SNU 4190.310
Programming Language

Prof. Kwangkeun Yi

ropas.snu.ac.kr/~kwang

컴퓨터공학부

프로그램밍 연구실

ropas.snu.ac.kr

(프로그래밍) 언어 배우기

겉모양 ——————————→ Syntax

– "We use it this way" : 문법 구조

속내용 ——————————→ Semantics

– "It means ..." : 의미 구조

문법과 의미 (겉과 속)이 과연 완전히 구분될 수는 없겠지만 ...

* abstract syntax
* semantic formalisms

To learn :

syntax . 문법 구조

abstract syntax . 문법 구조의 핵심만을 표현하는 방법

semantics . 의미구조

semantic formalisms . 의미구조를 정확히 표현하는 방법

warm-ups . 한번 해 보자 !

프로그래밍 언어를 공부하는 데 필요한 기본기의 첫 스텝

Question !

- How do infintely many programs exist?

- 무한히 많은 프로그램의 문법과 의미를 유한한 한 학기 동안 강의 할 수 있겠습니까?

* For all natural number n

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Can you prove it?

Inductive Definition

귀납적인 정의

"나는 귀납한다
고로 전산학을 한다."

방법 I

/ is natural number

if ★ is nat then /★ is also nat

방법 I'

$$\frac{\cdot}{/\cdot \text{ is nat}} \qquad \frac{\star \text{ is nat}}{/\star \text{ is nat}}$$

Natural number is a set of elements that can be checked by method I

방법 II

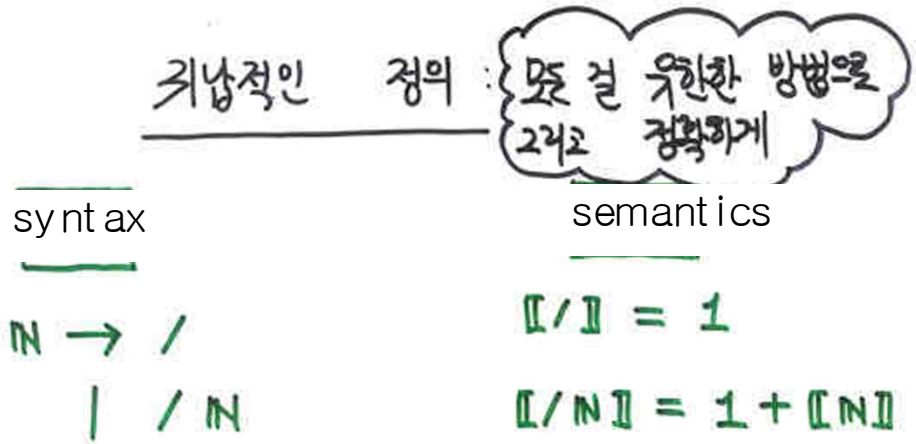
$$N \rightarrow /$$
$$N \rightarrow /N$$

or

$$N \rightarrow / \mid /N$$

Natural number is a set of elements that can be made by method I I

귀납법


귀납적인 정의 : 모든 걸 유한한 방법으로 그리고 정확하게
syntax
semantics
N → /
| / N
[[/]] = 1
[[/N]] = 1 + [[N]]

/

//

///

////

귀납적인 정의 : 모든 걸 유한한 방법으로 그리고 정확하게

syntax

semantics

$\mathbb{N} \to /$ | $/\,\mathbb{N}$

$[\![/]\!] = 1$

$[\![/\mathbb{N}]\!] = 1 + [\![\mathbb{N}]\!]$

$\triangle \to \circ$ | |  | 

$[\![\circ]\!] =$

$[\![$  $]\!] =$

$[\![$  $]\!] =$

$[\![$  $]\!] =$

귀납적인 정의 : 모든 걸 유한한 방법으로 그리고 정확하게

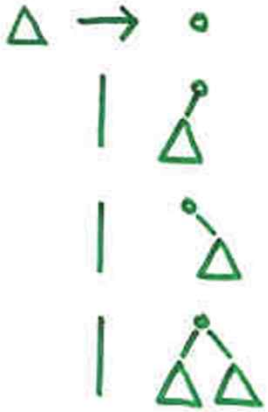
syntax

semantics

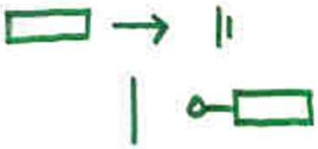
$\mathbb{N} \to /$ | $/\mathbb{N}$

$[\![/]\!] = 1$

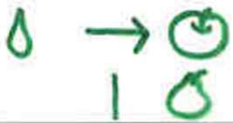
$[\![/\mathbb{N}]\!] = 1 + [\![\mathbb{N}]\!]$



