Compiler (컴파일러)
Semantic Analysis III
-Type Checking

2007년 2학기
충남대학교 전기정보통신공학부
조은선

Back to Type Checking

- 타입이란?
  - 수행 중 가지게 되는 '값'에 대한 기술
  - Predicate이기도 함.
    (예: C에서 "int x" 는 -2^31 <= x < 2^31 라는 뜻)
- 타입 오류 : 값의 부적절한 사용
- 타입 안전성:
  - "타입 오류가 없음"
  - 보장 방법
    - 타입을 선언하고 (=binding)
      - 명시적방법 (int x) vs. 암묵적 방법 (x=1;)
    - 이를 검사함 (= checking) 으로써 보장
      - 타입 규칙을 세우고 이에 대해 검사함

타입과 관련된 몇 가지 개념들

1. Static vs. dynamic checking
   - 타입 검사 시점 : 컴파일 시 vs. 수행시
2. Static vs. dynamic typing
   - 타입 정의 시점 : 컴파일전 vs. 수행시
3. Strong vs. weak typing
   - 타입 오류 : 엄격히 방지 vs. 조금 허용
4. Sound type systems
   - 모든 타입 오류를 방지 하는 타입 규칙 집합 및 enforce 시스템

타입 표현식 (Type Expressions)

- 언어에서 제공하는 기본 타입
  - int, float, char …
- 타입 표현식 (type expressions)
  - 기본 타입 및 이들로부터 작성되는 의미있는 타입들
  - Array types : T[], T[10], T[1..10], T[2][3]
    - 각 원소 접근, 지정 시 타입 유의
  - Structure types : $\{id_1: T_1, \cdots, id_n: T_n\}$
    - 필드 접근, 지정 시 타입 유의
  - Pointer types : T *
    - 내용 접근, 지정 시 타입 유의
  - Function types : $T_1 \times T_2 \times \cdots \times T_n \rightarrow T_{return}$
    - 호출 시 인자와 리턴 값 타입 유의

Implementation

- 타입 클래스 만들기
  - class BaseType extends Type {String name;}
  - class IntType extends BaseType { ... }
  - class FloatType extends BaseType { ... }
  - class ArrayType extends BaseType { ... }
  - class FunctionType extends BaseType { ... }
- 타입 비교 방법

  (1) 타입 트리 비교 방법 : T1.equals(T2) 등의 메소드내에서 해결

  (2) 매 타입 마다 하나씩 상수(또는 객체) 두는 방법 : 비교가 빠름!
    - 예) int [100] 은 1, int [1024]는 2 , float * 는 3 등

  Note : OO에서는 T1.subtypeOf(T2) 같은 것도 있어야 함

Implementation Example with Yacc & SDD w/o a Symbol Table

- 타입 객체 만들기 :

```
Type type;
type : INTEGER {$$= new IntType(id); }
       | ARRAY LBRACKET type RBRACKET
        {$$ = new ArrayType($3); } ;
```

- 타입 검사

```
expr:     expr PLUS expr {
                  if ($1 == IntType && $3 == IntType)
                        $$ = IntType;
                  else      TypeCheckError("+");
          }
```

Implementation with A Symbol Table

- Struct, class 정의등은 symbol table에 타입정의를 저장

```
class IdExpr extends Expr {
        Identifier id;
        Type typeCheck() {return id.lookupType(); }
}
```

  - 검사할 때는 symbol table에서 이 정의를 lookup

- Checking example (NonSDD style)

```
class Add extends Expr {
        Type typeCheck() {
                Type t1 = e1.typeCheck(), t2 = e2.typeCheck();
                if (t1 == CInt && t2 == CInt) return CInt;
                else TypeCheckError("+");
        }
}
```

타입 규칙 정의

- 타입 judgment (= static semantics)
  - 타입 규칙을 formal하게 정의함

    E : T *"E는 T 타입을 만족한다."*
  - 어휘분석에는 regular expression, 구문 분석에는 CFG, 타입 분석 (의미분석)에는 타입 judgment
    - 2 : int
    - true : bool
    - 2 * (3 + 4) : int
    - "Hello" : string
    - if (b) 2 else 3 : int
    - x == 10 : bool
    - y = 2 : int 또는 y=2 : unit // unit 는 well typed 임을 표현

Class Problem

- 다음 문장의 type 은?

