

プログラミング言語の基礎概念

導出システム ... プログラミング言語の意味論や型システムを
derivation system

semantics

記述するための枠組み

自然演繹 (数理論理学で形式的証明を記述するための) や
シーケント計算といった体系を一般化した汎用的な枠組み

議論の対象 (論理式、プログラム、型など) に対する
様々な判断を推論規則に従って導くための記述体系

判断の形式とそれに対する推論規則群を
与えることによって定められる

例) 自然数の大小比較を対象とした導出システム

判断の例: 3 は 5 より小さい

推論規則の例: 任意の自然数 n_1, n_2, n_3 について,
 n_1 が n_2 より小さく、かつ n_2 が n_3 より小さいならば、
 n_1 は n_3 より小さい

1.1 自然数の加算・乗算: 導出システム Nat

定義 1.1 導出システム Nat で我々が扱う判断の一般的な形式は
 n_1, n_2, n_3 をペアノ自然数

- n_1 plus n_2 is n_3
- n_1 times n_2 is n_3

のいずれかとする。

※ 形式上は正しいが内容的には正しくない判断も考えてよい

例) $S(S(z))$ plus $S(S(S(z)))$ is $S(z)$

今後、単に「判断」といったら、形式的に正しい判断を指す

定義 1.2 判断 " n_1 plus n_2 is n_3 " のための推論規則は
次のふたつである

規則 P-ZERO: 任意のペアノ自然数 n に対して、
 n plus n_2 is n_3 を導いてよい。

規則 P-SUCC: 任意のペアノ自然数 n_1, n_2, n_3 に対して、
 n_1 plus n_2 is n_3 ならば
 $S(n_1)$ plus n_2 is $S(n_3)$ を導いてよい。

規則 P-SUCC のような、「 J_1 ならば J_2 」という形の、^{premise} 前提のある推論規則は、前提となる判断 J_1 が既に得られている場合に限り J_2 の判断を導いてよい

定義 1.3 判断 " n_1 times n_2 is n_3 " のための推論規則は以下のふたつ

規則 T-ZERO: z times n is z ← 任意のペア/自然数
 規則 T-SUCC: n_1 times n_2 is n_3 かつ n_2 plus n_3 is n_4 ならば、 $S(n_1)$ times n_2 is n_4

$$\bullet n_1 \times n_2 = n_3$$

$$\bullet n_2 + n_3 = n_4$$

$$\boxed{S(n_1) \times n_2 = n_4}$$

$$n_2 + (n_1 \times n_2) = n_4$$

$$n_2 \frac{(n_1 + 1)}{S(n_1)} = n_4$$

$S(S(z))$ times $S(S(z))$ is $S(S(S(S(z))))$

規則 T-ZERO より z times $S(S(z))$ is $z \dots$ ①

"

z plus z is z

P-SUCC $\rightarrow S(z)$ plus z is $S(z)$

P-SUCC $\rightarrow S(S(z))$ plus z is $S(S(z)) \dots$ ②

規則 T-SUCC、①、② より $S(z)$ times $S(S(z))$ is $S(S(z))$

規則 P-ZERO より z plus $S(S(z))$ is $S(S(z)) \dots$ ③

P-SUCC $\rightarrow S(z)$ plus $S(S(z))$ is $S(S(S(z)))$

P-SUCC $\rightarrow S(S(z))$ plus $S(S(z))$ is $S(S(S(S(z)))) \dots$ ④

規則 T-SUCC、③、④ より $S(S(z))$ times $S(S(z))$ is

$S(S(S(S(z))))$ 終

規則の適用によって判断を導く過程を 導出 と呼ぶ。
 derivation

練習 1.2 (1) $S(S(S(z)))$ plus $S(z)$ is $S(S(S(S(z))))$

$n = S(z)$ とすると P-ZERO より z plus $S(z)$ is $S(z)$

P-SUCC より $S(z)$ plus $S(z)$ is $S(S(z))$

P-SUCC より $S(S(z))$ plus $S(z)$ is $S(S(S(z)))$

P-SUCC より $S(S(S(z)))$ plus $S(z)$ is $S(S(S(S(z))))$

プログラミング言語の基礎概念

(2) $S(z)$ plus $S(S(S(z)))$ is $S(S(S(S(z))))$ $n = S(S(S(z)))$ と P-ZERO より z plus $S(S(S(z)))$ is $S(S(S(z)))$ P-SUCC より $S(z)$ plus $S(S(S(z)))$ is $S(S(S(S(z))))$