$[\![\circ]\!] =$



$[\![\,|\,]\!] =$



$[\![$  $]\!] =$

$[\![$ $]\!] =$

Programming language is the same

e.g.) Integer expression

0 -1 2

1+2 3*5 6**7 -3-6 7/2

syntax — semantics

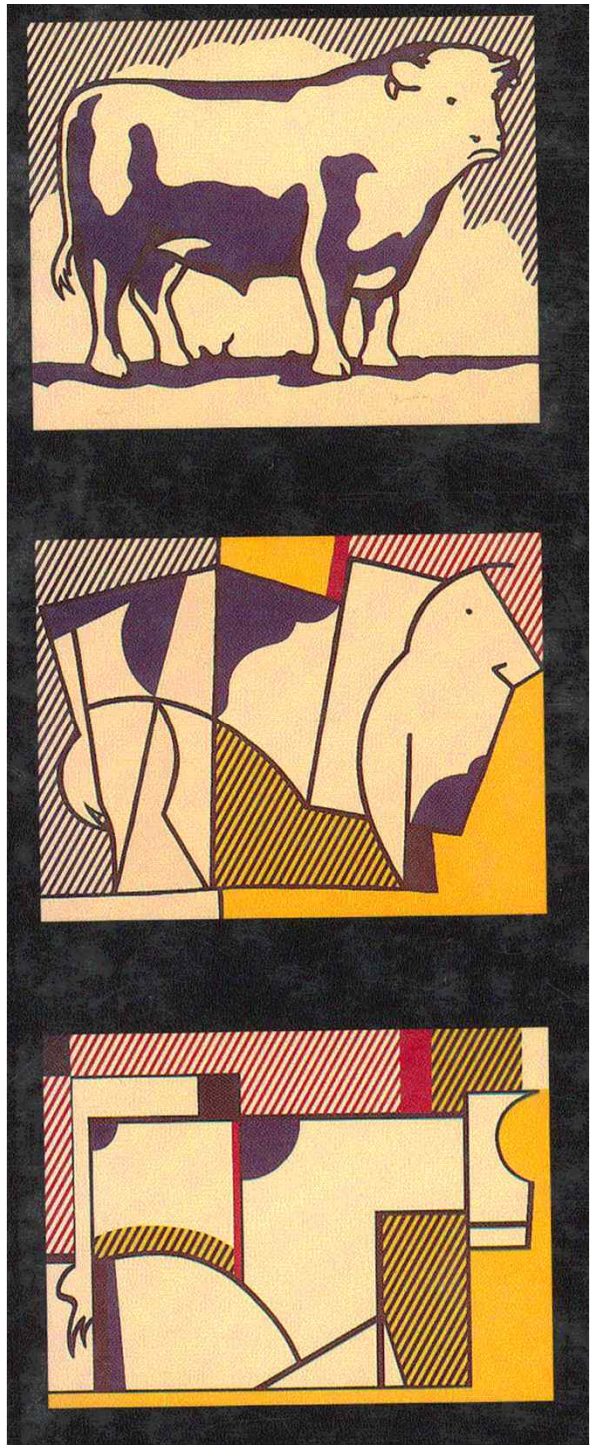
$$
\begin{aligned}
E &\rightarrow \mathbb{Z} \\
&\mid E + E \\
&\mid E - E \\
&\mid E * E \\
&\mid E / E \\
&\mid E ** E
\end{aligned}
$$

Abstract Syntax

- 문장을 만드는 방법
- 프로램을 만드는 방법

왜, 무엇때문에,
abstract syntax라고
하는 것인가?