  단 i로 시작하는 식별자는 int 변수이고, f로 시작하는 식별자는 float 변수라고 가정한다. S1은 문장이다.

  f1 [3]

  - i = i1 [i2]
  - while (i < 10) do S1
  - (i == 0 ? 4.0 : 1.0)

타입 Judgment 유도

- Consider if (b) 2 else 3 : int
- 이것을 위해 알아야할 것들
  - b 는 bool (b : bool)
  - 2 는 int (2 : int)
  - 3 는 int (3 : int)
- 타입 judgment notation
  - A ⊢ E : T “ A 상황에서 E는 T타입을 만족한다”
  - 예)

b: bool, x: int ⊢ b: bool

b: bool, x: int ⊢ if (b) 2 else x : int

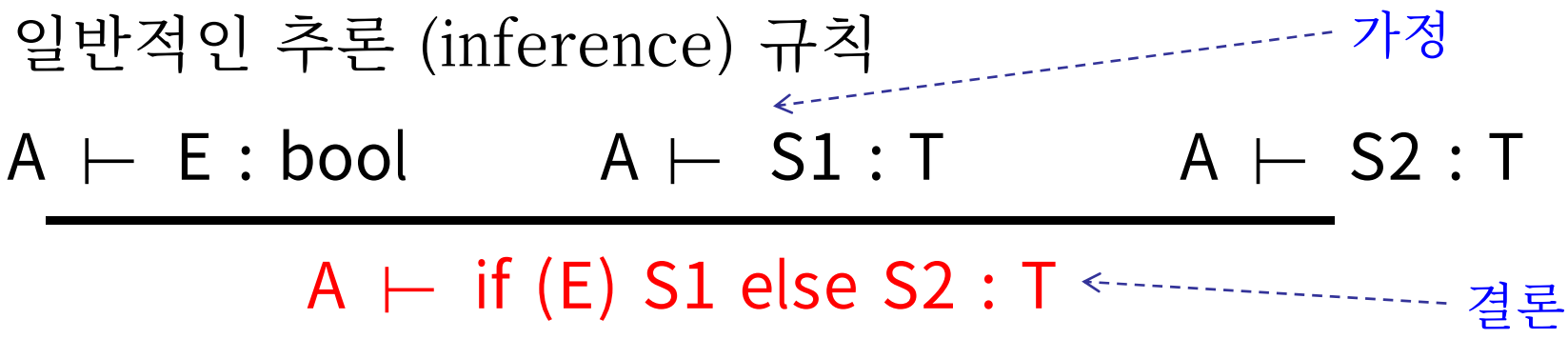
⊢ 2 + 2 : int

타입 Judgment 유도

- 이것을 보이기 위해서는
  - b: bool, x: int ⊢ if (b) 2 else x : int

  이것을 먼저 보여야함
  - b: bool, x: int ⊢ b : bool
  - b: bool, x: int ⊢ 2 : int
  - b: bool, x: int ⊢ x : int

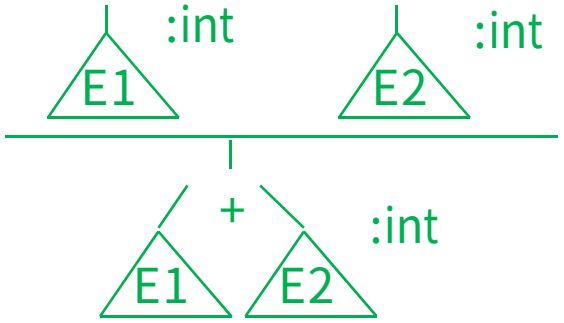
- 일반적인 추론 (inference) 규칙



"모든 E, S1, S2, A, T에 대해 가정이 성립하면 결론이 성립한다."

