ソフトウェア基礎論配布資料 (2)
算術式の言語

五十嵐 淳
京都大学 大学院情報学研究科知能情報学専攻
e-mail: igarashi@kuis.kyoto-u.ac.jp

平成 17 年 10 月 16 日

1 算術式の文法定義

自然数上の加算・乗算式の集合を算術式の集合 **Aexp** を定義する．構成要素は最小の自然数 `Z`, 「次の数」を示す `S`, 加算・乗算の `+`, `*` である．算術式は無限に存在するので，集合を厳密に定義するためには，要素を列挙するわけにはいかない．以下ではそのような集合を定義する方法をいくつか見る．

1.1 BNF 記法による定義

BNF 記法(*Backus-Naur form*) は，プログラミング言語の文法を定義する際の標準的な記法である．以下に実際の例を示すが,「::=」は「〜は以下のもので構成される」,「|」は「または」と読む.

1.1.1 定義: 算術式 (メタ変数 a) の集合 **Aexp** は以下の文法[1]で定義する.

$$a \in \mathbf{Aexp} ::= \texttt{Z} \mid \texttt{S(}a\texttt{)} \mid a \texttt{ + } a \mid a \texttt{ * } a$$

□

演算子の優先順位と抽象構文木　上の文法から，例えば `S(Z)` や `S(Z) + Z` は **Aexp** の要素である．これらを組み合わせて，`S(Z) + Z * S(Z)` も算術式である．しかし，`S(Z)` と `Z * S(Z)` を `+` で組み合わせても，(文字列として) 見た目が同じ算術式が得られる．別の言い方をすると，上の文法は，文字列からそれに対応する算術式が一意に得られないので曖昧である．より厳密には，上の文法は算術式を木構造として定義していると考えるのが正しい.

[1]細かい違いだが $a \in \mathbf{Aexp} ::= \texttt{Z} \mid \texttt{S(}\mathbf{Aexp}\texttt{)} \mid \mathbf{Aexp} \texttt{ + } \mathbf{Aexp} \mid \mathbf{Aexp} \texttt{ * } \mathbf{Aexp}$ と記述する方法もある.

このような木構造は言語処理系で**抽象構文木**(*abstract syntax tree*) と呼ばれるものである．また，このような (文字列の解析のためではない) 文法を**抽象構文**(*abstract syntax*) と呼ぶ．

もちろん，上の例のような「生成の仕方」の違いを区別する必要はでてくるので適宜 (メタな記号である) “(” “)” を使用し，(S(Z) + Z) * S(Z) などと記述する．以下では，木構造として「等しい」ことを示すために $\equiv$ という記号を用いる．また，記号の優先度を指定して括弧を省略していく．ここでは，メタ言語の常識に合わせて * は + よりも強く結合し，+, * ともに左結合するとする．よって，

$$\begin{array}{rcrcl} \texttt{S(Z) + Z * S(Z)} & \equiv & \texttt{S(Z) + (Z * S(Z))} & \not\equiv & \texttt{(S(Z) + Z) * S(Z)} \\ \texttt{Z + Z + Z} & \equiv & \texttt{(Z + Z) + Z} & \not\equiv & \texttt{Z + (Z + Z)} \\ \texttt{Z * Z * Z} & \equiv & \texttt{(Z * Z) * Z} & \not\equiv & \texttt{Z * (Z * Z)} \end{array}$$

である．

1.2 帰納的な定義

上の BNF 記法は，より厳密には以下のような**帰納的定義**(*inductive definition*) の簡潔な記法と考えられる．

1.2.1 定義: 算術式の集合 **Aexp** は，以下の条件を満たす集合 A のうち最小のものである．

1. $\{\texttt{Z}\} \subseteq A$
2. もし $a \in A$ ならば $\{\texttt{S(}a\texttt{)}\} \subseteq A$
3. もし $a_1 \in A$ かつ $a_2 \in A$ ならば $\{a_1\ \texttt{+}\ a_2, a_1\ \texttt{*}\ a_2\} \subseteq A$

□

「最小の」というのは，上の条件を満たす集合はいくらでも考えられるので,「ゴミ」が入っていないことを保証するための条件である．例えば，上の条件を満たす集合があったときに，1, S(1), 1+1, . . . などを加えたような集合も，同じ条件を満たすので，最小性を言うことは非常に大事である．BNF 記法での定義ではわざわざ言わないことが多い．