(3) $\left(\begin{array}{l} \frac{S(S(S(z)))}{n_1} \text{ times } \frac{z}{n_2} \text{ is } \frac{z}{n_4} \\ \frac{S(S(z))}{n_1} \text{ times } \frac{z}{n_2} \text{ is } \frac{z}{n_3} \\ \frac{z}{n_2} \text{ plus } \frac{z}{n_3} \text{ is } \frac{z}{n_4} \end{array} \right)$

 $n = z$ とすると P-ZERO より z plus z is $z \dots$ ①

$\left(\begin{array}{l} S(z) \text{ times } z \text{ is } z \\ z \text{ times } z \text{ is } z \text{ (} n = z \text{)} \end{array} \right)$

 $n = z$ とすると T-ZERO より z times z is $z \dots$ ②規則 T-SUCC, ①, ② より $S(z)$ times z is $z \dots$ ③規則 T-SUCC, ①, ③ より $S(S(z))$ times z is $z \dots$ ④規則 T-SUCC, ①, ④ より $S(S(S(z)))$ times z is z 終

練習 1.3 判断“ n_1 plus n_2 is n_3 ”が導出できる時、
その導出には何回の規則の適用が必要か?

1回目の適用
P-ZERO z plus $\overset{n_2}{\underbrace{(n)}} \text{ is } \overset{n_2}{\underbrace{(n)}} \quad \underline{(n_1 + 1) \text{ 回}}$

“ n_1 times n_2 is n_3 ”についてはどうか。

$(n_1 + 1)$ 回、T-SUCC (最後の1回だけはT-ZERO)を適用する
T-SUCCを適用することにより、 $(n_2 + 1)$ 回の適用が必要になる

T-SUCC & T-ZERO

 $(n_1 + 1) + (n_2 + 1) \times (n_1 + 1)$

1.2 推論規則と導出の記法

規則 X: J_1 かつ ... かつ J_n ならば J_0
 $\frac{J_1 \dots J_n}{J_0}$

※ 前提のない規則の場合 $n = 0$

$$(P-ZERO) \quad \overline{z \text{ plus } n \text{ is } n}$$

$$(P-SUCC) \quad \frac{n_1 \text{ plus } n_2 \text{ is } n}{S(n_1) \text{ plus } n_2 \text{ is } S(n)}$$

$$(T-ZERO) \quad \overline{z \text{ times } n \text{ is } z}$$

$$(T-SUCC) \quad \frac{n_1 \text{ times } n_2 \text{ is } n_3 \quad n_2 \text{ plus } n_3 \text{ is } n_4}{S(n_1) \text{ times } n_2 \text{ is } n_4}$$

derivation tree

導出木 ... 判断をノード、結論となる判断を根とする木構造

$$\begin{array}{l} S(S(S(z))) \text{ plus } S(z) \text{ is } S(S(S(S(z)))) \text{ by } P-SUCC \{ \\ \quad S(S(z)) \text{ plus } S(z) \text{ is } S(S(S(z))) \text{ by } P-SUCC \{ \\ \quad \quad S(z) \text{ plus } S(z) \text{ is } S(S(z)) \text{ by } P-SUCC \{ \\ \quad \quad \quad z \text{ plus } S(z) \text{ is } S(z) \text{ by } P-ZERO \{ \} \\ \quad \quad \} \\ \quad \} \\ \} \end{array}$$

$$\begin{array}{c} \overline{z \text{ plus } S(z) \text{ is } S(z)} \text{ } P-ZERO \\ \hline \overline{S(z) \text{ plus } S(z) \text{ is } S(S(z))} \text{ } P-SUCC \\ \hline \overline{S(S(z)) \text{ plus } S(z) \text{ is } S(S(S(z)))} \text{ } P-SUCC \\ \hline \overline{S(S(S(z))) \text{ plus } S(z) \text{ is } S(S(S(S(z))))} \text{ } P-SUCC \end{array}$$

$$\begin{array}{l} (3) \quad S(S(S(z))) \text{ times } z \text{ is } z \text{ by } T-SUCC \{ \\ \quad S(S(z)) \text{ times } z \text{ is } z \text{ by } T-SUCC \{ \\ \quad \quad S(z) \text{ times } z \text{ is } z \text{ by } T-SUCC \{ \\ \quad \quad \quad z \text{ times } z \text{ is } z \text{ by } T-ZERO \{ \} \\ \quad \quad \} ; \quad z \text{ plus } z \text{ is } z \text{ by } P-ZERO \{ \} \\ \quad \quad z \text{ plus } z \text{ is } z \text{ by } P-ZERO \{ \} \\ \quad \} ; \\ \quad z \text{ plus } z \text{ is } z \text{ by } P-ZERO \{ \} \\ \} \end{array}$$