- abstract syntax is the tool for the speaker
- "concrete syntax" is the tool for the listener

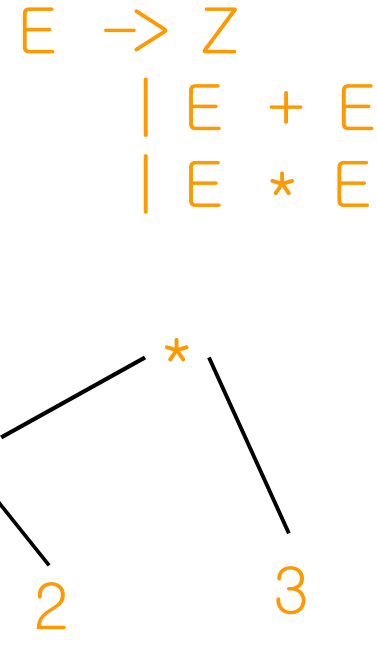


concrete

abstract

Abstract Syntax

- specifies how to construct sentences (programs)
- no more no less
- hence "abstract"



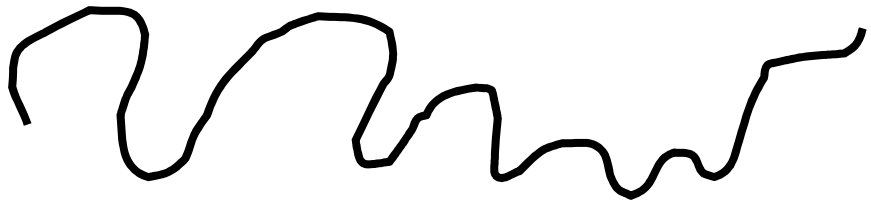
"1더하기 2하고, 결과에 3곱하기"

Concrete Syntax

- tells how to see sentences
- tells how to **re-construct** the abstract syntax tree from the strings!

"1+2*3"

1차원의 실



이 실에서 부터 말한 사람이
머리속에 구성했던 문장의 구조를
어떻게 재구성하나?

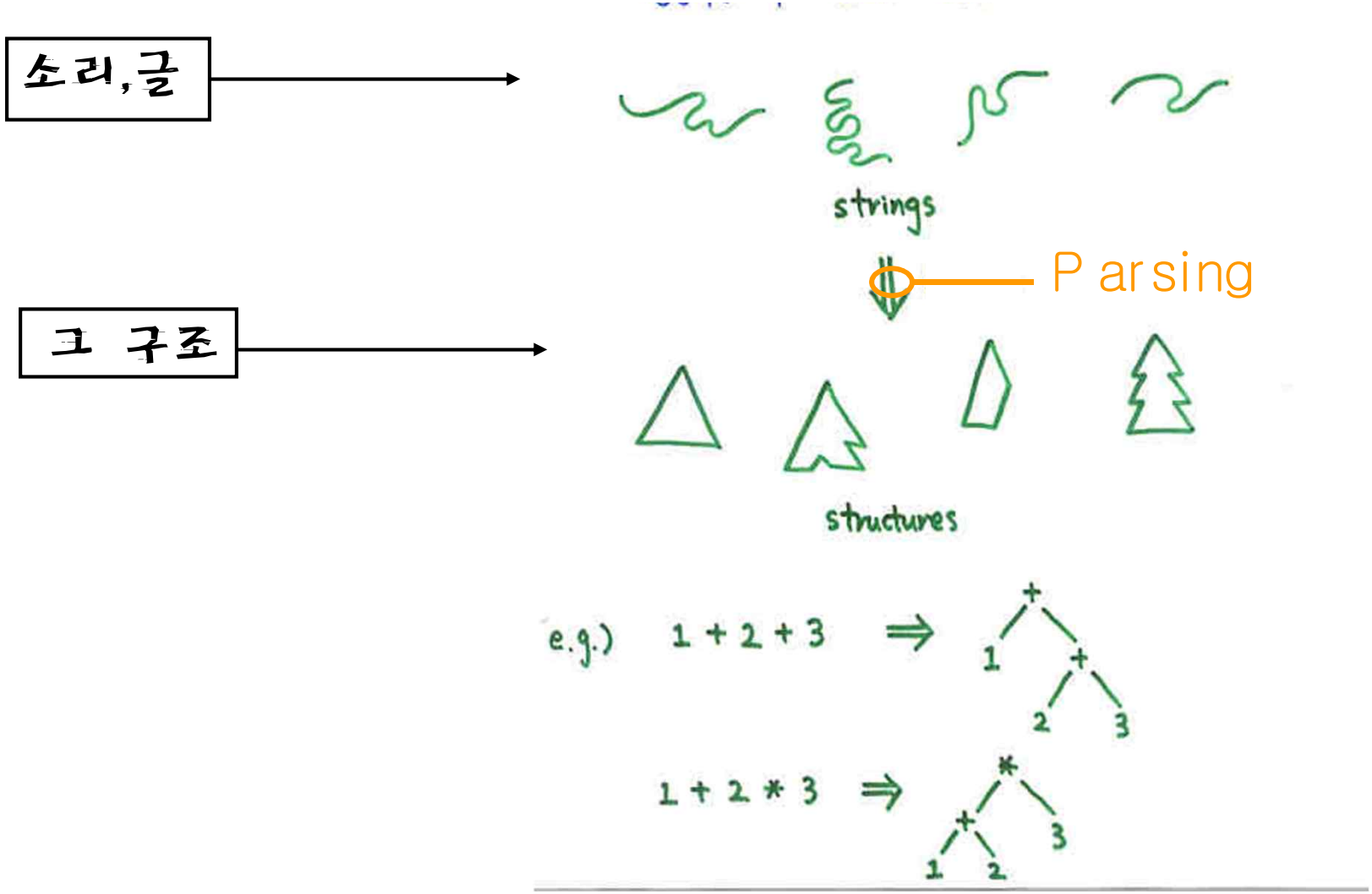
Concrete Syntax tells you how.