また，上の「定義」は **Aexp** の満たすべき条件のみが記述されており，その存在は自明ではない[2]が，このような集合が存在することも，証明することができる．

1.3 帰納法による証明・帰納法による関数定義

自然数 (を表現する算術式) の帰納的な定義

$$n \in \mathbf{Nv} ::= \texttt{Z} \mid \texttt{S(}n\texttt{)}$$

[2] このような定義を超越的な定義と呼ぶ

からは以下の数学的帰納法

$$(P(\mathtt{Z}) \,\&\, \forall n \in \mathbf{Nv}.(P(n) \Longrightarrow P(\mathtt{S}(n)))) \Longrightarrow \forall n \in \mathbf{Nv}.P(n)$$

が導かれる．これと同様に帰納的に集合を定義すると，それに対応した**帰納法の証明原理**(*induction principle*) が導かれる．

算術式の構造に関する帰納法　a が算術式ということは，その構成の仕方より以下のどれかが成立する．

1. $a \equiv \mathtt{Z}$.
2. $a \equiv \mathtt{S}(a_0)$ であり，a_0 は a より「小さい」算術式.
3. $a \equiv a_1 \ \mathtt{+}\ a_2$ であり，a_1, a_2 は a より「小さい」算術式.
4. $a \equiv a_1 \ \mathtt{*}\ a_2$ であり，a_1, a_2 は a より「小さい」算術式.

これより算術式の**構造に関する帰納法**(*structural induction*) の原理は以下のようになる．

1.3.1 定理 [算術式の構造に関する帰納法]: $\forall a \in \mathbf{Aexp}.P(a)$ を示すには,

1. $P(\mathtt{Z})$
2. $\forall a \in \mathbf{Aexp}.P(a) \Longrightarrow P(\mathtt{S}(a))$
3. $\forall a_1, a_2 \in \mathbf{Aexp}.P(a_1) \,\&\, P(a_2) \Longrightarrow P(a_1 \ \mathtt{+}\ a_2)$
4. $\forall a_1, a_2 \in \mathbf{Aexp}.P(a_1) \,\&\, P(a_2) \Longrightarrow P(a_1 \ \mathtt{*}\ a_2)$

を示せばよい. □

また，帰納法の原理から,「帰納法による関数定義」が可能になる．例えば，算術式の大きさを示す関数 $size \in \mathbf{Aexp} \to \mathbf{Nat}$ は以下の 4 行だけで定義したことになる (以下の 4 式を満たすような関係はただひとつしかないということが帰納法の原理から証明できる).

$$\begin{aligned}
size(\mathtt{Z}) &= 1 \\
size(\mathtt{S}(a_0)) &= size(a_0) + 1 \\
size(a_1 \ \mathtt{+}\ a_2) &= size(a_1) + size(a_2) + 1 \\
size(a_1 \ \mathtt{*}\ a_2) &= size(a_1) + size(a_2) + 1
\end{aligned}$$

1.3.2 例: $depth \in \mathbf{Aexp} \to \mathbf{Nat}$ を以下のように定義する.

$$\begin{aligned}
depth(\mathtt{Z}) &= 1 \\
depth(\mathtt{S}(a_0)) &= depth(a_0) + 1 \\
depth(a_1 \ \mathtt{+}\ a_2) &= \max(depth(a_1), depth(a_2)) + 1 \\
depth(a_1 \ \mathtt{*}\ a_2) &= \max(depth(a_1), depth(a_2)) + 1
\end{aligned}$$

このとき，$\forall a \in \mathbf{Aexp}.size(a) \leq 2^{depth(a)} - 1$ である.

Proof: 算術式の構造に関する帰納法で証明する.

1. $size(\mathtt{Z}) \leq 2^{depth(\mathtt{Z})} - 1$ は各関数の定義より明らか.
2. $size(a) \leq 2^{depth(a)} - 1$ を仮定して，$size(\mathtt{S}(a)) \leq 2^{depth(\mathtt{S}(a))} - 1$ を示す.

$$\begin{aligned} size(\mathtt{S}(a)) &= size(a) + 1 \\ &\leq 2^{depth(a)} \leq 2^{depth(a)+1} - 1 \\ &\leq 2^{depth(\mathtt{S}(a))} - 1 \end{aligned}$$

