属性文法を基にした制御系向け階層型プログラミング (1)*

6M－7

白倉隆雄†　冨樫陽太　中川裕之

キヤノンソフトウェア株式会社‡ システム研究所

1 はじめに

ソフトウェアに対する要求の正当性や実現可能性を高める手法の1つとして操作的手法が利用されることが多く、これまでにも操作的手法に基づく要求定義法、たとえばプロセスモデルに基づく手法、状態遷移モデルに基づく手法等が提案されてきた。

これらの手法に基づく仕様記述言語は記述性に優れ、システムの動的ふるまいを記述しているという点では比較的実行に有利である。しかし、ソフトウェアの保守作業時に問題となる仕様の可読性については必ずしも優れているとはいいがたかったり、大規模システムに適用する場合、仕様を記述する段階で抽象化やモジュール分割法といった別の方法論の適応が必要になるといった問題がある。そのため我々が記述対象とする制御システム、すなわち制御対象がどのような状態にあるかを判断し、その状態に応じて次の動作を決定するシステムの記述には適切ではない。

一方、コンパイラの記述に用いられる属性文法を基にした仕様記述言語は可読性と表現性をプログラムに与えてきた。これは文法構造と属性が本質的にもっている性質である。属性文法は単にコンパイラの記述だけではなく、階層的な構造をもつ様々なアプリケーションプログラムの記述への試みもなされてきている [1]。

そこで、我々は有限状態機械としてモデル化できる制御対象について、HFP の計算モデル [1] に基づき属性文法を制御システム向けに拡張した実行可能 (またはプログラムの自動生成が可能) な仕様記述方法の提案を行う。

2 記述方法

提案する記述は下記の3構成要素からなる。

1. 宣言の記述
2. 生成規則の記述
3. 意味規則の記述

1では、記述の中で使用する非終端記号・下請関数・遷移関数についての宣言を行う。遷移関数の名前と前提・後提条件は、ペトリネット (状態-事象ネット) を生成するために用いられる。

2.1 宣言部

宣言部は、(%から%) の間に記述され、各記号の記述は、%%ではじまる宣言子で識別する (図1の1～15行目まで)。

非終端記号 (*non-terminal*) は宣言子**%%NONTERM** 以降に、下請関数 (*function*) は宣言子**%%FUNCTION** 以降に記述し、その名前、付随する入力属性・出力属性の名前と型を記述する。構文を下記に示す。

non-terminal ::= #*name* {*decl_attribute*} ;.
function ::= #*name* {*decl_attribute*} ;.
decl_attribute ::= ([{*attributes*}],[{*attribute*}]).
attributes ::= (*attribute*)*.
attribute ::= *type attribute_name.*

遷移関数 (*transition_function*) は宣言子**%%TRANSITION** 以降に記述し、状態遷移を伴う動作を表現するもので、その名前、付随する入力属性・出力属性の名前と型、動作についての前提条件・後提条件を記述する。構文を下記に示す。

transition_function ::=
　　#*name* {*decl_attribute*} *decl_condition* ;.
decl_condition ::=
　　:([{*condition_list*}],[{*condition_list*}]).

たとえば図1の7から8行目の`#move ([int moveArg],[]):([SELECTNO],[SELECTOK]);` は、int 型の入力属性**moveArg** をもち、その前提条件が**SELECTNO**、後提条件が**SELECTOK** であるような遷移関数**move** を宣言している。

2.2 生成規則

属性文法型計算モデルでは、非終端記号をプログラムモジュールと解釈する。形式的には、$M \rightarrow$

*Hierarchical Programming Based on Attribute Grammer for RealTime System (1)

