『プログラム言語論』 資料 (2013/04/25, 亀山)

小さなプログラム言語を例として、構文、表示的意味、操作的意味、抽象機械がどのようなものであるか理解しよう。ここでの言語は、足し算、掛け算、自然数定数で構成される「式」がプログラムであるような言語である。式の優先順位をつけるため、括弧を使う。例: (1 + (3 * 2)) + 5

1 構文の定義

構文を、BNF (Backus Normal Form) で定義する。

$$n ::= 0 \mid 1 \mid 2 \mid \cdots \quad \text{(自然数定数)}$$
$$e ::= n \mid (e) \mid e + e \mid e * e \quad \text{(式)}$$

曖昧さの問題: この構文では、$1 + 2 * 3$ の構文が 2 通りに導出される。
改善された構文の定義:

$$n ::= 0 \mid 1 \mid 2 \mid \cdots \quad \text{(自然数定数)}$$
$$e ::= f \mid e + f \text{ (式)}$$
$$f ::= g \mid f * g \text{ (補助的な構文クラス)}$$
$$g ::= n \mid (e) \text{ (補助的な構文クラス)}$$

問題 1. $1 + 2 * (3 + 4) * 5$ が上記の e の定義にあてはまることを導出せよ。また、その導出が 1 通りしかないことを確認せよ。

これまで述べたのは、具体構文 (concrete syntax; あるいは表層構文 surface syntax) である。具体構文では、$1 + 2$ と $((1 + (2)))$ は異なるが (異なる文字列であるが)、これらを構文解析器を通して「木」の形に組みあげると、差異はなくなる。

抽象構文の定義:

$$e ::= \texttt{Int}(n) \mid \texttt{Plus}(e, e) \mid \texttt{Times}(e, e)$$

注. 抽象構文の定義で、Plusとか Timesというタグを使わないといけない、ということはない。普通に、$e ::= n \mid (e + e) \mid (e * e)$ と書いて、これを抽象構文とおもってもよい。(曖昧さがない構文定義であれば、何でもよい。) この原稿では、具体構文と抽象構文のどちらの話をしているかを明確に区別したかったので、$(e + e)$ と書かず、$\texttt{Plus}(e, e)$ と書いた、というだけの理由である。

2 意味の定義 (表示的意味論)

式の意味を決めるにあたり、「式 $1 + 2$ と式 $3 * 1$ は同じ意味を持つ」ように決めたい。表示的意味論の方法では、すべての式を、適当な (数学的にきちんと定まる) 集合の要素に対応付ける形で意味を与える。今回の小さなプログラム言語では、「適当な集合」として、自然数の集合を取れば良いので、そうしてみる。

式 e を自然数に対応付ける。対応付けられた自然数を $[\![e]\!]$ と書くことにする。

$$[\![Int(n)]\!] = n$$
$$[\![Plus(e_1, e_2)]\!] = [\![e_1]\!] \dot{+} [\![e_2]\!]$$
$$[\![Times(e_1, e_2)]\!] = [\![e_1]\!] \dot{*} [\![e_2]\!]$$

ただし、$\dot{+}$ は、式の構文にでてくる $+$ の記号ではなく、2 つの自然数の足し算をあらわす。$\dot{*}$ も同様

例. $[\![\texttt{Plus}(\texttt{Int}(1), \texttt{Times}(\texttt{Int}(2), \texttt{Int}(3)))]\!] = [\![\texttt{Int}(1)]\!] \dot{+} [\![\texttt{TimesInt}(2), \texttt{Int}(3)]\!] = 1 \dot{+} ([\![\texttt{Int}(2)]\!] \dot{*} [\![\texttt{Int}(3)]\!]) = 7$

3　意味の定義 (操作的意味論の 1 つ)

表示的意味論は、式が「結局、何を表しているか」だけを示すものであり、数学的にはこれでよいが、コンピュータ屋としては、式が「どういう風に計算されていって、最終的な答えになるか」という過程も大事にしたい。(また、今回のように単純な言語では、どういう順番に計算しても同じ結果になるが、print 文などの「副作用」がある言語では、式を計算する順番が大事になるので、上記のような単純な形で表示的意味論を与えることはできなくなる。)

