1S−1

ITSのためのプログラム生成システムCOMPEXの構築

手塚 祐一†, 池田 満‡, 奥田 健三†
†:宇都宮大学工学部　‡:大阪大学産業科学研究所

1 はじめに

ITS(Intelligent Tutoring System)において知識伝達の効率を向上させるためには,学習者とシステムの知識構造の類似性を高める必要がある[1]. プログラミング教育に関して言えば,課題からプログラムまでの中で利用する知識構造を学習者とシステムにとり相互に対応するように設定することが重要となる. 本報告では,この立場からプログラマがプログラム生成過程で描く概念構造に相当する抽象表現を設定し,それらを媒介とするプログラ生成システムCOMPEX(COgnitive Model of Programming EXpert)について述べる.

2 システムの構成

COMPEXは,Pascalプログラムを生成するシステムである. 本システムの基本的な構成を図1に示す.

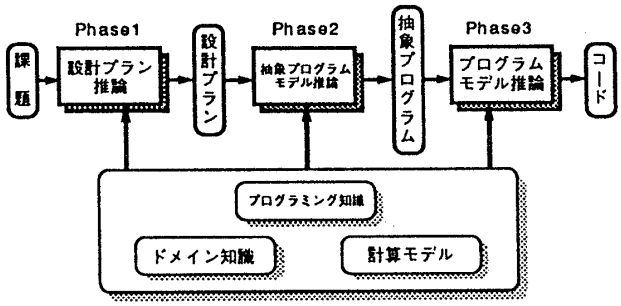

図1: COMPEXのシステム構成

COMPEXは,次の3つのモデルを用いてプログラムを生成する. 課題を記述する課題モデル,課題の操作を抽象的に定式化した抽象プログラムモデル,プログラム言語に依存したプログラムモデルである. 各モデルは,デバイスの集合によって定義される. デバイスは計算モデルの基本機能を提供する概念である. 例えば,プログラムモデルでは,演算処理,条件分岐,反復,といった機能が要求される. デバイスは,このような機能の内の一つ以上を提供するものである. デバイスは部品によって構成され,プログラマのプログラム生成過程で描く知識構造を捉えるのに利用する.

それぞれのモデルの表現について以下で述べる.

3 課題モデル

課題モデルには,プログラム作成のための十分な情報を記述することが要求される. 課題モデルの知識表現には,集合論的表現を採用している. 例えば,「文字列中の空白以外の字数を数えなさい」という課題は,集合論的表現で図2のように表現される. 記述形式は,操作記述と制約記述から構成される. 操作記述は,入力に対して制約を満たす操作をしたものが出力となることを表現している.

```
S1:{string}
ops(count,S1,N,Ca)
Ca:{belong(&X,S1) and notEqual(&X,' ')}
```

図2: 課題記述の例

この例では,” 文字列S1の要素のうち制約Ca(&XはS1の要素かつ空白でない)を満たす要素の集合の個数をNとする” ことを意味する. ここで, &Xは集合Xの1要素を表現している.

4 抽象プログラムモデル

課題モデルとプログラムモデル間の中間的な認識レベルでのプログラムの表現形式を抽象プログラムと呼ぶことにする. 抽象プログラムは,課題モデルに含まれる概念構造とプログラムの構造の対応づけに役立つ.

本研究では,プログラムの課題が持つデータ構造と制御構造を1次元配列・ループ・分岐に限定して,抽象プログラムの構造を設定している. COMPEXではこの内,列構造の逐次処理を表現する抽象プログラムの構造をストリームというデバイスとして実現した. ストリームの概念を図3に示す.

図3の例では,逐次処理の対象となった列構造として文字列S1がストリームと結び付き,逐次処理の対象となる文字&Xがストリームの部品Windowと結び付いている. 逐次処理のための初期化,終了処理,Windowの更新処理は,ストリームの部品Enter,Exit,Nextと対応付けられる. このストリームにより,文字列中の文字を走査する操作scanが実現される(図4参照).

