3U－8

フラグメントの部分間差分を用いた例からのプログラム自動合成

山口 文彦† 中西 正和‡
慶應義塾大学大学院理工学研究科計算機科学科

1. はじめに

Summers はS式をS式に変換するLISPのプログラムを、その入出力例から合成する一般的な手法を提案した[1]。そこでは、複数の例の間の構造的な差異を発見することで、特殊なヒューリスティックを含まないという意味で一般的な関数の合成を可能にしている。また彼の手法はその手続き中に探索をほとんど含まないという特徴を持つ。

しかし、Summers の手法では例の間の差異から帰納的関係を仮定するので、適当な順序に従って入出力例を与える必要がある。入出力例に対するこの条件は、目的とする関数に対する知識を仮定したもので、自動プログラミングという観点からは好ましくない。

本稿では構造的な差異を例と例からではなく一つの例の部分と部分から発見することで、与える例が必要とする条件を緩和し、自動プログラミングシステムとしてより扱い易いものにすることを考える。

2. 対象領域

1個の入出力関係を基本関数の合成によって記述したものを入出力フラグメントと呼ぶ(図1参照)。

入出力例:
$(A\ B\ C) \rightarrow ((A)(B)(C))$

入出力フラグメント:
$\lambda x.(cons(cons(car\ \ x)nil)$
$(cons(cons(car(cdr\ x))nil)$
$(cons(cons(car(cdr(cdr\ x)))nil)$
$nil)))$

図 1: 入出力例と入出力フラグメント

本稿では単数または複数の入出力フラグメントから帰納関数を推論する問題を扱う。

入出力フラグメントを容易に構築できる領域として、S式をS式に変換するLISPの関数を取り上げる。基本関数として *car*、*cdr*、*cons* の3つを用いる。また、帰納関数を構成する際には、適当な条件文および述語が必要となる。ここでは、条件文として *cond* 文を、述語として *atom* を用いる。

3. フラグメントの部分間差分

二つの n 変数入出力フラグメント F_1、F_2 と任意の入力 $x_1, \cdots, x_n$ に対し、

$$(F_2\ x_1\ \cdots\ x_n) = (a\ \ (F_1\ (b_1\ x_1)\ \cdots\ (b_n\ x_n))\ \ x_1\ \cdots\ x_n)$$

となる $n+1$ 変数入出力フラグメント a および1変数入出力フラグメント $b_1 \cdots b_n$ を求めることを差分をとるといい、a を F_2 の F_1 に対する差分フラグメント、$b_1 \cdots b_n$ を引渡しフラグメントという(図2参照)。

$F_1 = \lambda x.(cons\ (car\ x)\ nil)$
$F_2 = \lambda x.(cons\ (car\ (cdr\ x))\ nil)$
$F_2 = F_1(cdr\ x)$
i.e. $a = Id., b = cdr$

図 2: フラグメント間差分

関数が帰納的に定義されているとき、その条件文(および述語)を評価すると(基本)関数が繰り返して適用された形が得られる。例えば関数 f が以下のように定義されているとき、

$$(f\ x) = \begin{cases} (g\ x) & if(p\ x) \\ (h\ (f\ (d\ x))\ x) & otherwise \end{cases}$$

Program Synthesis from Examples using Differences between Parts of a Fragment
Fumihiko YAMAGUCHI†
Masakazu NAKANISHI‡
Department of Computer Science, Graduate School of Science and Technology, Keio University 3-14-1 Hiyoshi, Yokohama, Kanagawa 223, Japan

ある入力 a に対する出力 $(f\ a)$ は、例えば以下のようになる。

$$\begin{array}{l}(f\ a) = (h\quad (h\quad \ldots (h\quad (g\ (d\ (d\ \ldots\ (d\ a))))\\ \qquad\qquad\qquad\qquad\qquad\quad (d\ \ldots\ (d\ a)))\\ \qquad\qquad\qquad\quad \ldots\ (d\quad a))\\ \qquad\qquad a)\end{array}$$

入出力例 $a \rightarrow (f\ a)$ から作られた入出力フラグメントがこのような形をしているとき、h、g、d であるようなフラグメントおよび適当な述語 p を発見して関数の定義を推測する。