ということで、操作的意味論の出番である。ここでは、操作的意味論のなかでも代表的な、構造的操作的意味論 (structural operational semantics) という形で与える。

「式 e を計算すると自然数 n を得る」ことを意味する関係 $e \downarrow n$ の帰納的定義:

$$\frac{}{\mathtt{Int}(n) \downarrow n} \qquad \frac{e_1 \downarrow n_1 \quad e_2 \downarrow n_2 \quad (n_1 \dot{+} n_2 = n)}{\mathtt{Plus}(e_1, e_2) \downarrow n} \qquad \frac{e_1 \downarrow n_1 \quad e_2 \downarrow n_2 \quad (n_1 \dot{*} n_2 = n)}{\mathtt{Times}(e_1, e_2) \downarrow n}$$

例題。$\mathtt{Plus}(\mathtt{Int}(1), \mathtt{Times}(\mathtt{Int}(2), \mathtt{Int}(3))) \downarrow 7$ の導出は以下の通り。

$$\dfrac{\dfrac{}{\mathtt{Int}(1) \downarrow 1} \qquad \dfrac{\dfrac{}{\mathtt{Int}(2) \downarrow 2} \quad \dfrac{}{\mathtt{Int}(3) \downarrow 3} \quad (2 \dot{*} 3 = 6)}{\mathtt{Times}(\mathtt{Int}(2), \mathtt{Int}(3)) \downarrow 6} \qquad (1 \dot{+} 6 = 7)}{\mathtt{Plus}(\mathtt{Int}(1), \mathtt{Times}(\mathtt{Int}(2), \mathtt{Int}(3))) \downarrow 7}$$

4　抽象機械

抽象機械として有名なものは Landin の SECD 機械, Felleisen の CEK 機械であり、他にも多数ある。(3 年次実験 S-8 では Caml Light という言語の基盤になっている、ZINC 機械へのコンパイルを行なう。)

ここでは、CEK 機械を単純化した CK 機械を与えよう。この機械では、C と K が主役である。

- C ... control string (その抽象機械の機械語で書かれたプログラム; ここでは「(抽象構文の) 式」)
- K ... continuation (一般には「継続」と呼ばれるものだが、ここでは単にスタックを表す)

抽象機械の状態: $\langle eval, C, K\rangle$ と $\langle apply, C, K\rangle$ の 2 種類の状態がある。

抽象機械の動作 (状態遷移):

$$\begin{aligned}
\langle eval, \mathtt{Int}(n), K\rangle &\to \langle apply, K, \mathtt{Int}(n)\rangle \\
\langle eval, \mathtt{Plus}(e_1, e_2), K\rangle &\to \langle eval, e_1,\ \mathtt{push}((plus1, e_2), K)\rangle \\
\langle eval, \mathtt{Times}(e_1, e_2), K\rangle &\to \langle eval, e_1,\ \mathtt{push}((times1, e_2), K)\rangle \\
\langle apply, \mathtt{push}((plus1, e), K), n\rangle &\to \langle eval, e, \mathtt{push}((plus2, n), K)\rangle \\
\langle apply, \mathtt{push}((plus2, n_2), K), n_1\rangle &\to \langle apply, K, n\rangle \quad (n_1 \dot{+} n_2 = n) \\
\langle apply, \mathtt{push}((times1, e), K), n\rangle &\to \langle eval, e, \mathtt{push}((times2, n), K)\rangle \\
\langle apply, \mathtt{push}((times2, n_2), K), n_1\rangle &\to \langle apply, K, n\rangle \quad (n_1 \dot{*} n_2 = n) \\
\langle apply, \mathtt{init}, n\rangle &\to n \quad (\text{最終結果})
\end{aligned}$$

計算は $\langle eval, e, \mathtt{init}\rangle$ という状態からはじめる。

この抽象機械は、スタック 1 つを利用して、上記の式の計算を実行する形となっている。$\mathtt{Plus}(e_1, e_2)$ の形の式に対して、e_1 を評価するときは e_2 をスタックに積み、e_2 を評価するときは、e_1 の評価結果 n をスタックに積んでいる。