オブジェクト指向言語のための代入を持つ計算系と同一性

3F−7

大竹 和雄

日本電気(株) C&Cシステムインタフェース技術本部

1 オブジェクト指向とアイデンティティ

オブジェクト指向言語において、オブジェクトは内部状態を持ちこの内部状態が変化しても同一のアイデンティティを保つようなものである。このアイデンティティによってオブジェクト内部の変化にかかわらず、他から同一のものとしてアクセスすることができる。筆者はアイデンティティをオブジェクトの本質的な要素と考え、またアイデンティティを扱うためにオブジェクト指向の計算モデルが手続的である必要性を主張している[2][3]。そのような計算モデルとして本稿では手続き的な代入を持つ計算モデル L-rewriting system を提案する。

2 C-rewriting system

Felleisen 等は手続き的な代入を持つ高階プログラミング言語の計算モデルのために λ-calculus に代入を導入した λ_s-calculus を構築した[1]。この λ_s-calculus を構築するために補助的に用いられている C-rewriting system が本論文の出発点である。C-rewriting system において代入の行なわれる変数は

$$V^n$$

と書かれるラベル n と値 V の組である。これは名前のない変数に相当し、labeled value と呼ぶ。プログラム中の同じラベルを持つ labeled value はどれも同じ変数を表す。labeled value に代入が起きると他の場所で使われている同じ変数の内容がプログラム全体に渡って新しい値に書き換えられる。V^n の V は複雑な構造を持つようなものでもいいので labeled value としてオブジェクトを表現することができ、ラベルをアイデンティティの表現として用いることができる。

オブジェクト指向言語のモデルとして用いようとするときの C-rewrting system の問題点ひとつは labeled value 自身は値とみなされない点である。(これはすべての値が同時に適用可能な関数であるという λ-calculus の性質を保存しようとすることに起因するものであろう。)このことと、項がパラメータとして渡される時には自動的に値となるまで評価される(すなわち call by value)ので labeled value のままでパラメータとして渡すことはできない。従ってアイデンティティを持つオブジェクトとしてとり扱うことがきわめて困難になる。この問題を解決するために次の L-rewriting system を提案する。

3 L-rewriting system

L-rewriting system では後のシンタックスに示すように labeled value V^n も値 V とみなす。これにより labeled value を評価してもその値が取り出されることがなくなるのでそれを明示的に指定できなければならない。また V^n が値であるから特に禁止しない限り関数の位置にも来ることができる。したがってその場合の意味を定めておかなければならない。L-rewriting system ではふたつの問題をひとつにまとめて解決している。すなわち V^nU を V^n の値を取り出すための構文として用いる。このとき値 U は引金として用いられるだけで後は無視される。

シンタックス

L-rewriting system のプログラムを表す項 M は以下のように定義される。

$$\begin{array}{lll} x & : & \text{変数} \\ n & : & \text{ラベル} \\ M & ::= & x \mid \lambda x.M \mid \sigma X.M \mid MM \mid V^n \mid \text{new}\, M \\ V & ::= & (\text{閉包}) \mid V^n \\ X & ::= & x \mid V^n \end{array}$$

ラベルはどのようなものでも良いが変数と明確に区別が付くようにここでは自然数を用いる。V は値を表すもので、ここで閉包とは抽象(すなわち $\lambda x.M$ または $\sigma X.M$)であって自由変数を含まないものである。V^n は labeled value で名前を持たない変数(あるいはポインタ)を抽象化したものである。

代入は $(\sigma U^n.M)V$ という構文で行なわれ、labled value U^n に対して値 V が代入される。M は代入の後でこの代入の式の値を決めるために評価される項である。