E -> Z | E+T | T+E
T -> Z | T*T

Concrete Syntax

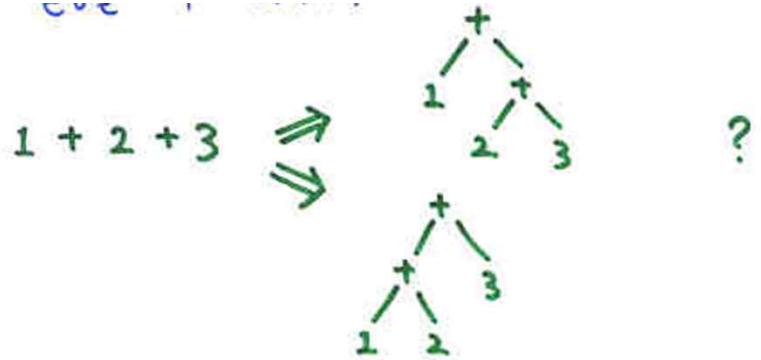
구체적인 문법

Syntax for finding out the syntax structure in a 1-line program

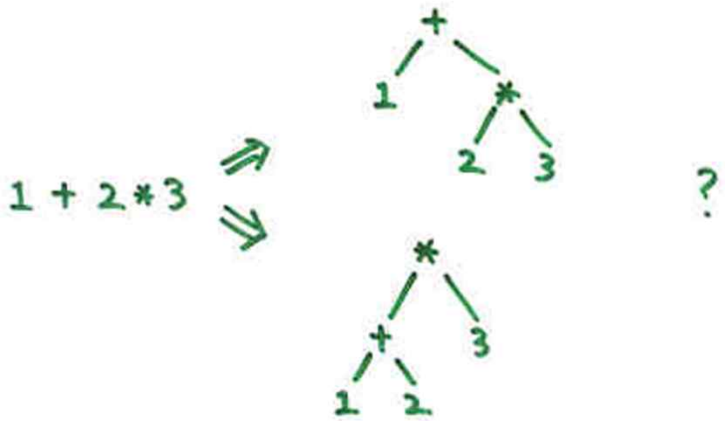


Concrete Syntax

1-line program should be able to be determined without confusing



방향성 associativity



우선순위 precedence

Abstract Syntax

Just represent core structure.

Integer expression

정수식을 만드는 방법.

$$E \to \mathbb{Z} \mid E + E \mid E * E$$

- E represents a set.
- Each production rule defines a function from a set to a set.

$$K_n : \mathbb{Z} \to E$$
$$K_+ : E \times E \to E$$
$$K_* : E \times E \to E$$

정수식을 만드는 방법 3가지.
2 세가지의 타입.
2 이외에는 알고 싶지 않다.
= 문법구조의 핵심. 상위레벨

The set E is $\bigcup_{i=0} E_i$

$$E_0 = \emptyset$$
$$E_1 = \{K_n(i) \mid i \in \mathbb{Z}\} \cup \{K_+(e,e') \mid e \in E_0, e' \in E_0\} \cup \{K_*(e,e') \mid e \in E_0, e' \in E_0\}$$
$$\vdots$$
$$E_k = \{K_n(i) \mid i \in \mathbb{Z}\} \cup \{K_+(e,e') \mid e \in E_{k-1}, e' \in E_{k-1}\} \cup \{K_*(e,e') \mid e \in E_{k-1}, e' \in E_{k-1}\}$$
$$\vdots$$