COMPEX : A Program Generating System for ITS
Yuichi TEZUKA† , Mitsuru IKEDA‡ , Kenzou OKUDA†
†:Faculty of Engineering, Utsunomiya University ‡:I.S.I.R., Osaka University

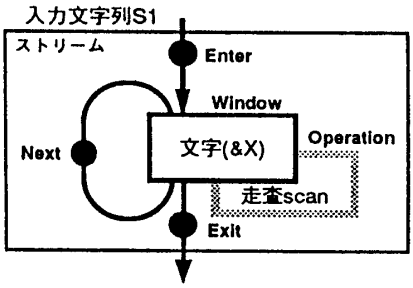

図 3: 抽象プログラム：ストリーム表現

5 プログラムモデル

プログラムモデルでは，プログラミング言語が規定する構文・意味規則からデバイス・部品・役割 (機能) を抽出し，これらを用いて定義する．例えば，ループの機能を持つデバイスとして，for 文,repeat 文,while 文に対応するデバイスが用意される．それぞれのデバイスは，構成要素と意味から各部品 (初期状態, 終了状態, ループ条件等) を設定している．

6 COMPEX の推論メカニズム

COMPEX の推論メカニズムは，以下に示す 3 つのフェーズからなる．

Phase1 課題から設計プランの生成: 課題から問題解決の設計プランを立てる．例えば，数字の系列の総和を求める操作は，数字を逐次走査する操作と数字を加算する操作に変換する．設計プランは，図 4 のような操作系列として表現される．操作系列とは，逐次的に処理が可能な操作の系列であり，操作の適用順序を示す．図 2 の仕様に適用されるルールの一例を示す．

```
if    ops(Ope,Input,Output,Const),
      入力 Input が列構造の性質を持つ,
      操作 Ope が逐次処理の操作系列 S を適用可能である
then 操作系列 S の各操作を設計プラン op として設定する
```

図 2 の課題に対してルールが適用されると，文字列 S1 を対象に制約 Const を満たす要素を数える操作 count は，文字列の中で制約を満たす要素を走査する scan と単純に要素を数える increment の 2 つの操作系列に変換される．また，操作 scan はより詳細な操作に変換され，単純に文字列を走査する scan と制約を満たす要素を取り出す check の 2 つの操作系列にわかれる．最終的に 図 2 の課題上の操作 (count) が，図 4 の操作系列 (scan,check,increment) に変換される．図 2 の課題に対する設計プランを 図 4 に示す．

Phase2 抽象プログラムの生成: 設計プランから 図 3 に示した形式の抽象プログラムを生成する．ストリームの各部分の制御ポイントに基づいて制御単位を設定し，その機能を抽象プログラムの語彙を用いて記述する．以下に設計プランからストリームを生成するルールの一例を示す．

```
op(scan,S1,&X,C1)        :文字列から要素を走査
C1:{belong(&X,S1)}
op(check,&X,&Y,C2 )      :要素が空白以外かチェック
C2:{notEqual(&X,' ')}     (&Y は空白以外の文字)
op(increment,Y,N,C3)     :要素&Y の個数をカウント
C3:{}
```

図 4: 課題から得られる設計プランの例

```
if    op(Ope,Input,Output,Const),
      列構造を持つ Input からその要素 Output を
      走査する操作 Ope である
then ストリーム全体の制御単位を設定する
```

このルールでは，列構造の逐次処理から 図 3 に示したようなストリームの全体を設定する．この例では，それぞれの設計プランの語彙の機能に従って文字列 S1 に対して Enter が入力処理 enter_process(string),Exit が出力処理 exit_process(string),Window が文字列 S1 の文字 character,Operation が走査 scan,Next が Window を更新する処理 next(character) が設定される．

また，ストリームの制御単位の一部を設定したり，複数のストリーム間の関係を設定するルール等もある．ストリーム間の関係を設定するルールは，プログラム構造 (ブロック構造等) を捉えるために必要となる．

Phase3 プログラムモデルの生成: ここでは，プログラミング言語の定義に基づいて変換ルールによって抽象プログラムをプログラムモデルに変換する．抽象プログラム上での機能を実現するためのプログラムモデルのデバイスが複数存在する場合 (例えば，ループの機能に対して for 文，repeat 文，while 文) は，それぞれのデバイスを用いたプログラムの生成が可能となる．

7 おわりに

本稿では，プログラム生成のための知識の構造を考察し，抽象プログラムを介した生成システムを提案した．抽象プログラムを媒介としたプログラム生成では，課題から直接プログラムコードへ対応付ける Johnson らの PROUST[2] にくらべ，より人間に近いプログラミングプロセスを表現できると考えられる．

今後の研究では，知識の蓄積・拡充につとめる必要がある．また，いろいろなケースに対するバグの生成に関しても検討すべき問題であると考えられる．

参考文献

[1] Wenger,E. : "Artificial intelligence and tutoring systems", Morgan Kaufmann Pub., Los Altos (1987)

[2] Johnson,W.L. et al. : "PROUST:Knowledge-Based Program Understanding", IEEE Trans. on Soft. Eng., Vol.SE-11, No.3, pp.11-19(1985)