3. $size(a_1) \leq 2^{depth(a_1)} - 1$ と $size(a_2) \leq 2^{depth(a_2)} - 1$ を仮定して，$size(a_1 \ \mathtt{+} \ a_2) \leq 2^{depth(a_1 \ \mathtt{+} \ a_2)} - 1$ を示す.

$$\begin{aligned} size(a_1 \ \mathtt{+} \ a_2) &= size(a_1) + size(a_2) + 1 \\ &\leq 2^{depth(a_1)} - 1 + 2^{depth(a_2)} - 1 + 1 \\ &\leq 2^{\max(depth(a_1), depth(a_2))} + 2^{\max(depth(a_1), depth(a_2))} - 1 \\ &\leq 2^{depth(a_1 \ \mathtt{+} \ a_2)} - 1 \end{aligned}$$

4. $size(a_1) \leq 2^{depth(a_1)} - 1$ と $size(a_2) \leq 2^{depth(a_2)} - 1$ を仮定して，$size(a_1 \ \mathtt{*} \ a_2) \leq 2^{depth(a_1 \ \mathtt{*} \ a_2)} - 1$ を示す．(省略)

□

2 規則による定義・判断と導出

帰納的な集合 (とくに関係) の定義を行う際に，特定の形の**規則**(*rule*) を使って記述することがある．具体例として，**Aexp** の定義を規則を使った定義に書き直してみる.

2.1 定義: 算術式の集合 **Aexp** は，以下の規則で定義される.

$$\frac{}{\mathtt{Z} \in \mathbf{Aexp}} \quad \text{(A-Zero)} \qquad \frac{a_1 \in \mathbf{Aexp} \qquad a_2 \in \mathbf{Aexp}}{a_1 \ \mathtt{+} \ a_2 \in \mathbf{Aexp}} \quad \text{(A-Plus)}$$

$$\frac{a \in \mathbf{Aexp}}{\mathtt{S}(a) \in \mathbf{Aexp}} \quad \text{(A-Succ)} \qquad \frac{a_1 \in \mathbf{Aexp} \qquad a_2 \in \mathbf{Aexp}}{a_1 \ \mathtt{*} \ a_2 \in \mathbf{Aexp}} \quad \text{(A-Mult)}$$

□

直感的には，各規則は「線の上の事柄が成立するならば，線の下の事柄が成立する」という意味で，例えば A-Plus は「a_1 が **Aexp** の要素で a_2 が **Aexp** の要素ならば，a_1+a_2 は **Aexp** の要素である」と読める．これを使うと Z + S(Z) $\in$ **Aexp** ということは A-Zero を二回，A-Succ を一回，A-Plus を一回使うことで導くことができる．このように，規則を有限回組合せて $a \in$ **Aexp** が言えるような a の集合を以って **Aexp** を定義したと考えるのが，規則で定義するということである．この規則と BNF 記法の対応は明らかであろう.

2.1 判断と導出

一般的に規則は

$$\frac{J_1 \qquad \cdots \qquad J_n}{J} \qquad \text{(規則名)}$$

という形をしている．ここで J_1 は**判断**(*judgment*) と呼ばれる何らかの事実について述べた文 (もしくは，そういう意味を持つ記号列) である．規則は，判断 $J_1, \ldots, J_n$ から 新しい判断 J を導くものであり，その判断を導く過程を**導出**(*derivation*) という．導出は，しばしば，木構造をもつ**導出木**(*derivation tree*) という形で表現される．例えば，$\texttt{Z + S(Z)} \in \mathbf{Aexp}$ という判断の導出は

$$\dfrac{\dfrac{}{\texttt{Z} \in \mathbf{Aexp}}\ \text{A-Zero} \quad \dfrac{\dfrac{}{\texttt{Z} \in \mathbf{Aexp}}\ \text{A-Zero}}{\texttt{S(Z)} \in \mathbf{Aexp}}\ \text{A-Succ}}{\texttt{Z + S(Z)} \in \mathbf{Aexp}}\ \text{A-Plus}$$

という導出木で表現することができる[3].