プログラミング言語の基礎概念

Compare Nat 1

$$\frac{}{n \text{ is less than } S(n)} \quad (L-Succ)$$

$$\frac{n_1 \text{ is less than } n_2 \quad n_2 \text{ is less than } n_3}{n_1 \text{ is less than } n_3} \quad (L-TRANS)$$

練習問題 1.5 (1) z is less than $S(S(z))$

$$\frac{z \text{ is less than } S(z) \quad \frac{}{S(z) \text{ is less than } S(S(z))} \quad (L-Succ)}{z \text{ is less than } S(S(z))} \quad (L-TRANS)$$

Compare Nat 2

$$\frac{}{z \text{ is less than } S(n)} \quad (L-ZERO)$$

$$\frac{n_1 \text{ is less than } n_2}{S(n_1) \text{ is less than } S(n_2)} \quad (L-Succ Succ)$$

1.5 (1) z is less than $S(S(z))$ $L-ZERO$

Compare Nat 3

$$\frac{}{n \text{ is less than } S(n)} \quad (L-Succ)$$

$$\frac{n_1 \text{ is less than } n_2}{n_1 \text{ is less than } S(n_2)} \quad (L-Succ R)$$

$$\frac{1.5 (1) \quad \frac{z \text{ is less than } S(z)}{z \text{ is less than } S(S(z))} \quad (L-Succ)}{z \text{ is less than } S(S(z))} \quad (L-Succ R)$$

- Nat を使って定義された加算が交換法則を満たすことを示す
 - Eval Nat Exp を使って定義された評価は、与えられた算術式に対して、その値を一意的に定める
- といった、具体的な導出システムにおける、判断についての一般的な性質を考える

→ このような、判断についての一般的な性質 (のうち数学的に証明

されたものを **メタ定理** と呼ぶ
meta theorem

帰納法 ここではメタ定理の
証明に用いる
induction

操作的意味論 ... 評価・簡約を使った意味論
operational semantics (ここでは算術式に意味を与えている)

ある判断が導出できているかどうかは、単なる記号操作の問題

Σ plus n is $S(n)$ (P-WRONG) といった規則を考えれば

Σ plus $S(\Sigma)$ is $S(S(\Sigma))$ という内容的には誤った判断を
P-ZEROしかないような導出システムでは導出できるが、
内容的には正しいはずの $S(\Sigma)$ plus $S(\Sigma)$ is $S(S(\Sigma))$ を

導出することができない
導出システムを「何らかの概念を表現した理論」と考えたとき、
こういった導出システムの「出来」を考察した理論は **メタ理論**
(理論に関する理論) である。

健全性 ... 導出された判断が全て内容的に正しいという性質

完全性 ... 内容的に正しい判断は全て導出できるという性質

構造帰納法 $\mathbb{N} \times \mathbb{N}$ / 自然数は自然数と1対1対応

- (1) 任意の $\mathbb{N} \times \mathbb{N}$ 自然数 n に対し、 Σ plus n is n
- (2) 任意の $\mathbb{N} \times \mathbb{N}$ 自然数 n に対し、 n plus Σ is n

(1) 規則 P-ZERO により、 n が実際にどんな $\mathbb{N} \times \mathbb{N}$ 自然数であか
によらず Σ plus n is n P-ZERO という導出木が作れる

(2) $\mathbb{N} \times \mathbb{N}$ 自然数に関する帰納法より、以下の2つを証明すればよい

(a) Σ plus Σ is Σ が導出できる

(b) 任意の $\mathbb{N} \times \mathbb{N}$ 自然数 k について k plus Σ is k が
導出できるなら、 $S(k)$ plus Σ is $S(k)$ が導出できる

※ここでは $P(n)$ は「判断 n plus Σ is n が導出できる」

(証明) (a) は P-ZERO を使えば導出できる

(b) k plus Σ is k の導出を

$\stackrel{\text{def}}{D}$

k plus Σ is k とすると、