프로그램의 문법구조 표현방식

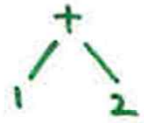
방식 I

$K_+(K_n(1), K_n(2))$

$K_+(K_n(1), K_*(K_n(2), K_n(3)))$

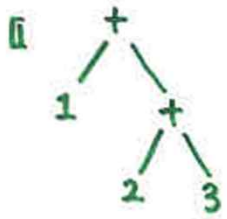
$K_*(K_n(1), K_+(K_n(2), K_n(3)))$

방식 II



Semantics

Once the syntax of a program is determined, Semantics should be determined without confusing

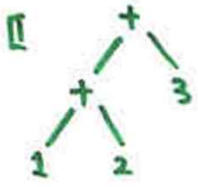


$$[\![\;]\!] = [\![1]\!] + [\![\;]\!] \quad \text{by def.}$$

$$= 1 + ([\![2]\!] + [\![3]\!]) \quad \text{by def.}$$

$$= 1 + (2 + 3) \quad \text{by def.}$$

$$= 6 \quad \text{by def.}$$



$$[\![\;]\!] = [\![\;]\!] + [\![3]\!] \quad \text{by def.}$$

$$= ([\![1]\!] + [\![2]\!]) + 3 \quad \text{by def.}$$

$$= (1 + 2) + 3 \quad \text{by def.}$$

$$= 6 \quad \text{by def.}$$

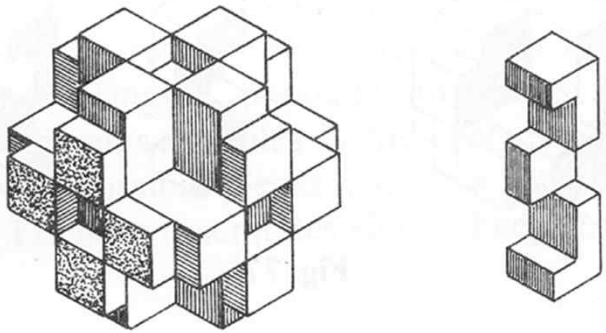


Fig. 75

Propositional Logic : Abstract Syntax & Semantics

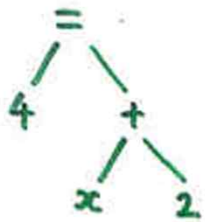
Expressions

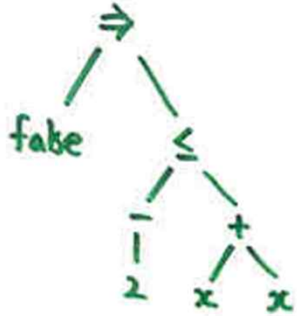
$$
\begin{aligned}
E &\rightarrow \mathbb{Z} \\
&\mid -E \\
&\mid E + E
\end{aligned}
$$

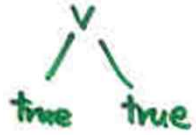
Assertions

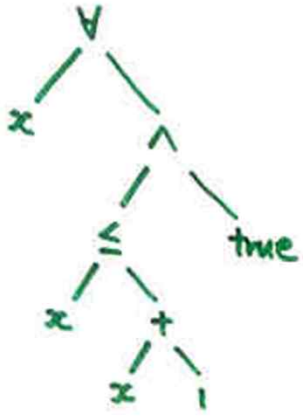
$$
\begin{aligned}
A &\rightarrow \text{true} \\
&\mid \text{false} \\
&\mid E = E \\
&\mid E \leq E \\
&\mid \neg A \\
&\mid A \wedge A \\
&\mid A \vee A \\
&\mid A \Rightarrow A
\end{aligned}
$$

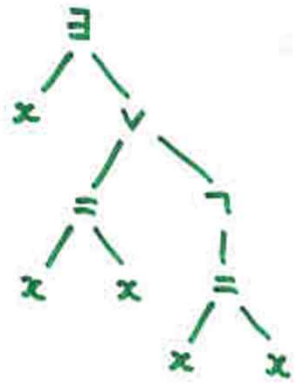
e.g.)