Proof Tree (타입 유도 트리)

$$\frac{A \vdash E1 : int \quad A \vdash E2 : int}{A \vdash E1 + E2 : int}$$



- 앞서 예에서 S1과 S2가 true 인지도 확인해야함
  - 반복됨, 결국 트리 형태 ➔ proof tree
- 만족하는 proof tree 가 있다 == 타입 오류가 없다
- 예)

$$\frac{\dfrac{A1 \vdash b : bool}{A1 \vdash !b : bool} \qquad \dfrac{A1 \vdash 2 : int \qquad A1 \vdash 3 : int}{A1 \vdash 2 + 3 : int} \qquad A1 \vdash x : int}{b : bool, x : int \vdash \text{if } (!b)\ 2 + 3 \text{ else } x : int}$$

Class Problem

- 다음과 같은 언어에 대해

  t ::= true
  | false
  | if t then t else t
  | 0
  | succ t
  | pred t
  | iszero t

- 다음 문장의 타입을 트리로 유도하시오.
  (1) if iszero 0 then 0 else pred 0 : int
  (2) pred(succ(iszero(succ(pred(0))))) : int

- 다음과 같은 타입 규칙이 있을 때

  0 : int
  true : bool
  false : bool

  ---

  t1: bool t2: T t3 : T
  if t1 then t2 else t3 : T

  ---

  t1 : int
  succ t1 : int

  ---

  t1 : int
  pred t1 : int

  ---

  t1 : int
  iszero t1 : bool

Assignment Statements

$$\frac{id : T \in A \qquad A \vdash E : T}{A \vdash id = E : T}$$ (variable-assign)

$$\frac{A \vdash E3 : T \qquad A \vdash E2 : int \qquad A \vdash E1 : array[T]}{A \vdash E1[E2] = E3 : T}$$ (array-assign)

If Statements

- if 는 각 절을 수행한 결과값의 타입

$$\frac{A \vdash E : \text{bool} \quad A \vdash S1 : T \quad A \vdash S2 : T}{A \vdash \text{if (E) S1 else S2} : T} \text{(if-then-else)}$$

- else 가 없으면 값을 부여하지 않는다

$$\frac{A \vdash E : \text{bool} \quad A \vdash S : T}{A \vdash \text{if (E) S} : \text{unit}} \text{(if-then)}$$

Class Problem

다음을 위한 추론 규칙을 정의해보시오.

1. while (E) S

2. Type id = E

3. E1 ? S1 : S2

Sequence Statements

- 첫번째 문장이 well typed 이고 그 나머지 문장이 well typed 이면 전체 문장도 well typed
  - 그럼 그 나머지 문장은? Recursive 한 (list 기반) 정의

$$\frac{A \vdash S1 : T1 \qquad A \vdash (S2; \ldots; Sn) : Tn}{A \vdash (S1; S2; \ldots; Sn) : Tn}\ \text{(sequence)}$$

선언

= unit if no E

$$\frac{A \vdash T\ \ id[\ =\ E\ ] : T1 \qquad A, id : T \vdash\ (S2;\ ....\ ;\ Sn) : Tn}{A \vdash\ (T\ \ id\ [\ =\ E\ ];\ S2;\ ....\ ;\ Sn) : Tn}\ \text{(declaration)}$$

선언의 타입은
symbol table에 entry를 삽입하는 과정을 유발

Function

- 선언
  - A 가 함수가 정의될 때 상황이라고 할 때,

$$\frac{A, a1 : T1 , \ldots , an : Tn \vdash E : Tr}{A \vdash \text{fun } (T1\ a1, \ldots , Tn\ an) = E \ : T1 \times T2 \times \ldots Tn \rightarrow Tr}$$

- 호출

$$\frac{A \vdash E : T1 \times T2 \times \ldots Tn \rightarrow Tr \qquad A \vdash Ei : Ti \quad (i \in 1 \ldots n)}{A \vdash E(E1, \ldots, En) : Tr} \text{ (function-call)}$$

Function (cont ')

- 리턴 (return)
  - 아래식은 너무 간단 (리턴이 제대로 되었나 알 수 없음)

$$\frac{A \vdash E : T}{A \vdash \text{return } E : \text{unit}}$$ (return-좀 부족)

  - Symbol table에 함수 리턴 값을 나타내는 새 식별자 (return_fun 등) 하나를 추가로 도입 ..함수 리턴 타입이 그 타입이 되도록 함.
  - 아래와 같이 수정

$$\frac{A \vdash E : T \qquad \text{return\_fun} : T}{A \vdash \text{return } E \ : T}$$ (return)