計算規則

代入があるので計算順序によって結果が異なってくる。したがって計算規則は計算の順序を決めておく必要がある。計算順序は書き換え可能なリデックスの中で項を木とみなしたときに先行順序で探して最初に見つかったリデックスを書き換えるとする。この計算順序を規定するために以下のように定義されるコンテクスト $C[\,]$ を用いる。

$$C[\,] ::= [\,] \mid V\,C[\,] \mid C[\,]\,M$$

以下の書き換え規則で $C[M]$ はコンテクスト $C[\,]$ に現われる $[\,]$ をすべて M で置き換えた項を表す。

$$C[(\lambda x.M)V] \mapsto C[M[V/x]] \quad (1)$$

$$C[(\sigma U^n.M)V] \mapsto C[M][n := V] \quad (2)$$

$$C[V^nU] \mapsto C[V[n := V]] \quad (3)$$

$$C[\text{new}\, V] \mapsto C[V^n] \quad (n \text{ は新しいラベル}) \quad (4)$$

ここで $M[V/x]$ は M に自由に現われる x をすべて V で置き換えることを表し、$M[n := V]$ は M に現われるすべての U^n というパターン(U は任意)を V^n で置き換えることを表す。$\sigma X.M$ という構文はあくまでも代入を表現するだけのもので、λ抽象とは異なり名前に対する新しいスコープを作らない。代入の効果は上の書き換え規則で明らかなように

A calculus with assignment for object oriented programming and identity of objects
Kazuo Otake
NEC Corporation

$(\sigma U^n.M)V$ に局所的な書換えではなくプログラム全体に渡る書換えとして実現される。すなわち、

$$(\ldots(\sigma U^n.M)V\ldots) \mapsto (\ldots M\ldots)[n := V]$$

のように代入は M の中だけでなく外側の ... においても同時に U^n を V^n で置き換える操作として表現される。

V^nU は labeled value を指している値に置き換えるもので U は単に引きがねとして用いられもので値は無視される。(単に V ではなく $V[n := V]$ である理由は V の中に自己参照がある場合があるからである。これについては後に述べる。) new V は逆に labeled value を作る構文である。

4 翻訳と計算例

手続き的プログラミング言語がどのように L-rewrting system に翻訳されるかを見よう。labeled value は名前を持たない変数であるが、プログラミング言語では変数は名前とともに宣言されることが多い。例えば次のような宣言 (プラス初期化) は以下のように翻訳される。

$$\texttt{int x=0; ...} \Longrightarrow (\lambda x.\ldots)(\text{new}\,0)$$

また C 言語の increment 演算子 ++ は変数を引数に取る次の項に翻訳できる。

$$++ \Longrightarrow \lambda x.(\sigma x.x[])(s(x[]))$$

ここで s は整数に対して 1 を足した値を返す関数の名前とする。また[]は任意の値である。

実行順序と変数への代入に依存する人工的な例として、次のプログラムを考える。(C 言語ではそうではないが引数の評価が左から右へ行なわれるとする)

```
int x=0; second(++x, ++x);
```

は L-rewrting system によって例えば以下のように翻訳し実行される。second は単に二番目の引数の値を返す関数で以下では $\lambda ab.b$ に翻訳されている。

$$\begin{aligned}
&(\lambda x.(\lambda ab.b)((\sigma x.x[])(s(x[])))((\sigma x.x[])(s(x[]))))(\text{new}\,0)\\
&\quad\mapsto (\lambda x.(\lambda ab.b)((\sigma x.x[])(s(x[])))((\sigma x.x[])(s(x[]))))0^n\\
&\quad\mapsto (\lambda ab.b)((\sigma 0^n.0^n[])(s(0^n[])))((\sigma 0^n.0^n[])(s(0^n[])))\\
&\quad\mapsto (\lambda ab.b)((\sigma 0^n.0^n[])(s0))((\sigma 0^n.0^n[])(s(0^n[])))\\
&\quad\mapsto (\lambda ab.b)((\sigma 0^n.0^n[])1)((\sigma 0^n.0^n[])(s(0^n[])))\\
&\quad\mapsto (\lambda ab.b)(1^n[])((\sigma 1^n.1^n[])(s(1^n[])))\\
&\quad\mapsto (\lambda ab.b)1((\sigma 1^n.1^n[])(s(1^n[])))\\
&\quad\mapsto (\lambda b.b)((\sigma 1^n.1^n[])(s(1^n[])))\\
&\quad\mapsto (\lambda b.b)2\\
&\quad\mapsto 2
\end{aligned}$$

