, state).boolValue())  
        return Eval(l, Eval (l.stmt, state));  
    else  
        return state;  
}
```

```
while  
  /  \  
Expr Stmt
```

▪ 혹은

```
State Eval (While l, State state) {  
    while (V(l.expr, state).boolValue())  
        state = Eval (l.stmt, state);  
    return state;  
}
```

실습 #3: 언어 S의 인터프리터 구현

1. let 문을 구현을 위한 allocate 함수와 free 함수를 구현하시오.

```
State allocate (Decls ds, State state) {  
    // 선언된 변수들(ds)을 위한 엔트리들을 상태 state에 추가  
}  
  
State free (Decls ds, State state) {  
    // 선언된 변수들(ds)의 엔트리를 상태 state에서 제거  
}
```

2. 언어 S의 문법에 따라 관계 및 논리 연산 수행 기능을 구현하시오.

binaryOperation() 확장하여 정수, 스트링의 관계 연산 및
부울값의 논리 연산을 구현한다.

실습 #3 :언어 S에 대한 인터프리터 구현

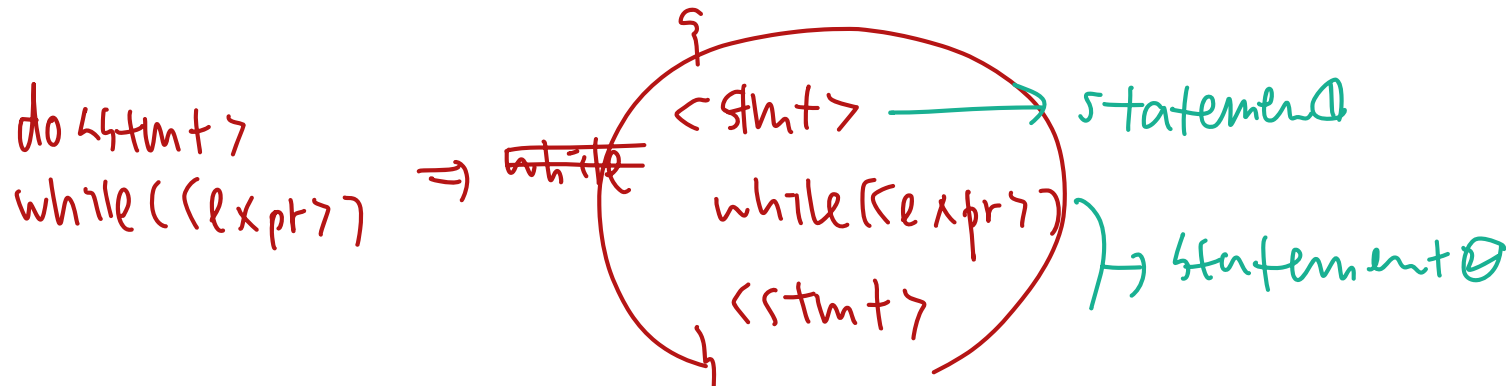
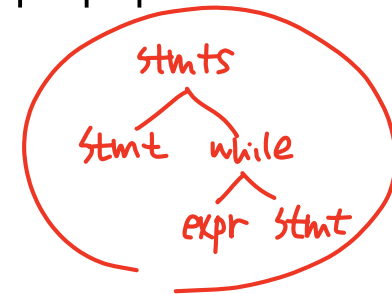
3. 언어 S의 확장

언어 S의 문장에 다음과 같이 do-while 문, for 문을 추가하고
이를 해석하는 인터프리터를 작성하시오.

<stmt> → ...

| do <stmt> while (<expr>);

| for (<type> id = <expr>; <expr>; id = <expr>) <stmt>



```

let
  <type> id = expr;
  while (<expr>) {
    <stmt>
    id = <expr>;
  }
end

```