繰り返しの発見は、全体との差分をとることのできる部分を探すことを基本とする。フラグメント $\boldsymbol{F}$ とその部分フラグメント $\boldsymbol{G}$ との差分をとることを考える。これを以下の手順に従って行なう。

例
$\boldsymbol{F}=$
$\lambda x.(cons(cons(car\ \ x)nil)$
$\quad (cons(cons(car(cdr\ x))nil)$
$\quad\quad (cons(cons(car(cdr(cdr\ x)))nil)$
$\quad\quad\quad nil)))$
$\boldsymbol{G}=$
$\lambda x.(cons(cons(car(cdr\ x))nil)$
$\quad (cons(cons(car(cdr(cdr\ x)))nil)$
$\quad\quad nil))$

1. $\boldsymbol{F}$中の $\boldsymbol{G}$ を Self に置き換えたものを $\boldsymbol{F'}$ とする

$$\boldsymbol{F'} = \lambda x.(cons(cons(car\ x)\ nil)\ \texttt{Self})$$

2. $\boldsymbol{F}$中で $\boldsymbol{G}$ の出現する位置と $\boldsymbol{G}$ 中で同じ位置にあるものを Self に置き換えたものを $\boldsymbol{G'}$ とする

$$\boldsymbol{G'} = \lambda x.(cons(cons(car(cdr\ x))\ nil)\ \texttt{Self})$$

3. $\boldsymbol{F'}$ と $\boldsymbol{G'}$ の変数を重複のないように呼び換えて両者の最汎単一化代入子 θ を求める。

$$\theta = [x_F := (cdr\ x_G)]$$

ここで θ が存在するとき、差分をとることができるといい、$\boldsymbol{F'}$ が差分フラグメントを、θ が引渡しフラグメントを表す。このように差分をとることのできる部分フラグメントを発見できたとき、$\boldsymbol{F}$ を帰納的に定義しうると推論する。

$$\boldsymbol{F} \simeq \lambda x.(cons(cons(car\ x)\ nil)(\boldsymbol{F}\ (cdr\ x)))$$

4. 矛盾点の修正

合成された関数に入力例を適用して入出力フラグメントと異なるものが得られる場合、そのとき関数に与えられた入力と出力すべきフラグメントから矛盾しないための入出力例を得る。

入出力例: $(A\ B\ C) \rightarrow ((A)(B)(C))$
$\boldsymbol{F} = \lambda x.(cons(cons(car\ x)\ nil)(\boldsymbol{F}\ (cdr\ x)))$
⇒
$(\boldsymbol{F}\ (cdr\ (cdr\ (cdr\ x)))) = nil$ となるべき

合成された関数と入出力フラグメントとが矛盾しない場合の入力と矛盾する場合の入力とを区別する述語を発見し、条件文を用いて関数を修正する。

$(atom\ '(A\ B\ C)) \neq T$
$(atom(cdr\ '(A\ B\ C))) \neq T$
$(atom(cdr(cdr\ '(A\ B\ C)))) \neq T$
$(atom(cdr(cdr(cdr\ '(A\ B\ C))))) = T$
⇒
$\boldsymbol{F}=$
$\lambda x.(cond((atom\ x)\ nil)$
$\quad (t(cons(cons(car\ x)\ nil)(\boldsymbol{F}\ (cdr\ x))))$
$\quad)$

この修正法は複数の例が与えられた場合にも適用されうる。

5. 実装

以上のような例からのプログラム自動合成システムを AUSTIN KYOTO COMMON LISP 上で実装した。

参考文献

1. PHILLIP D. SUMMERS: *A Methodology for LISP Program Construction from Examples*, JACM, Vol. 24, No. 1, pp. 161–175, January 1977.
2. ANGLUIN, D., AND SMITH C. H.: *Inductive Inference: Theory and Methods*, ACM Comput. Surv., Vol. 15, No. 3, pp. 237–269, Sep., 1983.
3. E. Y. SHAPIRO 著 有川 節夫 訳: 知識の帰納的推論, 共立出版, 知識情報処理シリーズ 第3巻, 1986.