厳密にいうと，規則には a などのメタ変数が含まれていることが多く，実際に導出する時には，それらのメタ変数を具体的なもので置換することになる．この具体化する前後の規則を区別して，前者を規則のスキーマ，後者を規則のインスタンスと呼ぶことがある．

2.2 部分式とパス

演習システムでは,「この式のこの部分を表示せよ」といった，式の一部を指定する表記が使われる．規則による定義に慣れるために，$\texttt{sub}\ p\ \texttt{of}\ a\ \texttt{is}\ a'$ という「算術式 a のパス p に位置する部分式は a」であるという判断を導出する規則を考えてみる．ここで**パス**(*path*) は `0` もしくは `1` を有限個並べた文字列であり，ファイルシステムのディレクトリ・パスのように，先頭から項の i 番目の部分式を辿っていく動作を表現する．このことを考えると，規則は以下のように定義できる．

$$\frac{a \in \mathbf{Aexp}}{\texttt{sub}\ \varepsilon\ \texttt{of}\ a\ \texttt{is}\ a} \quad \text{(Sub-Empty)} \qquad \frac{a_1 \in \mathbf{Aexp} \qquad \texttt{sub}\ p\ \texttt{of}\ a_2\ \texttt{is}\ a'}{\texttt{sub}\ \texttt{1}p\ \texttt{of}\ a_1\texttt{+}a_2\ \texttt{is}\ a'} \quad \text{(Sub-PlusR)}$$

$$\frac{\texttt{sub}\ p\ \texttt{of}\ a_0\ \texttt{is}\ a'}{\texttt{sub}\ \texttt{0}p\ \texttt{of}\ \texttt{S(}a_0\texttt{)}\ \texttt{is}\ a'} \quad \text{(Sub-Succ)} \qquad \frac{\texttt{sub}\ p\ \texttt{of}\ a_1\ \texttt{is}\ a' \qquad a_2 \in \mathbf{Aexp}}{\texttt{sub}\ \texttt{0}p\ \texttt{of}\ a_1\texttt{*}a_2\ \texttt{is}\ a'} \quad \text{(Sub-MultL)}$$

$$\frac{\texttt{sub}\ p\ \texttt{of}\ a_1\ \texttt{is}\ a' \qquad a_2 \in \mathbf{Aexp}}{\texttt{sub}\ \texttt{0}p\ \texttt{of}\ a_1\texttt{+}a_2\ \texttt{is}\ a'} \quad \text{(Sub-PlusL)} \qquad \frac{a_1 \in \mathbf{Aexp} \qquad \texttt{sub}\ p\ \texttt{of}\ a_2\ \texttt{is}\ a'}{\texttt{sub}\ \texttt{1}p\ \texttt{of}\ a_1\texttt{*}a_2\ \texttt{is}\ a'} \quad \text{(Sub-MultR)}$$

[3]導出木のことを単に導出と呼ぶことも多い.

$$\frac{}{a_0 \texttt{ + Z} \longrightarrow a_0}\ \text{(E-PlusZero)} \qquad \frac{a_1 \longrightarrow a_1'}{a_1 \texttt{ + } a_2 \longrightarrow a_1' \texttt{ + } a_2}\ \text{(E-PlusL)}$$

$$\frac{}{a_1 \texttt{ + S(}a_2\texttt{)} \longrightarrow \texttt{S(}a_1 \texttt{ + } a_2\texttt{)}}\ \text{(E-PlusSucc)} \qquad \frac{a_2 \longrightarrow a_2'}{a_1 \texttt{ + } a_2 \longrightarrow a_1 \texttt{ + } a_2'}\ \text{(E-PlusR)}$$

$$\frac{}{a_0 \texttt{ * Z} \longrightarrow \texttt{Z}}\ \text{(E-MultZero)}$$

$$\frac{}{a_1 \texttt{ * S(}a_2\texttt{)} \longrightarrow a_1 \texttt{ * } a_2 \texttt{ + } a_1}\ \text{(E-MultSucc)} \qquad \frac{a_1 \longrightarrow a_1'}{a_1 \texttt{ * } a_2 \longrightarrow a_1' \texttt{ * } a_2}\ \text{(E-MultL)}$$

$$\frac{a \longrightarrow a'}{\texttt{S(}a\texttt{)} \longrightarrow \texttt{S(}a'\texttt{)}}\ \text{(E-Succ)} \qquad \frac{a_2 \longrightarrow a_2'}{a_1 \texttt{ * } a_2 \longrightarrow a_1 \texttt{ * } a_2'}\ \text{(E-MultR)}$$

図 1: 簡約規則 (1)

ここでは，長さ 0 のパス (空文字列) を便宜上 ε と記述している．この判断を導出する規則には前提に $a \in \mathbf{Aexp}$ が現れているので，実際には，A-XX も組み合わせて導出することになる．

