概念制約式を用いたプログラミングとプログラム合成*

2B－4

小林弘明† 有田隆也† 川口喜三男† 曽和将容‡
†名古屋工業大学 電気情報工学科
‡電気通信大学 情報システム研究科

1 はじめに

あるプログラムと、そのプログラムが満たす要求仕様とが与えられたとする。この時、要求仕様の変化に応じてプログラムを適切に修正する作業を機械的に行なう、という問題を考えることが出来る。この問題は、下のように2つの部分問題に分けることが出来る。

(a) プログラム本体の“どの部分を”修正するか決定する
(b) (a) の結果を“どのように”修正するか決定する

本稿では (b) について議論する。議論を (b) のみに限定しうる例として、元のプログラムに類似した“プログラムそのもの”を変更要求として用いる場合を取り上げる。このような形態のプログラム修正を“合成”と呼ぶ。この例を通して、機械的に修正可能なプログラムを記述するためにプログラミング言語の側が提供するべき特性(能力)について考察する。そのような特性をプログラミング言語に与えるための仕組みとして、新たに概念制約式を提案する。概念制約式を用いることで“合成”可能なプログラムを記述できる実例をあげる。

2 プログラムの合成

2.1 従来の問題点

下のように3段のパイプラインから構成されるプログラムを考える。

前処理|sort|後処理　(1)

このプログラムに対する変更要求として別のプログラムを一つ与える。

前処理|辞書順 sort|後処理　(2)

これら2つのプログラムを1つに“合成”することを考える。この場合、(1) と (2) との相違点はパイプラインの2段目のみなので、修正の必要な場所は明らかである。残る問題は以下の2点である。

(a) “sort ∩ 辞書順 sort” を計算できない
これは、従来のプログラミング言語が“sort”と“辞書順 sort”という2つの“名前”同士の関連性(類似点/相違点)を定義する機構を持っていないためである。

(b) (1) において、“態度保留”を記述できていない
通常のプログラミング言語では、多くの場合“sort”という記述は特定の実体(サブルーチンやメソッド)に結び付いている。そのため、プログラム中で“sort”と書くことは、“ここではソートを行なう”という決断のみならず、“デフォルトの方法に従う = 辞書順にしない(する)”という余分な決断まで表現してしまうことになる。

2.2 言語が提供するべき特性

2.1節における問題 (a),(b) の解決法を探るため、一つの試みとして以下のような変更を行なう。

(i) sort という“名前”が純粋に“ソートを行なう”という決断のみを表現するよう、言語の中の“名前”の役割を変更する。
(ii) 引数の形で付加的な決断を受け付けるように sort を変更する。(1) には空の引数(=態度保留)を渡し、(2) には“辞書順にする”という決断を引数として与える。

この結果、下のような2つのプログラムを得る。

前処理|sort[ ]|後処理　(3)

前処理|sort[辞書順]|後処理　(4)

この変更により (3) と (4) とを合成することが可能になり、合成の結果は (4) になる。

しかしこの変更だけでは対処できない場合がある。例えば、(4) を下の“英大小文字を区別するようなソート” (5) と“合成”して (6) を得ようとする場合である。

前処理|sort[Case 区別]|後処理　(5)

前処理|sort[辞書順 ∩ Case 区別]|後処理　(6)

この場合、(6) の二つの引数(辞書順,Case 区別)を共存させても矛盾を生じないことを検証する必要がある。

以上の考察から、“合成”を実現するためには、以下の3つの特性を備えたプログラム記述要素でプログラムが書かれている必要があると考えられる。

---

*Combining Programs expressed by Conceptual Qualification
Hiroaki Kobayashi†, Takaya Arita†, Kimio Kawaguchi†, Masahiro Sowa‡
†Nagoya Institute of Technology
‡University of Electro-Communications

(a) “原子的” な決断を表現できること
ただし、“原子的” とは、それ以上分割できない、という意味である。これは、単一の選択に関する決断のみを表現し、余分な選択については “態度保留” を表現する能力が必要であるためである。

(b) 決断に決断を重ねることで、より詳しい決断を生成できること

(c) (b) により変化した部分と変化しなかった部分を区別して表現できること
これは、(b) の際に、変化した部分に重なりが無いことを判定する必要があるためである。

3 概念制約式

3.1 定義

2.2節の要請を満たすものとして、新たに “概念制約式” を提案する。概念制約式とは、概念制約を表現する式のことである。概念制約は、以下の (a),(b) により定義される。すなわち、概念制約全体のなす集合を Cとして、

(a) 概念記号の集合 $A(A \subset C)$

(b) 概念制約間に定義される 5 つの関係

(1) 一般・特殊関係 S
C上の半順序関係。オブジェクト指向プログラミングにおける、クラス・サブクラス関係に相当する。

(2) 大意・属性関係 P
Cと Γ の 2 項関係。ただし、Γ(これを属性列と呼ぶ) は Cの要素 1 個以上からなる列 (無限長も含む) 全体のなす集合である。従来の言語における、レコードとその属性名の関係に相当する。

(3) 属性・内容関係 V
Pと Cの 2 項関係。従来の言語における、レコードの属性名と内容の型との関係に相当する。

(4) 概念差分関係 D_c
Sと Δ の 2 項関係。ただし、Δ (これを差分と呼ぶ) は概念制約の集合族であり、2 つの概念の相違点を表している。

(5) 属性差分関係 D_p
$P \times P \times \Delta$ の部分集合。

A 中の記号は、“原子的” な決断を表現するよう注意して選ばれる。また、A に新たに記号を付け加える際は、その記号に関する一般・特殊関係 S(b1) と概念差分関係 D_c(b4) とを全て定義することをプログラマに義務づける。

3.2 実例

2.2節のプログラム (6) が矛盾のないプログラムであることを言語に伝えるために、以下のような概念制約を定義する。ただし “a.f.g ← h” は

$(\mathtt{f},\mathtt{g}) \in \Gamma$

$(\mathtt{a},(\mathtt{f},\mathtt{g})) \in P$

$((\mathtt{a},(\mathtt{f},\mathtt{g})),\mathtt{h}) \in V$

を意味する。

```
(sort[ ] , sort[辞書順])
            ∈ S
((sort[ ] , sort[辞書順]) , $dX)
            ∈ Dc
(sort[ ] , sort[Case 区別])
            ∈ S
((sort[ ] , sort[Case 区別]) , $dY)
            ∈ Dc

where
 $dX = sort.  並べ方. 文字毎の取り扱い
              . { 英文字 | 数字 | 空白文字 } 以外
              ←無視する
 $dY = sort.  並べ方. 文字毎の取り扱い
              . 英文字.{ 大文字 | 小文字 }
              ←区別する
```

このような概念制約を定義すると、2 つの概念差分 $dX と $dY に重なりが無いことが明らかになる。この結果 (6) を求めることが可能になり、合成の結果は、

```
前処理
| sort [ . 並べ方. 文字毎の取り扱い
         {
          .{ 英文字 | 数字 | 空白文字 } 以外
          ←無視する
          |
          . 英文字.{ 大文字 | 小文字 }
          ←区別する
         }
        ]
|後処理
```

となる。

4 まとめ

“名前” に代る新たなプログラム構成要素として概念制約式を提案し、これを用いて “合成” 可能なプログラムを記述する例をあげた。概念制約式を用いてプログラムを記述することは、“合成” に限らず機械的修正全般の助けとなると考えられる。

今後の課題としては、概念制約に関する推論体系を構築しプログラマの負担を減らして実用性を向上させることがあげられる。