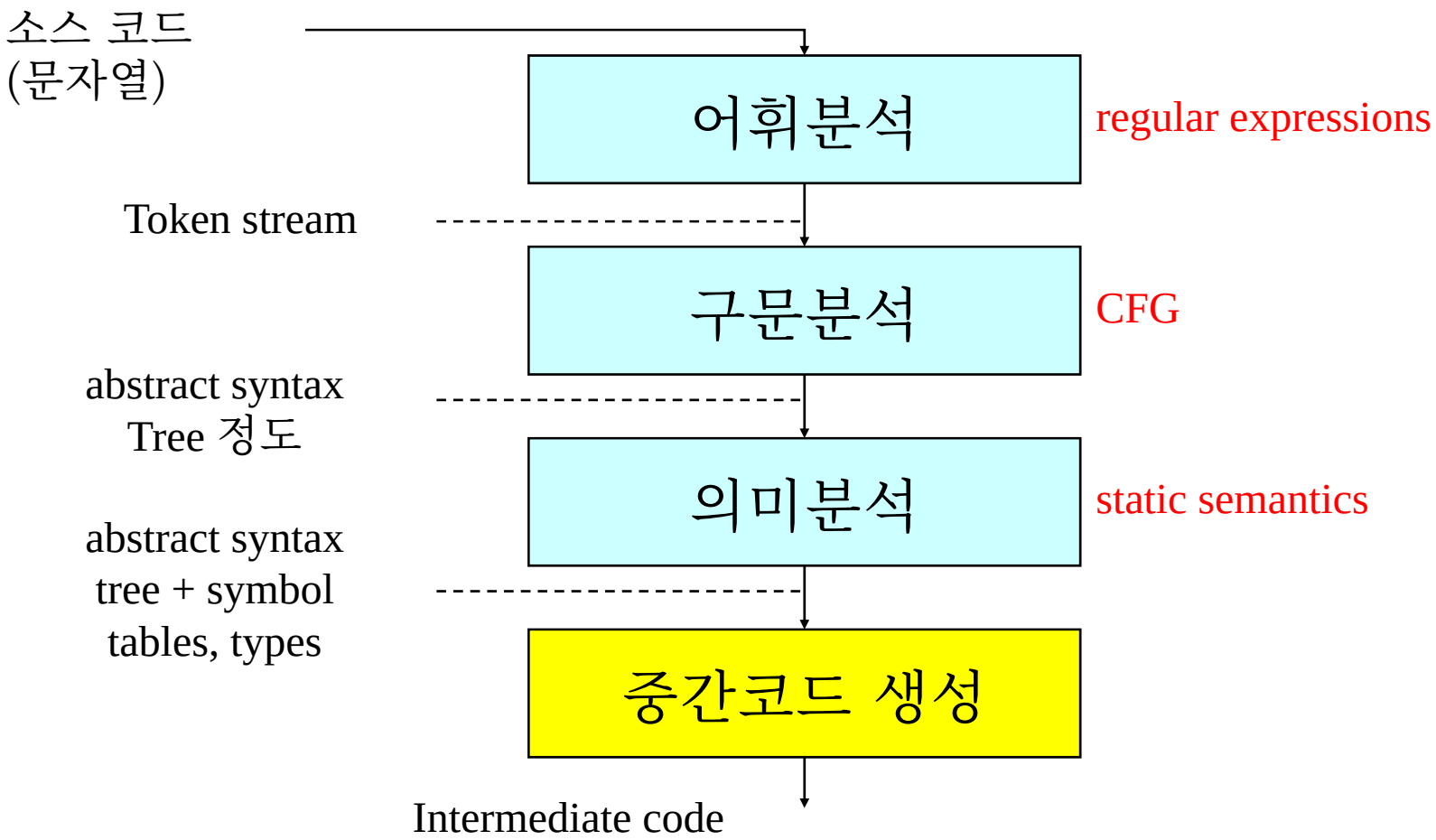
  - 결국 아래식이 검사 됨

$$A, a1 : T1, \ldots, an : Tn, \text{return\_fun} : Tr \vdash E : Tr$$

Control Flow 오류

- Scope와 Type check외에 하나 더
- “Control flow errors”
  - break 나 continue 가 while이나, for 속에 들어 있는지.
  - Goto labels 이 해당 함수 안에 있는지 등
  - AST 를 따라가면 어렵지 않게 알 수 있음.

Where We Are...


소스 코드
(문자열)
어휘분석
regular expressions
Token stream
구문분석
CFG
abstract syntax
Tree 정도
의미분석
static semantics
abstract syntax
tree + symbol
tables, types
중간코드 생성
Intermediate code