5 自己参照を持つ構造

L-rewiting system は代入があるために自己参照を持つような構造が作れる。一般に $(\ldots V^n\ldots)^n$ という形をした項が自己参照を持つ項である。具体的には次のような実行によって自己参照を持つ項が作られる。

$$\begin{aligned}
&(\lambda x.(\sigma x.x)(\lambda a.a1x))(\text{new}[])\\
&\quad\mapsto (\sigma[]^n.[]^n)(\lambda a.a1[]^n)\\
&\quad\mapsto (\lambda a.a1[]^n)^n
\end{aligned}$$

$\lambda a.aUV$ はペア $\langle U,V\rangle$ を λ 項で表現する標準的な方法なので、上の最後の行に得られた項は無限リスト $(111\cdots)$ を表現している。自己参照を持つ構造は値を取り出すことを繰り返しても labeled value がなくならない。実際、この例では一回値の取り出しを行なうことによって次の式を得る。

$$(\lambda a.a1[]^n)^n[] \mapsto \lambda a.a1(\lambda a.a1[]^n)^n$$

再帰的な関数も自己参照を持つ関数として実現できる。

6 考察

外部インタフェース

計算規則の (2) によって項の計算規則全体も文脈依存となるので、項の意味も局所的には定められない。これは手続き的な計算の扱いにくさを形式的に表現していると言える。しかしながら、ある項を実行した時の外から見た効果は最終的な値と labeled value に対するアクセスに集約される。これらが全く同じ項はプログラムとして等しいとみなすことは自然である。すなわち値と labeled value に対するアクセスを項の意味とみなして良い

ところで labeled value は変数を抽象化したものとして導入しているので、項の計算が終った時点で保持している値のみが意味を持つ。一方、最終的な値だけでなく項の計算中に代入された値の履歴に意味を持たせることも可能である。そのような labeled value はメモリマップド I/O を抽象化したものと考えられる。この方法によって入出力を L-rewriting system に導入することができる。

永続オブジェクト

オブジェクトのアイデンティティの問題はひとまとまりの計算と計算の間にわたってアイデンティティが保持されるような永続オブジェクトを想定する場面でよりはっきりと現われてくる。L-rewriting system において永続オブジェクトを扱う一つの方法は次のようなものである。一方、永続オブジェクトを扱うためにすべての labeled value を永続オブジェクトとみなして自動的に保存されるものとする。次の計算で保存されている永続オブジェクトを用いる時には labeled value V^n を実行の開始時に与えられてこれを用いる。(永続オブジェクトを考えない場合は、変数は典型的にはすべて int x; のように宣言によって導入されるので、そのようなプログラムを翻訳した L-rewrting system のプログラムには実行開始時には labeled value V^n は現われない。すべて new V の結果として出現する。)

以上の考察事項の他に、[3] で示した環境を明示的に持つ体系との融合を目指し、オブジェクト指向言語への適用をめざすことが今後の課題である。そのためにはさらに制御構造の部分も取り込んで拡張する必要がある。

参考文献

[1] Felleisen, M. and Friedman, D. P. : A Calculus for Assignments in Higher-Order Languages *Proc. Symp. on Principles on Programming Languages*, pp. 314-325 (1987)

[2] 大竹和雄:「オブジェクト指向のための手続き的計算モデルと型」情報処理学会 記号処理 60-6・プログラミング 2-6 合同研究会 (1991)

[3] 大竹和雄:「環境としてのオブジェクト — オブジェクト指向言語の実行モデルの考察 —」情報処理学会 第 43 回全国大会 講演論文集 (5) pp.221-222 (1991)