2.2.1 練習問題: 以下の判断を導出せよ:

1. `sub 01 of (S(S(Z))+Z)*S(Z+S(Z)) is Z`
2. `sub 101 of (S(S(Z))+Z)*S(Z+S(Z)) is S(Z)`

2.2.2 練習問題: $\forall a \in \mathbf{Aexp}.\exists a' \in \mathbf{Aexp}.\texttt{sub } \underbrace{\texttt{0...0}}_{depth(a)-1} \texttt{ of } a \texttt{ is } a'$ を証明せよ．

3 算術式の簡約と性質

3.1 簡約関係の定義

算術式の集合が定義できたところで，算術式を計算する過程を**簡約**(*reduction*) 関係 $\longrightarrow$ ($\subseteq \mathbf{Aexp} \times \mathbf{Aexp}$) として規則を使って定義する．$(a_1, a_2) \in \longrightarrow$ を $a_1 \longrightarrow a_2$ と書く．「簡約」とは式をより簡単な形に変形する，くらいの意味である．

3.1.1 定義: 関係 $\longrightarrow$ は，図 1 の規則で定義される．

3.1.2 練習問題: 以下の判断の導出木を示せ.

$$\texttt{(S(S(Z)) + Z) * S(Z+S(Z))} \longrightarrow \texttt{S(S(Z)) * S(Z+S(Z))}$$
$$\texttt{S(S(Z)) * S(Z+S(Z))} \longrightarrow \texttt{S(S(Z)) * S(S(Z+Z))}$$
$$\texttt{(S(S(Z)) + Z) * S(Z+S(Z))} \longrightarrow \texttt{(S(S(Z)) + Z) * S(S(Z+Z))}$$
$$\texttt{(S(S(Z)) + Z) * S(Z+S(Z))} \longrightarrow \texttt{(S(S(Z)) + Z) * (Z+S(Z)) + S(Z+S(Z))}$$

上の練習問題の例からもわかるように，足し算をするにも数ステップの簡約が必要なように形式化されている.

与えられた式の中で，実際に式の変形が起こりうる部分式 (例えば a_0`+ Z` や a_1`* S(`a_2`)` という形をしているもの) を**簡約基**$(redex)$ と呼ぶ.

上の関係は帰納的に定義されたので，やはり，それに対応する帰納法の証明原理が導かれる.

3.1.3 定理 [簡約関係の導出に関する帰納法]: $\forall a_1, a_2 \in \mathbf{Aexp}.a_1 \longrightarrow a_2 \Longrightarrow P(a_1, a_2)$ を示すには，以下を示せばよい.

1. $\forall a \in \mathbf{Aexp}.P(a_0\ \texttt{+}\ \texttt{Z}, a_0)$
2. $\forall a_1, a_2 \in \mathbf{Aexp}.P(a_1\ \texttt{+}\ \texttt{S(}a_2\texttt{)}, \texttt{S(}a_1\ \texttt{+}\ a_2\texttt{)})$
3. $\forall a \in \mathbf{Aexp}.P(a_0\ \texttt{*}\ \texttt{Z}, \texttt{Z})$
4. $\forall a_1, a_2 \in \mathbf{Aexp}.P(a_1\ \texttt{*}\ \texttt{S(}a_2\texttt{)}, a_1\ \texttt{*}\ a_2\ \texttt{+}\ a_1)$
5. $\forall a, a' \in \mathbf{Aexp}.P(a, a') \Longrightarrow P(\texttt{S(}a\texttt{)}, \texttt{S(}a'\texttt{)})$
6. $\forall a_1, a_1', a_2 \in \mathbf{Aexp}.P(a_1, a_1') \Longrightarrow P(a_1\ \texttt{+}\ a_2, a_1'\ \texttt{+}\ a_2)$
7. $\forall a_1, a_2, a_2' \in \mathbf{Aexp}.P(a_2, a_2') \Longrightarrow P(a_1\ \texttt{+}\ a_2, a_1\ \texttt{+}\ a_2')$
8. $\forall a_1, a_1', a_2 \in \mathbf{Aexp}.P(a_1, a_1') \Longrightarrow P(a_1\ \texttt{*}\ a_2, a_1'\ \texttt{*}\ a_2)$
9. $\forall a_1, a_2, a_2' \in \mathbf{Aexp}.P(a_2, a_2') \Longrightarrow P(a_1\ \texttt{*}\ a_2, a_1\ \texttt{*}\ a_2')$

□

3.2 簡約関係の性質

まず，簡約をしていくと必ずそれ以上計算できない状態になるという，簡約の停止性を証明する.