inductive semantic definition

Semantics (Propositional Logic Formula)

$[\![E]\!]$ = as before

$[\![\text{true}]\!]$ = T

$[\![\text{fake}]\!]$ = F

$[\![E_1 = E_2]\!]$ = $[\![E_1]\!]$ equal $[\![E_2]\!]$

$[\![E_1 \leq E_2]\!]$ = ($[\![E_1]\!]$ less $[\![E_2]\!]$)
or ($[\![E_1]\!]$ equal $[\![E_2]\!]$)

$[\![\neg A]\!]$ = not $[\![A]\!]$

$[\![A_1 \wedge A_2]\!]$ = $[\![A_1]\!]$ andalso $[\![A_2]\!]$

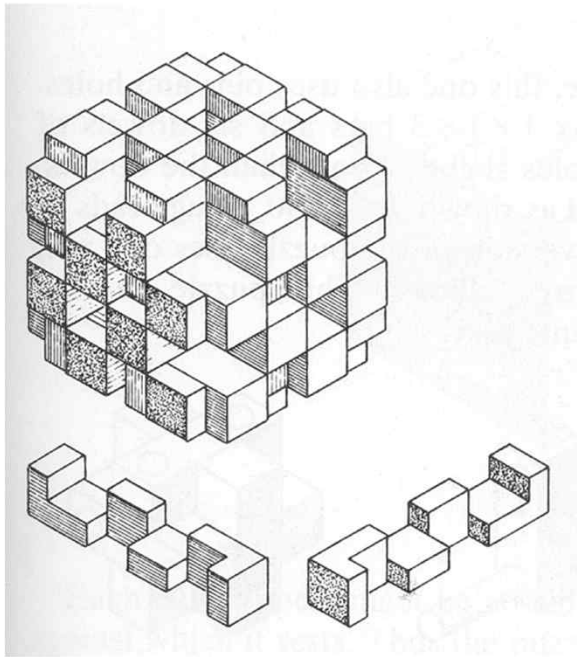
$[\![A_1 \vee A_2]\!]$ = $[\![A_1]\!]$ orelse $[\![A_2]\!]$

$[\![A_1 \Rightarrow A_2]\!]$ = $[\![A_1]\!]$ implies $[\![A_2]\!]$

$[\![\cdot]\!] \in \text{Expr} \to \text{Int}$

$[\![\cdot]\!] \in \text{Assertion} \to \text{Bool}$

$[\![e]\!] \approx \text{eval}(e)$



Predicate Logic : Abstract Syntax & Semantics

Expressions

$$E \rightarrow \mathbb{Z} \mid Id \mid -E \mid E + E$$

Assertions

$$A \rightarrow true \mid false \mid E = E \mid E \leq E \mid \neg A \mid A \wedge A \mid A \vee A \mid A \Rightarrow A \mid \forall_{Id}.\, A \mid \exists_{Id}.\, A$$

again,
and always,

inductive
semantic
definition

Semantics (Predicate Logic Formula)

주의 constant 가 아닌 이름들이
프로그램 텍스트에 있습니다.
(formula)

$[\![\cdot]\!] \in Expv \times Env \to Int$

$[\![\cdot]\!] \in Assertions \times Env \to Bool$

$\sigma \in Env = Id \to Int$

$[\![n]\!]\sigma = n$

$[\![-E]\!]\sigma = neg([\![E]\!]\sigma)$

$[\![E_1 + E_2]\!]\sigma = ([\![E_1]\!]\sigma)\ add\ ([\![E_2]\!]\sigma)$

$[\![x]\!]\sigma = \sigma(x)$

$[\![true]\!]\sigma = T \qquad [\![false]\!]\sigma = F$

$[\![E_1 = E_2]\!]\sigma = ([\![E_1]\!]\sigma)\ equal\ ([\![E_2]\!]\sigma)$

$\vdots$

$[\![\forall x.A]\!]\sigma = [\![A]\!]\sigma[x \mapsto 0]\ andalso$
$[\![A]\!]\sigma[x \mapsto 1]\ andalso$
$\vdots$

$[\![\exists x.A]\!]\sigma = [\![A]\!]\sigma[x \mapsto 0]\ orelse$
$[\![A]\!]\sigma[x \mapsto 1]\ orelse$
$\vdots$