†Takao Shirakura

‡Canon Software inc.,3-9-7 Mita, Minato-ku, Tokyo, Japan

$M_1M_2 \ldots M_n$と表現され、左辺のモジュールMが右辺のn個のモジュールに分割することをあらわす。我々は、これらのモジュールを状態遷移を実現するためのモジュールと読み換え、$M_1M_2 \ldots M_n$は非終端記号または遷移関数で構成されているとする。さらに、制御対象の状態に応じた動作記述のために、状態取得・状態のパターンマッチ・ガード機構を提供する。状態取得は、<>内に記述された下請関数により行う。その機構は、関数型言語のlet式の機能に類似しており、下請関数の結果の値を局所属性の値として保持させる。状態のパターンマッチは、局所属性に保存されている状態と[] 内に記述された状態名とで行う。

たとえば図 1の 20 から 23 行目は、下請関数 pickerstatus により状態を取得し、その状態が [SELECT],[FREE] のどちらにマッチするかを調べ、マッチしたガード式に続く生成規則が評価されるActionという動作をあらわしている。

2.3 意味規則

生成規則に付随する意味規則は、{ と } の間 (図 1の 17〜19 行目まで) に記述し、属性の値を下請関数による関数型の計算で決定する。異なるモジュール (非終端記号・遷移関数) 間で属性名の重複の可能性があるので、属性名だけを属性として扱うのは十分でなく、モジュールの生起 (M_i) における属性 (a) の生起を $M_i.a$ としてあらわし、属性とする。つまり、意味規則は下記の形式で表せる (Fは下請関数)。

$M_i.a = F(M_{k(1)}.b_1, M_{k(2)}.b_2, \ldots, M_{k(r)}.b_r)$

たとえば図 1の 18 行目の move.moveArg = noget (); は、遷移関数moveの入力属性moveArgが下請関数nogetの結果の値として決まることを記述している。

3 記述例

ロボットハンド入れ替えシステムのハンド交換記述 (図 1) を例にとる。

システムを構成する各機器は、下記の動作を行う。

- アームは固定位置で上下動 (up & down) する。
- アームの先端にはハンドをつかんだり (pick)、はなしたり (release) する機構がついている。
- ハンドキャリアは、ハンド番号を入力とし左右動 (move) する。

実際のハンド交換 (図 1の 16 行目) は、下記の手順で行う。

1. アームにハンドが装着されている場合 (SELECT状態) には、1aの処理後2を行う。、装着されていない場合 (FREE状態) には、2の処理を行う。
   (a) アームを下ろして (down)、ハンドをはなし (release) アームを上げる (up)。
2. ハンドキャリアを装着すべきハンドがpick可能な状態にする (move)。装着すべきハンドの番号は、noget関数の返値とする。
3. アームを下ろして (down)、ハンドを装着し (pick) アームを上げる (up)。

```
 1 (%
 2 %%TRANSITION
 3 #up :([DOWN],[UP]);
 4 #down :([UP],[DOWN]);
 5 #release :([DOWN,SELECT],[DOWN,FREE]);
 6 #pick :([DOWN,FREE],[DOWN,SELECT]);
 7 #move ([int moveArg],[])
 8         :([SELECTNO],[SELECTOK]);
 9 %%NONTERM
10 #HandChange;
11 #Action;
12 %%FUNCTION
13 #noget ([],[int syn]);
14 #pickerstatus ([],[int status]);
15 %)
16 HandChange ::= Action move down pick up
17 {
18     move.moveArg = noget();
19 };
20 Action ::= <pickerstatus>
21               [SELECT] down release up
22             | [FREE]
23             ;
```

図 1: 記述例

4 まとめ

属性文法を基に制御システム向けに拡張した実行可能な仕様記述について述べた。状態遷移を伴う処理とその他の処理を区別して記述することで、制御対象の状態遷移が明確になり操作手順の可読性を向上させた。

参考文献

[1] Takuya Katayama: HFP, a hierarchical and functional programming, In Proceeding of the 5th International Conference on Software Engineering, pp. 343-353(1981).

[2] 冨樫・白倉・中川: 属性文法を基にした制御系向け階層型プログラミング (2), 情報処理学会 第 51 回大会 (1995)