3.2.1 定義: $a \in \mathbf{Aexp}$ が，$\neg\exists a' \in \mathbf{Aexp}.a \longrightarrow a'$ の時，a を**正規形**$(normal\ form)$ である，という.

3.2.2 定義: 関係 $a \longrightarrow^* a'$ を，$a \longrightarrow^* a' \iff a \underbrace{\longrightarrow \circ \cdots \circ \longrightarrow}_{n} a'$ (ただし $n \geq 0$) である．)で定義する．(このように定義される関係を反射的推移的閉包という．)

3.2.3 定理 [簡約の停止性, termination of evaluation]: 任意の算術式 a に対し，$a \longrightarrow^* a'$ なる正規形の a' が存在する．

Proof: $a \longrightarrow \cdots \longrightarrow a' \longrightarrow \cdots$ なる無限列がないことを示す．まず，$w(a) \in \mathbf{Aexp} \to \mathbf{Nat}$ を以下のように定義する．

1. $w(\mathtt{Z}) = 1$
2. $w(\mathtt{S(}a\mathtt{)}) = w(a) + 1$
3. $w(a_1 \ \mathtt{+} \ a_2) = 2(w(a_1) + w(a_2))$
4. $w(a_1 \ \mathtt{*} \ a_2) = 3 \cdot w(a_1) \cdot w(a_2) + 1$

このとき，$\forall a, a' \in \mathbf{Aexp}.a \longrightarrow a' \Longrightarrow w(a) > w(a')$ である．また $\forall a \in \mathbf{Aexp}.w(a) > 0$ なので，$a \longrightarrow \cdots \longrightarrow a' \longrightarrow \cdots$ が成立したとすると，$w(a) > \cdots > w(a') > \cdots$ なる無限列が存在することになり矛盾. □

次の定理は，簡約の複数の過程で途中経過が異なっても，再び同一の式へと簡約することができる，という性質で**合流性**(*confluence*) と呼ぶ.

3.2.4 定理 [合流性, confluence]: $\forall a_1, a_2, a_3.a_1 \longrightarrow^* a_2 \ \& \ a_1 \longrightarrow^* a_3 \Longrightarrow \exists a_4.a_2 \longrightarrow^* a_4 \ \& \ a_3 \longrightarrow^* a_4$.

Proof: (省略) □

この結果より，ある式を簡約して正規形になっと時には，その形は一意であることがいえる.

3.2.5 系 [正規形の唯一性, uniqueness of normal forms]: $a \longrightarrow^* a'$ かつ $a \longrightarrow^* a''$ かつ a', a'' が正規形ならば，$a' \equiv a''$ である.

4 簡約戦略・eager/lazy な戦略

4.1 簡約戦略の定義

実際の (逐次) プログラミング言語の実行においては，上で定義した簡約関係のように与えられた式に対して複数の実行過程があることはなく，そのうちのひとつに固定されているのが普通である．そのように，式に対して簡約後の式を複数あり得る中からただ一つ定めることを**簡約戦略**(*reduction strategy*) を与えるという.

4.1.1 定義: $F \in \mathbf{Aexp} \rightharpoonup \mathbf{Aexp}$ が簡約関係 $\longrightarrow$ の戦略である，とは 任意の $a \in dom(F)$ について $a \longrightarrow F(a)$ が成立することである．($f \in A \rightharpoonup B$ に対して $dom(f)$ は f の定義域 A を示す．)

ここでは，二種類の戦略を考える．

最初に考える戦略は，多くのプログラミング言語に見られるもので,「演算子の引数はまず『値』になるまで計算する」という戦略である．つまり，(S(Z) * S(Z)) + Z のような式は (式全体が簡約基になっているが)，内側のかけ算から行なっていく．もうひとつの戦略は「演算子の引数は，計算を進めねばならなくなった時に始めて計算する」というもので，(S(Z) + S(Z)) * Z のような式は，S(Z) + S(Z) の計算をせずに (しなくても計算が進められるので) Z に簡約される．それぞれを eager な・lazy な簡約と呼ぶ．lazy な簡約 (の変形) を採用している言語に Haskell がある．

以下では，どちらの戦略も，まず規則を使って算術式上の関係 ($\longrightarrow_e$ と $\longrightarrow_l$) として定義した後，それが戦略となっていることを示す．

4.1.2 定義: 関係 $\longrightarrow_e$ と $\longrightarrow_l$ は，それぞれ図 2 と図 3 の規則で定義される．

ここで，規則のメタ変数を置き換える時には，メタ変数の名前に応じて具体化を行なうことが暗黙のうちに仮定されている．例えば，Ee-Plr を使用するときには， n_1 は上で定義した「自然数」(S(S(...(Z)...))) で具体化しなければならない．

4.1.3 例:

$$\texttt{(S(S(Z)) + Z) * S(Z+S(Z))} \longrightarrow_e \texttt{S(S(Z)) * S(Z+S(Z))}$$

は導出できるが,

$$\texttt{(S(S(Z)) + Z) * S(Z+S(Z))} \longrightarrow_e \texttt{(S(S(Z)) + Z) * S(S(Z+Z))}$$

や

$$\texttt{(S(S(Z)) + Z) * S(Z+S(Z))} \longrightarrow_e \texttt{(S(S(Z)) + Z) * (Z+S(Z)) + S(Z+S(Z))}$$

は導出できない．

$\longrightarrow$ と同様に，この規則より導出に関する帰納法の原理が導かれる．下には $\longrightarrow_e$ についてのみ示す．

4.1.4 定理 [eager 簡約関係の導出に関する帰納法]: $\forall a_1, a_2 \in \mathbf{Aexp}.a_1 \longrightarrow_e a_2 \Longrightarrow P(a_1, a_2)$ を示すには，以下を示せばよい．

1. $\forall n \in \mathbf{Nv}.P(n_0\ \texttt{+}\ \texttt{Z}, n_0)$
2. $\forall n_1, n_2 \in \mathbf{Nv}.P(n_1\ \texttt{+}\ \texttt{S(}n_2\texttt{)}, \texttt{S(}n_1\ \texttt{+}\ n_2\texttt{)})$

$$\frac{}{n_0 \texttt{ + Z} \longrightarrow_e n_0} \text{(Ee-PlZ)} \qquad \frac{a_1 \longrightarrow_e a_1'}{a_1 \texttt{ + } a_2 \longrightarrow_e a_1' \texttt{ + } a_2} \text{(Ee-PlL)}$$

$$\frac{}{n_1 \texttt{ + S(}n_2\texttt{)} \longrightarrow_e \texttt{S(}n_1 \texttt{ + } n_2\texttt{)}} \text{(Ee-PlSc)} \qquad \frac{a_2 \longrightarrow_e a_2'}{n_1 \texttt{ + } a_2 \longrightarrow_e n_1 \texttt{ + } a_2'} \text{(Ee-PlR)}$$

$$\frac{}{n_0 \texttt{ * Z} \longrightarrow_e \texttt{Z}} \text{(Ee-MuZ)}$$

$$\frac{}{n_1 \texttt{ * S(}n_2\texttt{)} \longrightarrow_e n_1 \texttt{ * } n_2 \texttt{ + } n_1} \text{(Ee-MuSc)} \qquad \frac{a_1 \longrightarrow_e a_1'}{a_1 \texttt{ * } a_2 \longrightarrow_e a_1' \texttt{ * } a_2} \text{(Ee-MuL)}$$

$$\frac{a \longrightarrow_e a'}{\texttt{S(}a\texttt{)} \longrightarrow_e \texttt{S(}a'\texttt{)}} \text{(Ee-Succ)} \qquad \frac{a_2 \longrightarrow_e a_2'}{n_1 \texttt{ * } a_2 \longrightarrow_e n_1 \texttt{ * } a_2'} \text{(Ee-MuR)}$$

図 2: eager な簡約

3. $\forall n \in \mathbf{Nv}.P(n_0 \texttt{ * Z}, \texttt{Z})$
4. $\forall n_1, n_2 \in \mathbf{Nv}.P(n_1 \texttt{ * S(}n_2\texttt{)}, n_1 \texttt{ * } n_2 \texttt{ + } n_1)$
5. $\forall a, a' \in \mathbf{Aexp}.P(a, a') \Longrightarrow P(\texttt{S(}a\texttt{)}, \texttt{S(}a'\texttt{)})$
6. $\forall a_1, a_1', a_2 \in \mathbf{Aexp}.P(a_1, a_1') \Longrightarrow P(a_1 \texttt{ + } a_2, a_1' \texttt{ + } a_2)$
7. $\forall n_1 \in NV, a_2, a_2' \in \mathbf{Aexp}.P(a_2, a_2') \Longrightarrow P(n_1 \texttt{ + } a_2, n_1 \texttt{ + } a_2')$
8. $\forall a_1, a_1', a_2 \in \mathbf{Aexp}.P(a_1, a_1') \Longrightarrow P(a_1 \texttt{ * } a_2, a_1' \texttt{ * } a_2)$
9. $\forall n_1 \in NV, a_2, a_2' \in \mathbf{Aexp}.P(a_2, a_2') \Longrightarrow P(n_1 \texttt{ * } a_2, n_1 \texttt{ * } a_2')$

□

4.2 eager / lazy な簡約に関する性質

ここでは $\longrightarrow_e$ が $\longrightarrow$ の戦略になっていることのみを示すが，$\longrightarrow_l$ に関しても同様に示すことができる．

4.2.1 定理: 任意の算術式 a, a' に対して $a \longrightarrow_e a'$ ならば $a \longrightarrow a'$ である．

Proof: $\longrightarrow_e$ を定義する規則のどのインスタンスも，$\longrightarrow$ を定義する規則のインスタンスになるので，ほぼ自明であるが，厳密には $a \longrightarrow_e a'$ の導出に関する帰納法で証明する．$P(a, a')$ は $a \longrightarrow a'$ である． □

$$\frac{}{a_0 \texttt{ + Z} \longrightarrow_l a_0} \quad (\textsc{El-PlZ}) \qquad \frac{a_2 \texttt{+} a_3 \longrightarrow_l a_2'}{a_1 \texttt{ + } (a_2 \texttt{ + } a_3) \longrightarrow_l a_1 \texttt{ + } a_2'} \quad (\textsc{El-PlPl})$$

$$\frac{}{a_1 \texttt{ + S(}a_2\texttt{)} \longrightarrow_l \texttt{S(}a_1 \texttt{ + } a_2\texttt{)}} \quad (\textsc{El-PlSc}) \qquad \frac{a_2 \texttt{*} a_3 \longrightarrow_l a_2'}{a_1 \texttt{ + } a_2 \texttt{ * } a_3 \longrightarrow_l a_1 \texttt{ + } a_2'} \quad (\textsc{El-PlMu})$$

$$\frac{}{a_0 \texttt{ * Z} \longrightarrow_l \texttt{Z}} \quad (\textsc{El-MuZ})$$

$$\frac{}{a_1 \texttt{ * S(}a_2\texttt{)} \longrightarrow_l a_1 \texttt{ * } a_2 \texttt{ + } a_1} \quad (\textsc{El-MuSc}) \qquad \frac{a_2 \texttt{ + } a_3 \longrightarrow_l a_2'}{a_1 \texttt{ * } (a_2 \texttt{ + } a_3) \longrightarrow_l a_1 \texttt{ * } a_2'} \quad (\textsc{El-MuPl})$$

$$\frac{a \longrightarrow_l a'}{\texttt{S(}a\texttt{)} \longrightarrow_l \texttt{S(}a'\texttt{)}} \quad (\textsc{El-Succ}) \qquad \frac{a_2 \texttt{ * } a_3 \longrightarrow_l a_2'}{a_1 \texttt{ * } (a_2 \texttt{ * } a_3) \longrightarrow_l a_1 \texttt{ * } a_2'} \quad (\textsc{El-MuMl})$$

図 3: lazy な簡約

4.2.2 定理 [eager 簡約の決定性]: $\longrightarrow_e \in \mathbf{Aexp} \rightharpoonup \mathbf{Aexp}$ である．すなわち，$\forall a, a', a'' \in \mathbf{Aexp}. a \longrightarrow_e a' \,\&\, a \longrightarrow_e a'' \Longrightarrow a' \equiv a''$.

Proof: これも直感的には，どの規則も結論部の $a \longrightarrow_e a'$ の左側の形に重なりがないので，ほぼ自明に見えるが，厳密には $a \longrightarrow_e a'$ の導出に関する帰納法で証明する．$P(a, a')$ は $\forall a'' \in \mathbf{Aexp}. a \longrightarrow_e a'' \Longrightarrow a' \equiv a''$ である． □